



**Behavioral Synthesis
Reference Manual
(Version 6.2)**

NEC Confidential

Copyright © 1988-2017 NEC Corporation. All rights reserved.

- The information obtained from this manual shall not be disclosed outside the scope of agreement.
- This document shall not be reproduced, altered, reprinted, distributed and disclosed without prior permission from NEC in writing.
- NEC shall not guarantee or grant license for the Intellectual Property or other rights belonging to it or any third party (company) for the usage of products or information mentioned in this manual. NEC shall not own any responsibility in event of any problem arising owing to violation of rights of any third party caused by the above-mentioned usage.
- This manual can be modified without any prior notice. Hence confirmation of latest version should be performed while using the same.
- This product may fall under restricted items according to regulation of Foreign Exchange and Foreign Trade Control Law, therefore caution is required.
- CyberWorkBench is a registered trademark of NEC Corporation. Any other product name, company name are the registered trademarks or brands of their respective company.

SCAD-CY-1400009

The following notations are used in this manual to indicate the option etc. of a command.

Notation	Meaning of the notation
#	Indicates the numeric character
?	Indicates the character string
(a b)	Any of either a or b can be specified
[c]	c indicates that omission is possible
[:d...]	d can be specified multiple times using ":" as a separator
value	Indicates the variable name <i>value</i> , which is specified by architect

Frequently used variable names		
Variable name	Meaning of variable	Type
<i>filename</i>	File name	Character string
<i>filename</i> [.IFF]	File name, where extension ".IFF" can be omitted	Character string
<i>directory</i>	Directory name	Character string
<i>word</i>	Optional character string	Character string
<i>comment</i>	Comment	Character string
<i>name</i>	Name	Character string
<i>kind</i>	Kind	Character string
<i>unit</i>	Unit specification of delay	Character string
<i>id</i>	Label for time constraint	Character string
<i>interval</i>	Format for time constraint	Character string
<i>font</i>	Font name	Character string
<i>st1</i>	State number	Numeric character
<i>st2</i>	State number	Numeric character
<i>stn</i>	State count	Numeric character

Table of Contents

Table of Contents	4
1 Introduction	26
1.1 Overview	26
1.2 System Design and Synthesis flow	26
1.3 Synthesized Circuit Structure	29
2 Synthesis Modes and Constraint Modes.....	32
2.1 Synthesis Modes	32
2.1.1 Manual Scheduling	32
2.1.2 Automatic Scheduling.....	33
2.2 Constraint Modes	33
2.2.1 Functional unit constraint-None-use mode/Use-mode	35
2.2.2 Memory constraint generation/Use mode	36
2.3 Summary	37
3 Attributes	38
3.1 Overview	38
3.2 Attributes.....	38
3.3 Placement of Attribute Description	39
3.4 Macro substitution of Attribute.....	41
3.5 Constant substitution of attribute value	43
3.6 Attributes and Options.....	44
3.6.1 Order of Priority	44
3.7 Summary	47
4 Options Specification	48
4.1 Overview	48
4.1.1 “bdltran” –Options	48
4.2 Option Specification Order - Exceptions	48
4.3 Default Option Specification File.....	49
4.4 Parameter File.....	50
4.5 Input File Specification.....	50
4.6 Summary	52
5 License related Options.....	53
5.1 License Retry functionality	53
6 Synthesis Circuit Specification	54
6.1 Overview	54
6.2 Physical Type Variables	54
6.2.1 Physical type variables in Manual scheduling mode.....	54
6.3 Bit Order Specification - Ascending/Descending	55
6.3.1 Related Options.....	55

6.3.2	Related attributes	55
6.3.3	Input BDL – Specification Order.....	56
6.3.4	RTL Output – Specification Order.....	56
6.4	Signed / Unsigned Specification.....	57
6.4.1	Related Options.....	57
6.4.2	Description	57
6.5	Carry-Over in Calculation	57
6.5.1	Description	57
6.5.2	Operation in which carry is considered.....	58
6.6	Operation Bit Width - Specification.....	58
6.6.1	Related Attributes.....	58
6.6.2	Description	58
6.7	Signal Name	59
6.7.1	Related Options.....	59
6.7.2	Signal Name - Prefix Specification.....	60
6.7.3	Functional Unit Name - Prefix Specification.....	60
6.7.4	Synonym Variable.....	63
6.7.5	Unrolling Structure Variable	63
6.8	Comment	63
6.8.1	Related Attributes.....	63
6.8.2	Description	64
6.9	Options for Sub-system	66
6.9.1	Related Options.....	66
6.9.2	Options for equivalence verification tool bdlcmp.....	66
6.10	Synthesis Constraint Option	66
6.10.1	Related Options.....	66
6.10.2	Synthesis time limit.....	67
6.10.3	Limit of total number of functional unit constraints	67
6.10.4	Restrictions.....	68
6.11	Summary.....	69
7	Input Output Ports	70
7.1	Overview	70
7.1.1	Related options	70
7.1.2	Related Attributes.....	72
7.2	Port Sequence.....	73
7.3	Single Port Sharing	74
7.4	Negative Logic Port Specification	75
7.5	Output Variable Reference	76
7.6	HiZ (high impedance) Output	76
7.7	Port Generation - State Number Output	76
7.8	Unused Ports Handling	76
7.9	Output Ports - Specifying Default Value	78

7.10 Port Validating Signal (s)	79
7.11 Declaration of valid signal explicitly (Functionality for simulation) ..	82
7.11.1 Related attribute	82
7.11.2 Basic usage.....	82
7.11.3 Simulation of multiple processes combined with sub-tool.....	83
7.11.4 Applicable notation of valid_sig(_neg)_bind attribute.....	85
7.11.5 Notes and limitations.....	89
7.12 Wait valid signal in SystemC	90
7.13 Output port attributes of “out var” variable.....	91
7.14 Generation of synchronization circuit between out-of-phase clocks	93
7.14.1 Synchronization circuit generation function between out-of-phase clocks ..	94
7.14.2 Function that generates synchronization circuit between out-of-phase clocks at the input port.....	97
7.15 Register generation in input output port	98
7.16 Summary.....	100
8 Clock	101
8.1 Overview	101
8.1.1 Related Options.....	101
8.1.2 Related Attributes.....	102
8.2 Clock Cycle Constraint Specification.....	102
8.3 Clock Port Name Specification.....	103
8.4 Clock Signal Specification	105
8.5 Summary.....	106
9 Reset	107
9.1 Overview	107
9.1.1 Related Options.....	107
9.1.2 Related Attributes.....	109
9.2 Reset Port Specification	111
9.3 Reset techniques	113
9.3.1 Fully asynchronous/synchronous reset.....	113
9.3.2 Reset Synchronizer.....	116
9.4 Reset State Specification	119
9.5 Specification of default reset.....	122
9.6 Reset Method Specification of Register with Initial Value.....	123
9.7 Register wise specification of reset signal and logic	125
9.8 Specification of register without reset.....	126
9.9 Reset Generation - Specific Output.....	126
9.9.1 Outline	126
9.9.2 Restrictions.....	127
9.10 Summary.....	128
10 Handling start and completion of Process Function	130

10.1 Overview	130
10.1.1 Related Options	130
10.1.2 Related Attributes	130
10.2 Variable Declaration	131
10.2.1 Overview	131
10.2.2 Variable Initialization - Timing Change	132
10.2.3 Initialization Circuit Generation	132
10.3 Reset Value Specification Method	133
10.3.1 Overview	133
10.3.2 Related Attributes	134
10.3.3 Description	134
10.4 Handling of Process Function Completion	135
10.4.1 Overview	135
10.4.2 Destination State - Execution Ends Neither Through Exit nor Return Statements	135
10.4.3 Destination State - Execution Ends Through Return Statement	135
10.4.4 Destination State on Exit	135
10.4.5 Value of each variable at process function completion	136
10.5 Summary	138
11 Array / Memory	139
11.1 Overview	139
11.1.1 Related Options	139
11.1.2 Related Attributes	145
11.2 Synthesis of Arrays	150
11.2.1 Memory	150
11.2.2 Decoder Enabled Register Array	151
11.2.3 Combinatorial Circuits	152
11.2.4 Variable Unrolling/Expansion	152
11.3 Array Synthesis Specification	154
11.3.1 Specifying Memory	154
11.3.2 Internal Memory/ External Memory Specification	155
11.3.3 Decoder Enabled Register Array - Specification Method	156
11.3.4 Combinatorial Circuit - Specification Method	157
11.3.5 Variable Unrolling/Expansion - Specification	158
11.3.6 Array Variable Synthesis - Automatic Decision	159
11.4 Array Variable Specification	160
11.4.1 Multidimensional Arrays - Partial Unrolling	160
11.4.2 Multidimensional Arrays - Partitioned Memory	161
11.4.2.1 In case array for dimension accessed by all constant subscripts is unrolled	161
11.4.2.2 In case array related to other dimensions are unrolled	162
11.4.3 Converting to One-Dimensional Arrays - Subscript Specification	164
11.4.4 Subscript Specification - Array Reference/ Assignment	167

11.4.5 Exclusion Specification - Array Reference/ Assignment in Loop Folding.....	169
11.4.6 Merging Multiple Arrays.....	169
11.4.7 Speculative Execution Specification	171
11.4.8 Converting Array Accesses Using Switch.....	173
11.4.9 Initial Value of an Array.....	176
11.4.10 Variable Name after Unrolling the Array	176
11.5 Detailed Memory Specification	177
11.5.1 Port Specification of Memory.....	177
11.5.1.1 Read-Write Port – Data Port	178
11.5.1.2 Read-Write Port – Read-Enable Port	179
11.5.1.3 Chip-Select Port.....	180
11.5.1.4 Byte-Enable Port	181
11.5.1.5 Clock Port for Synchronous SRAM.....	183
11.5.1.6 Treatment of Unused Ports	184
11.5.2 Specification related to chaining of memory access.....	184
11.6 Access Timings Specifications	185
11.6.1 Synchronous read and write memory (Pipeline memory).....	187
11.6.1.1 Specification Method of Pipeline Memory	189
11.6.1.2 Limitations	190
11.6.1.3 Pipeline Memory Access by Automatic Control	190
11.6.1.4 Pipeline Memory Access by Manual Control	193
11.6.1.5 Pipeline Memory of Manual Scheduling Mode of CWB5.1.2 and prior	199
11.6.2 Asynchronous read and synchronous write memory	209
11.6.2.1 Specification method of asynchronous read and synchronous write memory	209
11.6.3 Asynchronous read asynchronous write memory	211
11.6.3.1 Specification of memory access timing of manual scheduling	211
11.6.3.2 Specification of memory access timing of automatic scheduling	213
11.7 Insert Register in Pipeline Memory Port	217
11.7.1 Note	219
11.8 Synthesis function for shared memory without read enable and shared register array.....	220
11.8.1 Overview	220
11.8.2 Related option	221
11.8.3 Description	221
11.9 Connection specification of control signal, reset, clock of memory	223
11.10 Decoder enabled Register Array - Detailed Specification	227
11.10.1 Specification of the Maximum Number of Decoders	227
11.10.2 Decoder Number Specification	229

11.10.3 Decode Delay of decoder enabled register array - Specification	230
11.10.4 Partitions of the Decoder Register Array - Function Specification	231
11.11 Universal Memory Interface Specification.....	233
11.11.1 Overview	233
11.11.2 Related Attributes.....	233
11.11.3 Port types of MLIB.....	233
11.11.4 Memory Read Timing.....	235
11.11.5 Memory Write Timing.....	236
11.11.6 Behavioral Specifications on using Stall Control	237
11.11.7 Example of description by BDL of Memory Controller.....	237
11.11.8 Limitations.....	240
11.12 Generation of Memory Description for FPGA Logical Synthesis	241
11.12.1 Overview	241
11.12.2 Related attribute	241
11.12.3 MEM_SYNTH Entry for MLIB.....	241
11.12.4 Example from the Action Description till Logical Synthesis	242
11.12.5 Limitations	244
11.13 Summary	245
12 External/ Shared Memory, Shared register / ter / var, External functional unit.....	246
12.1 Overview.....	246
12.2 External/ Shared memory	246
12.3 Shared register/ ter/var	246
12.3.1 Overview	246
12.3.2 Initial value of shared register/var	248
12.3.3 Activate Clock signal/Reset signal of shared register/var	249
12.4 Value of array/shared register when not used	250
12.5 External functional unit	251
12.6 Summary.....	253
13 Pointer.....	254
13.1 Overview.....	254
13.2 Related option.....	254
13.3 Description	254
13.4 Restrictions	254
13.5 Summary.....	254
14 Structure	256
14.1 Overview.....	256
14.1.1 Related Options.....	256
14.1.2 Related Attributes.....	256
14.2 Variable Name after unrolling Structured Variable	256
14.3 Attribute packed Synthesis of Structured Variable	258
14.3.1 Notes in Hierarchical Description.....	260

14.3.2 Notes in User-Defined Functional Unit	260
14.3.3 Restrictions.....	261
15 Function	264
15.1 Overview.....	264
15.1.1 Related Options.....	264
15.1.2 Related Attributes.....	265
15.1.3 Function Description Restrictions.....	266
15.2 Function Parameter Types	266
15.3 Type of Function Return Value	267
15.4 Synthesis Method	267
15.4.1 Inline Expansion	267
15.4.2 Goto Conversion	269
15.4.3 Conversion of a Function into a Functional Unit	269
15.4.4 Function Synthesis Method Specification	269
15.5 Conversion of User defined functional unit	272
15.5.1 Overview	272
15.5.2 Synthesis options.....	273
15.5.3 Target Function Synthesis - Specification Method.....	275
15.5.3.1 Attribute specified for function synthesized as functional unit	
276	
15.5.3.2 Individual specification of synthesis method of function	
targeted for functional unit conversion.....	276
15.5.3.3 Individual specification of generated circuit.....	277
15.5.3.4 Port name of automatically generated port	278
15.5.4 Functional Unit Count File (lower module specification file)	279
15.5.5 Target Function Instance specification.....	279
15.5.6 Target Function Global Variables - Access Mode Specification	279
15.5.7 Option specification if stall signal generation of function targeted for	
functional unit conversion.....	280
15.5.8 Static Variables of Target Function	280
15.5.9 Calling of Function for Converting Functional Unit (in case of new	
function/top module).....	281
15.5.10 Input Values - Default Specification	281
15.5.11 Parent Function Conversion to a Functional Unit	283
15.5.12 Restrictions.....	283
15.6 Initialization of Local Variable	290
15.7 Summary.....	290
16 for loop	292
16.1 Overview.....	292
16.1.1 Related Options.....	292
16.1.2 Related Attributes.....	293
16.1.3 "for" loop with fixed repeat count.....	294

16.2 "for" Loop unrolling	295
16.2.1 Unrolling Types	295
16.2.2 Specification Method	296
16.3 Optimization of Operations	298
16.3.1 Related Options.....	298
16.3.2 Related Attributes.....	298
16.3.3 Description	299
16.4 Merge Capability of 'for' Loop.....	301
16.4.1 Related Options.....	301
16.4.2 Related Attributes.....	301
16.4.3 Description	302
16.4.4 Conditions for Merging.....	304
16.4.4.1 Determining whether the Format is same for "for" Loops.....	305
16.4.4.2 Determining whether loop execution depends only on the loop return condition	306
16.4.4.3 Determining Whether the Execution Count of Entire Merged Loop is the Same	306
16.4.4.4 Deciding Condition to Save Data Dependency Relation within the Loop	307
16.4.4.5 Retaining data dependency Relation between loops.....	308
16.5 Summary.....	310
17 Loop Folding.....	311
17.1 Common Scheduling Modes.....	311
17.1.1 Overview	311
17.1.2 Related Attribute.....	315
17.1.3 Constraints	315
17.2 Automatic Scheduling mode	317
17.2.1 Related Options.....	317
17.2.2 Related Attributes.....	318
17.2.3 Overview	320
17.2.4 Register Hazard	321
17.2.5 I/O port hazard	323
17.2.6 Array Hazard.....	325
17.2.7 Scheduling constraint setting during specification of AVOID	329
17.2.8 Loop folding break operation.....	330
17.2.9 Loop folding wait operation.....	332
17.2.10 Release of Nested loop folding.....	333
17.2.11 Restrictions in auto scheduling mode	337
17.3 Manual Scheduling Mode	339
17.3.1 Related Attributes.....	339
17.3.2 Overview	339
17.3.3 Loop Folding of "while" Statement.....	342

17.3.4 Loop Folding of "for" Statement.....	344
17.3.5 Loop Folding of 'do- while' Statement - 1.....	346
17.3.6 Loop Folding of 'do- while' Statement - 2.....	347
17.3.7 Loop Folding and Loop Unrolling of 'for' loop	349
17.3.8 Array Reference - Specifying the Exclusivity of Subscript	350
17.3.9 Restrictions in manual scheduling mode	352
17.3.10 "break" statement in the loop folding	355
17.3.11 Loop folding break statement limitations	356
17.3.12 "wait" Operation in loop folding.....	358
17.3.13 Multiple "wait" Loop Folding Operation.....	361
17.3.14 Operation for Coexistence of "wait" and "break" Loop Folding	362
17.3.15 Limitations for Loop Folding "wait"	363
17.4 Foldable loop in SystemC description.....	366
17.4.1 Foldable loop in auto scheduling mode.....	366
17.4.2 Loop folding stall description in auto scheduling mode.....	367
17.4.3 Foldable loop in manual scheduling mode	369
17.5 Summary.....	370
18 watch statement	371
18.1 Overview.....	371
18.1.1 Related Attributes.....	371
18.1.2 "watch" statement and I/O variable - dependency	371
18.2 Summary.....	372
19 Automatic Scheduling	373
19.1 Overview.....	373
19.2 Input /output timing specification	373
19.2.1 Overview	373
19.2.2 Related Options.....	373
19.2.3 Related Attribute.....	374
19.2.4 Description	374
19.3 Speculative Execution based Scheduling	376
19.3.1 Overview	376
19.3.2 Related options	376
19.3.3 Related attributes	376
19.3.4 Description	376
19.3.5 Restrictions.....	379
19.4 Conditional exclusion based on data dependency	379
19.4.1 Related attributes	379
19.4.2 Description	379
19.5 Comprehensive Parallelization of 'if' Statement	381
19.5.1 Overview	381
19.5.2 Related Options.....	381
19.5.3 Description	381
19.6 Conditional Branch Scheduling.....	385

19.6.1 Overview	385
19.6.2 Related Options.....	385
19.6.3 Description.....	385
19.7 Register Based Scheduling	387
19.7.1 Overview	387
19.7.2 Related Options.....	387
19.7.3 Related Attributes.....	387
19.7.4 Description.....	387
19.8 Scheduling Considering Basic Library component Delay	390
19.8.1 Overview	390
19.8.2 Related Options.....	390
19.8.3 Description	390
19.8.4 Restrictions.....	390
19.9 Dependency relation of array and I/O specification	391
19.9.1 Overview	391
19.9.2 Related Options.....	391
19.9.3 Description	391
19.9.4 Restrictions.....	391
19.10 Dependency relationship of array, I/O Specification	392
19.10.1 Related attributes	392
19.10.2 Description	392
19.11 Summary	395
20 Resource Allocation	396
20.1 Overview.....	396
20.1.1 Related Options.....	396
20.1.2 Related Attributes.....	399
20.2 Signed, unsigned in Resource Allocation	400
20.2.1 Signed and Unsigned Variables in the Register Sharing	401
20.2.2 Signed and Unsigned Variables in Functional Unit Allocation	401
20.3 Sharing of Register during Manual Scheduling	402
20.4 Register Sharing During Auto Scheduling	403
20.5 Register Sharing among Variable with different bit width	403
20.6 Register Name	405
20.7 Sharing of Functional Unit and Memory allocation	407
20.8 Destination Registers Specification for Variables	411
20.8.1 Basic Function	411
20.8.2 Function available when specified in Local Variable within Function	411
20.9 Specifying Destination Memory for Array	412
20.10 Specification of functional unit type in operator allocation.....	415
20.10.1 Detailed specification method of allocated functional unit Detailed specification method o.....	415
20.11 Summary	417

21 Time Constraints	418
21.1 Overview	418
21.2 Scheduling Block	418
21.2.1 Overview	418
21.2.2 Related Options.....	419
21.2.3 Related Attributes.....	419
21.2.4 Overview	420
21.2.5 Limitations.....	421
21.2.6 Usage Example	422
21.3 Time Constraint Specification for Port	425
21.3.1 Overview	425
21.3.2 Description.....	425
21.3.3 Points to be noted.....	426
21.3.4 Limitations.....	427
21.4 Specifying the latency constraints of loop	428
21.4.1 Overview	428
21.4.2 Related attributes	428
21.4.3 Description.....	429
21.4.4 Limitations	429
21.5 Specifying time constraints of array access	430
21.5.1 Related attributes	430
21.5.2 Specifying time constraints for consecutive array accesses (interval_array).....	432
21.5.3 Time constraints for all array accesses (group_interval_array)	432
21.5.4 Difference between group_interval_array and interval_array	432
21.6 Specifying time constraint using id	433
21.6.1 Related attributes	433
21.6.2 id attribute usage method	436
21.6.2.1 id attribute of statement	436
21.6.2.2 id attribute of element.....	437
21.6.2.3 id attribute of assignment destination	437
21.6.3 Specifying step time constrains between ids (interval_id)	437
21.6.4 Specifying path time constraint between ids (path_interval_id).....	439
21.7 Time constraint specification (TLMT) file	442
21.7.1 Related options	442
21.7.2 Description.....	442
21.8 Time constraint violation	443
21.8.1 Description	443
21.8.2 Message output at the time of time constraint violation	443
21.8.3 Support at the time of time constraint violation	443
22 Automatic Pipeline Scheduling	446
22.1 Overview	446
22.1.1 Related Options.....	447
22.1.2 Related Attributes.....	447

22.2 Pipeline circuit.....	447
22.3 Restrictions of Automatic pipeline scheduling	449
22.3.1 Restrictions Regarding Control Structure.....	449
22.3.2 Restrictions Regarding Pipeline Structure Hazards	450
22.3.3 Synthesis Specific Restrictions of Automatic Pipelined circuits	450
22.4 Forwarding Register	451
22.4.1 Suppression of the Forwarding Registers	454
22.5 Structural Hazards.....	455
22.5.1 Structural Hazards of Functional units	455
22.5.2 Structural Hazards of Input and Output Ports.....	455
22.5.3 Avoiding Structural Hazards for Input and Output Ports.....	456
22.5.4 Suppressing Detection of Structural Hazards of Input/Output Port.....	458
22.5.5 Structural Hazards of Memory Port.....	458
22.5.6 Avoiding Structural Hazards of Memory Ports.....	459
22.6 Data Hazards of Memory Port	459
22.6.1 Overview	459
22.6.2 Options and Attributes to counter data hazards.....	460
22.7 Summary.....	461
23 Stall Functionality.....	462
23.1 Overview.....	462
23.1.1 Related option	462
23.1.2 Related attributes	462
23.2 BDL Description for Stall Functionality.....	462
23.3 Stall - Overview of the Operation	463
23.4 Memory Operations during Stall - Synchronous Pipeline Memory..	463
23.5 Memory Operations during stall - Synchronized Non-Pipelined Memory.....	464
23.6 Input Output Operations during stall	464
23.7 Operation of Built-in Operator during Stall	465
23.8 Operation for Converting Functional Unit for User-Defined Functions during stall	467
23.9 Pipeline Stall Compensated Circuit.....	468
23.10 Summary	469
24 Hierarchical Setting.....	471
24.1 Overview	471
24.2 Related options	471
24.3 Recommended Synthesis Flow.....	471
24.4 Merits of Bottom-Up Synthesis	472
24.4.1 Port Consistency Check and Adjustment Functionality	473
24.4.2 Synthesis Area Reduction effect	475
24.5 Synthesis functionality for lower module	478
24.5.1 Related option	478

24.5.2 Description	478
25 Constraint File Generation	479
25.1 Overview	479
25.1.1 Related Options.....	479
25.2 Functional unit constraint: None-use/Use mode and Memory constraint: None-use/use mode	482
25.2.1 Functional unit constraint- None-use/use-mode	484
25.2.2 Memory Constraint: None-use/Use-mode	485
25.3 Generation of Functional Unit Library (FLIB) file and Functional Unit Count (FCNT) files	487
25.3.1 Overview	487
25.3.2 Related options	487
25.3.3 Generation of FCNT file	495
25.4 Generation of Memory Library (MLIB) file, Memory Count (MCNT) file.....	499
25.4.1 Overview	499
25.4.2 Related Option	499
25.4.3 Related Attributes.....	500
25.4.4 MCNT File Generation	500
25.4.5 Generation of MLIB, MCNT File.....	503
25.4.6 MCNT File Generation in case of Hierarchical Description.....	506
25.4.7 Precautions.....	507
25.5 Summary.....	507
26 MultiCycle functional unit/ MultiCycle Memory	509
26.1 Overview	509
26.1.1 Related Options.....	509
26.1.2 Related Attributes.....	509
26.2 Pipeline Functional Unit	510
26.3 Multicycle Functional Unit/Multicycle Memory	515
26.3.1 Specification Method in Manual Scheduling	515
26.3.2 Specification Method in Automatic Scheduling	517
26.3.3 Implementation Method of Multicycle Functional Unit/Memory in Automatic Scheduling.....	519
26.3.3.1 Multiplexer Select Signal	519
26.3.3.2 Input Variable Directly Used in MultiCycle Functional Unit /Memory	521
26.4 Arithmetic Macro Functional Unit.....	522
26.4.1 Related attributes	522
26.4.2 Overview	523
26.4.3 Synthesis Method of Arithmetic Macro Functional Unit	523
26.4.3.1 Attribute Specification of target Operational expression	523

26.4.3.2	FLIB file, FCNT file specification for Arithmetic Macro Functional Unit	524
26.4.4	Points of Concern	524
26.5 Summary	525
27 Language Level Conversion Function	526
27.1 Overview	526
27.2 Invariant Expression in Loops	526
27.2.1	Related Options.....	526
27.2.2	Description	526
27.3 Common Sub-Expressions Elimination	528
27.3.1	Related Options.....	528
27.3.2	Related Attributes.....	529
27.3.3	Description	529
27.4 Functionality to reduce the stages of operation tree	533
27.4.1	Related option	533
27.4.2	Related Attributes.....	533
27.4.3	Description	533
27.4.4	Points to be noted.....	534
27.5 Bit Width Reduction Function for Variables	535
27.5.1	Related Option	535
27.5.2	Related Attributes.....	536
27.5.3	Overview	536
27.5.4	Specification of Variable for Bit Width Reduction.....	537
27.5.5	Input Range Specification of Input Variable	538
27.5.6	Display of variables reduced	539
27.5.7	Restrictions.....	539
27.6 Summary	541
28 Logical level conversion functionalities	542
28.1 Overview	542
28.1.1	Related Options.....	542
28.1.2	Related Attributes.....	543
28.2 Merging Similar Condition of Multiplexers	543
28.3 Simplification of Logic Condition of Multiplexers	544
28.4 Distribution of “ter variable” in Manual Scheduling	545
28.5 Deletion of Unreferenced Registers	547
28.6 Converting a register variable, having an assignment only in the initialized declaration, to a constant	548
28.7 Generation of Rewrite Circuit of Register Value	549
28.8 Optimization of Redundant Logical Functions	550
28.9 Flattening of multistage multiplexer	551
28.10 Summary	552
29 Combinatorial Loop	554

29.1 Overview	554
29.1.1 Related Options	555
29.2 "True" Loop Detection in Manual scheduling	556
29.3 False loop, Combinatorial Loop Detection Function	557
29.4 False Loop Avoidance Scheduling	558
29.5 Loop Avoidance in Resource Allocation	560
29.6 Summary	561
30 Gated clock circuit	562
30.1 Overview	562
30.1.1 Related options	562
30.1.2 Related Attributes	562
30.2 Circuit for Gated Clock	563
30.3 Output format of Gated Clock Circuit and Register	563
30.4 Summary	567
31 Generation of Wrapper for Bypassing Internal Memory	568
31.1 Functional Overview	568
31.2 Related options	568
31.3 Remarks	569
31.4 Summary	569
32 Interrupt ("allstates" Statement)	570
32.1 Overview	570
32.2 Interrupt Using the "allstates" Statement	570
32.3 Operations Executed at every Step in Automatic Scheduling	571
32.4 Restrictions of "allstates" Statement	571
32.5 Summary	574
33 FSM generation	575
33.1 Overview	575
33.1.1 Related Options	575
33.2 Handling of Invalid States	575
33.3 Partition of State Register	576
33.4 Summary	578
34 Macro Option	579
34.1 Overview	579
34.2 Related Options	579
34.3 Description	579
34.4 Summary	580
35 Dump Output	581
35.1 Overview	581
35.1.1 Related Attributes	581
35.2 Input Description	581
35.3 Handling of "dump" attribute in Verilog-HDL output	582

35.3.1 Related Options.....	582
35.3.2 Functional Description.....	582
35.3.3 Example of dump output.....	583
35.4 Handling of “dump” attribute in VHDL output	583
35.4.1 Option.....	583
35.4.2 Example of “dump” Output	584
35.4.3 Constraints	584
35.5 Summary.....	584
36 Error / Warning	585
36.1 Overview	585
36.1.1 Related Options.....	585
36.1.2 Related Attributes.....	588
36.2 Types of Error and Warning.....	588
36.3 Changing the message language.....	590
36.4 Error Level Control	591
36.5 Functionality to Check Multiple Assignment in Manual Scheduling	591
36.6 Functionality to check ‘nmux’ statement of continuous assignment	593
36.7 Bit Width Check Function of XOR Operation	595
36.8 Function for Checking the Bit Width of Allocated Functional unit ..	596
36.9 Function to Check Attributes	597
36.10 Functionality to Check the Portion of Behavior Deviating from C Language in Manual Scheduling	599
36.10.1 Usage.....	599
36.10.2 Function Overview.....	600
36.10.3 BDL Rules for Using This Function	601
36.10.4 Appendix.....	602
36.10.4.1 Collection of BDL Description Techniques	602
36.10.4.2 Collection of error messages generated in Clog file	602
36.11 Functionality to check Delay Violation Path in Manual Scheduling	604
36.11.1 Overview	604
36.11.2 Usage.....	604
36.11.3 Restrictions.....	605
36.12 Functionality to generate Warning about Description influenced by Specification Changes in BDL version 3.0	606
36.12.1 Overview	606
36.12.2 Warning List related to specification change in BDL3.0	607
36.12.3 Restrictions.....	610
36.13 Functionality to warn any adverse effect of &&, 	610
36.14 Summary	611
37 Information File	613

37.1 Overview.....	613
37.1.1 Related Options.....	613
37.2 List of Output Information File	616
37.3 Summary File (.SUMM)	617
37.4 Error File (.err)	623
37.5 Log File (.LOG.gz)	623
37.6 Lower module specification File (.LMSPEC)	624
37.7 Hint file (.tips)	624
37.8 Synthesis Information Files (.CSV Format).....	624
37.9 BDL File for Analysis (.BDL)	626
37.10 Resource Allocation Information File (.INF)	629
37.10.1 Functional Unit Usage (Operator Usage Table)	631
37.10.2 Memory and Register Array Usage (Memory Usage Table).....	632
37.10.3 Module Information (Module Usage Table).....	633
37.10.4 Register Transfer Information (Register Transfer Table).....	633
37.10.5 Terminal Transfer Information (Terminal Transfer Table).....	634
37.10.6 Condition Information (Condition Table)	635
37.10.7 Input Output Information (IO Usage table).....	636
37.11 Circuit Quality Report (.QOR/.QOR.HTML).....	637
37.11.1 Summary.....	637
37.11.1.1 Latency Index	641
37.11.2 Important Information.....	641
37.11.3 Delay Information.....	643
37.11.4 Input output/Memory Information.....	643
37.11.5 Functional Unit Information	644
37.11.6 Register/Multiplexer/Decoder Information	645
37.11.7 Detailed Delay Information.....	647
37.11.8 Wiring Information.....	650
37.11.8.1 Wiring Information	650
37.11.8.2 Register "fan-in / fan-out" Information	652
37.11.8.3 Wiring Characteristics/Routability Information	653
37.11.8.4 Count of Wiring Information.....	654
37.11.8.5 Count of Wiring Characteristics Information/Routability	655
37.11.9 Loop Latency Information	656
37.11.10 Text Version Circuit Quality Report (.QOR).....	657
37.11.11 Options Concerning the Circuit Quality Report	670
37.12 State Transition Diagram file (.dty) in Graphviz Format.....	678
37.13 Power consumption evaluation file for FPGA (_pwr.csv,_pwr.xpe)	680
37.13.1 Xilinx Power consumption Evaluation (XPower Estimator)	680
37.13.2 Altera Power consumption evaluation (PowerPlay Early Power Estimator). 682	682
37.13.3 Important Points	684
37.14 File Output Destination Directory Specification	685

37.15 Summary	686
38 CDFG Window	687
38.1 Overview	687
38.1.1 Related Options.....	687
38.2 Operations during Display	687
38.3 Summary.....	689
39 File specification	690
39.1 Overview	690
39.1.1 Related options	690
39.2 Functional Unit Constraint (LMT) File	692
39.3 Memory Constraint (MLMT) File	696
39.4 Port Constraint (PLMT) File	698
39.5 Port Relation (PREL) file	700
39.6 Basic Library (BLIB) File.....	701
39.6.1 Basic Library (BLIB) file	701
39.6.1.1 Overview	701
39.6.1.2 Basic library element entry	703
39.7 Functional Unit Library (FLIB) File.....	705
39.7.1 Overview	706
39.7.2 Functional Unit Entry.....	707
39.8 Functional Unit Count (FCNT) File	712
39.8.1 Overview	713
39.8.2 Functional Unit Count Entry	714
39.9 Lower module specification file	715
39.9.1 Overview	716
39.9.2 Entry Content	717
39.10 Memory Library (MLIB) File.....	718
39.10.1 Overview	720
39.10.2 Memory Entry.....	720
39.11 Memory Count (MCNT) File	727
39.11.1 Overview	727
39.11.2 Memory count entry.....	728
39.12 Summary	731
A. Logical Synthesis Script Generation Tool	732
A.1. Functional Overview	732
A.2. Outline flow	732
A.3. Usage	733
A.4. Logical synthesis method with Generated file	740
A.5. Timing Exception Path	741
A.5.1 Synthesis Procedure.....	742
A.5.2 Limitations.....	743

A.6. Points to be noted while editing the synthesis script (Template) .	745
A.7. Other Points to be noted	745
B. BDL Input Command	746
B.1. Functional Overview	746
B.2. Usage	746
B.3. Alarm Output Control Option (-w, -Wall)	747
B.4. BDL Output Option (-OB)	747
B.5. Error File Control Option (-EF0)	748
B.6. Code Related Option (-Zmix_signed, -Zunsigned)	748
B.7. Ascending/ Descending Bit Order Specification Option (-Zdefault_bit_order)	748
B.8. Ascending/ Descending Bit Order Specification Attribute (default_bit_order)	749
B.9. Checking option of Descending Bit Order Description (-Zchk_bit_description)	749
B.10. Parameter File Input Option (-p, -p:)	749
B.11. assert function option (-assert_on, -assert_off)	750
B.12. Error check control of attribute made for source code encryption tool	750
B.13. include Path	751
B.14. Multiple Process and fragmented compilation	751
B.15. Character code Limitations	752
C. VHDL Output Command	753
C.1. Functional Overview	753
C.2. Usage	753
C.3. Circuit Configuration	760
C.4. Output Files	760
C.5. Regarding “-nmux” Option	761
C.6. Regarding -case_default_x=YES NO Option	762
C.7. Regarding “-dec” Option	763
C.8. Black Box Hierarchy	764
C.9. Register Output Format	764
C.10. Pipeline Functional Unit Simulation Model	764
D. Verilog-HDL Output Command	765
D.1 Functional Overview	765
D.2 Usage	765
D.3 Circuit Configuration	772
D.4 Output file	774
D.5 Regarding “-nmux” option	775
D.6 Regarding -case_default_x=YES NO Option	777

D.7	Regarding -keep_starc_rule Option	778
D.8	Regarding “–dec” option.....	778
D.9	Black Box Hierarchy	779
D.10	Regarding Register Output Format	779
D.11	Pipeline Functional Unit Simulation Model	780
D.12	Regarding Absolute Delay Insertion in Register Assignment Statement	780
E.	Combinational Loop Detection Command.....	782
E.1.	Functional Overview	782
E.2.	Usage	782
E.3.	Functional Description	782
F.	Overflow Detection Tool	787
F.1.	Functional Overview	787
F.2.	Usage	787
F.3.	Functional Description	787
F.4.	Warning Control Options	789
F.5.	Report File Output Option	789
G.	Signal Display Tool.....	791
G.1.	Functional Overview	791
G.2.	Usage	791
G.3.	Functional description	792
H.	Basic Library Generation Tool	794
H.1.	Functional Overview	794
H.2.	Usage	794
H.3.	Functionality Description	798
H.4.	User Script File Specification	798
H.5.	Setting regarding Logical Synthesis Tool.....	798
I.	Functional Unit Library Delay and Area Setting Tool.....	799
I.1.	Functional Overview	799
I.2.	Usage	799
I.3.	Functionality Description	803
I.4.	Standard Functional Unit Library (Standard FLIB)	804
I.5.	Estimation Mode.....	806
I.6.	Delay Specification.....	806
I.7.	Functional Unit Chain Effect Specification.....	807
I.8.	Delay Constraints in Functional Unit Synthesis	808
I.9.	Pipeline Functional Unit Support	808
I.10.	Settings related to Logical Synthesis Tool	812
I.10.1	Delay Unit of Report Result of Logical Synthesis Tool	812
I.10.2	64 bit mode specification of logic synthesis tool.....	812

I.10.3 User Script File Specification	812
I.10.4 Obtaining License of Logical Synthesis Tool	813
I.10.5 Settings related to Logic Synthesis Tool for FPGA	813
I.11. Points to be noted.....	814
I.12. Limitations.....	814
J. Memory library delay area setting tool	815
J.1. Functional overview.....	815
J.2. Usage	815
J.3. Functionality Description	817
J.4. Settings related to logic sysnthesis tool.....	817
K. BDL Output Tool	818
K.1. Functional Overview	818
K.2. Usage	818
K.3. Function Description	819
L. Top Hierarchy Generation Tool.....	821
L.1. Functional Overview	821
L.2. Usage	821
L.3. Generated File Name and Higher Hierarchy.....	824
L.4. Port Connections between Lower Hierarchy	824
L.5. Variable Connection in Behavioral Level Mode	825
L.5.1 Functional Description.....	825
L.5.2 Connection of in/out variable.....	826
L.5.2.1 Connection of Non Structural Variables.....	826
L.5.2.2 Connection of Structural Variable	829
L.5.3 Connection of "shared" variable	833
L.5.3.1 Connection of unstructured variable	833
L.5.3.2 Connection of Structured Variable.....	833
L.5.4 MLIB/MCNT Template Generation.....	837
L.5.5 Connection between Ports with Different Name	838
L.5.6 Multiple Instance Generation Functionality.....	839
L.5.6.1 Overview	839
L.5.6.2 Related attributes	840
L.5.6.3 Description	840
L.5.6.4 Usage example.....	844
L.6. Variable connection in structural level Mode	848
L.6.1 Connection of shared reg	848
L.6.2 Connection of "outside mem" and "shared mem" variables	849
L.6.2.1 "outside mem" variable	849
L.6.2.2 "Shared mem" Variable	850
L.6.3 Connection of "out" port of same name	854
L.6.4 Monitor tap Output Specification	857

L.7. Limitations.....	858
L.7.1. Bidirectional ports are not supported.....	858
L.7.2. shared var/ ter variable in structural level mode are not supported	859
M. Topology checker among hierarchies (Connection system design rule check)	860
 M.1. Functional overview.....	860
 M.2. Usage	860
 M.3. Functional description.....	860
N. Source Code Encryption Tool	862
 N.1. Source Code Encryption Tool Functionality.....	862
O. Attribute List.....	864
 O.1 Attributes added in items.....	864
 O.2 Attributes added in process function.....	865
 O.3 Attributes added in sentence	865
 O.4 Attributes added in signal declaration.....	866
 O.5 Attributes added in Function definition, Function declaration	869
 O.6 Attributes added in Block.....	870
 O.7 Attributes for SystemC.....	870

1 Introduction

Cyber is a behavioral synthesis system that can be used to generate hardware implementation for a system. It takes behavioral description in Behavior Description Language (BDL) or System C as input. Then, it generates the RTL description for this input. The RTL description can be used for processing various logical synthesis tools. The processing of the logical synthesis tools are done for design generation in the specified language, design validation, optimizations, model simulations, functional and formal verifications, and actual gate-level synthesis. The gate-level synthesis is sent to foundry for actual system device fabrication.

BDL is a subset of "C" language with extensions to describe hardware specific details, such as bit width etc. (See "BDL Reference Manual").

1.1 Overview

This section covers the following concepts:

- The main system architecture and components of Cyber.
- The structure of synthesized circuit.
- Two types of synthesis mode: Manual scheduling mode and Automatic scheduling mode.

1.2 System Design and Synthesis flow

Figure 1 shows the main system architecture of behavioral synthesis system Cyber.

Figure 2 shows the system architecture of various tools included in Cyber.

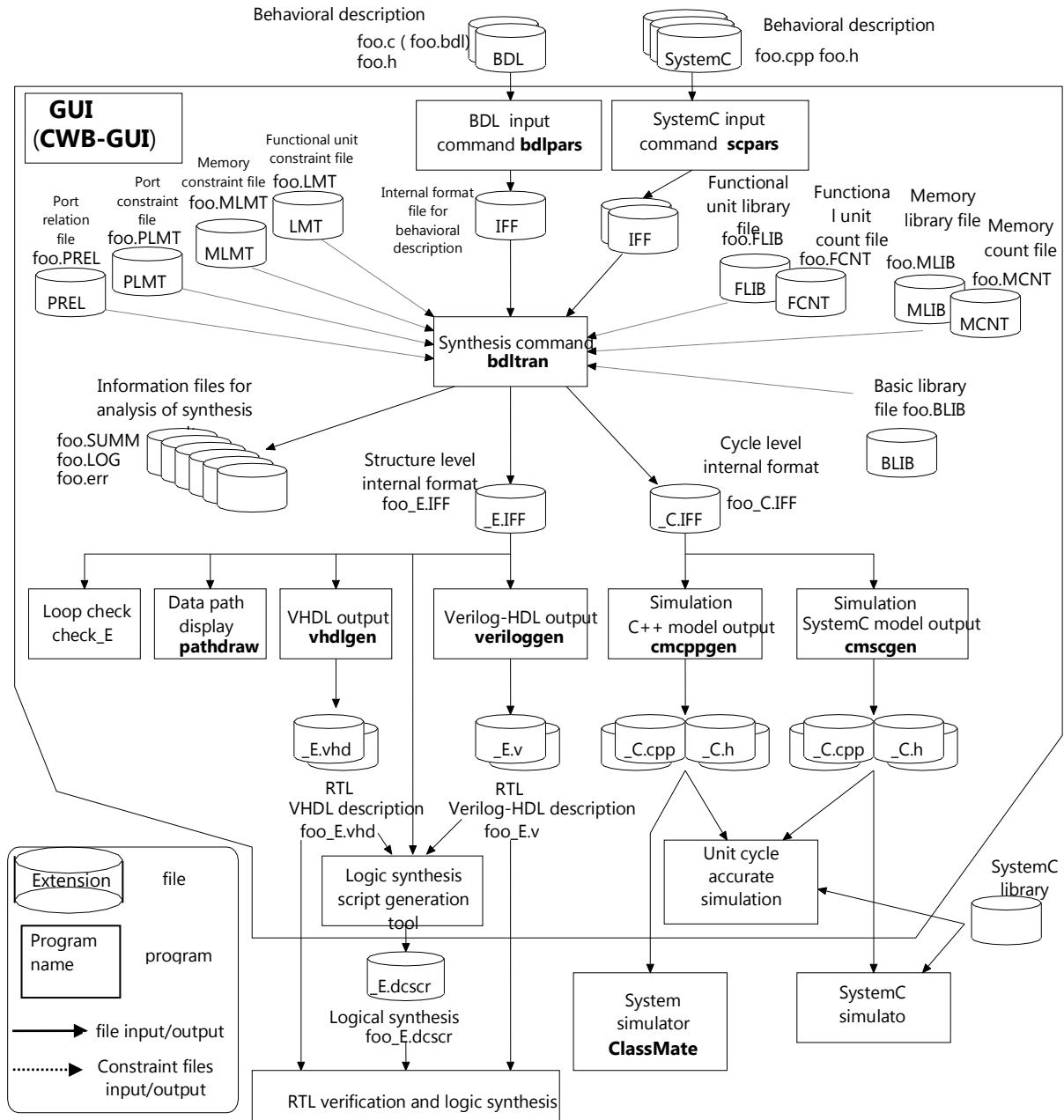


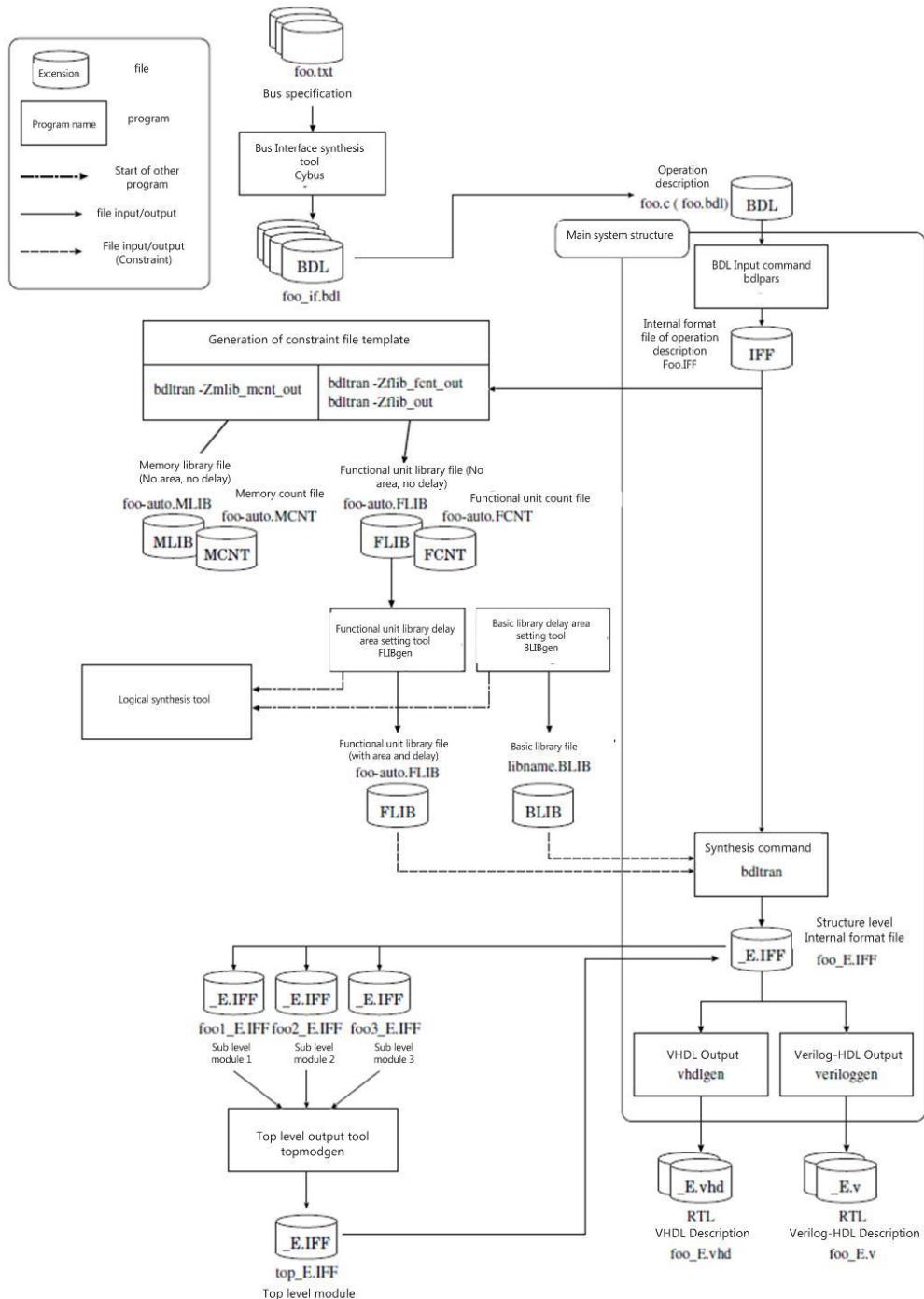
Figure 1: Cyber - Main System Architecture

Cyber primarily consists of the following five components:

Cyber Component	Function
bdlpars, scpars, rtlpars	These components are the tools to convert behavioral description to the internal format.

Cyber Component	Function
Bdltran	This component is a tool for carrying out behavioral synthesis.
vhdlgen, veriloggen, cmveriloggen, cmspeccgen	These components are the tools used to generate synthesized RTL circuit description or cycle accurate simulation model description in each language.
cwbexplorer, iff2bdl, check_E, check_overflow, mkmfsm, signallist, FLIBgen, BLIBgen, MLIBgen, topmodgen, scmodgen, bmcppgen, LSscrgen, tbgen, cybus, CybusM, bdldraw, pathdraw, pathcheck	These components are other sub tools.
CWB-GUI	This component is the Graphical User Interface (GUI) to manage the tools.

Note: This manual describes the *bdltran* command for behavioral synthesis on the basis of BDL. Other sub tools, commands, and their usage are described in the appendix.

**Figure 2:** System diagram of Cyber sub tool

1.3 Synthesized Circuit Structure

Cyber takes the behavioral descriptions of functions as inputs to synthesize a circuit. The synthesized circuit implements the specified functions for infinite cycle execution.

Each function is declared as a process. Each process is synthesized as a module. The process is referred to by the name specified in the function definition.

Synthesized circuit consists of two modules - Finite State Machine (FSM) and Data Path (see **Figure 3**). However, FSM is redundant for synthesized circuits with only one state. In such circuits, only the Data Path bus is generated and FSM is not generated.

Note: In this manual, a function which is declared using the keyword "process" shall be referred to as "process".

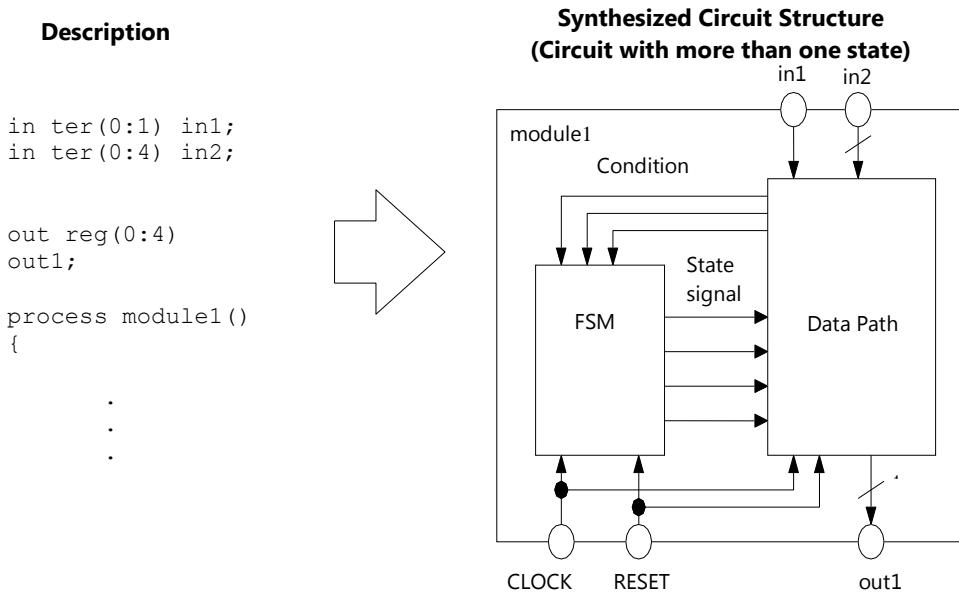


Figure 3: Synthesized Circuit - Module Structure

The current version of Cyber allows designers to specify only one process for each behavioral description input. In order to synthesize complex circuits consisting of multiple processes, each process should be synthesized individually. In addition, a top module should be implemented for integration of all processes. Cyber also has a tool to generate top module automatically (refer to **Appendix L**).

The current version of Cyber accepts only one behavioral description file at the command line. Functional descriptions specified in multiple files for a process should be explicitly included in a single file.

2 Synthesis Modes and Constraint Modes

2.1 Synthesis Modes

Cyber has two types of synthesis modes: *manual scheduling mode* and *automatic scheduling mode*.

Option	Description
-sN	Manual scheduling mode (Default)
-s	Automatic scheduling mode

Auto pipeline scheduling mode is a part of auto scheduling mode.

2.1.1 Manual Scheduling

In manual scheduling, execution timing for each operation can be explicitly specified by setting the clock cycle boundary "\$" in the functional description. Based on the setting, a circuit is synthesized. This synthesized circuit incorporates:

- The scheduling of operations and memory accesses as per the timing specified in the description.
- Sharing of registers, functional computing units and other resources, thereby minimizing the required area.

In manual scheduling, timing can be specified directly in the functional description. Therefore, manual scheduling mode is suitable where timings are generally pre-determined such as for synthesis of control circuits, interface circuits, etc.

```
in ter(0:1) flag;
out reg(0:1) ready;

process module1()
...
  while(flag == 0) {
    ready = 1;
    $
  }
  ready = 0;
  $
  ready = 1;
  $
...
}
```

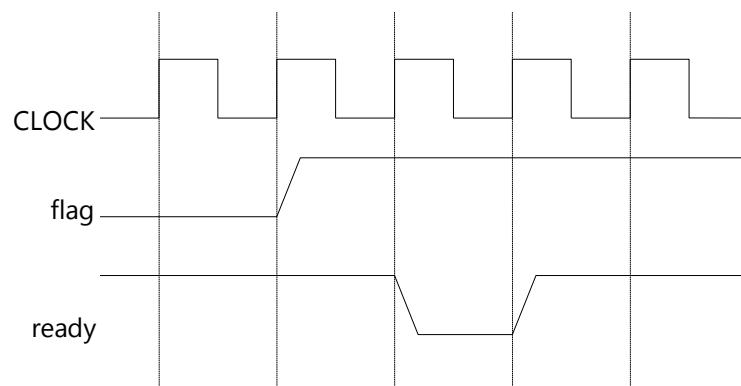


Figure 4: Manual Scheduling - Example

2.1.2 Automatic Scheduling

Automatic scheduling enables the selection of optimal circuit architecture. The selection is done by evaluating the trade-offs between execution cycles and required circuit area for various alternative circuits. These alternative circuits implement the same functionality by altering clock cycles and functional units with the specified constraints.

```

in ter(0:1) in_req;
in ter(0:8) data_in;
out ter(0:1) in_ack;

out reg(0:8) data_out;
out ter(0:1) out_req;
in ter(0:1) out_ack;

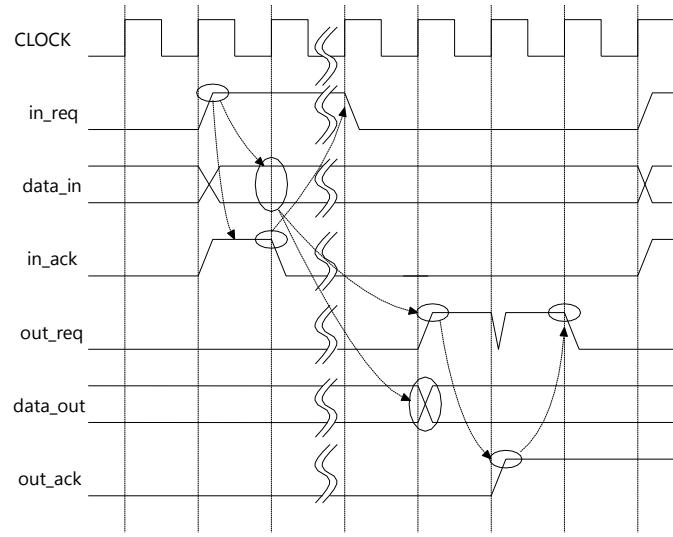
process module1() {
    var(0:8) data;

    do{
        data = data_in;
        in_ack = 0;
    }while( !in_req );
    in_ack = 1;

    /* processing */
    ....

    do{
        data_out = data;
        out_req = 1;
    }while( !out_ack );
}

```



The number of execution cycles (required for processing of data) from data input to processing completion will change according to the synthesis constraints.

Figure 5: Automatic Scheduling - Example

In automatic scheduling mode, circuits are initially synthesized to minimize the number of execution cycles within the scope of the specified constraints, such as clock cycles, the limitations of functional units, etc. Later, the circuits are configured to minimize the area requirement by sharing of registers, functional units, and other resources.

Automatic scheduling mode is suitable where fixed algorithms are implemented without any fixed timing requirements such as the synthesis of encryption circuits, compression/decompression circuits, etc.

2.2 Constraint Modes

In constraint mode, there are two modes to synthesize. In first mode, user synthesizes without specifying the functional unit and memory resource constraint by selecting "None-use (see **Figure 6**)" mode. In second mode, user synthesizes after specifying functional unit and memory resource constraint by selecting "Use-mode (see **Figure 7**)".

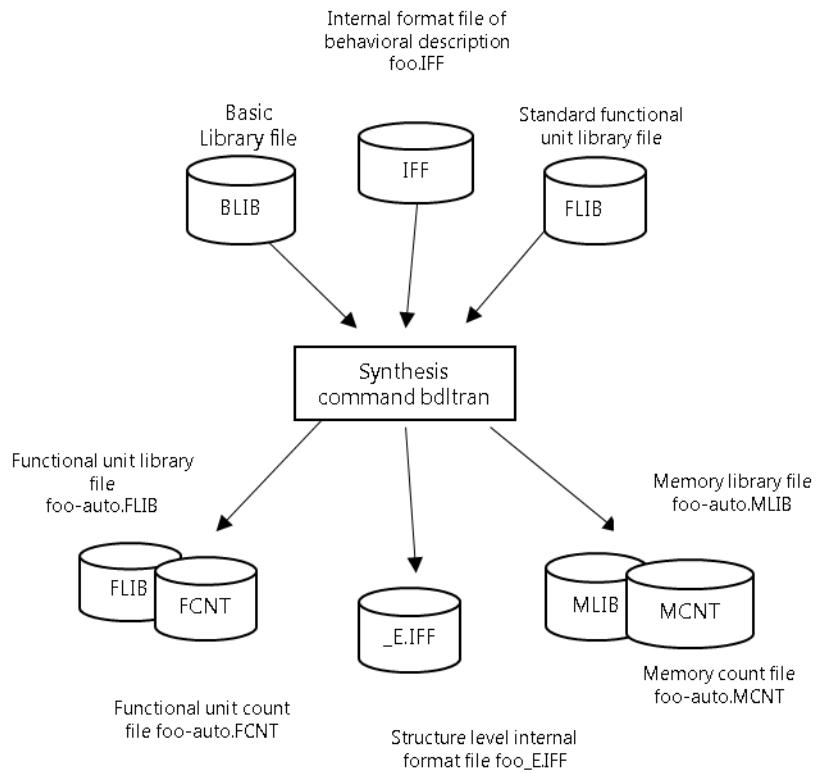
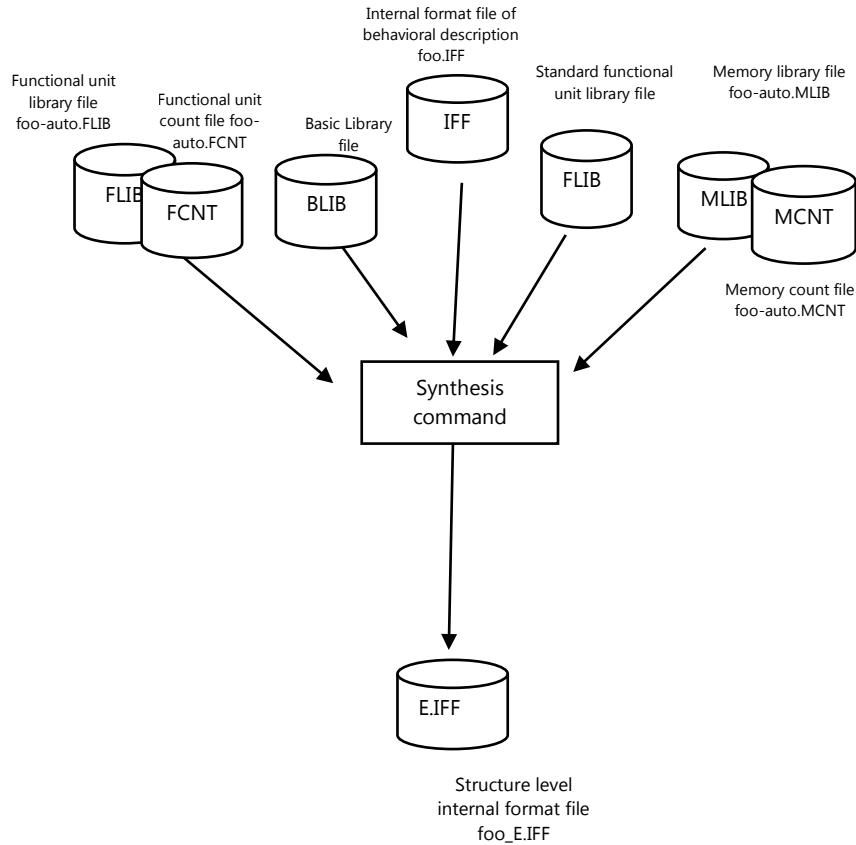


Figure 6: None-use mode

**Figure 7:** Use Mode

2.2.1 Functional unit constraint-None-use mode/Use-mode

- Functional unit constraint-None-use mode

In None-use mode, user synthesizes without specifying resource constraint of functional unit. In this mode it is not necessary to specify resource constraint as it is appropriately calculated in Cyber Behavioral synthesis system.

Calculated resource constraint of functional unit is generated in "Functional unit library file (FLIB)" and "Functional unit count file (FCNT)".

It is necessary to specify "Basic library file (BLIB)" and "Standard functional unit library file (Standard FLIB)" in order to synthesize using None-use mode.

- Functional unit constraint-Use-mode

In use-mode, user synthesizes after specifying FLIB and FCNT files. It is used when user wants to assign constraint in behavioral synthesis system.

Synthesize using use-mode if resource constraint is required to be specified manually on the basis of circuit or device specification.

Resource constraint can be easily specified if FLIB and FCNT file to be specified in use-mode are the files automatically generated in none-use mode.,

Synthesize using "none-use mode", if input description (.c|.bdll|.cpp) has been analyzed or, if resource constraint of functional unit need not to be specified. Synthesize using "use mode" if resource constraints are to be specified manually.

Refer to section **39.7** or **39.8** for FLIB and FCNT files respectively.

Refer to section **25.2.1** for the details of functional unit constraint-None-use mode/Use-mode.

2.2.2 Memory constraint generation/Use mode

- Memory constraint-None-use mode

In none-use mode, user synthesizes without specifying resource constraint of memory. It is not necessary to specify resource constraint as it is appropriately calculated in behavioral synthesis system.

Calculated resource of memory resource constraint is generated in "Memory library file (MLIB)" and "Memory count file (MCNT)".

It is necessary to specify "Basic library file (BLIB)" for synthesizing by selecting None-use mode.

- Memory constraint-Use-mode

In use-mode, user synthesizes after specifying MLIB and MCNT files. It is used when constraint is assigned in behavioral synthesis system.

Synthesize using use-mode when resource constraint need to be specified manually on the basis of circuit or device specification.

Resource constraint can be easily specified in use-mode if MLIB and MCNT files are the files automatically generated in none-use mode..

Synthesize using "none-use mode", if input description (.c|.bdll|.cpp) has been analyzed or, if resource constraint of functional unit need not to be specified. Synthesize using "use mode" if resource constraints are to be specified manually.

Refer to section **39.10** or **39.11** for MLIB and MCNT file respectively.

Refer to section **25.2.2** for the details of memory constraint-None-use mode/Use-mode.

2.3 Summary

This section covered the following:

- Cyber consists of the following five components:
 - a. The tools to convert behavioral description to the internal format.
 - b. The tool for carrying out behavioral synthesis.
 - c. The tools used to generate output synthesizable circuit (rtl) description in each language.
 - d. The other sub tools.
 - e. The GUI to manage the tools.
- Cyber synthesizes circuits by taking the behavioral description of functions as inputs.
- A function declared using the keyword process is referred to as process.
- Synthesized circuit consists of two modules - FSM and Data Path.
- Cyber has two types of synthesis modes: Manual scheduling mode and Automatic scheduling mode.
- Manual scheduling mode is suitable where timings are generally pre-determined.
- Automatic scheduling mode is suitable where fixed algorithms are implemented without any fixed timing.

- Glossary Terms -

Cyber: Cyber is a behavioral synthesis system that can be used to generate hardware implementation for a system.

Manual scheduling: Manual scheduling is a type of scheduling where timing can be specified explicitly in the functional description.

Manual scheduling Mode: Manual scheduling mode is a type of synthesis mode and is suitable where timings are generally pre-determined such as the synthesis of control circuits, interface circuits, etc.

Automatic scheduling: Automatic scheduling is a type of scheduling where the selection of optimal circuit architecture is done by evaluating the trade-offs between execution cycles and required circuit area for various alternative circuits.

Automatic scheduling mode: Automatic scheduling mode is a type of synthesis mode and is suitable where fixed algorithms are implemented without any fixed timing such as the synthesis of encryption circuits, compression/decompression circuits, etc.

3 Attributes

3.1 Overview

This section covers the following:

- Attribute definition, use, and format.
- The placement of attribute description.
- Advantages and disadvantages of using attributes and options.
- Constant substitution of attribute value.
- Rules for the order of priority.

3.2 Attributes

An attribute is used to specify supplementary information related to behavioral synthesis. The supplementary information is provided with appropriate statements in behavioral description.

An attribute can be specified as a comment that begins with the keyword "Cyber". An attribute consists of two fields – a name and a value. In few cases, the attribute value is not required.

The format for an attribute is shown below:

```
/* Cyber Attribute Name = Attribute Value */  
// Cyber Attribute Name = Attribute Value  
/* Cyber Attribute Name */  
// Cyber Attribute Name
```

The format for an attribute is subjected to the following rules:

1. The first word of the comment should begin with keyword "Cyber".
 - Keyword "Cyber" is not differentiated with small or big fonts. For example: "cyber" is also treated similarly as "Cyber". Zero or more Spaces or Tabs are permitted between "/ *" or "/*" and keyword "Cyber" or "cyber" in the comment.
 - One or more Spaces or Tabs are mandatory between keyword "Cyber" and "Attribute Name".
 - In case Cyber keyword is not specified, the comment will be treated as a normal comment.
2. Multiple attributes can be specified with one keyword. When specifying multiple attributes, comma delimiter (",") is used for separating each attribute from the other attributes.

```
/* Cyber Attribute Name 1 = Attribute Value 1,  
Attribute Name 2 = Attribute Value2 */
```

3.3 Placement of Attribute Description

There can be 4 types of placement for attribute descriptions. These placements are as follows:

1. Attributes for terms (expression elements, array access)

In the first type of placement, attribute descriptions are specified just after the term definition. Following are examples for specifications of the first type of placement:

- Attribute specification example for "+" (expression element)
`y = a + /* Cyber ope2fu = add08 */ b;`
- Attribute specification example for array reference (array access))
`y = M [i] /* Cyber mu_port = 1 */ ;`
- Attribute specification example for array assignment (array access))
`M [i] /* Cyber ex_group = 1 */ = x ;`

2. Attributes for statements such as function definition statement, "for" statement, etc.

In the second type of placement, attribute descriptions are specified just before the statement. Following are examples for specifications of the second type of placement:

- Attribute specification before process function

```
/* Cyber comment = version2.1.2 */
process module2() {
}
```

- Attribute specification before function definition statement

```
/* Cyber func = inline */
var(0.8)
func1( var(0:8) a, var(0:8) b) {
}
```

- Attribute Specification for "for" statement, etc.

```
/* Cyber unroll_times = all */
for( i = 0; i< 10; i++) {
```

3. Attributes for variables (reset variable, array, input-output variable, etc.)

In the third type of placement, attribute descriptions are specified just after the variable name declaration in a statement and before ";". Even in the case of initial value assignment during

variable declaration, the attribute descriptions can be specified just after the variable name and before “=”. Following are examples for specifications of the third type of placement:

- Attribute specification example for variable

```
int x /* Cyber share_name= x */ ;
```
- Attribute specification example for variable with initial value assignment

```
out reg(0:8) y /* Cyber valid_sig_gen = v_y */ =1 ;
```
- Attribute specification example for variable with initial value assignment (array)

```
reg(0:8) M[6] /* Cyber rw_port = RW1 */ = {3,4,5,6,7,8} ;
```
- Attribute specification example for structural member

```
struct ST1 {
    reg(0:8) a /* Cyber clock_sig = CLK */;
    reg(0:8) b;
} s1, s2;
→clock_sig attribute is valid for s1.a, s2.a
```
- Attribute specification example for structural variable

```
struct ST1 {
    reg(0:8) a;
    reg(0:8) b;
} s1 /* Cyber clock_sig = CLK */, s2;
→clock_sig attribute is valid for s1.a, s1.b
```
- Attribute specification example for individual member of structural variable

```
struct ST1 {
    reg(0:8) a;
    reg(0:8) b;
} s1 /* Cyber .a = {clock_sig = CLK}, reset_sig = RST */, s2;
→clock_sig, reset_sig attributes are valid for s1.a
```
- Attribute specification example for individual member of nested structural variable

```
struct ST1 {
    reg(0:8) a;
    reg(0:8) b;
    struct ST2 {
        reg(0:8) c;
        reg(0:8) d;
    } v;
} s1 /* Cyber .v.c = {clock_sig = CLK, reset_sig = RST} */;
→clock_sig, reset_sig attributes are valid for s1.v.c
```

4. Attributes for function declaration part

In the fourth type of placement, attribute descriptions are specified just after function declaration part and before “;”. Following are examples for specification of the fourth type of placement:

- Attribute specification example for function declaration part

```
var(0:8) func2( var(0:8) a) /* Cyber func = inline */;
```

5. Attribute for block

In the fifth type of placement, attribute descriptions are specified just before the block (As shown in the following example).

```
/* Cyber scheduling_block */
{
}
```

3.4 Macro substitution of Attribute

The effect of attribute can be changed by doing macro substitution within the comment.

Macro substitution within the string can also be done in a similar manner as the input description. When macro is defined using #define then macro substitution of string within the comment is also done in the same way as a normal input description. This makes it possible to externally control the effect of attribute described within the command that begins with "Cyber".

Effects of attributes can be changed using macro conversion by the following methods.

- Substitution of Attribute value
Macro substitution of attribute value is possible. Below example shows the substitution of unroll_times attribute value with UNROLL macro.

Before Macro Substitution

```
#define UNROLL all
/* Cyber unroll_times = UNROLL
 */
for( i = 0; i < 10 ; i++){
    x = m[i];
}
```

After Macro Substitution

```
/* Cyber unroll_times = all
 */
for( i = 0; i < 10 ; i++){
    x = m[i];
}
```

- Grouping of attribute

Macro substitution of multiple macros can be done simultaneously. In the below example, array attribute and rw_port are substituted together in the macro in order to specify port count while implementing array in RAM memory.

Before Macro Substitution

```
#define A1
array=RAM, rw_port=RW2
#define A2 array=REG
char m1[32]
/* Cyber A1 */;
char m2[32]
```

After Macro Substitution

```
char m1[32]
/* Cyber
array=RAM, rw_port=RW2 */;
char m2[32]
/* Cyber array=REG */;
```

- Valid/Invalid toggle of attribute

When comment title is "Cyber" then attribute becomes valid. Hence, multiple attributes can be toggled between valid or invalid using macro substitution.

Before Macro Substitution

```
#define GROUP1 Cyber
#define GROUP2
/* GROUP1 folding = 1
*/
/* GROUP2 unroll_times
= all */
for( i = 0; i < 10 ;
i++) {
x = m[i];
}
/* GROUP1 folding = 1
*/
/* GROUP2 unroll_times
= all */
```

After Macro Substitution

```
/* Cyber folding = 1 */
for( i = 0; i < 10 ; i++){
x = m[i];
}
/* Cyber folding = 1 */
for( i = 0; i < 10 ; i++){
y = n[i];
}
```

3.5 Constant substitution of attribute value

Even if constant calculation method is described in attribute value, it is not constantly calculated and handled as character string. If constant calculation method is encircled by \$(), then it can be substituted to constant value.

```
#define FOO 4
#define BAR 2
#define FLG 1
out ter out1 /* Cyber latency_set = $(FOO + 1) */;
out ter out2 /* Cyber latency_set = $(FOO + 2) */;
out ter out3 /* Cyber latency_set = $(FLG ? FOO * (BAR + 1) : FOO)
```

↓

```
out ter out1 /* Cyber latency_set = 5 */;
out ter out2 /* Cyber latency_set = 6 */;
out ter out3 /* Cyber latency_set = 12 */;
```

- Operator which can be used in constant calculation method
 - Arithmetic operation (+,-,*,/,%)
 - Comparison operation (<,<=,>,>=,!=,==)
 - Logical operation (!,&&,||)
 - Bit operation (<<,>,&,&|,^,~);
 - Ternary operation (? :);
- Restrictions in constant substitution
 - Constants are calculated as signed 32bit
 - No support and error in case of constants (Constants other than -2147483648 ~ 2147483647) exceeding signed 32 bit
 - In case the result of constant calculation exceeds signed 32bit, then highest bit is ignored and changes to the value of lowest signed 32bit
 - Error as nesting is not supported (Ex \$(1 + \$(3 + 2)))

3.6 Attributes and Options

There are two methods to control the synthesis of a system. The first method is to use **attributes** to specify the appropriate synthesis technique in the attribute description itself. The second method is to use command-line **options** to specify the appropriate synthesis techniques to be used during synthesis.

The advantages and disadvantages of attributes and options are as follows:

- **Attributes**

Advantages

- Attribute can be specified for each variable and function individually.
- Attributes are given preference over options.

Disadvantages

- When making any changes in behavioral specification, the attribute descriptions need to be rewritten.
- Attributes need to be specified for all applicable terms separately.

- **Options**

Advantages

- Options can be easily specified for the overall system circuit.
- New circuit can be synthesized by changing the option at just one location.

Disadvantages

- An option cannot be specified separately for different variables and functions.

3.6.1 Order of Priority

The order of priority for attributes is subjected to following rules:

1. Attributes versus options

In case both attributes and options have been specified for a term, then attributes are given priority and synthesis is done as per the attributes.

Note: Attributes are preferred among Options and Attributes.

The following example illustrates the order of priority between attributes and options. While the option specifies that the "for" loop should not be unrolled, the attribute specifies that the "for" loop should be unrolled. The first "for" loop will be unrolled because the attribute will be given priority over the option.

In the second "for" loop, an option specifies that the "for" loop should not be unrolled. In the absence of any attribute specification, the second "for" loop will not be unrolled as per the option specification.

```

/* Cyber unroll_times = all */
for( i=0; i<10; i++) {
    :
    :
}
    :
    :
for(j=0; j<20; j++) {
    :
    :
}

```

2. Multiple specifications of same class of attributes

In case same class of attributes have been specified multiple numbers of times, the last one will be considered and given priority. However, a commented attribute will be an exception to this case (refer to **Section 6.7**).

Note: In case of specification of multiple same class of attributes, **last attribute specification is given priority.**

The following example illustrates the order of priority for similar class of attributes that have multiple numbers of specifications. In this example, "share_name = a" and "no_share" are two attributes of same class. As "no_share" is the last attribute specified, it will be considered and given priority over "share_name = a".

```
reg(0:8) data /* Cyber share_name = a, no_share */ ;
```

And, in case same type of attribute is specified in structural variable & member variable then the attribute specified in structural variable is given priority. However, if an attribute is specified in individual member of structural variable then it will be given priority.

In the below example, "clock_sig = CLK1" of member variable is ignored and "clock_sig = CLK2" of structural variable becomes valid.

```

struct ST {
    reg(0:8) data /* Cyber clock_sig = CLK1 */;
};
reg struct ST x/* Cyber clock_sig = CLK2 */;
```

And, in the below example, for x.v.data1 the individually specified becomes valid "clock_sig = CLK5" is valid, and for members of x other than that, "clock_sig = CLK6" as specified in x is valid.

And regarding y, for y.v the "clock_sig = CLK3" specified in v is valid, only for y.data4 "clock_sig = CLK4" as specified in member is valid.

```
struct ST1 {
    reg struct ST2 {
        reg(0:8) data1 /* Cyber clock_sig = CLK1 */;
        reg(0:8) data2 /* Cyber clock_sig = CLK2 */;
        reg(0:8) data3;
    } v /* Cyber clock_sig = CLK3 */;
    reg(0:8) data4 /* Cyber clock_sig = CLK4 */;
};

reg struct ST1 x/* Cyber .v.data1={clock_sig = CLK5},
                  clock_sig = CLK6 */;

reg struct ST1 y;
↓
x.v.data1 → clock_sig = CLK5 (Member Individual Specification
gets top priority)
x.v.data2 → clock_sig = CLK6 (Structural Variable
Specification gets priority)
x.v.data3 → clock_sig = CLK6 (Structural Variable
Specification gets priority)
x.data4 → clock_sig = CLK6 (Structural Variable Specification
gets priority)

y.v.data1 → clock_sig = CLK3 (Structural Variable v
Specification gets priority)
y.v.data2 → clock_sig = CLK3 (Structural Variable v
Specification gets priority)
y.v.data3 → clock_sig = CLK3 (Structural Variable v
Specification gets priority)
y.data4 → clock_sig = CLK4 (Member Variable Specification is
```

3.7 Summary

- o This sectt word of the comment should begin with keyword "Cyber".
- o Multiple attributes can be specified with one keyword
- o There can be five types of placement for attribute specification.
 - a. Attributes can be specified just after the term definition.
 - b. Attributes can be specified just before the statement.
 - c. Attributes can be specified just after the variable name declaration in a statement and before ";".
 - d. Attributes can be specified just after function declaration part and before ";".
 - e. Attribute can be specified for a block.
- o The rules for setting the order of priority for attributes are as follows:
 - a. When an attribute and an option are specified for a term, the attribute is given priority.
 - b. When similar types of attributes are specified multiple number of times, the attribute specified last is considered and given priority.

- Glossary Terms -

Attribute: An attribute is a comment that begins with the keyword "Cyber" and contains a name and a value.

4 Options Specification

4.1 Overview

This section covers the following:

- Command-line options specification rules for “**bdltran**”.
- Exceptions where the order of specification of the options is considered.
- Default Option specification file.
- Usage of parameter file (.prm).
- Input file specification method.

4.1.1 “**bdltran**” –Options

The “bdltran” command is used for Behavioral synthesis. The command-line options for “bdltran” and their description are as follows:

Options	Description
-p	Use Options specified on line number 1 of parameter file “bdltran.prm”
-p#	Use Options specified on line number # of parameter file “bdltran.prm”
-pfilename	Use Options specified on line number 1 of parameter file “filename”
-p: filename	Use Options specified on line number 1 of parameter file “filename”

4.2 Option Specification Order - Exceptions

The “bdltran” command doesn’t require options to be specified in a particular sequence on the command line. Therefore, the options of the “bdltran” command can be specified in any order excluding few exceptions. These exceptions where the order of specification of the options is considered are as follows:

1. Multiple options of the same class specified

In case several options of the same class are specified a number of times, the option that is specified the last is considered and given precedence over the previous options.

In the following example, options “-Ad” and “-Aa” are of similar type. Therefore, the previous option “-Ad” is ignored, and the last option “-Aa” is considered and used.

```
>bdltran -Ad lfl foo.FLIB -lfc foo.FCNT foo.IFF -s -Aa
```

2. Options with “+”

The following options are meant for specifying multiple files. These options are allowed to be specified multiple times without any priority given for later ones.

```
+lf, +lm, +lfl, +lfc, +lml, +lmc
```

3. When a “+” changes to “-”

When a later option uses a “-” instead of a “+” the options following the “-” will be considered valid and is given precedence. For example, in the case of specification given below, the “+” before “lfl fileC.FLIB” is changed to a “-”. As a result, “-lfl fileA.FLIB” and “+lfl fileB.FLIB” are ignored and the specifications after –lfl fileC.FLIB (including +lfl fileD.FLIB) are considered valid and are used.

```
-lfl fileA.FLIB +lfl fileB.FLIB - lfl fileC.FLIB +lfl fileD.FLIB
```

And it is necessary to specify “-p” option in the beginning.

A maximum of 255 options can be specified, and the total character count of options should not exceed 1023.

4.3 Default Option Specification File

The options specified in the default option specification file are applied by default. The default option specification file should have the name “cyber”. The default option specification file can be placed at any of the following locations

1. In case the environment variable CYBER_ENV is defined, a file with name “.cyber” is placed in the directory specified by CYBER_ENV.
2. In case CYBER_ENV is not defined, a file with name “.cyber” is placed in the directory specified by the environment variable HOME.

Cyber assumes that the default option specification file doesn’t exist in case it is not found in any of the specified locations.

Options that are specified in the default option specification file are specified much earlier than the options specified at the execution time. Therefore, the options specified at the execution time are given preference. In other words, if there are options specified at the execution time that are contradicting the options defined in the default option specification file, then the options specified at the execution time will override the options specified in the default option specification file.

The formatting rules regarding the default option specification file are as follows:

- Lines starting with # are considered as comment lines
- Starting from the first line, all text till the line that starts with #@END is treated as options. In case #@END is not specified, text till end of the file is treated as options.
- Parameter file specification option -p cannot be specified inside the default option specification file.

4.4 Parameter File

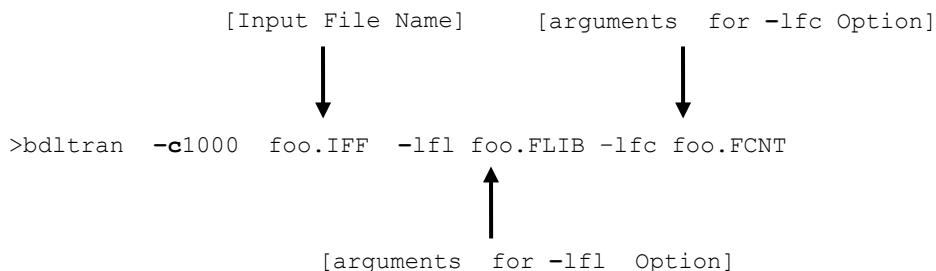
First line of parameter file can be specified as an option by specifying parameter file using option **-p**.

If only **-p** is specified, the file "bdltran.prm" present in the current directory will be considered as the parameter file.

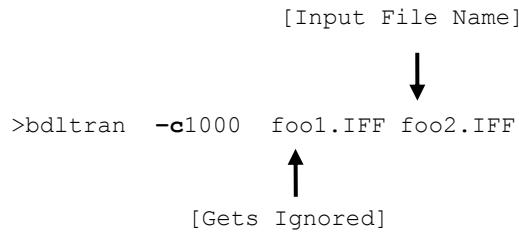
The **-p** option must be specified as the first option. An option specified in parameter file is given preference over default option, but it is given less preference as compared to option specified at time of execution.

4.5 Input File Specification

Arguments not starting with "-" are assumed to be the name of input files. However, arguments that follow an option, that requires argument specification, are not treated as input files even if they don't start with "-". For example, in the case of "-p: filename", the argument is not treated as an input file because it is following an option requiring an argument specification.



If the input file name has been specified a number of times, the last one will be given preference over others. Therefore, the file name specified in the last will be considered.



Internal format file (IFF), which is generated by BDL input command, is specified as an input file. Even when input file name doesn't include the file extension ".IFF", Cyber assumes that the file extension has been omitted and adds file extension ".IFF" as suffix to the file name.

If file name ends with the extensions ".c" or ".bdl", it is assumed that file names entered for BDL input command have been specified. In this case, the extensions ".c" or ".bdl" are deleted prior to adding a file extension suffix ".IFF". In all 4 examples given below, it will be assumed that input file name is "foo.IFF".

```
>bdltran -c1000 foo.IFF -lfl foo.FLIB -lfc foo.FCNT
>bdltran -c1000 foo. -lfl foo.FLIB -lfc foo.FCNT
>bdltran -c1000 foo.c -lfl foo.FLIB -lfc foo.FCNT
>bdltran -c1000 foo.bdl -lfl foo.FLIB -lfc foo.FCNT
```

4.6 Summary

This section covered the following:

- The command-line options for "bdltran" include "-p", "-p#", "-pfilename", and "-p:filename".
- The "bdltran" command doesn't require options to be specified in a particular sequence on the command line. However, there are certain exceptions to this rule.
- The options specified in the default option specification file are applied by default.
- The options specified at the execution time are given preference over the options specified in the default option specification file.
- Arguments not starting with "-" are assumed to be the name of input files.
- Arguments that follow an option, that requires argument specification, are not treated as input files even if they don't start with "-".

- Glossary Terms -

Default option specification file: The default option specification file contains the options specifications that are applied by default.

5 License related Options

5.1 License Retry functionality

Options	Description
-lic_wait=#	License has already been taken by some other process so in case license cannot be obtained then repeat retry for #minute(s) [0 minute(s)]

License has already been taken by some other process, hence in this case license cannot be obtained then it results in the following error.

```
Unable to obtain FLEXlm license.
Because
Licensed number of users already reached
Feature: Cyber-Synthesis
License path: *****
FLEXlm error: -4,132
For further information, refer to the FLEXlm End User Manual,
available at "www.macrovision.com".
F_BT1922: Failed to obtain license due to FLEXlm
[Countermeasure] Revise the settings of FLEXlm license
```

In such a case, -lic_wait=# option is used to re-gain the license. License re-take is carried out for #minutes at an interval of 1 minutes by specifying -lic_wait=#. The argument can be specified from 0 minutes to 2880 minutes (24 hours). In case of 0 minutes it becomes an error since it does not carry out retry. As the default is set at 0 minutes therefore in case it is not specified then it results in an error as the license is not obtained.

```
%bdltran foo.bdl -lic_wait=2
.....
option foo.bdl -lic_wait=2
All licenses are already in use. Waiting for a license up to 2 minute(s).
All licenses are already in use. Waiting for a license up to 1 minute(s).
....
```

In case an error occurs like communication with license server fails or the feature does not exist then the license cannot be retried.

6 Synthesis Circuit Specification

6.1 Overview

This section covers the following:

- Physical types of variables in case of manual scheduling
- Ascending/descending bit order specification
- Signed/unsigned specifications
- Carry-over in calculation
- Specification of operation bit width
- Prefix specification for generated signal name and functional unit name
- Synthesis time limit and its restrictions

6.2 Physical Type Variables

6.2.1 Physical type variables in Manual scheduling mode

In manual scheduling, the variables where the physical type has not been specified, are treated as "var" type. For 'var' type variables in manual scheduling mode, multiple assignments and referencing can be done in same cycle.

Following are the limitations for "var" type variables, in manual scheduling mode:

A "var" type variable cannot be used in "allstates" statements and in assignment statements with continuous assignment.

Attribute reg_bind cannot be specified for a "var" type variable

The input() function cannot be used multiple times in same cycle for a "in var" type of variable.

The output() function cannot be used multiple times in the same cycle for a "out var" type variable.

In addition, "var" type arrays that are not unrolled have a restriction. "Var" type arrays can only be synthesized when the exclusion of subscripted array is explicitly evident in the case of reading or writing after writing at the same cycle. This restriction also applies to "var" type arrays that are without any physical type specification. When an error occurs due to this restriction in a case when subscript is not exclusive, however it is exclusive in reality, then synthesis can be done by adding "ex_group" attribute to the array.

6.3 Bit Order Specification - Ascending/Descending

6.3.1 Related Options

Option	Description
-Zbit_order_force=NO	In output RTL description, signal bit order is put in same order as the declaration in the behavior description. (default)
-Zbit_order_force=up	All signals are put in ascending order in output RTL description
-Zbit_order_force=down	All signals are put in descending order in output RTL description.

Following three options are available for the BDL input command "bdlpars":

Option	Description
-Zdefault_bit_order=up	Variables without any ascending/descending order specification are assumed to be in ascending order
-Zdefault_bit_order=down	Variables without any ascending/descending order specification are assumed to be in descending order
-Zchk_bit_description	Warns about ascending order bit declaration and bit reference like (m:n)

6.3.2 Related attributes

Attribute	Description	Setting Target
/* Cyber default_bit_order = up */	Constants and variables without any ascending/descending order specification are assumed to be in ascending order	Process Function
/* Cyber default_bit_order = down */	Constants and variables without any ascending/descending order specification are assumed to be in descending order	Process Function

6.3.3 Input BDL – Specification Order

There are two ways of declaring bit order of a variable in BDL: ascending order and descending order.

```
var(0:8)      data1;      //ascending
var(0..7)     data2;      //ascending
var(7..0)     data3;      //descending
char          data4;      //variable without ascending/descending specification
```

The variable, which has been declared in descending order, gets declared in descending order in the RTL description also. Similarly, the variable, which has been declared in ascending order, gets declared in ascending order in the RTL description also.

Ascending and descending order of 1 bit variable, variable without ascending/descending order specification, and internally generated variable used in register sharing are undefined. In the cases of such variables, Cyber decides ascending or descending order for these variables depending upon the situation.

Order of any variables and constants with undefined ascending/descending order is decided based upon the order of specification of other variables. If the other variable is in ascending order, the bit order of variable / constant will be in ascending order and vice versa. If the other variables have a mixed of ascending and descending order, variables / constants with undefined order are processed in ascending order.

To explicitly specify ascending / descending order for variables and constants with undefined order as described above, either option “-Zdefault_bit_order=up”, “-Zdefault_bit_order=down” of BDL input command bdlpars or attribute default_bit_order can be used.

```
/* Cyber default_bit_order = down */
Process foo() {
    .
    .
}
```

6.3.4 RTL Output – Specification Order

Since the RTL description generally has a descending bit order, there are instances where all variables are desired to be in descending order even though there are some variables in ascending order in the BDL input. In such a case, if option “-Zbit_order_force=down” is specified, all variables in the RTL description are generated in descending order. Further, an option “-Zbit_order_force=up”, which generates all variable in ascending order, is also available.

6.4 Signed / Unsigned Specification

6.4.1 Related Options

Option	Description
-Zmix_signed	Signal signs (signed / unsigned) are synthesized according to the respective signal declaration (Default)
-Zunsigned	All signals are synthesized as unsigned

The two options mentioned above are available for the BDL input command "bdlpars".

6.4.2 Description

Cyber supports two modes in regards to processing of signed / unsigned variables:

1. First mode synthesizes each variable according to the sign specification (signed/unsigned) provided with the declaration. This mode is applied by default.
2. Second mode synthesizes all variables as unsigned.
In the unsigned mode, the results are also assumed to be in the "unsigned" mode. Therefore, one needs to be careful as it may cause tentative results with negative values to be improperly processed as large positive numbers.

6.5 Carry-Over in Calculation

6.5.1 Description

Output bit width for synthesis of operation is decided by considering carry-over operation. For instance, in the following case, bit width required for addition of two 4 bit variables a and b is 5. When variable c is added to the 5-bits resultant value from earlier step, the bit width required is 6. Therefore, circuit that prevents the overflow is generated appropriately.

Example:

```
d(0:6) = (a(0:4) + b(0:4)) + c(0:4) ;
```

Further, when output bit width has not been explicitly specified in functional unit constraint file, the output bit width of functional unit is decided based upon carry-over consideration of addition, subtraction, multiplication, shift operations etc.

6.5.2 Operation in which carry is considered

In following four operations, output bit width has been specified based upon carry-over consideration:

Operator	Operation name	Output bit width
+	Addition	(Maximum bit width of input variable)+1 (8bit + 4bit → 9bit)
*	Multiplication	(Bit width of left item)+(bit width of right item) (8bit * 4bit → 12bit)
<<	Left shift	Bit width of left item + maximum value of right item (8bit<<4bit → 23bit) * 1
-	Subtraction	(Maximum bit width of input variable)+1 (8bit-4bit → 9bit)

※1 The bit width of right side variable is 4 bits. Therefore, the maximum value of right side variable is 15. Thus, after combining 8 bit width of the left-side variable, the output bit width is 23 bits.

6.6 Operation Bit Width - Specification

6.6.1 Related Attributes

Attribute	Description	Settings Target
/*Cyber bitw = # * /	Bit width of operation output is # bits.	Formula element

6.6.2 Description

This section explains the attribute "bitw", which is used to specify bit width of operation output. It is decided based upon operand bit width and category.

However, there are cases when operation output bit-width is required to be reduced. This can be seen typically when the upper bit of operation result is not required or when the range of value stored in variable is fixed.

In the case of following example, the result of adding a, b is of 5 bits. Therefore, 5 bits are required for the adder that uses these inputs.

```
data(0:5) = (a(0:4) + b(0:2)) + c(0:4)
```

When it is assured that result of addition of a and b can fit into 4 bits, the following cast can be used to specify the addition in 4 bits:

```
data(0:5) = (var(0:4))(a(0:4) + b(0:2)) + c(0:4)
```

In addition, there is another method of specification i.e. using "bitw" attribute.

It is specified by using attribute "bitw". Attribute "bitw" is an attribute that specifies the output bit width of operation. And in this case, the output bit width can be reduced to 4 bits by specifying 4 bits attribute "bitw" for addition. However, it is recommended to use cast method of specification.

```
data(0:5) = (a(0:4) + /*Cyber bitw = 4 */ b(0:2)) + c(0:4)
```

6.7 Signal Name

6.7.1 Related Options

Options	Description
-Zprefix=word	Add 'word' before generated signal name
-Zsig_name1state	Add a state name before the generated signal name
-Zname_sensitive	Consider upper case / lower case characters for distinguishing variable names while checking (case sensitive)
-Zname_sensitive=NO	Do not consider upper case / lower case characters for distinguishing variable names while checking (case insensitive) (default)

Options for VHDL output command and Verilog-HDL output command

Option	Description
-module_prefix	Add "process name_" to prefix of sub-hierarchy name (default)
-module_prefix=prefix	Add prefix to prefix for sub-hierarchy name
-module_prefix=	Do not add prefix to sub-hierarchy name

6.7.2 Signal Name - Prefix Specification

This option adds a specified character string before the signal name generated by Cyber.

When this option is used, the specified character string will be prefixed to signal names generated during synthesis.

Usual synthesis result

```
unsigned reg(0:8) RG_01;
RG_01(0:8) ::= nmux{
:
:
}
```

In case -Zprefix=module1 is specified

```
unsigned reg(0:8) module1_RG_01;
module1_RG_01(0:8) ::=nmux{
:
:
}
```

Also, the option “-Zsig_name1state” adds the used state name to a variable name. This is done in order to find out in which state variable is going to be used, such as when generated variable is used for only one state.

Usual synthesis result

```
unsigned ter(0:1) U_01;
U_01 : : = ST1_02d & gop081ot;
```

In case -Zsig_name1state is specified

```
unsigned ter(0:1) U_ST1_02_01 ;
U_ST1_02_01 : : = ST1_02d & gop081ot;
```

Restrictions

- The maximum number of characters for “word” that can be specified using option – Zprefix=word is 8.

6.7.3 Functional Unit Name - Prefix Specification

The functional unit output in generated circuit is given in the following sub hierarchy.

When “ALIAS”(=ADD) of functional unit count file is omitted from the sub-hierarchy name, then “process_name_” has added to the functional unit name “ADD” and used. This is done in order to avoid the presence of sub hierarchy having the same name in respective circuits when behavioral synthesis is done for multiple circuits in the same functional unit library file and a single circuit is generated by combining those circuits which are generated in synthesis.

Input Description Functional unit count file

```
in  ter(0:8) a, b;      #@VERSION{1.00}
out reg(0:8) c;        #@CNT{foo_cnt}
process                #@CLK 1000
add_module() {
    add_          #@UNIT 1/100ns
    module       @FCNT{
        c = a + b;           NAME add08_08u
    }                      LIMIT 1
                           ALIAS ADD
    }
#@END{foo_cnt}
```

Usual Verilog-HDL Output Result

```
:
add_module_ADD INST_FUNC_0 (.i1(ADD1i1), .i2(ADD1i2),
                           .o1(ADD1o1));
:
module add_module_ADD ( i1 ,i2 ,o1 );
input  [0:7] i1 ;
input  [0:7] i2 ;
output [0:7] o1 ;
assign o1 = ( i1 + i2 ) ;
endmodule
```

Usual VHDL Output Result

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity add_module_ADD is
    port (
        i1 : IN std_logic_vector (0 to 7) ;
        i2 : IN std_logic_vector (0 to 7) ;
        o1 : OUT std_logic_vector (0 to 7)
    );
end add_module_ADD;
architecture VHDLgen of add_module_ADD is
begin
    o1(0 to 7) <=i1(0 to 7) + i2(0 to 7);

end VHDLgen;
:
:
component add_module_ADD
    port (
        i1 : IN std_logic_vector (0 to 7) ;
        i2 : IN std_logic_vector (0 to 7) ;
        o1 : OUT std_logic_vector (0 to 7)
    );
end component;
begin
    VHDLgen_Label1002 : add_module_ADD port map(i1=>ADD1i1,
                                                i2=>ADD1i2,o1=>ADD1o1);
:

```

There are following options to specify the sub-hierarchy prefix in VHDL output command and Verilog-HDL output command. Prefixes can be changed through these options.

Option	Description
-module_prefix	Add "process name_" to prefix of sub-hierarchy name (default)
-module_prefix=prefix	Add prefix to prefix of sub-hierarchy name
-module_prefix=	Do not add prefix to sub-hierarchy name

However, functional unit name "NAME" is not prefixed as sub-hierarchy name if there exists the functional unit, where conversion of the function into the functional unit has occurred, or if there

exists the functional unit, where EXIST is specified for net list module (NETLIST) in the functional unit library file.

6.7.4 Synonym Variable

Options	Description
-Zname_sensitive	Consider upper case / lower case characters for distinguishing variable names while checking (case sensitive)
-Zname_sensitive=NO	Do not consider upper case / lower case characters for distinguishing variable names while checking (case insensitive) (default)

C, BDL and Verilog-HDL languages are case sensitive languages. This means that upper case characters and lower case characters are treated differently in variable name. However, VHDL is a case insensitive language.

Therefore, while considering the case of VHDL, the synonym check for variable name will be done in a case insensitive way by default. However, a case sensitive synonym check option, "-Zname_sensitive", has been provided for use in the case of Verilog-HDL and similar cases.

When two variables are found to be synonymous, one of them will be changed. The information regarding the change is either displayed for case insensitive language or treated as an error for case sensitive language.

6.7.5 Unrolling Structure Variable

Structure variable is unrolled with variable name such as "Structure Variable Name"_"Member Variable Name" for each member variable.

Refer to chapter **14.2** for details.

6.8 Comment

6.8.1 Related Attributes

Attribute	Description	Settings Target
/*Cyber comment word = comment */	Adds comment 'comment' to either resulting VHDL output or Verilog-HDL output.	Process function

6.8.2 Description

Any attribute with name starting with "comment" is treated as a comment attribute. The character string specified in attribute value is put-up as a comment to the top of the VHDL or Verilog-HDL output.

Input description

```
/*Cyber comment_Version = version 2.1.3 */
/*Cyber comment_date = Tue Jun 4 2002 */
/*Cyber comment_owner = Taro Nichiden */
in ter(0:8) a;
out reg(0:8) b;

process module1( ) {
    :
    :
}
```

Verilog-HDL output result

```
// verilog_out version 2.20

/* version 2.1.3
   Tue Jun 4 2002
   Taro Nichiden */

module module1( a, b, CLOCK, RESET ) ;
    :
    :
```

VDHL Output Result

```
-- vhdlgen version 3.31.1, SADL version 2.90i
:
:
-- end of command Option Status

--      version 2.1.3
--      Tue June 4 2002
--      Taro Nichiden
---

library IEEE;
use IEEE.std_logic_1164.all;
entity module1 is
    port (
        a : IN std_logic_vector(0 to 7)
        b : BUFFER std_logic_vector(0 to 7);
        CLOCK : IN std_logic;
        RESET : IN std_logic
    );
end module1;
:
:
```

As specified in description of Section **6.10.4**, there is limit on characters that can be specified in attribute value. Workaround for this limitation is to specify multiple comment attributes. If a character string is specified with the attribute beginning with "comment", the attribute will be considered as a comment attribute. The character string after "comment" will not be interpreted. However, it is not recommended to use the same "comment" attribute name twice or more.

Description for comment attribute value can also be written in Japanese, however, recommended character should be used. There is no assurance if description is written in other than recommended character codes.

UNIX(Solaris/Linux)	EUC
Windows	Shift-JIS

A comment attribute will be added to process function when it is described at any location. The output order of sequence generated at file header of either the VHDL or Verilog-HDL output will be in the order of input description.

6.9 Options for Sub-system

6.9.1 Related Options

Option	Description
-Zbdlcmp	Carry out the synthesis, checking, and correspondence relation output for equivalence verification tool bdlcmp.
-Zbdlcmp=NO	Do not carry out the synthesis, checking, and correspondence relation output for equivalence verification tool bdlcmp. (default)

6.9.2 Options for equivalence verification tool bdlcmp

When equivalence between behavioral description and RTL description needs to be confirmed by equivalence verification tool bdlcmp, there is need to specify -Zbdlcmp option at the time of synthesis. When -Zbdlcmp option is specified, the following process is executed at the time of bdltran.

- synthesis processing necessary for executing the bdlcmp command
- output of correspondence relation
- checking whether input description falls under restriction of bdlcmp command or not.

If it falls under the restriction, the following error message will be displayed¹. Refer to "bdlcmp user's manual" for restriction related other details.

F_BT1655: Equivalence verification tool does not support the deletion of loop optimization.

[Countermeasure] Refer to manual of equivalence verification tool

Points to be noted:

- In case -Zbdlcmp option is specified, it is compared with the case when option is not specified, and difference is generated in synthesis result from difference of optimization.

6.10 Synthesis Constraint Option

6.10.1 Related Options

Option	Description
-Ztimeout=#	Specifies the time for timeout '#' in seconds
-Zfu_total_limit=#	Increases the limit of total number of functional unit constraints to # times

¹Any one from 1F_BT1648 to F_BT1656 error messages will get output.

6.10.2 Synthesis time limit

In the case of time-consuming synthesis, the desire can be to stop the synthesis. For example, to explore for suitable architecture, various circuit variations are generated featuring diverse synthesis options and functional unit constraints. Such an exploration can result in a time-consuming synthesis. In such a case, the timeout duration can be specified in seconds by specifying the option “-Ztimeout” when synthesis termination is desired midway.

Restrictions

- Option “-Ztimeout” may not have any effect in some cases.

6.10.3 Limit of total number of functional unit constraints

Since sometimes resource allocation is time-consuming, there is a limit of total number of functional unit constraints. The limit is upto 2048 or 4096 constraints depending upon the scheduling mode and resource allocation option. If limit is crossed, the following error message is displayed.

F_BT2847: It may take several minutes to synthesize as total number of functional unit constraints 4097 is more than 4096.

[Countermeasure] Specify the -Zfu_total_limit=# option that increases the total number of functional unit constraints.

The limit of total number of functional unit constraints can be increased to #times by specifying the -Zfu_total_limit=# option.

6.10.4 Restrictions

Cyber has following restrictions.

File path length	Less than 256 characters
Function name	Less than 256 characters
Signal name	Less than 128 characters
Label name	Less than 128 characters
Signal Bit width	32768 bit or less
Initialization bits for a signal	32768 bit or less
Functional unit name in functional unit constraint file	Less than 256 characters
Memory name of memory constraint file	Less than 256 characters
Attribute name	Less than 256 characters
Attribute value	Less than 1024 characters
Number of array dimensions	16 dimensions or less
Number of nested hierarchy in conditional branches	256 steps or less
Length of 1 line of parameter file	4096 characters or less
Number of specified options (*1)	512 or less
Number of characters for specified options (*1)	4096 characters or less

*1 This also includes options specified in default option file.

6.11 Summary

This section covered the following:

- The "Zdefault_type_var=NO" option is used to convert arrays to either of ter / reg / mem type, and non-array variables to "ter" type of variable.
- The variable, which has been declared in descending order, gets declared in descending order in the RTL description also.
- Similarly, the variable, which has been declared in ascending order, gets declared in ascending order in the RTL description also.
- The attribute "bitw" is used to specify bit width of operation output. It is decided based upon operand bit width and category.
- The "-prefix_fu" option prefix "<process_function_name>_" to the functional unit name.
- Similarly, the "-prefix_fu=word" option prefix optional string "word" to functional unit name.
- A case sensitive synonym check option, "-Zname_sensitive", has been provided for use in the case of Verilog-HDL and similar cases.
- Any attribute with name starting with "comment" is treated as a comment attribute.
- The timeout duration can be specified in seconds by specifying the option "-Ztimeout" when synthesis termination is desired midway.

7 Input Output Ports

7.1 Overview

This section covers the following points on the input output (I/O) ports:

1. Port sequence.
2. Port constraint file and port relation file.
3. Negative logic port specification.
4. Output variable reference.
5. HiZ (high impedance) output.
6. Generation of port that outputs status number.
7. Handling of unused ports.
8. Default value of output port.
9. Valid port that outputs the input-output timing.
10. Generated port and port naming convention.
11. Attribute of "out var" variable.

7.1.1 Related options

There are 3 types of I/O ports in a synthesized circuit: input port, output port, and bidirectional port. The options regarding I/O ports are as follows:

Option	Description
-lpA	Generate one port for every input-output variable (default)
-lp filename[.PLMT]	Use specified port constraint file filename[.PLMT]
-lpr filename[.PREL]	Use specified port relation file filename [.PREL]
-Zgen_stateno_out	Generate a named "ST1out" to output the state signal
-Zgen_stateno_out=name	Generate a port with port name <i>name</i> to output state signal
-Zport_asitis	Also generate unused ports (default)
-Zport_optimize	Do not generate unused ports
-Zport_optimize_auto	Only delete automatically generated unused port (default)
-api:o,-apA:o,-ap#:o	Do not generate unused ports for external memory
-Zport_valid_sig_gen[=YES]	Generate valid signal for each I/O port as output port
-Zport_valid_sig_gen=NO	Do not generate valid signal for each I/O port (default)
-Zinternal_valid_sig_gen[=YES]	Generate valid signal for each I/O port as internal signal
-Zinternal_valid_sig_gen=NO	Do not generate valid signal for each I/O port (default)

-Zmem_re	Generate both read enable port and write enable port for read-write memory port
-Zmem_re=NO	Generate only write enable port for read-write memory port (default)
-Zmem_cs	Generate chip select port for memory
-Zmem_cs=NO	Do not generate chip select port for memory (default)
-Zmem_be	Generate byte-enable port for memory ports
-Zmem_be=NO	Do not generate byte-enable port for memory ports (default)
-Zmem_clocking=none	Do not generate memory clock port for sync SRAM (default)
-Zmem_clocking=enable	Generate memory clock port for sync SRAM to output memory clock signal only during memory access (equivalent to -sync RAM option, which was available before Cyber 3.6)
-Zmem_clocking=always	Generate memory clock port for sync SRAM to always output memory clock signal (equivalent to -syncRAM2 option, which was available before Cyber 3.6)
-Zrw1=1port	Generate read-write data bidirectional port for read and write of external memory
-Zrw1=2port	Generate write-data output port and read-data input port for read and write of external memory (default)
-Zsynchronizer	Generate synchronization circuit between out-of-phase clocks for data signal
-Zsynchronizer=#1:#2	Generate synchronization circuit between out-of-phase clocks for data signal. The number of stages of register for inserting enable and data signal in sending-clock domain is taken as # 1. The number of stages of register for inserting enable signal in receiving-clock domain is taken as # 2.
-Zin_port_synchronizer=#	Generate synchronization circuit between out-of-phase clocks at input port The number of stages of register for synchronization circuit between out-of-phase clocks is taken as # stages.
-Zin_port_reg_stage=#	Generate the register of # stages at input port
-Zout_port_reg_stage=#	Generate the register of # stages at output port

7.1.2 Related Attributes

Attribute	Description	Settings Target
<code>/*Cyber sig_pol = neg */</code>	Set specified input, output or I/O port as Active Low (negative logic)	input, output or I/O variables
<code>/*Cyber default_value = value*/</code>	Set default value of output port to ' <i>value</i> '	out ter
<code>/*Cyber valid_sig_gen = name*/</code>	Generate valid signal <i>name</i> of specified I/O variables as output port of positive logic	input, output or I/O variables
<code>/*Cyber valid_sig_neg_gen = name*/</code>	Generate valid signal <i>name</i> of specified I/O variables as output port of negative logic	input, output or I/O variables
<code>/*Cyber mem_re = create */</code>	Generate read enable port for read-write memory array port	Array variable
<code>/*Cyber mem_re = ignore */</code>	Do not generate read enable port for read-write memory array port.	Array variable
<code>/*Cyber mem_clocking=enable*/</code>	Generate memory clock port for sync SRAM to output memory clock signal only during memory access	Array variable
<code>/*Cyber mem_clocking=always*/</code>	Generate memory clock port for sync SRAM to always output memory clock signal	Array variable

/*Cyber port_type = TER */	Set output attribute of 'out var' variable to 'ter' type	out var variable
/*Cyber port_type = REG */	Set output attribute of 'out var' variable to 'reg' type	out var variable
/*Cyber port_type = VAR */	Set output attribute of 'out var' variable as 'var' type (default)	out var variable
/* Cyber synchronizer */	Generate synchronization circuit between out-of-phase clocks for data signal	input, output or I/O
/* Cyber synchronizer = #1:#2 */	Generate synchronization circuit between out-of-phase clocks for data signal. The number of stages of register for inserting enable and data signal in sending-clock domain is taken as # 1. The number of stages of register for inserting enable signal in receiving-clock domain is taken as # 2	input, output or I/O
/* Cyber port_synchronizer = # */	Generate synchronization circuit between out-of-phase clocks at input port. The number of stages of register for synchronization circuit between out-of-phase clocks is taken as # stages.	Input
/* Cyber port_reg_stage = # */	Generate the register of # stages at input/output port	input, output or I/O

Note: valid_sig(_neg) attribute is changed into valid_sig(_neg)_gen attribute and then discontinued.

7.2 Port Sequence

I/O ports in synthesized circuit are sequenced according to the order of I/O variables declaration. Parameters are given higher priority between parameters for process function and global variables.

In the following examples, "a", "b", and "c", are global variables, and "d", "e", and "f", are process function parameters. The process function parameters are given priority over the global variables. I/O variables, which are not array, for process function parameter are planned to be not supported in the future.

```
in      ter(0:8) a;
out     ter(0:8) b;
inout   ter(0:8) c;

process moduleA(in ter(0:8)d, out reg(0:8) e, inout ter(0:8)f) {
    .
    .
    .
}
```

Verilog-HDL output

```
module moduleA( d, e, f, a, b, c, CLOCK, RESET ) ;
  input  [0:7]  d ;
  output [0:7]  e ;
  inout  [0:7]  f ;
  input  [0:7]  a ;
  output [0:7]  b ;
  inout  [0:7]  c ;
  input          CLOCK;
  input          RESET;
  .
  .
end module
```

VDHL Output

```
entity moduleA is
  port (
    d : IN std_logic_vector(0 to 7)
    e : BUFFER std_logic_vector(0 to 7)
    f : INOUT std_logic_vector(0 to 7)
    a : IN std_logic_vector(0 to 7)
    b : BUFFER std_logic_vector(0 to 7)
    c : INOUT std_logic_vector(0 to 7)
    CLOCK : IN std_logic;
    RESET : IN std_logic
  );
end moduleA;
```

7.3 Single Port Sharing

Options	Description
---------	-------------

-IpA	Generate one port for every input-output variable (default)
-Ip filename[.PLMT]	Use specified port constraint file filename[.PLMT]
-lpr filename[.PREL]	Use specified port relation file filename [.PREL]

In automatic scheduling, the maximum number of ports are restricted to the number of I/O variables. Therefore, one port can be shared among multiple I/O variables through time sharing. There are two methods to specify this, which are as follows:

1. I/O variables are allocated to ports automatically by specifying only the type and number of ports using a port constraint file.
2. I/O variables are explicitly allocated to ports by using a port constraint file and port relation file.

Refer to sections **39.4** and **39.5** for details on the port constraint file and port relation file.

In the first method, only the port constraint file is specified. In this method, only the number and types of I/O ports are specified and allocation of I/O variables to ports is decided automatically. The I/O variable is allocated to a port that has same or higher bit width.

In the second method, the port constraint file and port related file are specified. In this case, the allocation of the I/O variable to ports needs to be clearly specified in the port relation file.

7.4 Negative Logic Port Specification

Attribute	Description	Settings Target
/*Cyber sig_pol = neg */	Set specified input, output or I/O port as Active Low (negative logic)	input, output or I/O variables

In order to set the specified input, output or I/O port as Active Low (negative logic) port, an attribute "sig_pol" can be specified along with the variable declaration.

```
in    ter(0:8) foo1 /* Cyber sig_pol = neg */;
out   ter(0:8) foo2 /* Cyber sig_pol = neg */;
out   reg(0:8) foo3 /* Cyber sig_pol = neg */;
inout ter(0:8) foo4 /* Cyber sig_pol = neg */;
```

7.5 Output Variable Reference

An output variable cannot be referenced using a "ter" type, but it can be referenced with the "reg" and "var" types.

7.6 HiZ (high impedance) Output

HiZ (high impedance) can be explicitly assigned to a "ter" type output variable. The bidirectional "ter" type variable has a default value of HiZ. In addition, HiZ cannot be assigned to other variable types.

```
out ter(0:8) foo;

process bar() {

    foo = HiZ;
    $ 
    foo = a;
    $ 
    foo = 0;
    $ 
}
```

7.7 Port Generation - State Number Output

Options	Description
-Zgen_stateno_out	Generate a port named "ST1out" to output state number
-Zgen_stateno_out= <i>name</i>	Generate a port with port name <i>name</i> to output state signal

When the option "-Zgen_stateno_out" is specified, a "ter" type output port with the name "ST1out" is generated that outputs the current state number.

In case a port name is explicitly specified by using the option -Zgen_stateno_out, the port is generated with the port name "*name*" and current state number is output.

However, if FSM is not generated due to there being just one state, the port is not generated.

7.8 Unused Ports Handling

Options	Description
-Zport_asitis	Also generate unused ports (default)

-Zport_optimize	Do not generate unused ports
-Zport_optimize_auto	Only delete auto generated unused ports (default)
-apI:o, -apA:o, -ap#:o	Do not generate unused ports for external memory.

- **Input-Output Ports**

The existence of unused 'reg type' output port results in an error. If physical type is not defined or if 'ter type', 'var type' output port is not used then 0 is fixed with a warning message. If the option "-Zport_optimize" is specified, unused ports are not generated.

When an input port is not used, it is generated as a floating port. Unused input ports are deleted only when the option "-Zport_optimize" is specified.

When bidirectional ports are not used, then they are fixed at High Impedance along with a warning message. Unused bidirectional ports are generated even when option "-Zport_optimize" has been specified.

- **External Memory**

The use of external memory leads to the generation of few ports for connecting to the memory. In case external multi-port memory is having a port attached to circuit being synthesized, where this port is not being used by circuit, then it becomes unused. By default, the unused ports are deleted even if memory port is specified/unspecified in port information (DEFMOD) of memory library file (MLIB).

Specifying ":o" (-apA:o, -ap:o, -ap#:o) after the external memory usage related options (-apA, -ap, -ap#) specifies Cyber not to generate these unused ports.

However, in case the memory port is specified in port information (DEFMOD) in memory library file (MLIB), these options are invalid and ports are generated even though they are unused.

And when -Zport_asitis option is specified, then Cyber will take care of it by generating:

- a. Unused output ports with output fixed to "0"
- b. Unused input port with floating input
- c. Unused bidirectional port with high impedance output

- **Shared Register**

When a shared register is specified, three shared register ports are generated. For example, When values in a shared register are only referred and not updated, the write-enable port and write-data port will remain unused. In case of default and when -Zport_optimize is specified then they are to be deleted. Also, when '-Zport_asitis' option is specified then unused output ports for shared registers are generated with output fixed to "0", and unused input ports are generated with floating inputs.

Note: Handling of unused ports may change in future.

7.9 Output Ports - Specifying Default Value

Attribute	Description	Settings Target
/*Cyber default_value = value */	Set default value of output port to value	out ter

In the case of "out ter" type variables, if the output is not specified explicitly in the program, a default value of "0" is driven in output. This default output value can be changed by specifying the attribute "default_value" along with an "out ter" variable.

The value that can be specified is either high impedance (HiZ), numeric value expressed as a 32bit signed number, indefinite value (Don't Care). Numeric values can be specified in a way similar to that of the BDL output values in any of the binary, octal, decimal or hexadecimal format. High impedance can be specified using the string "HiZ". When bitwidth specification is not done, all the bits are set to HiZ. Indefinite value can be specified with "DontCare".

```
out ter(0:8) foo1 /* Cyber default_value = 0xFF */;
out ter(0:8) foo2 /* Cyber default_value = 255 */;
out ter(0:8) foo3 /* Cyber default_value = 0b11111111 */;
out ter(0:32) foo4 /* Cyber default_value = -1 */;
out ter(0:8) foo5 /* Cyber default_value = HiZ */;
out ter(0:8) foo6/* Cyber default_value = DontCare */;
```

In case the attribute "sig_pol = neg" is specified for a "ter" type output port, where the default value is also specified, both the attributes will become valid. In other words, inversed default value is output in a state that is without any specification of output value.

```
out ter(0:8) foo4 /* Cyber default_value =0xF0, sig_pol = neg */;
→ 0x0F is output
```

In case the value specified exceeds the limit that can be specified using a 32bit signed number, the value will be set to 0x7FFF_FFFF. Further any value that surpasses upper limit of "out ter" variable's bit-width, will be ignored.

```
out ter(0:8) foo1 /* Cyber default_value =1023 */;
→ 0xFF gets set (higher bits beyond range are ignored)

out ter(0:32) foo2 /* Cyber default_value = 0xFFFFFFFF */;
→ 0xFFFFFFFF gets set (outside 32-bit signed range)

out ter(0:8) foo3 /* cyber default_value= hiz */;
→ 0 gets set (incorrect character string)
```

Value of "out ter" variable becomes 0 when none-use mode is used. However, if it is specified with "default_value" attribute, it will generate attribute value. However, in case of none-use mode, 0 is generated when "default_value=DontCare" is specified.

When the option "-Zport_optimize" is specified, it will be deleted while using none-use mode even if the "default_value" attribute is specified in the "out ter" variable.

7.10 Port Validating Signal (s)

Options	Description
-Zport_valid_sig_gen[=YES]	Generate valid signal for each I/O port
-Zport_valid_sig_gen=NO	Do not generate valid signal for each I/O port (default)
-Zinternal_valid_sig_gen[=YES]	Generate valid signal for each I/O port as internal signal
-Zinternal_valid_sig_gen=NO	Do not generate valid signal for each I/O port (default)

Attribute	Description	Settings Target
<code>/* Cyber valid_sig_gen = name */</code>	Generate valid signal name of specified I/O variable as output port of positive logic.	input, or output, or I/O variable
<code>/* Cyber valid_sig_neg_gen = name */</code>	Generate valid signal name of specified I/O variable as output port of negative logic.	input, or output, or I/O variable

Valid signals are required to validate input, output or I/O variables while communicating with other interfacing modules like FIFO. In such cases, the valid signals are generated when the attribute "valid_sig" is specified to I/O variables.

Refer the table below that describes various scenarios for generating port validating signals.

- To generate active high (positive polarity) validating signals, an attribute "valid_sig" can be specified along with input, output or I/O variable. Based upon variable type, appropriate port(s) will be generated that will be asserted when input or output variable value is valid.
- To generate active low (negative polarity) validating signals, an attribute "valid_sig_neg" can be specified along with input, output or I/O variable.

In some scenarios, a valid port asserted for only one cycle is needed, for ex. when synthesis circuit acquires an input value or provides an output value to the interface of other modules such as FIFO. In such cases, to generate the valid signal, attribute valid_sig is added to I/O variable that outputs the timing.

If valid_sig attribute is specified for input, output or bi-directional variables, then a port of attribute value (active HIGH) is generated and when variable value is input or output, this port is asserted for one cycle and it is kept de-asserted for rest of the cycles.

If valid_sig_neg attribute is specified for input variable, output variable or bi-directional variable, then a port of attribute *value* (active LOW) is generated and when variable value is input or output, this port is asserted (LOW) for one cycle and it is kept de-asserted (HIGH) for rest of the cycles.

If option –Zport_valid_sig_gen is specified, valid signal will be generated for all I/O variables. Valid signal will be generated as an output port of *name*, where “_v” is specified after supporting I/O variable *name*.

Generated port is asserted HIGH for one cycle whenever variable value is input or output and de-asserted for all other cycles, same as in case of the attribute valid_sig.

If option –Zinternal_valid_sig_gen is specified, then valid signal will be generated for all I/O variables. Valid signal will be generated as internal ter type variable of *name*, where “_v” is added after supporting I/O variable *name*. Generated signal is asserted HIGH for one cycle whenever variable value is input or output and de-asserted for all other cycles, same as in case of the attribute valid_sig.

Example below demonstrates how to specify the attribute for input, output, and I/O variable for all the “ter”, “reg” and “var” types. Related timing diagram for these variables and corresponding validating signal has been shown below in **Figure 8**. One needs to carefully handle validating ports when they are not generated for ‘out reg’ type variable, since hazards will occur at clock boundary as in case of ‘out ter’ type.

In the case of “inout ter” type variable, names of both input and output validating port needs to be specified by separating them with “：“.

```
in  ter(0:8) a    /* Cyber valid_sig = a_sel */;
in  reg(0:8) b    /* Cyber valid_sig = b_sel */;
in  var(0:8) c    /* Cyber valid_sig = c_sel */;
out ter(0:8) d    /* Cyber valid_sig = d_sel */;
out reg(0:8) e    /* Cyber valid_sig = e_sel */;
out var(0:8) f    /* Cyber valid_sig = f_sel */;
inout ter(0:8) g   /* Cyber valid_sig = in_sel:out_sel */;
```

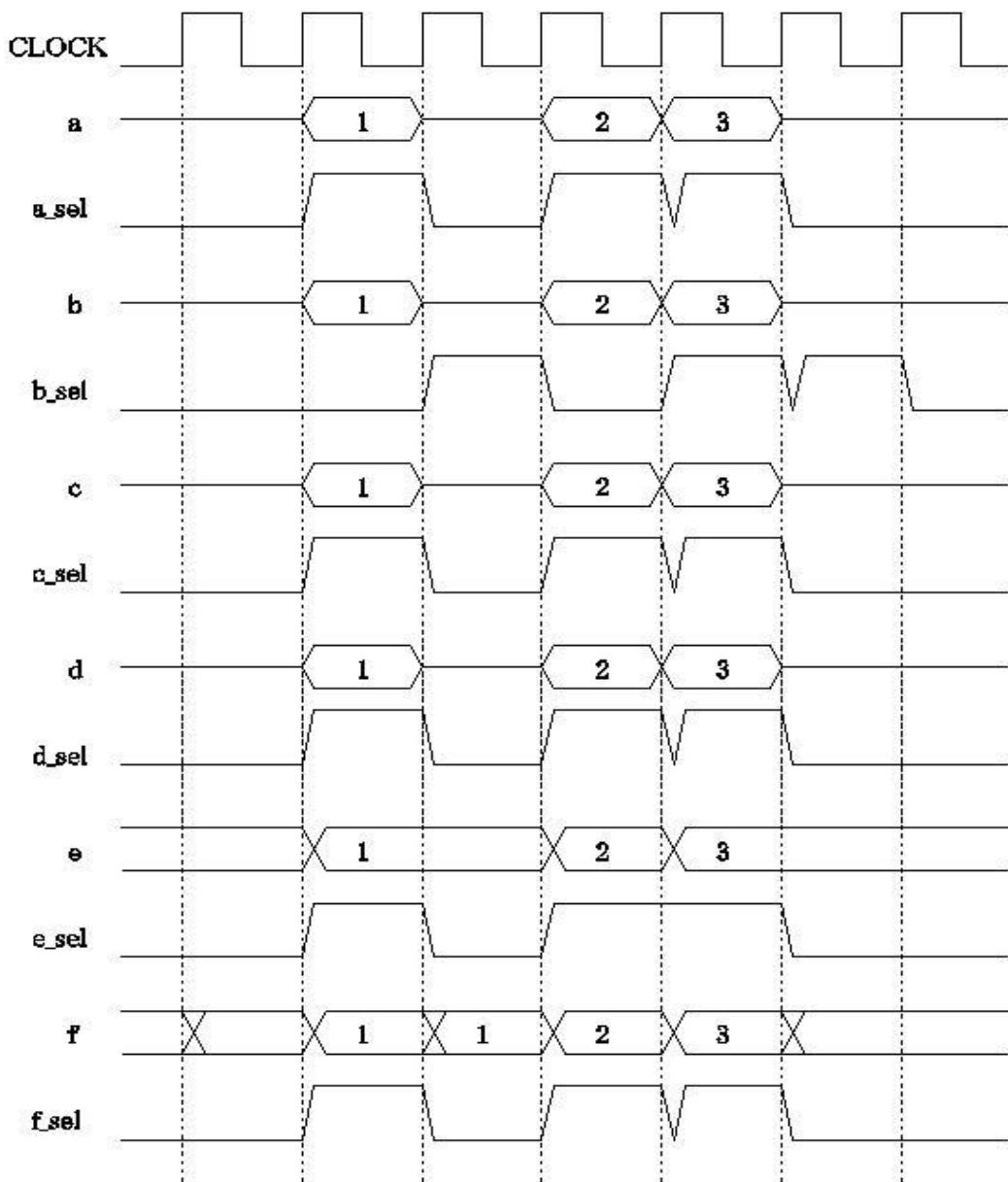


Figure 8: Timing chart of valid signals

7.11 Declaration of valid signal explicitly (Functionality for simulation)

In this section, it explains about the functionality of (valid_sig(_neg)_bind attribute) that declares valid signal explicitly.

The attribute valid_sig(_neg)_bind is mainly used for simulation. Actually, it is an attribute which enables multiple processes simulation functionality combined with SystemC/Verilog cycle accuracy model generating tool (cmscgen/cmveriloggen) or auto RTL test bench generation tool (tbgen).

In section **7.11.2**, it explains basic use of valid_sig(_neg)_bind attribute. In **7.11.3**, it is shown that more efficient verification is possible by combining valid signal with SystemC/Verilog cycle accuracy model generating tool or auto RTL test bench generation tool. In section **7.11.4**, it explains advanced notation where valid_sig(_neg)_bind attribute is used. In the last section **7.11.5**, it explains limitations of valid_sig(_neg)_bind attribute.

7.11.1 Related attribute

Attribute	Description	Settings Target
<code>/* Cyber valid_sig_bind= name */</code>	Use user declared external out port name as valid signal of positive logic.	input, output or I/O variables
<code>/* Cyber valid_sig_neg_bind= name */</code>	Use user declared external out port name as valid signal of negative logic.	input, output or I/O variables

7.11.2 Basic usage

User can use the output signal declared explicitly as valid signal by using valid_sig(_neg)_bind attribute.

In case of valid_sig(_neg)_gen attribute explained in section **7.10**, signal specified by attribute and its output description are generated automatically. But, in case of valid_sig(_neg)_bind attribute, user must declare output signal specified by the attribute and user must describe output timing of output signal also.

In case of valid_sig_bind attribute, user can use output signal declared explicitly as valid signal of positive logic. But, in case of valid_sig_neg_bind attribute user can use output signal declared explicitly as valid signal of negative logic.

Though specification format of the attribute is same as valid_sig(_neg)_gen but, in this case valid signal can be used for multiple I/O ports.

Output timing of valid signal can be controlled by using id attribute and scheduling/blocks.

In below example, for structure member input signal st.r_in, st.g_in and st.b_in, one valid signal st.valid is used. Valid signal st.valid can be asserted in same cycle of the input timing of these input signals by using scheduling block decription.

```

struct ST {
    in ter (0:8)r_in,g_in,b_in;
    out ter (0:1) valid;
}st
/*Cyber .r_in={valid_sig_bind=st.valid},
.g_in={valid_sig_bind=st.valid},
.b_in={valid_sig_bind=st.valid}*/;
out ter (0:8)o;
process main(){
    int r,g,b;
    /*Cyber scheduling_block*/
    {
        r = st.r_in;
        g = st.g_in;
        b = st.b_in;
        st.valid=1;
    }
    ...
}

```

7.11.3 Simulation of multiple processes combined with sub-tool

In this section, it is shown that transaction mode simulation of multiple processes is possible by combining valid signal with SystemC/Verilog cycle accuracy model generation tool (cmscgen/cmveriloggen) and auto RTL test bench generation tool (tbgen). And it is shown that verification operation can be executed efficiently.

As shown in below example desired specification description (for instance, JPEG encoder and triple DES, etc.) is divided into processes and functional verification is executed by summarizing these processes.

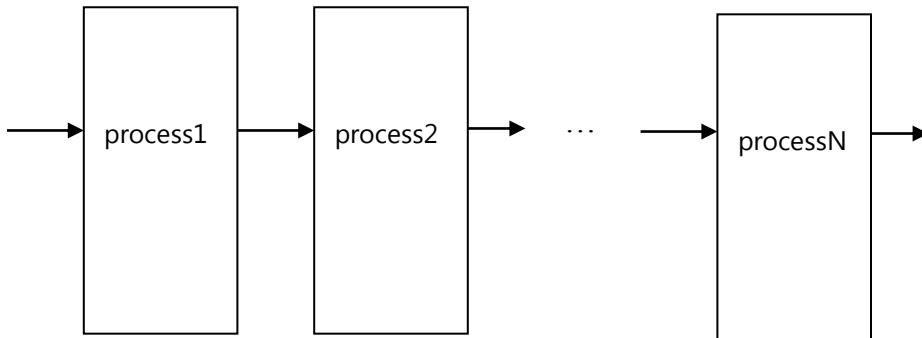


Figure 9: Example for dividing the processes

I/F of these processes are generally changed based on I/F specification decided by user while designing HW. Therefore, appropriate protocol conversion description (example is shown below) is prepared by using valid signal. SystemC cycle accuracy model generation tool and auto RTL test bench generation tool connects valid signals of processes seamlessly by setting top hierarchy description (it can be generated through top hierarchy generation tool (see section L)), and transaction mode simulation of multiple processes can be executed (**Figure 10**).

As a result, test bench used in transaction mode simulation of multiple processes and test bench used in C verification execution before process division, can be shared and used and hence, **verification hours can be reduced greatly**.

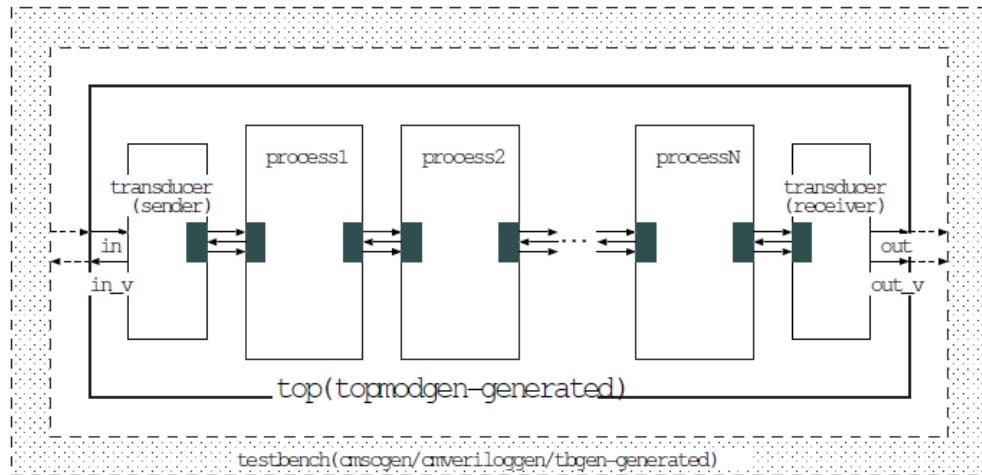


Figure 10: After addition of protocol converter (transducer)/ I/F change

Example of protocol converter (sending end):

```
/* I/F of verification process */
in ter req_i;
out ter ack_i;
out ter (0:8)data_i;
/* I/F of test bench*/
in ter (0:8) data_in
/*Cyber
valid_sig_bind=data_in_v*/;
out ter data_in_v;

process sender() {
    reg (0:8)data_r;

    while (1) {
        data_r = data_in;
        data_in_v = 1;
        ack_i = 0;
        $
        wait(req_i) {
            ack_i = 1;
            data_i = data_r;
        }
        $
    }
}
```

Example of protocol converter (receiving end):

```
/* I/F of verification process */
out ter req_t;
in ter ack_t;
in ter (0:8)data_t;
/* I/F of test bench */
out ter (0:8)data_out
/*Cyber
valid_sig_bind=data_out_v*/;
out ter (0:1)data_out_v;

process receiver() {
    reg (0:8)data_r;
    while(1) {
        wait(ack_t) {
            req_t = 1;
        };
        data_r = data_t;
        $
        req_t = 0;
        data_out = data_r;
        data_out_v = 1;
        $
    }
}
```

7.11.4 Applicable notation of valid_sig(_neg)_bind attribute

Transaction mode simulation using SystemC/Verilog cycle accuracy model generation tool and auto RTL test bench generation tool can be executed normally with valid signal specified by valid_sig(_neg)_bind attribute, even if it is not asserted in the same cycle of specified signal, (Refer to the user manual of SystemC/Verilog cycle accuracy model generation tool and auto RTL test bench generation tool for the details of transaction mode simulation.)

- If signal that specifies valid_sig(_neg)_bind attribute is an input signal, transaction mode simulation can be executed normally if **valid signal is asserted till the cycle just before when next input value is read**.
- If signal that specifies valid_sig(_neg)_bind attribute is of out var type, output signal of out reg type, transaction mode simulation can be executed normally if **valid signal is asserted till the cycle just before when next output value is written**. However, if signal which specifies valid_sig(_neg)_bind attribute is of out ter type, valid signal must be asserted in same output timing.

Examples and timing chart of valid signal of input signal (in ter type,in var type) are shown below. Since valid signal x_v and x2_v are asserted in the cycle just before when next input value x and x2 are read hence, transaction mode simulation is executed normally using SystemC/Verilog cycle accuracy model generation tool (cmscgen/cmveriloggen) and auto RTL test bench generation tool (tbgen).

```
in ter (0:8)x
/*Cyber valid_sig_bind=x_v*/;
out ter x_v;
in var (0:8)x2
/*Cyber valid_sig_bind=x2_v*/;
out ter x2_v;

out ter (0:8)y
/*Cyber valid_sig_bind=y_v*/;
out ter (0:1)y_v;
out var (0:8)y2
/*Cyber valid_sig_bind=y2_v*/;
out ter (0:1)y2_v;

process sample() {
    int a;
    /*Example of describing valid signal for in ter signal*/
    a = x;
    $;
    y = a;
    x_v = 1;
    y_v = 1;
    $
    a = x;
    x_v = 1;
    $;
    y = a;
    y_v = 1;
    $
    /*Example of describing valid signal for in var signal*/
    input(x2);
    $;
    y2 = x2;
    $;
    x2_v = 1;
    $;
    output(y2);
    y2_v = 1;
    $;
    y2 = x2;
    x2_v = 1;
    $;
    output(y2);
    $;
    y2_v = 1;
}
```

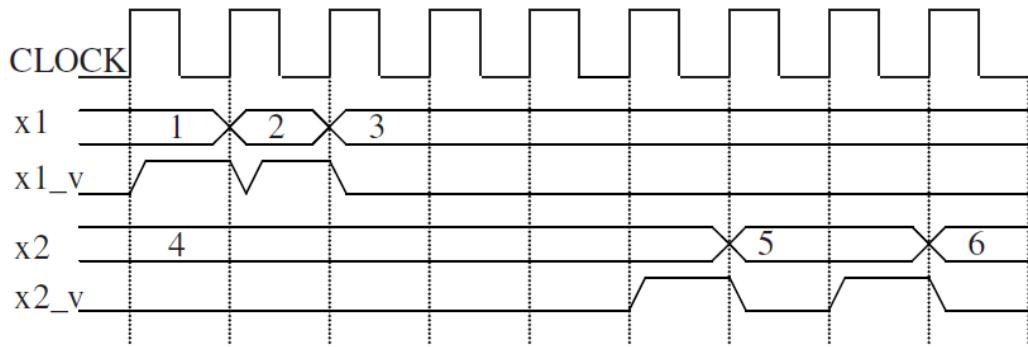


Figure 11: Input signal and timing chart of the valid signal

Further, example of valid signal ($y_1_v \sim y_3_v$) for output signal ($y_1 \sim y_3$) and the timing chart are shown below.

- Valid signal y_1_v for out ter type output signal y_1 is asserted in the same cycle of y_1 .
- Valid signal y_2_v for out var type output signal y_2 is asserted in the cycle just before when next output value is written.
- Valid signal y_3_v for out reg type output signal y_3 is asserted in the cycle just before when next output value is written.

Therefore, transaction modesimulation using SystemC/Verilog cycle accuracy model generation tool (cmscgen/cmveriloggen) and auto RTL test bench generation tool (tbgen) is executed normally.

```

out ter (0:8)y1/*Cyber valid_sig_bind=y1_v*/;
out ter (0:1)y1_v;

out var (0:8)y2/*Cyber valid_sig_bind=y2_v*/;
out ter (0:1)y2_v;

out reg (0:8)y3/*Cyber valid_sig_bind=y3_v*/;
out ter y3_v;

process sample2() {
    int a,b,c;

    /*Example of describing valid signal for out ter signal*/
    y1 = 1;
    y1_v = 1;
    $;
    /*Example of describing valid signal for out var signal*/
    y2 = 2;
    $;
    output(y2);
    $;
    $;
    y2 = 3;
    y2_v = 1;
    $;
    output(y2);
    y2_v = 1;
    $;
    /*Example of describing valid signal for out reg signal*/
    y3 = 4 ;
    $;
    y3_v = 1;
    y3 = 5;
    $;
    $;
    y3_v = 1;
}

```

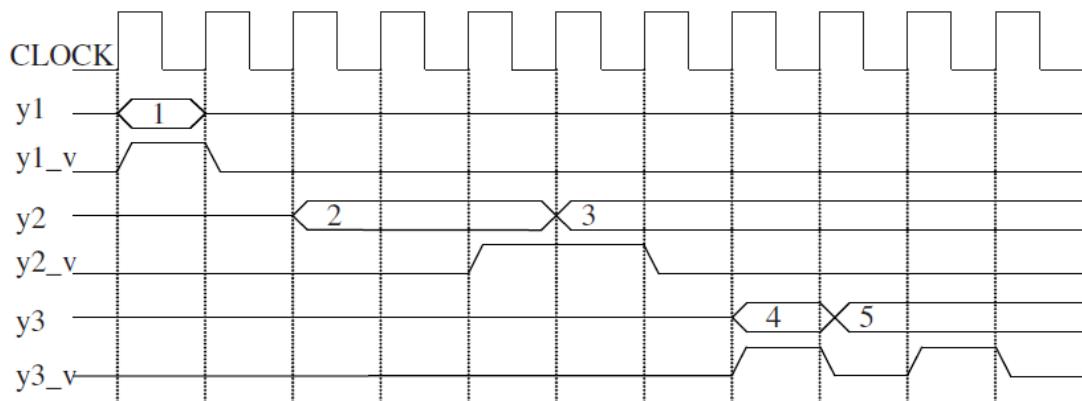


Figure 12: Output signal and timing chart of the valid signal

7.11.5 Notes and limitations

- This attribute can be specified only for in/out/inout signal and cannot be specified for pointer variable, clock/reset/set signal, outside/shared signal and defmod signal.
- Only user declared output variable of 1 bit can be specified in this attribute.
- If signal specified by this attribute, valid_sig(_neg) _gen attribute and sig_pol attribute cannot be specified for the same signal.
- Valid_sig_bind attribute and valid_sig_neg_bind attribute cannot be specified at the same time.
- When specifying -Zport_valid_sig_gen/-Zinternal_valid_sig_gen, generating valid signal is out of the scope for variable specifying valid_sig(_neg)_bind attribute/ variable specified in valid_sig(_neg) _ bind attribute.

7.12 Wait valid signal in SystemC

Options	Description
-Zwait_valid=OUT	Generates wait_valid signal as an output port.
-Zwait_valid=INTERNAL	Generates wait_valid signal as an internal signal.
-Zwait_valid=NO	Does not generate wait_valid signal (default)

There is a specific output valid signal for SystemC (SC_CTHREAD) inputs, "wait_valid signal". The wait_valid signal is asserted HIGH for one cycle whenever wait() statement is executed in SystemC description and for all other timings it is kept de-asserted.

Such signals are required because it is impossible to know through normal valid port of each port in SystemC description that how many times input port is actually processed. In case of SystemC description, even the input which is not actually used, gets read internally. In the example given below, it is not fixed whether in0 value between two wait() statements will be used or not, till the time of execution. However, value of in0 gets fetched during first wait statement execution, therefore, even if the value of in0 is not used, valid signal corresponding to in0 may be asserted . In that case, it is impossible to know whether the input is used or not through valid port of each I/O port. Therefore, valid_sig attribute and -Zport_valid_sig_gen / -Zinternal_valid_sig_gen options cannot be used for SystemC description.

```

sc_in<int> in0, in1;
sc_out<int> out0;
.

.

.

wait();
s = t + u + v;
if(in1.read()) {
    out0.write(in0.read() + s); // in0 is used only when in1 is true
} else {
    out0.write(0);
}
wait();

```

Generally, in case of SystemC (SC_CTHREAD) description synthesis, wait statement is executed in every cycle in manual scheduling. However, in automatic scheduling mode, synthesis between two wait() statements can be carried out as multiple cycles, therefore, sometimes interval of 2 cycles or more is required for executing wait statement. On the other hand, in case of SystemC (SC_CTHREAD) description, the value of I/O port between two wait() statements is updated only once. Therefore behavioral level simulation result and its comparison can be done by observing each input and output at the time when wait() is executed. Hence, in other words, after using

wait_valid signal, post behavioral synthesis circuit can be easily verified by checking the assertion of this port (which specifies that wait() statement was executed).

However, in case synthesis is done with System C or with Auto scheduling, then in order to do optimization of reducing the cycle count by ignoring unnecessary wait for updating the input-output value, then the input description may have synthesis result of different cycle count or wait timing. In such cases, since the input-output timing between input description and synthesis result is different, therefore the value of input-output cannot be verified using 'wait_valid'. It is better to specify "-Zboundary_wait=YES" option at the time of scpars input in order to carry out synthesis without ignoring wait in auto scheduling.

By default wait_valid signal is not generated, however, it can be generated as external port or internal signal through -Zwait_valid option.

7.13 Output port attributes of "out var" variable

The following table shows attributes through which the output port attributes of "out var" variable can be modified:

Attribute	Description	Setting Object
/*Cyber port_type = TER */	Set output attribute of 'out var' variable to 'ter' type	out var variable
/*Cyber port_type = REG */	Set output attribute of 'out var' variable to 'reg' type	out var variable
/*Cyber port_type = VAR */	Set output attribute of 'out var' variable as 'var' type (default)	out var variable

For an "out var" variable, the timing for the output value changes according to the attribute specified. In case nothing is specified for the output port attribute, it is configured as a "var" type (default).

Figure 13 shows timing diagram for every output attribute type for an "out var" variable and **Figure 14** shows circuit generated for respective attribute.

- **When attribute "port_type = TER" is specified**

Value is driven only in cycle in which output is generated using an "output" function. When the output is not generated in a cycle, a "0" is driven.

- When attribute “port_type = REG” is specified

After an output is generated in a cycle using an “output” function, the value is driven at the next cycle. The value is retained and kept driven till next output is generated.

- When attribute “port_type = VAR” is specified

Value is driven in a cycle in which an output is generated using an “output” function. The value is retained and kept driven till the next output is generated.

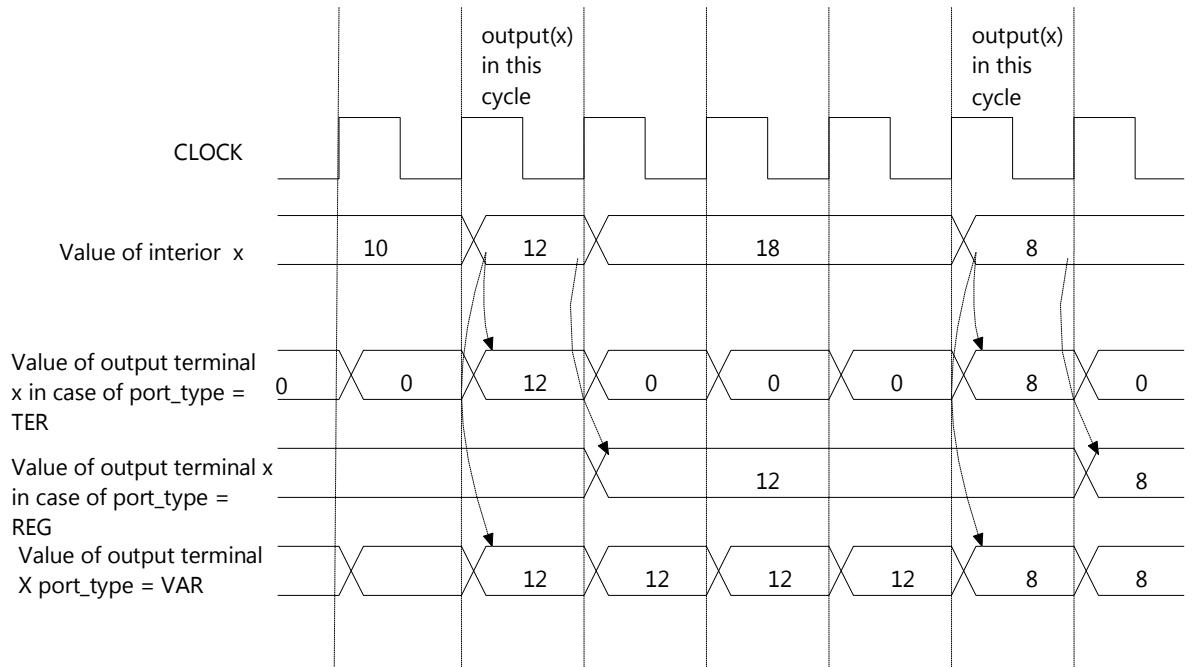


Figure 13: Timing chart of outputs for each attribute type of “out var” variable

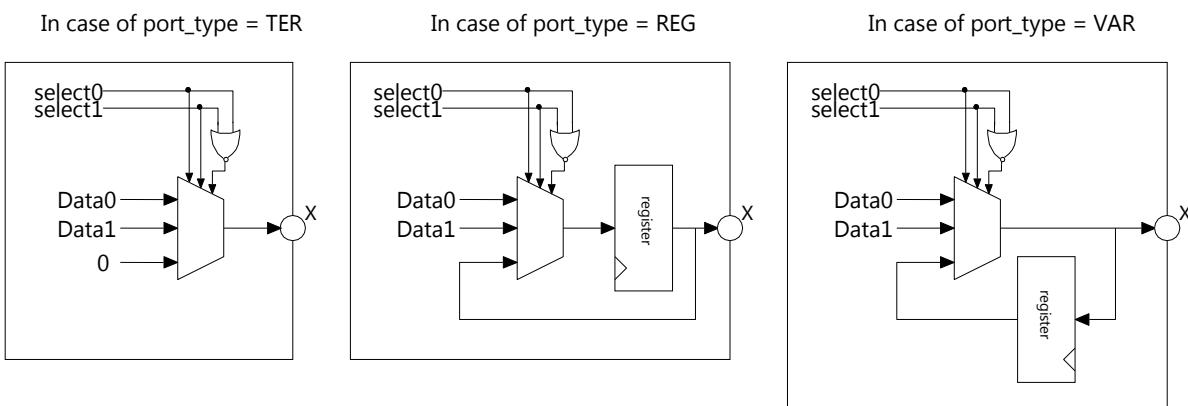


Figure 14: Output port circuits for each attribute type of “out var” variable

7.14 Generation of synchronization circuit between out-of-phase clocks

Options	Description
-Zsynchronizer	Generate synchronization circuit between out-of-clocks for data signal
Zsynchronizer=#1:#2	Generate synchronization circuit between out-of-phase clocks for data signal. The number of stages of register for inserting enable and data signal in sending-clock domain is taken as # 1. The number of stages of register for inserting enable signal in receiving-clock domain is taken as # 2.
-Zin_port_synchronizer=#	Generate synchronization circuit between out-of-phase clocks at input port. The number of stages of register for synchronization circuit between out-of-phase clocks is taken as # stages.

Attribute	Description	Setting Object
/* Cyber synchronizer */	Generate synchronization circuit between out-of-clocks for data signal	Input, output or I/O
/* Cyber synchronizer = #1 : #2 */	Generate synchronization circuit between out-of-phase clocks for data signal. The number of stages of register for inserting enable and data signal in sending-clock domain is taken as # 1. The number of stages of register for inserting enable signal in receiving-clock domain is taken as # 2.	Input, output or I/O

<pre>/* Cyber port_synchronizer = # */</pre>	<p>Generate synchronization circuit between out-of-phase clocks at input port.</p> <p>The number of stages of register for synchronization circuit between out-of-phase clocks is taken as # stages.</p>	<p>Input</p>
--	--	--------------

Meta-stable problem occurs when data is transferred between out-of-phase clock domains. In order to prevent the Meta-stable problem, function that obtains data by using synchronization circuit is explained.

7.14.1 Synchronization circuit generation function between out-of-phase clocks

This section explains function that generates synchronization circuit between out-of phase clocks. As shown in the below example, it considers the cases where data is sent between out-of-phase clocks domain (C1, C2). In this case, synchronization circuit is inserted between out-of-phase clocks during top module synthesis by specifying attribute synchronizer in I/O signal (Data) which are the sending-receiving target data. Further, synchronization circuit between out-of-phase clocks is determined automatically as per the clock frequency of sending and receiving. When “–Zsynchronizer” option is specified during synthesis, synchronization circuit between out-of-phase clocks is generated among all I/O variables that perform asynchronous transmission. For example, if clock frequency of receiving data is more than the sending data, circuit (MUX Multiplexer) is generated in top module as shown in the **Figure 15**. Example of timing is shown in **Figure 16**.

Module foo (data sending description):

```
clock C1;
in ter (0:8) x;
out reg (0:8) Data/*Cyber
    synchronizer,
    valid_sig_gen=Data_en */;
process foo() {
    Data = x;
}
```

Module bar (data receiving description):

```
clock C2;
in ter (0:8) Data/*Cyber
    synchronizer*/;
in ter Data_en;
out reg (0:8)y;
process bar() {
    int tmp_en, tmp_data;
    wait(tmp_en) {
        if (tmp_en = Data_en) {
            tmp_data = Data;
        }
    }
    y = tmp_data;
}
```

Top module top (use top module output tool “L”):

```
defmod foo {
    clock      C1;
    in ter(0:8)      x;
    out ter(0:8)     Data;
    out ter(0:1)     Data_en;
} INST_foo;

defmod bar {
    clock      C2;
    in ter(0:8)      Data;
    in ter(0:1)      Data_en;
    out ter(0:8)     y;
} INST_bar;

in ter(0:1)  C1;
in ter(0:8)  x;
in ter(0:1)  C2;
ter(0:8)     Data/* Cyber synchronizer */;
ter(0:1)     Data_en;
out ter(0:8) y;

process top() {
    INST_foo.C1 ::= C1;
    INST_foo.x ::= x;
    Data ::= INST_foo.Data;
    Data_en ::= INST_foo.Data_en;
    INST_bar.C2 ::= C2;
    INST_bar.Data ::= Data;
    INST_bar.Data_en ::= Data_en;
    y ::= INST_bar.y;
    return;
}_
```

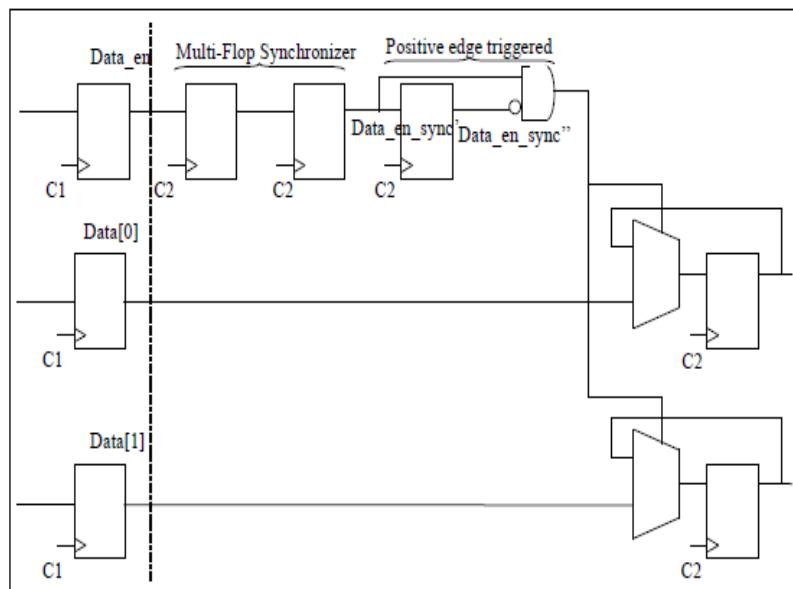


Figure 15: Synchronization circuit between out-of-phase clocks (when clock frequency is three times or more)

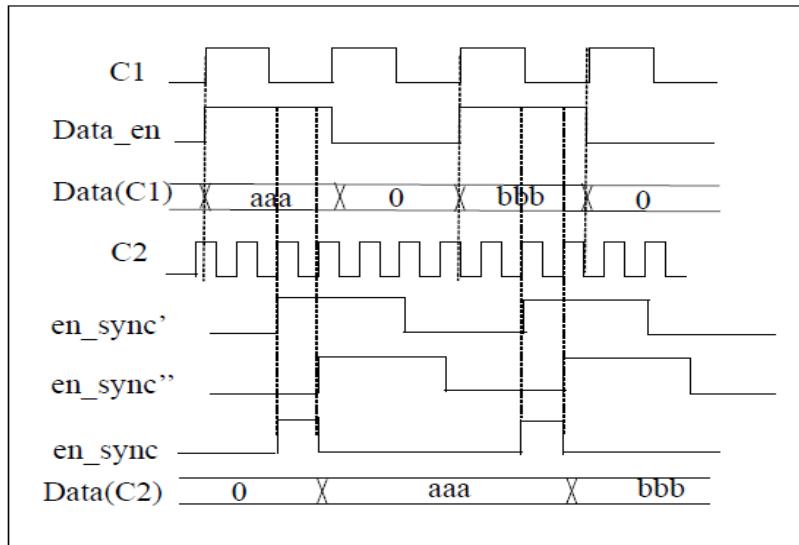


Figure 16: Timings example of data transmission between out-of-phase clocks (when clock frequency is three times or more)

7.14.2 Function that generates synchronization circuit between out-of-phase clocks at the input port

This section explains the functionality that inserts synchronization circuit between out-of-phase clocks at the input port.

A functionality exists that automatically inserts such synchronization circuit between out-of-phase clocks at the input port.

In case "port_synchronizer" attribute is specified in input signal to be inserted, then it automatically inserts the number of stages of register that is specified with attribute value for synchronization circuit between out-of-phase clocks.

In the below example, two stage insertion of register is done for synchronization circuit between out-of-phase clocks at input start signal.

```

in  ter(0:1) start /* Cyber port_synchronizer = 2 */;

in  ter(0:1) finish;
in  ter(0:8) in1,in2;
out reg(0:8) out1;

process foo(){
    while( start == 0 );

    while( finish != 0 ){
        out1 = in1 & in2;
    }
}

```

In case "-Zin_port_synchronizer=#" attribute is specified, then it inserts the # stages of register for synchronization circuit between out-of-phase clock in all the target input signals.

Registers for input are generated without reset. Reset signal of register to be inserted can be modified by specifying "reset_sig" attribute along with input signal. Clock signal of register to be inserted can be modified by specifying "clock_sig" of register to be inserted can be modified. "port_synchronizer" attribute can be specified in stall signal which has been specified with "stall_control" attribute.

In case of stall signal, registers to be inserted are not stalled with stall signal.

Reset signal is not supported currently and it cannot be specified so it shows error.

In case input is specified with "valid_sig" attribute then, there is a limitation that cycle of valid signal that becomes 1 from importing the actual value would get delayed only by the number of stages in register. It means that in case "port_synchronizer = 2" is specified in the signal that specifies valid_sig, then value of 2 cycles before value signal becomes 1, is imported.

7.15 Register generation in input output port

Option	Description
-Zin_port_reg_stage=#	Generate the register of # stages in input port
-Zout_port_reg_stage=#	Generate the register of # stages in output port

Attribute	Description	Target Setting
/* Cyber port_reg_stage = # */	Generate the register of # stages in input/output port	Input, output

When controlling the delays or when deciding a circuit specification, there are cases when register is inserted in input port, output port.

In such cases, there is a function which automatically inserts register in specified input port, output port.

If port_reg_stage attribute is specified in input signal, output signal inserting the register, then register of specified stages is automatically generated by attribute value.

In the below example, register is inserted in input start signal for 2 stages, and register is inserted in output end signal for 1 stage.

```

in ter(0:1) start /* Cyber port_reg_stage = 2 */;

in ter(0:1) finish;
in ter(0:8) in1,in2;
out reg(0:8) out1;
out ter(0:1) end /* Cyber port_reg_stage = 1 */;

process foo() {
    while( start == 0 );

    while( finish != 0 ){
        out1 = in1 & in2;
    }
    end = 1;
}

```

If option -Zin_port_reg_stage=# is specified, then register is inserted in all target input signal for the specified number of stages. Further, option-Zout_port_reg_stage=# is specified, then register is inserted in all target output signal for the specified number of stages.

Register generated in input signal is generated without reset. Reset signal of inserted register can be specified by specifying input signal according to reset_sig attribute. Clock signal of register to be inserted can be modified by specifying "clock_sig" attribute along with the input signal.

Register generated in output signal is generated with reset. Reset signal of inserted register can be changed by specifying output signal according to reset_sig attribute. Clock signal of register to be inserted can be modified by specifying "clock_sig" attribute along with the input signal.

In case of stall signal, inserted register is stalled by stall signal.

In case valid_sig_gen attribute is specified in input, it has a restriction that, from cycles which import the value, only the registered stage(s) gets delayed in the cycle wherein valid signal becomes 1. In particular,

when port_reg_stage=# is specified in *in ter* signal specifying valid_sig_gen attribute, then import the value before 2 cycles of valid signal having timing 1.

When valid_sig_gen attribute is specified in output, then timing is assumed to be same because register of same stage is generated in the created valid signal.

In case of using valid_sig_bind attribute, installation of register is not done automatically for valid signal specified in attribute value.

7.16 Summary

This section covered the following:

- I/O ports in synthesized circuit are sequenced according to the order of I/O variables declaration.
- Parameters are given higher priority between parameters for process function and global variables.
- The following two methods can be used for sharing one port among multiple I/O variables through time sharing:
 - I/O variables are allocated to ports automatically by specifying only the type and number of ports using a port constraint file.
 - I/O variables are explicitly allocated to ports by using a port constraint file and port relation file.
- Generation of unused ports is handled using the "-Zport_asitis", "-Zport_optimize", "-ap:o", "-apA:o", and "-ap#:o" options

Port validating signals are generally required to qualify (validate) input, output or the I/O variables while communicating with other interfacing modules like FIFO.

8 Clock

8.1 Overview

This section covers the following:

- Clock cycle constraint specification.
- Specification of the clock port name for the entire circuit.
- Clock signal specification for each register.

8.1.1 Related Options

The options for clock signal specifications are as follows:

Option	Description
-c#	Specifies clock cycle in #
-cuunit	Specifies time unit of clock in 'unit' (example 1/100ns)
-cd#	Specifies clock uncertainty (uncertainty of clock or clock skew) in #
-Zbase_unitunit	Specifies smallest delay unit in unit (example 1/100ns)
-Fca	Generate a clock signal if required (default)
-Fcf	Generate a clock signal irrespective of its requirement.
-Fc:name	Generates clock port with name 'name' (Default clock port name is "CLOCK")
-Fc+	Generate circuit that operates at rising edge of clock (default)
-Fc-	Generate circuit that operates at falling edge of clock
-FcE	Generate clock pin separately for register, which synchronizes at negative phase.

8.1.2 Related Attributes

Attribute	Description	Setting Target
<code>/*Cyber clock_edge = pos */</code>	Synthesize circuit such that it operates at rising edge of clock	clock, variable
<code>/*Cyber clock_edge = neg */</code>	Synthesize circuit such that it operates at falling edge of clock	clock, variable
<code>/*Cyber clock_sig = name */</code>	Use variable 'name' as clock for the register of variable for which this attribute has been specified.	variable
<code>/*Cyber clock_pol = pos */</code>	Register of variable for which attribute has been specified is operated at rising edge of clock	variable
<code>/*Cyber clock_pol = neg */</code>	Register of variable for which attribute has been specified is operated at falling edge of clock	variable

8.2 Clock Cycle Constraint Specification

Clock cycle must be specified as a constraint in automatic scheduling.

Clock cycle is specified by using the “-c” option. Moreover, clock cycle can be specified only with a positive integer.

By default, clock cycle unit is 1/100 nano seconds (ns). Clock cycle unit can be changed using the “-cu” option. Using the “-cu” option, following 18 types can be specified for clock cycle unit.

10000ps	10000ns
1000ps	1000ns
100ps	100ns
10ps	10ns
1ps	1ns
1/10ps	1/10ns
1/100ps	1/100ns
1/1000ps	1/1000ns
1/10000ps	1/10000ns

In case clock cycle is not specified, higher value out of following is taken as constraint clock cycle for synthesis:

- The maximum delay value of functional unit type specified in the functional unit constraint (refer to section **39.2**).
- The maximum delay value of memory type specified in the memory constraint (refer to section **39.3**).

In case no clock cycle has been specified, an error will occur if there is at least one delay value specification by "cycle specification (refer section **39.2**) in the functional unit constraint file or the memory constraint file.

When the clock uncertainty (uncertainty of clock or clock skew etc.) has been specified, then the data path delay constraint is determined by calculating the difference between the constraint clock cycle and the indefinite period. The unit of clock uncertainty is similar to -c.

In manual scheduling, since scheduling is determined only in description, clock cycle is not used as a constraint.

The minimum delay value for synthesis can be specified by -Zbase_unit option. The units which can be specified though this option are same as the units used with -cu option. 1/100ns unit is defined by default. In case a unit smaller than 1/100ns is specified by -cu option or functional unit constraint, minimum delay value needs to be lowered by specifying this option.

8.3 Clock Port Name Specification

When no clock port name has been specified in either options specified or specification description, "CLOCK" will be used for clock port name, and circuit operating on rising edge of this signal will be generated.

However when clock is not required, clock port is not generated. Clock is typically not required when no module requiring clock in circuit exists. If the "-Fcf" option is specified then Clock port is generated even when clock is not required.

There are two methods for explicitly specifying the clock port name. One method is to specify the clock port name in description using the "clock" declaration. The other method is to specify the

clock port name using the synthesis option. If both the methods are used simultaneously, an error will occur.

- **Using the “clock” declaration**

When a variable is declared using clock declaration, clock port will be generated using specified variable name.

```
clock CLK; // generation by port name CLK
```

By default, the circuit generated will operate on the rising edge of clock. In case the circuit operating at the falling edge of clock needs to be generated, the “clock_edge” attribute can be specified with the clock variable.

For clock signals, only “in clock” attribute is permitted. Therefore, the “out clock” and “inout clock” attributes is treated as an error.

```
clock CLK /* Cyber clock_edge = neg */; // falling edge specification method
```

- **Using the synthesis option**

When the “–Fc+” option is specified, the entire circuit will operate at the rising edge of clock. Otherwise, when the “–Fc-” option is specified, the entire circuit will operate at the falling edge of clock.

Moreover, when the “–Fc:name” option is specified, a clock port with the name “name” will be generated.

Note: The “–Fc” options need to be specified all at once. The syntax for specifying the “-Fc” options are as follows:

–Fc[a|f][:name][+|-]

8.4 Clock Signal Specification

In case of clock signal different from the entire circuit which is to be specified separately for each register by input output variable registers, etc. or at the time of defining the structure description, variable name of the register which is to be changed to clock should be specified using the `clock_sig` attribute. This attribute is typically used to specify the clock signal that is different from the default clock signal of the entire circuit.

The “`clock_edge`” attribute is used to configure the polarity as clock of signal that is specified using the “`clock_sig`” attribute.

In case clock polarities, such as rising / falling edge, need to be specified separately for each register, it can be specified using the “`clock_pol`” attribute.

In case attribute “`clock_pol`” contradicts with attribute “`clock_edge`”, attribute “`clock_pol`” is given priority.

```
in ter (0:1) clock2;  
  
// register, which connects to signal "clock3", is operated at the rising  
edge  
in ter (0:1) clock3 /*Cyber clock_edge = pos */;  
  
// register r1 is operated at falling edge of signal clock 2  
reg (0:8) r1 /*Cyber clock_sig = clock2, clock_pol = neg */;  
  
// connection is made to clock signal clock3 of output register variable o1  
out reg (0:8) o1 /*Cyber clock_sig = clock3 */;  
  
// register o2 is operated at falling edge of signal clock 3  
out reg (0:8) o2/*Cyber clock_sig = clock3, clock_pol = neg */;
```

In the example above, ‘clock’ declarator has not been used for variable “`clock2`”, which is used as a clock signal. This is because if a variable is declared as clock using the “`clock`” declaration, the variable will represent the default clock for the entire circuit. Therefore, from second clock onwards, the clock signals must be declared as normal 1-bit variable without using any “`clock`” declaration.

8.5 Summary

This section covered the following:

- Clock cycle is specified by using the “-c” option.
- The default clock cycle unit is 1/100 ns.
- The two methods for specifying the clock port name for the entire circuit is as follows:
 - Using “clock” declaration
 - Using the synthesis option
- The clock signal specification for each register is done using the “clock_sig” attribute

9 Reset

9.1 Overview

This section explains about reset specification for a circuit and method to specify reset signal for each register individually. It covers following:

- Reset port specification
- Circuit reset technique (Asynchronous/synchronous)
- Specification of reset state.
- Reset of registers with the initialization provided with initial value.
- Specification of individual reset signal for each register.
- Specification of register without reset.

9.1.1 Related Options

The options for specifying reset signal is as follows:

Option	Description
-Fra -FrF	Adds the reset signal whenever required (default). Adds reset signal.
-Fr+ -Fr-	Generates reset at Active High (positive logic) (default) Generates reset at Active Low (negative logic)
-Fr:name	Generates reset with port name name (default port name is "RESET")
-Zdefault_reset	Specifies reset signal which connects to the module having only one reset.
-Zsync -Zasync -Zreset_synchronizer -Zreset_synchronizer=#	Generates fully synchronous reset Generates fully asynchronous reset (default) Resets at asynchronous clock and generates circuit that removes reset at synchronous clock. Resets at asynchronous clock and generates circuit that removes reset at synchronous clock. (The number of stages of register for inserting reset input is set as #.)
-Zreset_state=Auto -Zreset_state=Yes -Zreset_state = NO	It is decided automatically whether to generate reset state (default) Generates reset state Don't generate reset state (default)

-Zreg_min_reset -Zreg_min_reset = NO	Don't generate reset in register without initial value Generates reset in register without initial value (default)
---	---

Option that becomes valid at the time of specifying “-Zreg_min_reset” are as follows:

Option	Description
-Zreg_reset_cone	Generates reset in register that is connected to output signal
-Zreg_reset_cone = NO	Don't generate reset even for register that is connected to output signal (default)
-Zreg_reset_cone_valid_sig	Generates reset in register that is connected to valid_sig output signal
-Zreg_reset_cone_valid_sig = NO	Don't generate reset for register that is connected to valid_sig output signal

Following two options are supported by the commands “vhdlgen” and “veriloggen”.

Option	Description
-Zreset_macro=individual	Generates reset of register with initial value same as provided at asynchronous reset
-Zreset_macro=inverter	By using inverter logic, generates reset of register with initial value same as provided at asynchronous reset.

9.1.2 Related Attributes

Attribute	Description	Target Object
/*Cyber reset_mode = sync */	Generates fully synchronous reset.	Reset, Variable
/*Cyber reset_mode = async */	Generates fully asynchronous reset.	Reset, Variable
/* Cyber reset_synchronizer */	Resets at asynchronous clock and generates circuit that removes reset at synchronous clock.	Reset, Variable
/* Cyber reset_synchronizer = # */	Resets at asynchronous clock and generates circuit that removes reset at synchronous clock.	Reset, Variable

Attribute	Description	Target Object
	(The number of stages of register for inserting reset input is set as #.)	
/*Cyber reset_active = high */ /*Cyber reset_active = low */	Generates reset by Active High (positive logic). Generates reset by Active Low (negative logic).	Reset, Variable Reset, Variable
/* Cyber default_reset */	Specifies reset signal which connects reset signal to the module having only one reset.	Reset, Variable
/*Cyber reset_macro = individual */ /*Cyber reset_macro = inverter */	Considers the reset of register, which are provided with an initial value, as asynchronous reset. Considers the reset of register, which are provided with an initial value, as asynchronous reset and implements in system where inverter is used.	Variable Variable
/*Cyber reset_sig = name */ /*Cyber reset_sig = void */	Converts the reset signal of register, where a specified variable has been allocated, to variable name. Do not generate the reset of register where a specified variable has been allocated.	Variable Variable
/*Cyber reg_reset_cone = YES */	Generates the reset in register that is connected to specified output signal even at the time of “-Zreg_min_reset”.	Output Variable

Attribute	Description	Target Object
/*Cyber reg_reset_cone_valid_sig = YES */	Generates reset in register that is connected to the "valid_sig" signal of specified output signal even at the time of "-Zreg_min_reset".	Output Variable

9.2 Reset Port Specification

When a design contains no reset port or options specifications and synthesized circuit are sequential, a reset port with positive logic is generated with a port name "RESET". A reset port is not generated when none of the modules in a design requires a reset port. However, when the "-Fr" is specified option is used, a reset port is generated even if it is not required.

There are two methods to specify a reset port name. One method is to specify the reset port name in the "reset" declaration within the design description. The other method is to specify the reset port name using the synthesis option. If both the methods are specified simultaneously, an error will occur.

- Using the "reset" declaration

When using the "reset" declaration, a reset variable is specified in the declaration. The reset port is generated using the name of the reset variable specified.

All variables will become reset port in case multiple variables are declared by using reset declaration.

```
reset RST; // generates by port name "RST"
```

By default, the generated reset port is set to the active high behavior (positive logic). In case active low (negative logic) behavior is needed, the "sig_pol" attribute should be specified with the reset variable.

```
// Specification method for Active Low (negative logic)
reset Reset /*Cyber reset_active = low */;
```

- Using the synthesis option

If the “-Fr+” option is specified, the reset signal becomes positive logic thereby causing circuit to reset on **High** signal. When the “-Fr-”option is specified, the reset signal becomes negative logic thereby causing circuit to reset on **Low** signal.

Moreover, if “-Fr: *name*” option is specified, the reset port name is generated with the name specified in the “*name*” attribute.

All “-Fr” options must be collated and specified as per the following syntax:

```
-Fr[{:a|f}] [:name] [{+|-}]
```

9.3 Reset techniques

9.3.1 Fully asynchronous/synchronous reset

This section explains fully asynchronous and synchronous reset as reset techniques. By default it synthesizes with fully asynchronous reset which has less circuit area (for reset synchronizer, a functionality that resets at asynchronous clock and removes reset at synchronous clock (refer section **9.3.2**).

Designers can generate either asynchronous or synchronous reset for a circuit. By default, Cyber generates asynchronously reset circuits, which requires less "die" area.

The **Figure 17** and **Figure 18** show the circuit design and timing diagram in fully asynchronous reset mode. Similarly, the **Figure 20** show the circuit design and timing diagram in fully synchronous reset mode.

Although fully asynchronous reset is vulnerable to noise due to its asynchronous behavior, its advantage is that it requires less "die" area. On the other hand, fully synchronous reset comparatively requires larger "die" area, but is less susceptible to noise as it reacts only at clock edges.

There are two methods to explicitly specify fully asynchronous or synchronous modes. One method is to specify using the "reset_mode" attribute with a variable declared in the "reset" declaration. The other method is to specify using the synthesis option. If both the methods are specified, the attribute overrides over the synthesis option.

- Using the "reset_mode" attribute

The modes are specified using the "reset_mode" attribute with a variable declared using the "reset" declaration.

```
// specification of synchronous reset mode
reset Reset /* Cyber reset_mode = sync */ ;
```

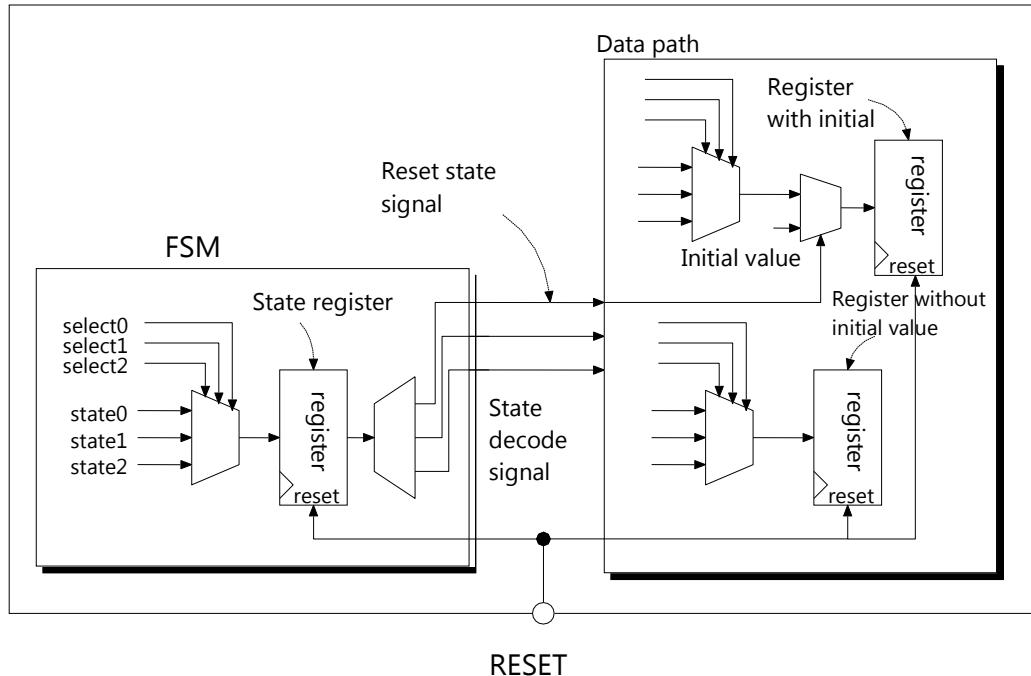


Figure 17: Fully asynchronous reset mode - circuit design (when reset state is deleted)

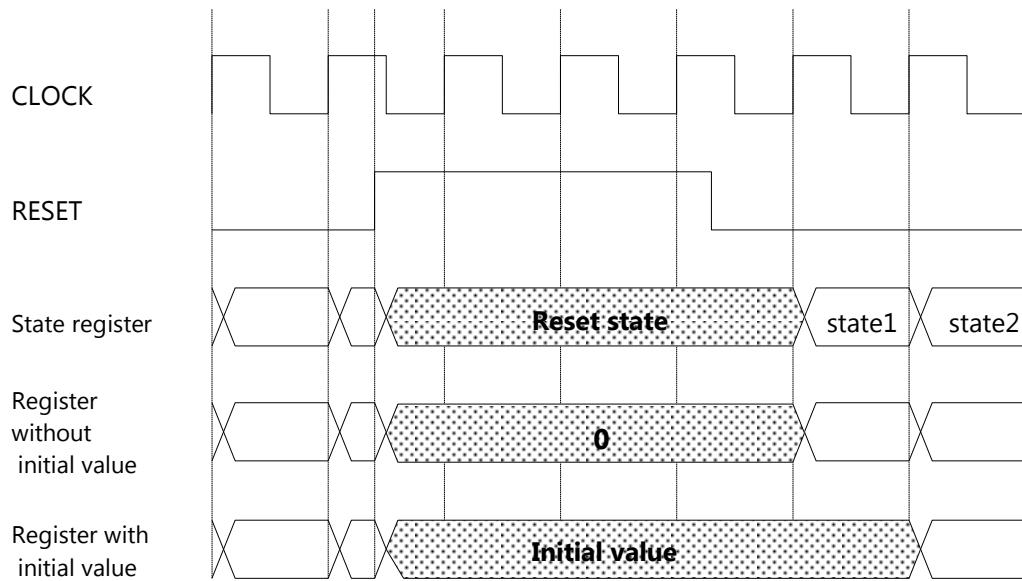


Figure 18: Fully asynchronous reset mode – timing diagram (when reset state is deleted)

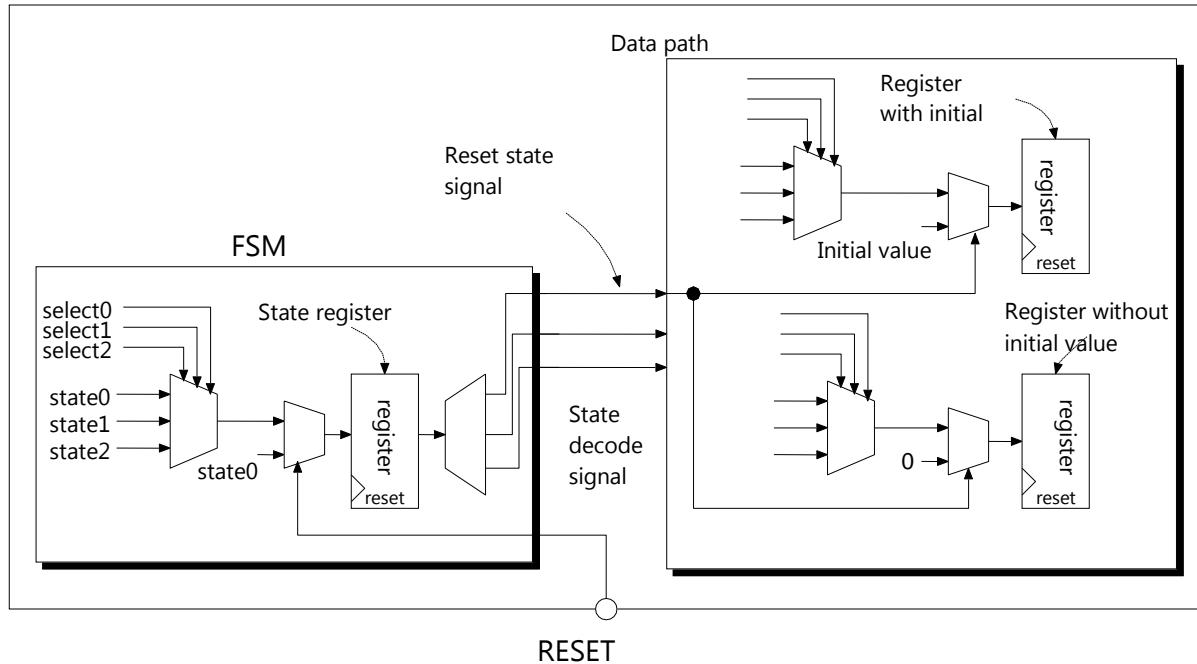


Figure 19: Fully synchronous reset mode - circuit design (when reset state is deleted)

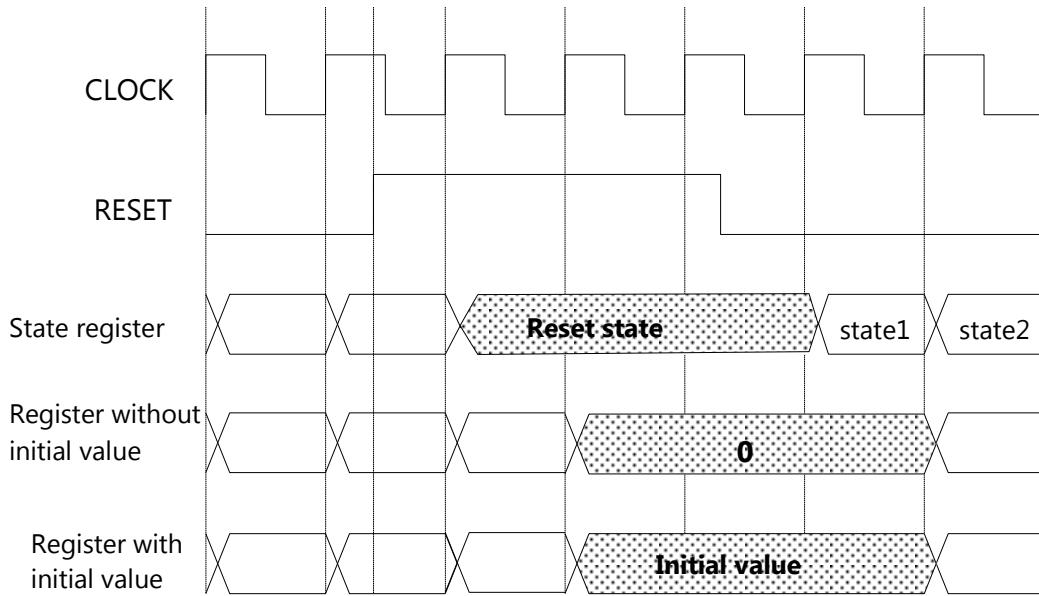


Figure 20: Fully synchronous reset mode - timing diagram (when reset state is deleted)

```
// Specification of fully synchronization reset
reset Reset/* Cyber reset_mode = sync */;
```

```
// Specification of fully asynchronous reset
reset Reset/* Cyber reset_mode = async */;
```

```
// Reset signal of fully asynchronous reset.
// Declaration of reset signal of fully synchronous reset.
reset aReset /* Cyber reset_mode = async */;
reset sReset /* Cyber reset_mode = sync */;
```

- Using the synthesis option

By default, a fully asynchronously reset mode is selected. This is the same as specifying “-Zasync” option. However, if the “-Zsync” option is specified, a fully synchronous reset mode is selected.

If variable has the initialization value and states count of circuit after synthesis is one, by default the circuit will have only data path without generating any FSM. **Figure 21** shows the circuit design if there is reset state of fully asynchronous reset circuit which has state count 1. **Figure 22** shows the circuit design if there is reset state of fully synchronous reset circuit which has state count 1.

9.3.2 Reset Synchronizer

Generally, there is metastable² issue in fully asynchronous reset. However, as shown in the below circuit design, if reset function is used (reset synchronizer) that removes the reset in clock asynchronous, counter measures for metastable was taken to achieve a highly secure reset.

There are two methods to use this reset function. First method is to specify using “reset_synchronizer” attribute with variable declared in reset declaration. The other method is to specify using synthesis option -Zreset_synchronizer. Further, it is possible to specify integer value (number of stages of register to be inserted for reset input) in attribute and option (2 can be set if numeric specification is omitted).

² Phenomenon where FF value becomes inaccurate in skew if asynchronous reset gets deleted and clock timing gets started almost at the same time.

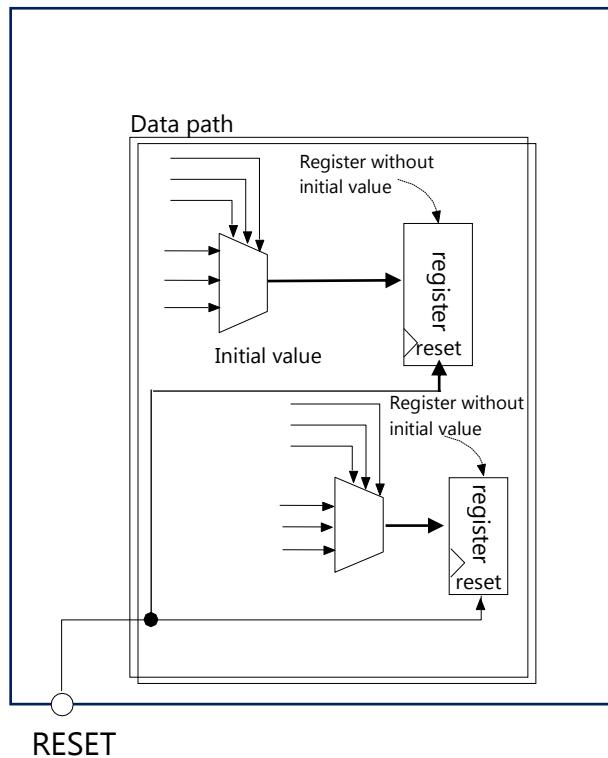


Figure 21: Fully Asynchronous reset mode – circuit design (when number of states is just 1, reset state is deleted)

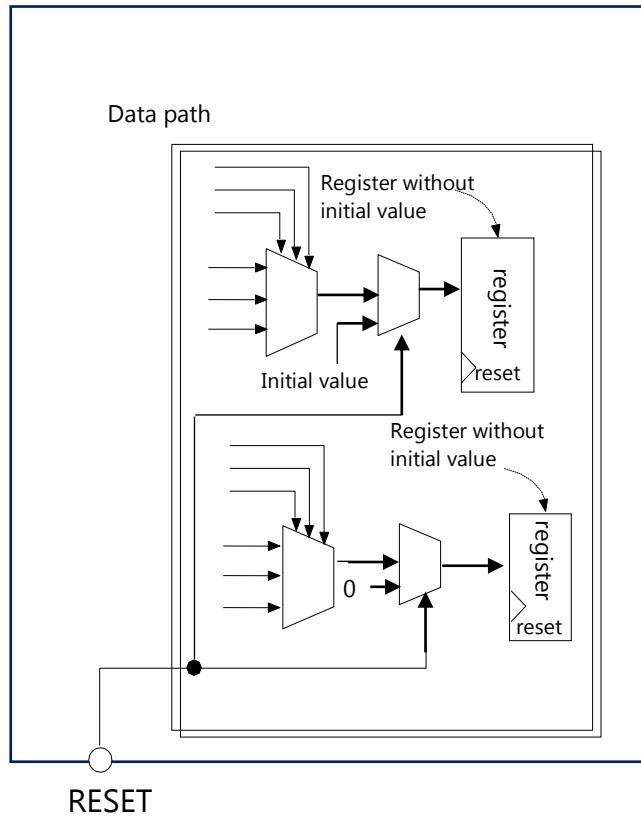


Figure 22: Fully synchronous reset mode – circuit design (when number of states is just 1, reset state is deleted)

```
// Specification of reset synchronizer
reset Reset /* Cyber reset_synchronizer */;
```

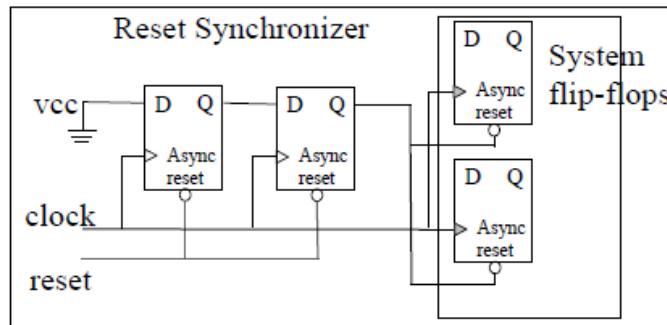


Figure 23: Reset synchronizer - Circuit design

9.4 Reset State Specification

Generation of the reset state of circuit can be controlled by using the “-Zreset_state” option.

By default, if the “-Zreset_state” option is set as NO, all registers including registers with initialization value in the data path, get reset by external reset signal without generating the reset state. Since setting of initial value of variable is also controlled by external reset, it is executed simultaneously with the reset of register without initial value.

If “-Zreset_state” option is set as AUTO, a FSM having reset state is generated if circuit has multiple behavioural states or, circuit has variable with initialization value. A circuit is generated with data path only without any reset state and without generating any FSM if circuit has single behavioral state and there is no variable with initialization value.

If “-Zreset_state” option is set as YES, reset mode is certainly generated. In other words, FSM is also generated after reset state gets generated even if behavioral state is just 1 and variable with initialization value does not exist.

Figure 24 and **Figure 25** shows the circuit design and timing of asynchronous reset mode when reset state exists.

Figure 26 and **Figure 27** shows the circuit design and timing of synchronous reset mode when reset state exists.

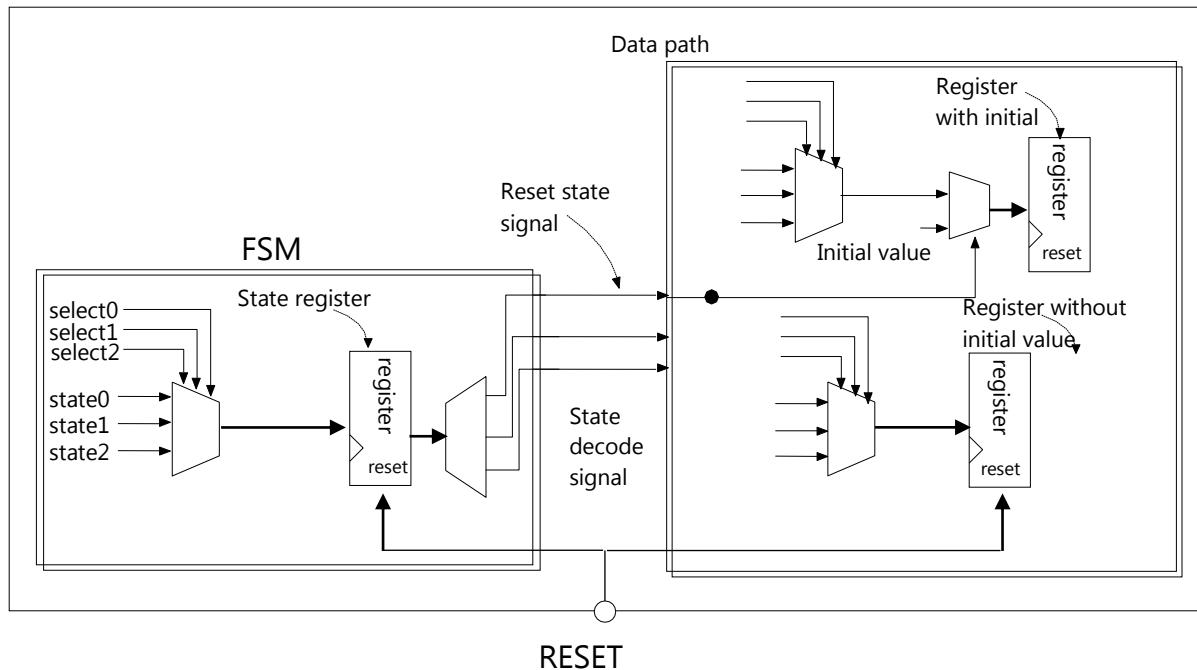


Figure 24: Asynchronous reset mode-Circuit design (when reset state exists)

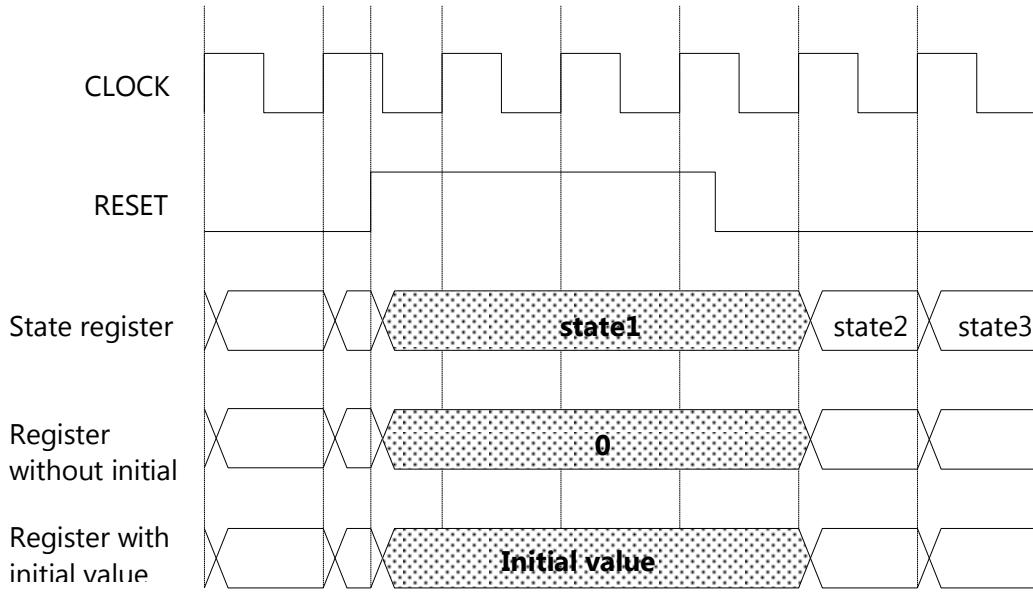


Figure 25: Asynchronous reset mode-Timing chart (when reset state exists)

Reset state is generated in synchronization reset also for register with initialization value if register state is generated. This is because they get initialized by passing through a reset state based on behavior of exit() statement in BDL description. If register with initialization value is to be reset asynchronously, it can be achieved either by doing the specification that does not generate reset state as explained in this section or by particularly specifying method explained in section 9.6. If there is no reset state, reset cannot be done during the operation of circuit hence, state after execution of exit() statement becomes the first state.

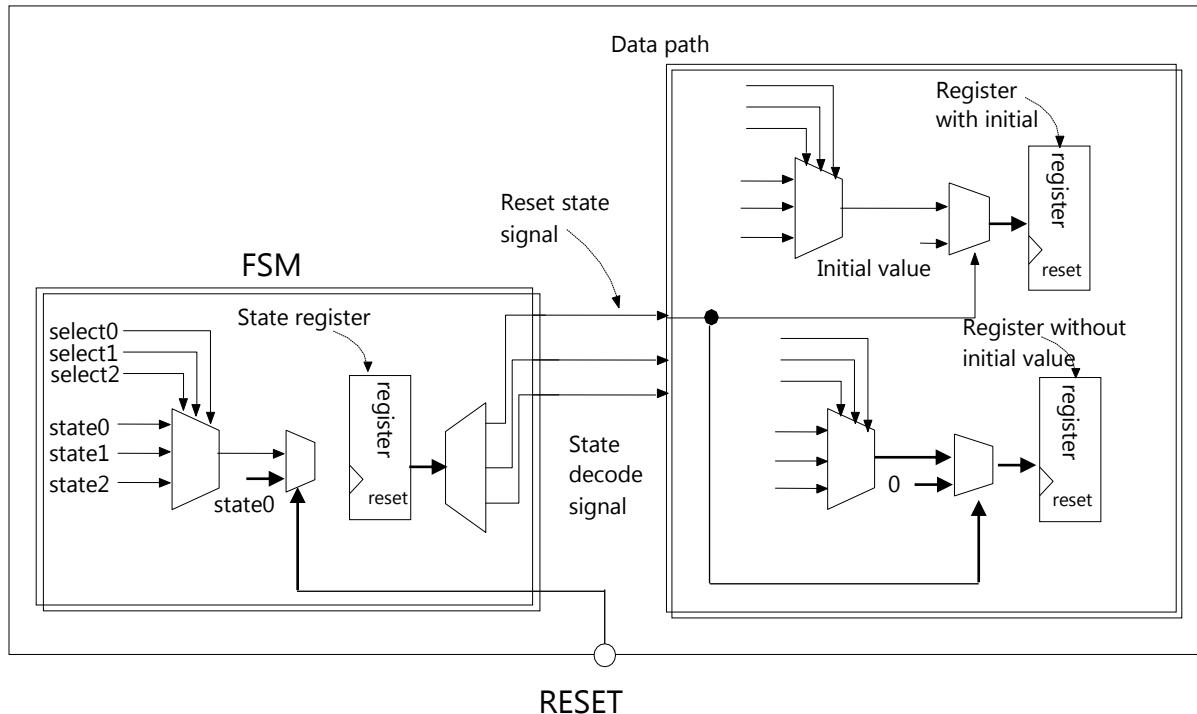


Figure 26: Synchronization reset mode - Circuit design (when reset state exists)

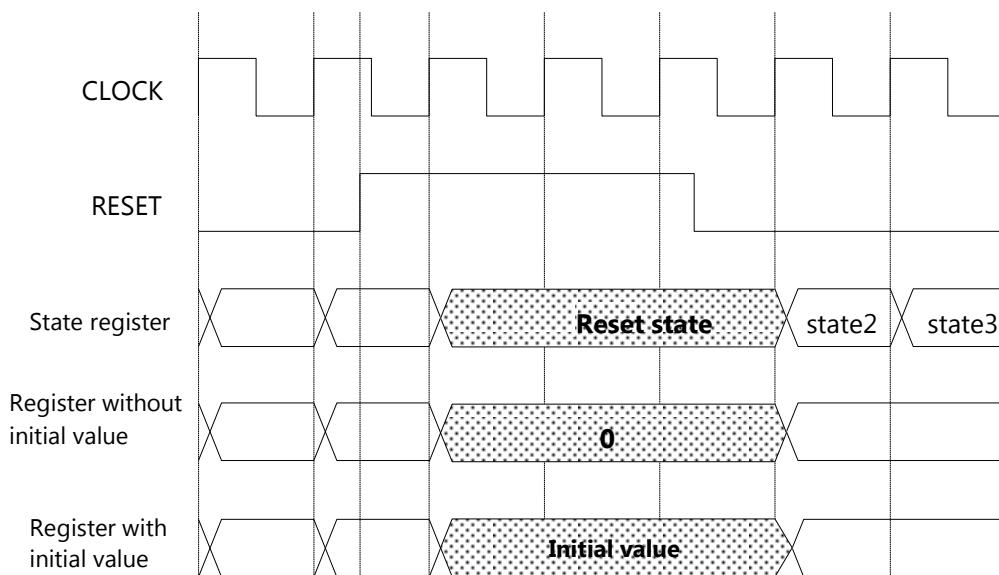


Figure 27: Synchronization reset mode - Timing (when reset state exists)

9.5 Specification of default reset

In case of multiple reset signals, which reset signal will be connected to the module having only one reset, can be specified by using default_reset attribute or –Zdefault_reset option.

There are modules which have only one reset port in memory and functional unit. In case of multiple reset signals, default reset signal is the signal of reset port which connect to the module having only one reset port. If many reset signals are declared, default reset signal is determined based on below rules.

- Variable specified by default_reset attribute
- Variable specified by –Zdefault_reset option if no variable has been specified with default_reset attribute,
- Reset variable defined in starting if non of the default_reset attribute and –Zdefault_reset option is specified

default_reset attribute is specified at the time of declaration of the reset signal. In below example, asynchronous reset signal arst and synchronous reset signal srst are defined. Here, srst is the default reset signal as it is specified as default_reset attribute.

```
reset arst/* Cyber reset_mode = async */;
reset srst/* Cyber reset_mode = sync, default_reset */;
```

Specify variable name as value of the option if reset signal is specified with -Zdefault_reset option. For example, if you specify -Zdefault_reset=srst, srst will be the default reset signal.

9.6 Reset Method Specification of Register with Initial Value

Reset method of registers with initialization value can be specified by attribute 'reset_macro' and synthesis options. Currently, it can be specified in the following two formats:

- The first format is called individual type. In this format, reset values are described as shown in the example below. In this format, reset values depend on the logical synthesis tool and library. However, it is assumed in this format that a set port exists in the Flip-Flop, and this Set Port can be used.
- The alternate format is known as inverter type. This format is intended to be used for multi-bit registers and Flip Flops with only reset port. Generally, multi-bit registers and Flip Flop with only reset port cannot be asynchronously set with values other than 0. Therefore, the value 1 is set by inserting an inverter in the input/output of the bit. However, adequate care should be taken during the simulation. This is because the value to be stored in the register gets inverted.

output of individual type

Example:

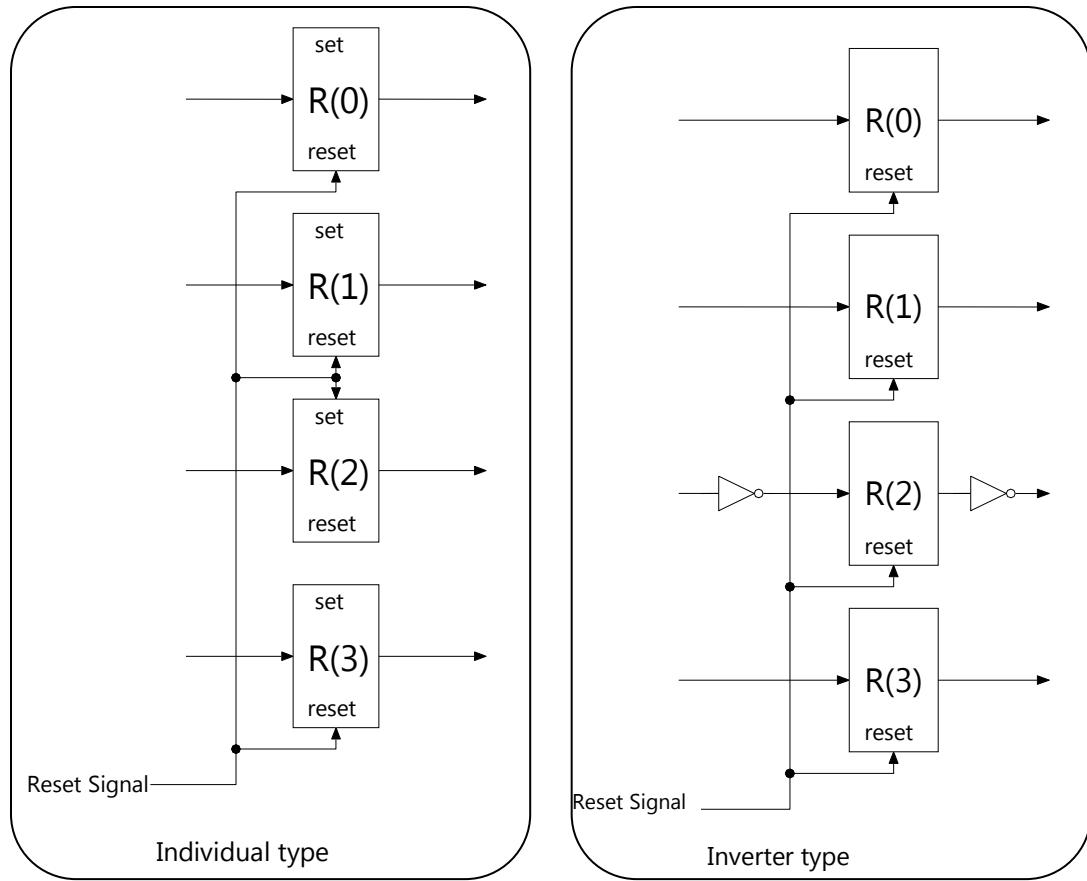
```
reg(0:4) R = 2;
```

VHDL Output:

```
VHDLgen_Label1003 :process (CLOCK, RESET)
begin
    if ( RESET = '1' ) then
        R <= "0010";
    elsif ( CLOCK'event and CLOCK = '1' ) then
        R <= VHDLgen_tmp5003(0 to 3);
    end if;
end process;
```

Verilog-HDL Output:

```
always @ ( posedge CLOCK or posedge RESET)
    if ( RESET )
        r <= 4'h2 ;
    else
        r <= VerilogOut_tmp0 ;
```



There are two methods to specify the above mentioned types of generations. One method is specifying through the attribute. The other method is specifying through the options of the VHDL output command "vhdlgen" and the Verilog-HDL output command "verilogen".

- Specification using the attribute
A register with an initial value is annotated with the "reset_macro" attribute to generate asynchronous reset.

```
reg(0:8) x/* Cyber reset_macro = individual */ = 1;
reg(0:8) y/* Cyber reset_macro = inverter */ = 2;
```

- Specification using the options of the VHDL output command and the Verilog-HDL output command.

Following options can be specified with both these output commands to generate asynchronous reset for a register variable with initialization value:

Option	Description
-Zreset_macro=individual	reset of register with initial value is assumed to be asynchronous reset.
-Zreset_macro=inverter	reset of register with initial value is assumed as asynchronous reset and it is implemented by the method of using inverter.

In case attribute contradicts with options, attribute is given priority.

Normally, when reset state is reached using the exit () statement in the BDL description, the value of each signal gets initialized to its initialization value specified during declaration. However, in case register with initialization value is asynchronously reset using above mentioned specification, the value of each signal doesn't get initialized even if it has reached the reset status by exit () statement in the BDL description.

9.7 Register wise specification of reset signal and logic

When reset signal different from the entire circuit is to be specified individually for each register in input-output variable or structure description, specify variable name that is to be used as reset by specifying attribute reset_sig along with variable. Specify name of variables by separating with ":" when multiple reset signals are to be specified for registering.

Logic of reset signal (positive logic or negative logic) can be specified by attaching reset_active attribute to the signal specified as reset signal. Similarly it is possible to specify synchronous or asynchronous reset with reset_mode attribute for the signal which is to be reset.

```
// signal reset2 is specified as negative logic reset signal
in ter(0:1) reset2 /* Cyber reset_active = low */;
in ter(0:1) reset3;

// signal reset2 is specified as reset signal of register r1
reg(0:8) r1 /* Cyber reset_sig = reset2 */;

// signal reset3 is specified as reset signal of register of
output variable o2
out reg(0:8) o2 /* Cyber reset_sig = reset3 */;

// signal reset2 and reset3 are specified as reset signal of
register r3
out reg(0:8) r3 /* Cyber reset_sig = reset2:reset3 */;
```

Although the variable "reset 2" is used as reset, the "reset" declaration is **not** used for this variable. This is because the variable declared using the "reset" declaration indicates the reset of circuit. Hence, "reset2" is declared as normal 1 bit variable without using the "reset" declaration.

9.8 Specification of register without reset

Reset is not generated in register by applying the “-Zreg_min_reset” option. However, in the following registers, reset is generated even on specifying the “-Zreg_min_reset” option.

- Register with initial value

In case initial value is specified at the time of register declaration, reset is generated in that register. This is similar to the static variable of a function.

- Register where the generation of reset is specified

When either the “reg_reset_cone” attribute or the “reg_reset_cone_valid_sig” attribute has been specified in output signal reset is generated in register related to this specification. Similarly, reset is generated when either the “-Zreg_reset_cone” option or the “-Zreg_reset_cone_valid_sig” option has been specified.

By using this option, it is possible to control reset signal application only to the necessary register. For example, reset is connected to register that links to important output signal such as enable output signal. However, reset is made not to connect to register that links only to other signals. For example, output signal whose value is ‘don’t care’ at time other than when data get output. This is useful when fan-out of reset signal is to be reduced.

9.9 Reset Generation - Specific Output

9.9.1 Outline

The specifications that generates reset in register affecting the specific output port are the “reg_reset_cone” attribute, “reg_reset_cone_valid_sig” attribute, “-Zreg_reset_cone” option, and “-Zreg_reset_cone_valid_sig” option.

- In case the “reg_reset_cone” attribute is applied to output signal, reset is generated in register that affects its output.
- In case the “reg_reset_cone_valid_sig” attribute is applied to output signal, reset is generated in register that affects the valid signal of that output.
- In case the “-Zreg_reset_cone” option is specified, it gives the same effect as the “reg_reset_cone” attribute applied to the entire output signal.
- In case the “-Zreg_reset_cone_valid_sig” option is specified, it gives the same effect as the “reg_reset_cone_valid_sig” attribute applied to entire output signal.

```

in ter(0:8) v0, v1, u0, u1 ;
out ter(0:8) out0 /* Cyber reg_reset_cone = YES*/;
out ter(0:8) out1 ;
process main(){
    reg(0:8) r0 /* Cyber no_share */;
    reg(0:8) r1 /* Cyber no_share */;
    reg(0:8) r2 /* Cyber no_share */;
    reg(0:8) r3 /* Cyber no_share */;
    reg(0:8) r4 /* Cyber no_share */;
    reg(0:8) r5 /* Cyber no_share */;

    r0 = v0 ; r1 = v1 ;
    r2 = u0 ; r3 = u1 ;
$    r4 = r0 + r1 ;
    r5 = r2 + r3 ;
$    out0 = r4 ;
    out1 = r5 ;
$}
}

```

In the case of the above description, initial value is not specified in any register. Therefore, when “-Zreg_min_reset” option is applied, all the registers become without reset. However, the “reg_reset_cone” attribute is specified in output signal out0. Therefore, reset is generated in register r0, r1, r4 that affects the output port.

9.9.2 Restrictions

The following restrictions exist with respect to the specification of reset generation of register connecting to specific output:

- When array is connected to specific output
 - In case the array is connected to specific output, warning is generated.
 - In case this array is synthesized with individual registers separately (EXPAND), reset is generated only in related elements.

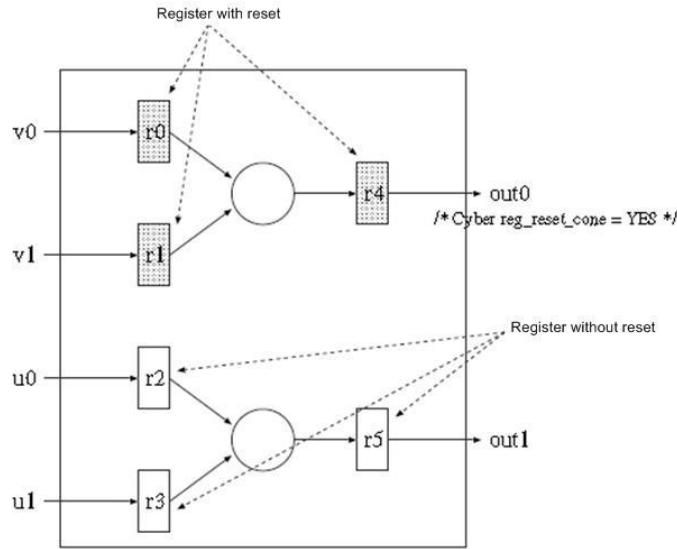


Figure 28: Reset Generation of Register connected to Specific Output

- o In case the array is synthesized with decoder enabled array (REG), reset is generated in all the elements.
- o In case the array is synthesized with memory (ROM/RAM), reset is not generated.
- In case specific output is connected to a function that becomes functional unit, reset is generated to register which is connected to all the arguments of that function without any relation to logic inside the function that became operator.
- In case of synthesizing as operator conversion function that becomes an operator (in case of synthesizing using option -Zlower_module), option -Zreg_reset_cone is applied automatically. Reset is generated in register that is connected to all the outputs.
- In the pipeline synthesis, one needs to be cautious as it is not assured that the circuit outputs certain value even when it is not used. This holds true even in the case of specifying the reset generation of register connected to specific output.

9.10 Summary

This section covered the following:

- When a design contains no reset port or options specifications, a reset port with positive logic is generated with a port name "RESET".
- Generation of the reset state of circuit can be controlled by using the "-Zreset_state" option.
- There are two methods to specify a reset port name.
 - o Using the "reset" declaration
 - o Using the synthesis option
- Although asynchronous reset is vulnerable to noise due to its asynchronous behavior, its advantage is that it requires less "die" area.

- Synchronous reset comparatively requires larger "die" area, but is less susceptible to noise as it reacts only at clock edges.
- Registers with initialization value can be reset by specifying the "reset_macro" attribute and the synthesis options. Currently, it can be specified in the following two formats:
 - Individual type
 - Inverter type
- Reset is not generated in register by applying the "-Zreg_min_reset" option.
- The specifications that generates reset in register affecting the specific output port are the "reg_reset_cone" attribute, "reg_reset_cone_valid_sig" attribute, "-Zreg_reset_cone" option, and "-Zreg_reset_cone_valid_sig" option.

10 Handling start and completion of Process Function

10.1 Overview

This section covers the following:

- Handling of variable declaration with initial value during process start.
- Reset value specification method other than initial value.
- Handling of variables at process completion.

10.1.1 Related Options

Option	Explanation
-Zinit=reset	Initialize variables at reset state (Default)
-Zinit=st_1	Initialize variables at initial state.

10.1.2 Related Attributes

Attributes	Explanations	Setting Object
/*Cyber mem_init = create*/	Generate a circuit (component) that will perform memory initialization.	Array Variable
/*Cyber mem_init = ignore*/	Do not generate circuit (component) that will perform memory initialization.	Array Variable
/* Cyber exit = st_1_state*/	Transit process function to initial state on exit.	Exit statement
/* Cyber exit = reset_state*/	Transfer process function to reset state on exit.	Exit statement

10.2 Variable Declaration

10.2.1 Overview

The value of variable at the beginning of a process function depends on following;
The variable, whose initial value is explicitly specified in declaration with initialization, is initialized through reset signal or in reset state. The variable, whose initial value is not specified in declaration, is treated as undefined even if it is a global variable.

However, there exist the following exceptions

- Initial value of array is ignored during memory implementation. This is because memory cannot be initialized in 1 cycle therefore these specifications assume that memory initialization is set-up in advance before starting the circuit. However, only at the time of automatic scheduling, initialization circuit can be generated during behavioral process instead of reset state by specifying the attribute Cyber mem_init = create (For more details refer section **10.2.3**).
- In case the local variable (excluding static variable) of a function has been assigned an initial value, the initialization of the variable to that value is performed at start of every function invocation, as is done in 'C' language. This is equivalent to having an assignment statement specified at the start of the function as shown in example below.

However, the array that is synthesized as a memory similarly ignores the initial value. Therefore, array initialization or assignment statements are not recommended at the start of function.

- In case static is specified in local variable (**excluding array, structural variable**) of function, for example, "initialized by zero".
- If initial value is described for out ter variable, then BDL synchronizes with state signal of reset state and output the initial value.
- In case initial value is described in ter variable other than out, referencing behavior cannot be described in reset state and it becomes a value that cannot be referenced. Therefore, initial value of ter variable other than out gives an error in manual scheduling. (However, in case of automatic scheduling, ter type is treated as var type therefore, error is not generated)
- By specifying -Zinit option, initialization during the declaration can be changed so that initialization is carried out in initial state. (Refer to section **10.2.2**)
- In case of auto scheduling, initial values that are not referred are ignored.
- A circuit which is initialized only by reset signal will be synthesized when reset_macro attribute is specified for certain register or in case specification is done that does not carry out initialization in reset state by specifying -Zreset_macro option by VHDL output command, Verilog-HDL output command, or in case reset state is not generated by default.
For this reason, there are following two restrictions.
 - After completion of process function, initialization will not be carried out during re-start.
 - out ter signal does not generate initial value

10.2.2 Variable Initialization - Timing Change

Option	Description
-Zinit=reset	Initialize variables at reset state(Default)
-Zinit=st_1	Initialize variables at initial state.

In case the global variable, local, or static variable inside a process function are assigned an initial value, a circuit (component) is generated to initialize the registers during reset by default.

When the “-Zinit=st_1” option is specified, the option ensures that the generated circuit is initialized during initial state and not reset state. This is equivalent to rearranging initialization code as specified below.

Before rewriting	After rewriting
<pre>reg (0:8) r = 3; process main () { : : }</pre>	<pre>reg (0:8) r ; process main () { r = 3 : }</pre>

10.2.3 Initialization Circuit Generation

If the initial values are specified in declaration of an array variable that is implemented as memory, then an initialization circuit is not generated and the initial values are ignored. It is assumed that the initial values have already been set in the memory when the circuit operation started.

However, if the “mem_init=create” attribute is specified along with array signal declarations, a circuit (component) is generated that performs memory initialization.

Attributes	Description	Setting target
/*Cyber mem_init = create*/	Generate a circuit (component) that will perform memory initialization.	Array Variable
/*Cyber mem_init = ignore*/	Do not generate circuit (component) that will perform memory initialization.	Array Variable

Therefore, when the "mem_init=create" attribute is specified for array variables being implemented as memory, specification (code) is generated that performs the initialization of array variables.

If "mem_init = create" attribute is used in manual scheduling mode, an error is generated specifying that it is not supported. This is because memory initialization does not end in 1 cycle & required cycle count fluctuates depending upon number of array elements to be initialized. Therefore, in manual scheduling mode, there is explicit requirement to specify assignment description according to memory.

When the "mem_init=create" attribute is used in auto scheduling, the specification that initializes all data elements will be generated. Therefore required number of execution cycles for initialization will change depending upon memory constraints.

```

mem(0:8) M[4] /* Cyber mem_init =create */ = {10,13,15,17};
process main(){
    :
    :
}
↓
/* in the case of auto scheduling */
mem(0:8) M[4] /* Cyber mem_init =create */ = {10,13,15,17};
process main(){
    M[0] = 10;
    M[1] = 13;
    M[2] = 15;
    M[3] = 17;
    :
    :
}
```

10.3 Reset Value Specification Method

10.3.1 Overview

In addition to initialization, a reset block can be used during declaration when the reset value of a variable is specified in Cyber.

A reset block is a functionality that can specify a particular control block as the block executed during reset using the "reset_block" attribute. Moreover, using this functionality, it is possible to describe the specification of reset value using loop or reset value for the structure member variable.

In the example provided below, reset_block is specified for the added control statement (if (1) block) using the "reset_block attribute". Therefore, the complete statement within the reset_block is executed in reset state.

In this situation, if a "for" loop is present in reset_block, which is not unrolled, an error will be generated.

```

reg(0:8) ary[16];
struct st1 {
    int x, y;
} s;
process main () {
    char i;
    var (0:64) t;
    /* Cyber reset_block*/
    {
        t = 0xFFFFFFFF: :0x00000000;
        for(i = 0; i <16; i++) {
            ary[i] = i;
        }
        s.x = 0
        s.y = 1
    }
    $ // not required in auto-scheduling
mode
    ....
}

```

10.3.2 Related Attributes

Attributes	Description	Settings target
/* Cyber reset_block */	Specify as executed block during reset of control statement	Control statement

10.3.3 Description

The "reset_block" attribute can only be added to the first if statement/switch statement/while statement/do-while statement/for statement of process function or function to be converted to functional unit. Further, in the manual scheduling mode, \$ cannot be described in the reset block and \$ is mandatory after completion of the reset block.

Only specifications of reset values by constants are allowed such as constant assignment. However, the expression is allowed if it becomes the specification of reset value by constant as the result of loop unrolling. The subscript of an array also must be constant when it is present at the left hand side in an assignment in a reset block. Further, multiple assignment to same variable cannot be described in reset block except when the constant value is the same. There is a restriction on call out of functional unit within the reset block, and it displays F_BT4325 error.

10.4 Handling of Process Function Completion

10.4.1 Overview

Synthesized circuits are generated to continuously execute process functions specified in behavioral description. This subsection explains about transfer destination states at process function completion, re-invocation, and the values of variables at function completion.

- * Destination state when process function execution is completed (except through exit or return statement).
- * Value of each variable at process function completion

If an infinite loop has been specified explicitly, process function execution will never be over. Therefore, this case will be handled separately.

10.4.2 Destination State - Execution Ends Neither Through Exit nor Return Statements

When the process function execution is completed by a method other than the return statement or exit statement, the process function is transferred to its initial state. The process function can also be transferred into reset state using the appropriate attribute.

Attributes	Description	Settings Target
<code>/*Cyber return = st_1 */</code>	Process function will transit to initial state at execution completion (where execution ends neither due to return nor exit statement).	Process Function
<code>/*Cyber return = reset */</code>	Process function will transit to reset state at execution completion (where execution ends neither due to return nor exit statement).	Process Function

10.4.3 Destination State - Execution Ends Through Return Statement

In case the process function execution ends by a return statement, the process function is transferred to its initial state.

10.4.4 Destination State on Exit

A process function is transferred to a reset state in case its execution is terminated using an exit statement. It can also be configured to transit into its initial state by specifying the appropriate attribute.

Attributes	Description	Settings Target
------------	-------------	-----------------

/* Cyber exit = st_1_state*/	Transfer process function to initial state on exit.	Exit statement
/* Cyber exit = reset_state*/	Transfer process function to reset state on exit.	Exit statement

However, in case reset state is lost on specifying option **-Zreset_state=NO**, all of the above three cases after process completion will transfer to initial state.

10.4.5 Value of each variable at process function completion

The handling of value of each variable, when process function is reiterated again after completing the process function, will be decided according to transfer state during process function completion and presence of initialization value during variable declaration.

Among the variables, global variable and static local variable of function retains the value however, a non static local variable of function may not necessarily retain the value.

```

in ter(0:1) flag;
in ter(0:8) in1,in2;
out reg(0:8) out1;
reg(0:8) r1; /* Global variable : Retain */
process module3(){
    reg(0:8) r2; /* Local variable : Not confined to retain */
    static reg(0:8) r3; /* static Local variable : Retain */
    .
    .
}
void func( ){
    reg(0:8) r4; /* Local variable : Not confined to retain */
    static reg(0:8) r5; /* static Local variable : Retain */
    .
    .
}
```

In the following example, when process function is reiterated, registers will not be allocated to retain the value of "r1", therefore, when flag is false, value of variable "r1" becomes undefined.

For this reason, in the following example, in case synthesis is done in automatic scheduling mode after specifying only one adder, which takes 1 cycle, as functional unit constraint, it is judged that lifetime of variable "r1" and "r2" will not overlap, and registers of variables are common, therefore, when flag is false, variable ""r1" will refer to the value of variable "r2", which is allocated to the same register. In **auto scheduling**, the value of every variable is not allocated to register in order to retain it across process invocations.

```
in    ter(0:1) flag;
in    ter(0:8) in1,in2;
out reg(0:8) out1;

process module3(){
    reg(0:8) r1,r2;

    if( flag ){
        r1 = in1 + in2 ;
    }
    /*---clock cycle ---*/
    out1 = r1;
    r2 = r1 + in2;
    /*---clock cycle ---*/
    out1 = r2;
}
```

However, only because retention by local variable of function is not assured, there could be cases where unshared variable such as non expanding array is left with a value even when process function is completed and reiterated again. Therefore, caution is required in such cases as retention of local variable values beyond process re-invocation may occur for some cases but it's not a guaranteed behavior.

Regarding the state at the process function completion, in case initial value is written in variable during declaration, when process function is reiterated again after a successful completion, if process transfers to an initial state, then initialization will not be carried out however, if it transfers to reset state, initialization will be carried out.

Points to remember and Limitations

- * Even if the initial value is described for non static local variable inside the process function at the time of declaration, the first time variable will get initialize by reset signal however, it will not get initialized when process function is reiterated after transiting to initial state, therefore, attention is required.

In case synchronous reset is specified, and –Zreg_min_reset is not specified, the register allocated to the variable without initialization is initialized at the reset state and therefore, the value will not be retained. So the value can be allocated in a register to retain it by specifying either of the following:

- An infinite loop explicitly or by
- The “last_reg=re_use” attribute along with the process function
- –The “Zlast_reg=re_use” option

However, the value will not be allocated to a register to retain it if there is an exit statement that transfers the process function to its reset state.

10.5 Summary

This section covered the following:

- In case the global variables, local variables in a process function, and static variables in functions are assigned an initial value, a circuit (component) is generated to initialize the registers during reset by default.
- The “–Zinit=st_1” option ensures that the generated circuit is initialized during initial state and not reset state.
- If the initial values are specified in declaration of an array variable that is implemented as memory, then an initialization circuit is not generated and the initial values are ignored.
- To generate a circuit (component) that performs memory initialization, the “mem_init=create” attribute is specified along with the array signal declaration.
- A reset block can be used during declaration when the reset value of a variable is specified in Cyber.
- A process function is transferred to a reset state in case its execution is terminated using an exit statement.
- A process function can be transferred to its initial state by using appropriate options and attributes.

11 Array / Memory

11.1 Overview

This section covers the following:

- The four methods of synthesizing the arrays: "Memory" "decoder enabled register array", "combinatorial circuit", and "variable unrolling".
- The circuit synthesis and specification mechanism for the methods listed above.
- The external and internal memory.
- The array variable specifications and related aspects.
- The multidimensional and one-dimensional array conversions.
- The items that are used while specifying a memory.
- The memory access timing specification for automatic scheduling and manual scheduling.
- The four items that can be specified while an array is synthesized as a register array provided with decoder.

As a limitation, the "input output array variable" is supported only by "variable unrolling" synthesis method. If the data has to be input by arrays from outside then it is recommended to synthesize it as external memory.

11.1.1 Related Options

Option	Description
-lm filename [.MLMT]	Specifies File filename [.MLMT] as memory constraint file.
+lm filename [.MLMT]	Specifies File filename [.MLMT] as additional memory constraint file.
-lml filename[.MLIB]	Specifies filename [.MLIB] as memory library file.
+lml filename[.MLIB]	Specifies filename [.MLIB] as additional memory library file.
-lmc filename[.MCNT]	Specifies filename [.MCNT] as memory count file.
+lmc filename[.MCNT]	Specifies filename [.MCNT] as additional memory count file.

Option	Description
-am -ar -a#	Achieves the array by memory. Achieves the array with the register. In case the array bit width x word length is more than # then it is realized as memory else it is realized as registers. (default-a1024)
-apI -apA -ap# -ap ?:o	Consider all memory as internal memory (default). Consider all memory as external memory. The array is synthesized as external memory if the number of elements is less than #, whereas the array is synthesized as internal memory if the number of elements is more than #. The unused external memory port is not generated (-ap:o, -apA:o, -ap#:o).
-Zno_aryinit -Zaddress_calc_mult = <i>kind[:kind...]</i> -Zaddress_concat = <i>kind[:kind..]</i>	Ignore the initial value of the array. Performs the address calculation using the multiplication expression, while laying out multidimensional array as a one dimensional array. Performs the address calculation using the :: (concatenation operation), while laying out multidimensional array as a one dimensional array.
-aeu -ae#	Array which is accessed only by constant subscript is normally synthesized without decoder (default). If array size is more than # then synthesize it with decoder.

Option	Description
-mr?w?	Specifies the memory access timings (refer to Section 11.6).
-mc -mcD -mcN	Allows chaining of the memory access for reading the memory. Allows the chaining of the memory access towards a different memory for memory reading. Does not chain the memory access during memory reading.
-Zmem_re -Zmem_re=NO	Generates read enable port and the write enable port in the read write two way port. Generates the write enable port in the read write two way port (default).
-Zmem_cs -Zmem_cs=NO -Zmem_be -Zmem_be=NO	Generates the chip select port in the two way port. Does not generate the chip select port (default). The data bit width allocates all the arrays with multiples of 8 in the byte enable port memory. Does not allocate the array in the memory having the byte enable port.
-Zrw1=1 port -Zrw1=2 port	Generates the read write data two-way port in the read write two-way port of the external memory (default). Generates the write data output port and read data input port in the read write two-way port of external memory.

Option	Description
-Zmem_read_speculation -Zmem_read_speculation = NO	<p>At the time of automatic scheduling, it performs the speculative execution of memory reference.</p> <p>At the time of automatic scheduling, it does not perform the speculative execution of memory reference (default).</p>
-au -al	<p>Does not restrict the number of decoders of the decoder enabled register array in manual scheduling (default).</p> <p>Restricts the number of decoders of the decoder enabled register array at the time of manual scheduling.</p>
-Zdecoder_div -Zdecoder_div=no -Zdecoder_div=ave -Zdecoder_div=pre -Zdecoder_div_unit=# -Zdecoder_div_over=#	<p>Divides the decoder of decoder enabled register array into a small decoder and multiplexer.</p> <p>Does not divide the decoder of register array (default).</p> <p>Divides the decoder of register array with average formula.</p> <p>Divides the decoder of register array with left justified formula.</p> <p>Divides the decoder of register array with # bit unit.</p> <p>The input bit divides the decoder which is more than #.</p>
-Zpipemem1=1	<p>Use the pipeline memory with the manual scheduling.</p>
-Zarray2multi_mem	<p>In the memory allocation of the multidimensional array, allocation is done after dividing it in multi memory (default).</p>

Option	Description
-Zarray2mult_mem=NO	In allocation of multidimensional memory, does not divide it in multi memory.
-Zmem_clocking=none -Zmem_clocking=enable -Zmem_clocking=always	Do not generate memory clock port for synchronous SRAM (default). Generate memory clock port for synchronous SRAM and outputs memory clock signal only at the time of memory access. (Corresponds to the –syncRAM option of the earlier Cyber 3.6). Generate memory clock port for Synchronous SRAM and outputs a steady memory clock signal. (Corresponds to the –syncRAM2 option of the earlier Cyber 3.6).
-Zmem_reg=none -Zmem_reg=in -Zmem_reg=out -Zmem_reg=in:out	Register is not input in input and output side of the access of pipeline memory. (default) Register is input in input side of the access of pipeline memory. Register is input in output side of the access of pipeline memory. Register is input in input and output side of the access of pipeline memory.
-Zoutside_mem_reg=outside -Zoutside_mem_reg=inside	Synthesize register of Input/Output side of external memory as external (Default). Insert Input/Output side register of external memory as internal.
-Zshared_mem_re_gen [=YES] -Zshared_mem_re_gen =NO	Read enable signal is generated that controls the multiplexers address of shared memory. (default) Read enable signal is not generated that controls the multiplexer address of shared memory (memory without read enable and chip select).

Option	Description
-Zshared_reg_lutram_re_gen [=YES]	Read enable signal is generated that controls multiplexer address of shared register array (LUT memory format).
-Zarray_port_other_DC[=YES]	Value of address signal, right data signal for arrays changes to Don't Care (default) when not used.
-Zarray_port_other_DC=NO	Value of address signal, right data signal for arrays changes to 0 when not used.

Option	Description
-Zset_lower_clock_reset=ALL	Set clock reset of lower hierarchy which carries out writing in clock reset signal of shared and registered memory. (default)
-Zset_lower_clock_reset=SHARED_MEM	Set clock reset of lower hierarchy which carries out clock reset signal of the shared memory. (default)
-Zset_lower_clock_reset=NO	Set clock reset signal of current hierarchy to the clock reset of shared and registered memory.

11.1.2 Related Attributes

Attribute	Description	Settings target
/*Cyber array=RAM*/	Achieve the array signal as a memory.	Array variable
/*Cyber array = ROM */	Achieve the array signal as a read only memory.	Array variable
/*Cyber array = REG*/	Achieve the array signal as a decoder enabled register array.	Array variable
/*Cyber array =LOGIC*/	Achieves the array signal with the combinatorial circuits.	Array variable
/*Cyber array =EXPAND*/	Expand the array signal into the individual variables.	Array variable
/* Cyber reg_array = LUTRAM */ /* Cyber reg_array = DECODER */	Achieve array as LUT memory format. Achieve array as decoder enabled register array	Array Array
/* Cyber expand_dim = #[:#...] */	Unroll the multi-dimensional array by a dimension on # number from right.	Array
/*Cyber array_merge=name*/	Merges the multiple arrays in declaration sequence in Array Name 'name'.	Array variable
/*Cyber array_merge=name:# */	Merge in the array named 'name' so that it begins with the element number #.	Array variable
/*Cyber array _merge=name @#*/	Merge as # no. of array in Array Name 'name'.	Array variable

Attribute	Description	Settings target
/* Cyber address_calc=MULT*/ /*Cyber address_calc=CONCAT */	<p>The address calculation is done by multiplication at the time of changing the multidimensional array to a single dimensional one.</p> <p>The address calculation is done by concatenation operation at the time of changing the multidimensional array to a single dimensional one.</p>	Array variable Array variable
/* Cyber array_index=const*/	Convert the Array reference, assignment to the description where Switch was used.	Array, Statement, Expression element, Assignment point
/*Cyber array_index_suffix=word*/	Specify the character string of suffix to be added to the name of variables to which the array is converted.	Array variable
/* Cyber speculation = YES */ /* Cyber speculation = NO */	<p>Performs the speculative execution of array reference.</p> <p>Does not perform the speculative execution of array reference.</p>	Array variable Array variable

Attribute	Description	Settings target
<pre data-bbox="251 304 523 382">/* Cyber ex_group = # [:#...]/</pre> <pre data-bbox="251 544 567 623">/* Cyber folding_ex_group = #[:#...]/</pre>	<p>Specifies the exclusion of subscript at the time of assignment and Array reference.</p> <p>Specifies the exclusion of subscript In loop folding.</p>	<p>Array reference, Assignment</p> <p>Array reference, Assignment</p>
<pre data-bbox="251 642 577 677">/* Cyber sig2mu = name */</pre> <pre data-bbox="251 804 535 882">/* Cyber sig2mu_port = name.r#w# */</pre> <pre data-bbox="251 1051 540 1087">/* Cyber mu_port = # */</pre>	<p>Allocates the array in memory, of 'memory type' name.</p> <p>Allocates the array in memory of 'memory type' name and specifies the port to be used.</p> <p>Specifies the port that is used by Array reference, assignment.</p>	<p>Array variable</p> <p>Array variable</p> <p>Array reference, Assignment</p>
<pre data-bbox="251 1191 561 1248">/* Cyber rw_timing = r?w?</pre>	<p>Specify the timing of memory access.</p>	<p>Array variable</p>
<pre data-bbox="251 1290 584 1326">/* Cyber mem_cs = create*/</pre> <pre data-bbox="251 1417 551 1453">/* Cyber mem_cs=ignore</pre>	<p>Creates the chip select port in the port.</p> <p>Does not create the chip select port in the port.</p>	<p>Array variable</p> <p>Array variable</p>
<pre data-bbox="251 1558 561 1615">/* Cyber mem_re = create</pre>	<p>Generates the read enable port in the read- write dual port of array memory.</p> <p>Does not Generate the read enable port in the read write dual port of array memory.</p>	<p>Array variable</p> <p>Array variable</p>

Attribute	Description	Settings target
/* Cyber mem_be = create */ /* Cyber mem_be = ignore */	Allocate to the memory having byte enable port. Does not allocate to the memory having the byte enable port.	Array variable Array variable
/* Cyber mem_init = create */ /* Cyber mem_init = ignore */	Circuit that initializes memory is generated. Circuit that initializes memory is not generated.	Array variable Array variable
/*Cyber mem_clocking = enable*/ /*Cyber mem_clocking = always*/	Generates the memory clock port for synchronous SRAM and outputs the memory clock signal only during memory access. Generates the memory clock port for synchronous SRAM and outputs memory clock signal on a steady basis.	Array variable Array variable
/* Cyber rw_port = RW# */ /* Cyber rw_port= R#.W# */ /* Cyber rw_port = R# */ /* Cyber rw_port = W# */	Specifies the maximum number of decoders for reference & assignment of decoder enabled register array as #. Specifies the maximum number of assignment decoders, number of reference decoders of decoder enabled register array respectively as #. Specifies the maximum number of reference decoders of decoder enabled register array as #. Specifies the maximum number of	Array variable Array variable Array variable Array variable

Attribute	Description	Settings target
/* Cyber sig2dec_port = r#w# */	assignment decoders of decoder enabled register array as #.	
/* Cyber dec_port = # */	Specifies the default port number of decoder enabled register array used in array reference, assignment.	Array variable
/* Cyber rw_delay= #*/	Specifies the individual port number of decoder enabled register array used in array reference and assignment.	Array reference Array assignment
/* Cyber decoder_div = #:#[#:...]*/	Specifies the delay of the decoder enabled register array by #.	Array variable
	Divides the decoder of decoder enabled register array into small decoder and multiplexer.	Array variable
/* Cyber mem_reg= none */	Does not insert register in input/output side of pipeline memory access (default).	Array variable
/* Cyber mem_reg = in */	Inserts register in input side of pipeline memory access.	Array variable
/* Cyber mem_reg = out */	Inserts register in output side of pipeline memory access.	Array variable
/* Cyber mem_reg = in:out */	Inserts register in input/output side of pipeline memory access.	Array variable
/* Cyber univ_mem_if= create */	Generates Universal Memory Interface for the array.	Array variable
/* Cyber univ_mem_if= ignore */	Does not generate Universal Memory Interface for the array.	Array variable

Attribute	Description	Settings target
/* Cyber pipemem = x# */	Specifies the stages of pipeline memory	Array variable
/* Cyber pipemem_io */	Pipe line memory input output variable	Signal
/* Cyber mem_synth = LUT */ /* Cyber mem_synth = BLOCK */ /* Cyber mem_synth = NONE */	Uses the memory achieved by LUT in FPGA synthesis. Use the block memory in FPGA synthesis. Do not specify the kind of memory in FPGA synthesis.	Array variable Array Variable Array Variable

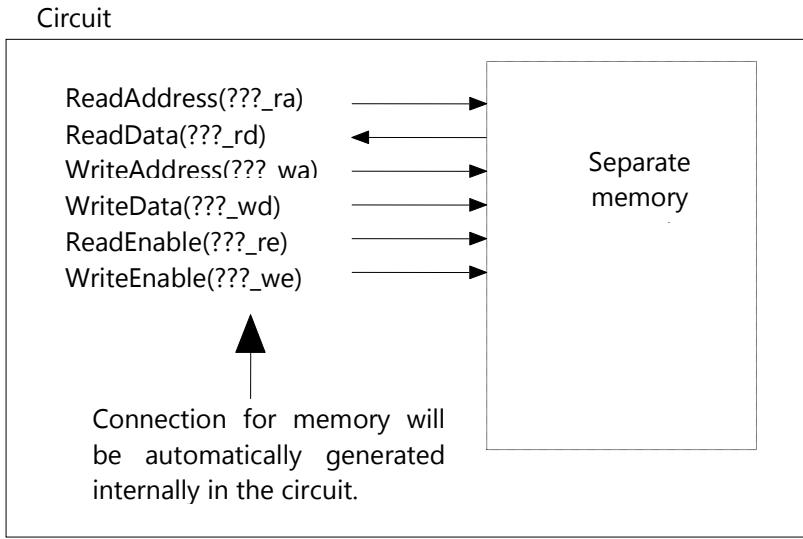
11.2 Synthesis of Arrays

11.2.1 Memory

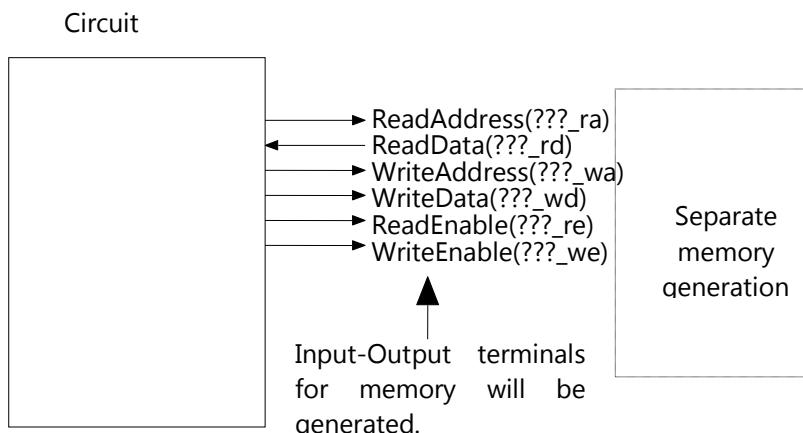
There are two methods of synthesizing memory. The first method is to synthesize it as "internal memory" that is lower hierarchy module inside the circuit to be synthesized. The second method assumes that the module exists outside the circuit to be synthesized i.e. "external memory". In the case of "external memory", terminal ports are generated in the circuit to be synthesized.

- Internal Memory
The memory is created within the circuit to be synthesized as the lower hierarchy module and the ports as decided by the memory type.

In this case, the internal (RTL) description of the memory is not generated. Hence, the memory module behaves as a black box. The synthesized circuit simply has connections generated to this black box. The designer has to specify the internal memory model separately in this scenario.

**Circuit**

- External Memory
- In this case, the memory exists outside the circuit to be synthesized. The input output ports connecting the memory are generated according to the memory type.
It is necessary for the designer to provide a higher level module and memory module that includes synthesized circuit and memory.

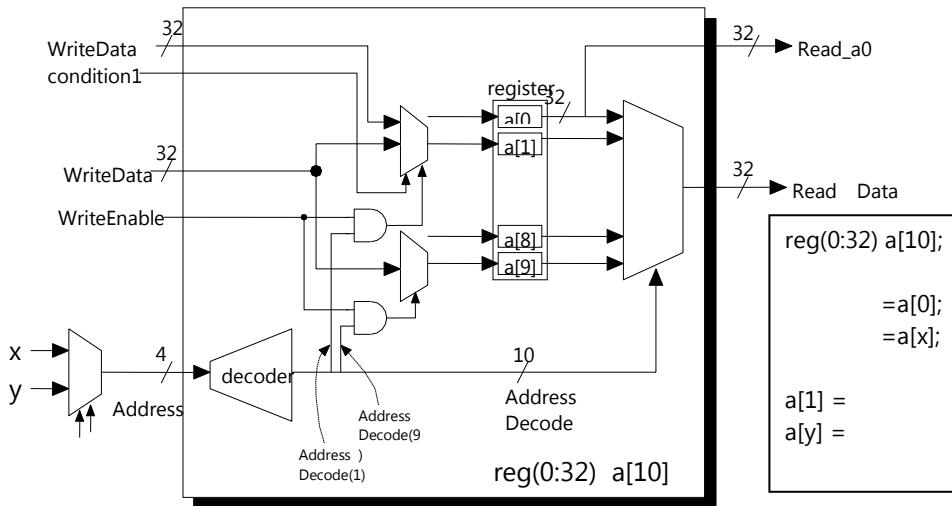


11.2.2 Decoder Enabled Register Array

The array is synthesized with a register array that is equal to the number of elements of the array and a decoder that decodes the address. Further, decoder enabled register array can be allocated to LUT memory based on FPGA synthesis. Single cycle can be referred simultaneously and assigned number can be changed by specifying the number of ports.

Assignments/references can be done without using a decoder where array index is a constant (e.g. $a[1]$ etc). This has the advantage that array elements can be assigned/read in a single cycle without being restricted by the number of decoders. Although the circuit area increases by

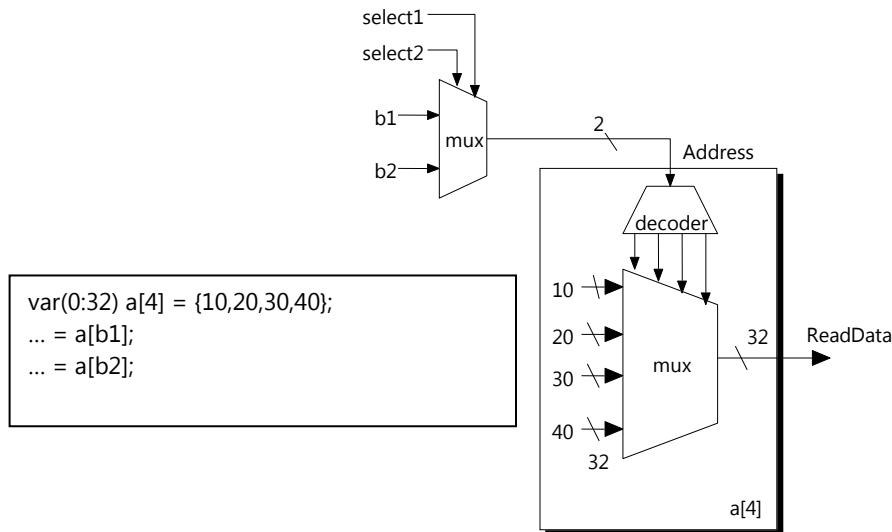
increasing the number of decoders, but the performance gain justifies this marginal increase in area.



11.2.3 Combinatorial Circuits

The read-only arrays, which have been suitably initialized in the specification, are synthesized as combinatorial circuits rather than as memory or decoder enabled register arrays.

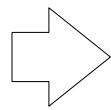
Since no assignment will be made, multiple embodiments can be created.



11.2.4 Variable Unrolling/Expansion

When all the subscripts are constants during array reference or assignment, then the array variables can be treated as individual variables. The number of individual variables is equal to the number of array elements. This is called a variable unrolling or variable expansion.

```
int M[4];  
  
... = M[0];  
... = M[1];  
.  
.  
M[0] = b1;  
M[1] = b2;
```



```
int M_a00,  
    M_a01,  
    M_a02,  
    M_a03;  
  
... = M_a00;  
... = M_a01;  
.  
.  
M_a00 = b1;  
M_a01 = b2;
```

11.3 Array Synthesis Specification

This section covers the method of array synthesis that has been specified by the designer.

11.3.1 Specifying Memory

In example 1, the array variable declared as physical type "mem" is synthesized as memory. In example 2 and 3, arrays designated with attribute /* Cyber array =ROM*/ or /* Cyber array=RAM* / are synthesized as "Memory", even when array has a physical type (example 3). Please note that the array variables of the "var" type are also synthesized as "Memory" when specified with the "array=kind" attribute.

Example)

```
mem(0:32)      a[10];          /* example1 */
int           b[10]/* Cyber array = ROM */;    /* example2 */
var(0:32)      c[10]/* Cyber array = RAM */;  /* example3 */
var(0:32)      d[300];        /* example4 */
```

When logical array variables (i.e. non-physical/non-hardware types) are specified without the use of attribute /* Cyber array =kind */, the logical array variables may be synthesized either as "memory" or "decoder enabled register array" based on the options given below. The same is true for "var" type objects. However, it is preferred to synthesize combinational logic or using variable unrolling, wherever possible.

- If there is no specification of the memory constraint file (MLMT), memory library (MLIB), memory count file (MCNT), then it is achieved by decoder enabled register array.
- If MLMT/ MLIB/ MCNT file is specified, then it is achieved according to the following three options (-am, -ar, -a#), i.e. either by memory or by decoder enabled register array.

Option	Description
-lm filename[.MLMT]	Specify file filename[.MLMT] as memory constraint file.
+lm filename[.MLMT]	Specify filename[.MLMT] as additional memory constraint file.
-lml filename[.MLIB]	Specify filename[.MLIB] in first memory library file.
+lml filename[.MLIB]	Specify filename[.MLIB] as additional memory library file.
-lmc filename[.MCNT]	Specify filename[.MCNT] in first memory count file.
+lmc filename[.MCNT]	Specify filename[.MCNT] as additional memory count file.

-am	Achieve the array with memory.
-ar	Achieve the array with the decoder enabled register array.
-a#	The array is synthesized as decoder enabled register file if number of elements are less than #. Otherwise, the array is realized as "memory". (default-a1024)

In order to synthesize the array as memory, it is necessary to specify the memory used in a MLMT, MLIB & MCNT file. The detailed explanation about MLMT is given in the section **39.3**, MLIB is given in the section **39.10** & MCNT file is given in the section **39.11**. Furthermore, there is an option "-Zmlib_mcnt_out", which automatically creates templates of MLIB, MCNT files. Automatic creation of the template of MLIB, MCNT file is given in section **25.4**.

By default in option "-a#" the array bit width x word length is more than 1024 then it is realized as memory else it is realized as register.

11.3.2 Internal Memory/ External Memory Specification

There are two methods of synthesizing memory: as internal memory and external memory. Further, there are two stages at which the internal memory/external memory can be specified. The first method involves specifying the individual array variable using the "outside" and "shared" declaration types. The memory allocated to the array variables declared with these types is synthesized as external memory as shown in table below.

Type declaration	Description
Outside	The memory allocated to the array variable is achieved by external memory.
Shared	The memory allocated to the array variable is realized by shared memory. (When it is shared by multiple processes.)

Note: Both "outside" and "shared" generate the necessary I/O terminals as external memory. However, when "shared" is specified, other processes also access it. Thus, optimization of I/O terminals is restricted.

Example)

```
outside mem (0:32) x[200]
shared mem (0:32) y[200]
outside var (0:32) d[300] /*Cyber array= RAM */
```

The alternative specification method is to use the global command line options. These options are of three types as given below. By default, the "-apI" option is specified so it is synthesized as internal memory.

Option	Description
-apI	The memory is taken as internal memory. (default)
-apA	Memory is taken as external memory.
-ap#	The array, where number of elements is less than #, is synthesized as external memory. Whereas array where no. of elements are more than # is synthesized as internal memory.

11.3.3 Decoder Enabled Register Array - Specification Method

Decoder enabled register array is determined when below conditions are mentioned.

- In **example 5**, the array variable whose physical type is declared as type "reg" is synthesized.
- In **example 6**, the array variables specified with the attribute /* Cyber array = REG */ and not declared as a physical type are also synthesized.
- In **example 7**, the array variables of the "var" type are synthesized as decoder enabled register array as well.

Array that is determined as decoder enabled register array can be synthesized using the allocated format in LUT memory by specifying reg_array = LUTRAM attribute. LUT memory is the default format in FPGA synthesis.

```
Example)
reg(0:32) e[10];                                /* example5 */
int      g[10]/* Cyber array = REG */;           /* example6 */
var(0:32) f[10]/* Cyber array = REG */;          /* example7 */
var(0:32) h[10];                                /* example8 */
```

In **example 8**, the array variables of the "var" type, where attribute /*Cyber array = kind*/ has not been specified, are achieved either by memory or by decoder enabled register array according to the option described in the section **11.3.1**. Similarly, the array variables, where physical type has not been declared, are also achieved either by memory or by decoder enabled register array.

However, there is one exception to the rules stated above. If the circuit can be realized with combinational logic or by variable unrolling, then they are preferred to the decoder based register array implementation due to circuit efficiency reasons.

However, there are option -Zinit_reg=asis and attribute init_reg that implements the combinational logic implemented register arrays by decoder enabled register arrays (Refer to section **28.6** for details.).

Furthermore, there exists a limitation that the number of array elements to be achieved by decoder enabled register array must be less than 4096.

11.3.4 Combinatorial Circuit - Specification Method

The following conditions are necessary to achieve the array by combinatorial circuit:

- The array is initialized at the time of declaration.
- There array is read but not written to. This means that the array does not appear as "lvalue" in any assignment statement.

The "reg" type array variables, which fulfill the above conditions, are synthesized as combinatorial circuit. Otherwise, it is synthesized as decoder enabled register array as explained in section **11.3.3**.

The terminal ("ter") type variables, which fulfill the above conditions, are synthesized using combinational logic as shown in example 9. For terminal array, which does not fulfill the above conditions are synthesized using variable unrolling.

The following are synthesized as combinatorial circuit:

- The array variable without a physical type for which the attribute /*Cyber array = LOGIC */ has been synthesized as shown example 10.
- Array variable of "var" type as shown in example 11.

Specification is erroneous if the variables do not comply with necessary conditions.

The "var" type array variable in which the attribute /*Cyber array = kind */ has not been specified (**example 12**) and fulfills the above conditions is synthesized as combinatorial circuit. The array

variable, which does not fulfill the above conditions, is synthesized by either memory or decoder enabled register array according to the description given in the options in section **11.3.1**.

Examples)

```
ter(0:6) i[4] = {32,4,10,3}; /* example 9 */
char j[2] /* Cyber array = LOGIC */={32,64}; /* example 10 */
var(0:4) k[3] /* Cyber array = LOGIC */={3,2,1}; /* example 11 */
var(0:32) h[10]={9,8,7,6,5,4,3,2,1,0}; /* example 12 */
```

This method of synthesis can convert an array into multiple combinational circuits, similarly as a function is converted into multiple functional units. The specification of a functional unit constrained file / functional unit count file includes specifying the following:

- The "@ array name" in the operator type in.
- The highest actual number in the constrained numbers. It specifies the minimum bit width that can represent the number of elements as input-bit width. It also specifies the bit-width of array variable in the output-bit width.

The section **39.2** explains about functional unit constrained file and section **39.8** explains about functional unit count file in detail.

Examples to specify Functional Unit Count File:

Functional unit count file (Example 9)

```
@FCNT {
    NAME      logicA
    KIND      @i
    LIMIT     4
    BITWIDTH (2),6
    DELAY     100
    AREA      40
}
```

Functional unit count file (Example 12)

```
@FCNT {
    NAME      logicD
    KIND      @h
    LIMIT     1
    BITWIDTH (4),32
    DELAY     500
    AREA      400
}
```

11.3.5 Variable Unrolling/Expansion - Specification

The condition that is necessary to carry out the variable unrolling of an array is that at the time of array reference or assignment all the subscripts are constants.

There is no restriction on array size, however it can be controlled by the following option:

Option	Description
-aeu	Synthesize those arrays normally without decoder which are accessed only by constant subscript (default).
-ae#	Synthesize those arrays with decoder whose size is more than #.

The array variables of the physical type "ter" undergo variable expansion if the following conditions are met:

- It cannot be synthesized as a combinatorial circuit as shown in **example 13**.
- It fulfills the above condition for variable unrolling of an array. The rest of the variables that do not fulfill the above condition result in an error.

Similarly, the variables of the type "reg" that cannot be synthesized as combinatorial circuits also undergo variable expansion if they fulfill the condition for variable unrolling of an array. Otherwise, they are synthesized by decoder enabled register array as explained in section **11.3.3**.

The following undergoes variable unrolling:

- The logical type array variable specified with the attribute /*cyber array = EXPAND*/, (example 14).
- The array variable of the "var" type (example 15).

If the required condition of variable unrolling is not fulfilled, it results in an error.

The following is also unrolled as per norm:

- The "var" type array variable in which the attribute /*cyber array = kind*/ has not been specified (see example 16) and which fulfills the array expansion condition.

The variable which does not fulfill the above array expansion condition is synthesized by either memory or decoder enabled register array as explained in section **11.3.1**.

Example)

```
ter(0:6) i[4];                                /* example 13 */
char      j[2]/* Cyber array = EXPAND */;        /* example 14 */
var(0:4)  k[3]/* Cyber array = EXPAND */;        /* example 15 */
var(0:32) h[10];                               /* example 16 */
```

11.3.6 Array Variable Synthesis - Automatic Decision

This section describes the decision process of array variable synthesis.

- If the array is of physical type "mem":
 - It will be synthesized as memory.
 - If the data type of array is not "mem" or unspecified
 - The two necessary conditions for combinatorial circuit are being fulfilled.
 - It is synthesized as combinatorial circuit.
1. Otherwise
 - * The necessary condition for variable unrolling/expansion is being fulfilled
 - o Variable is unrolled
 - * Else (necessary variable expansion condition is not met)
 - o A specification of the "ter" type and I/O array variables will result in error.
 - o Array variables of the "reg" type are synthesized by decoder enabled register array.
 - o Array variable of the "var" type or array variable of non-physical type are synthesized as decoder enabled register array or memory. This synthesis is based on the specification in the memory constraint file and array size.

The default behavior regarding the number of array elements and their synthesis method is described in the following table:

Total bit count below 1024	Register array
Other cases	Memory

It is possible to change "1024", the boundary of total bit count of array, in accordance with the option described in the section **11.3.1**.

11.4 Array Variable Specification

This section covers the following nine aspects related to array variable specification:

- Unrolling of dimensions in the parts of a multidimensional array.
- The allocation of multidimensional arrays into multi memory partitions.
- The specification of subscript evaluation expression at the time of converting multidimensional arrays into one dimensional array.
- The specification of exclusion of subscripts at the time of array reference or assignment.
- The specification of the subscript at the time of array reference or assignment in loop unfolding during manual scheduling.
- The specification in which multiple arrays are merged into one array.
- The specification of speculative execution of array reference.
- The specification which converts the array reference or assignment into description that uses switch.
- The initialization of array variables.

11.4.1 Multidimensional Arrays - Partial Unrolling

When all the references and assignments of a multidimensional array contain a constant subscript along a fixed dimension, then that specific dimension is unrolled by the tool.

If the array variable is the input-output array variable, no unrolling is done. In other words, the array variable that has to be converted into combinatorial circuit, decoder enabled register array or memory is unrolled.

In the example below, the extreme left subscripts are all constants at the time of array reference or assignment. Therefore after unrolling of these subscripts they are converted into two arrays.

Original description	unrolling after partition of dimensions
<pre>reg (0:8) M[2][3][10]; =M[0] [x] [y]; ...=M[1] [x] [y]; M[0] [x] [y]= ...; M[1] [x] [y]=...;</pre>	<pre>reg (0:8) M_0_a_A[3] [10]; reg (0:8) M_1_a_A[3] [10]; =M_0_a_a[x] [y]; ...=M_1_a_a[x] [y]; M_0_a_a[x] [y]= ...; M_1_a_a[x] [y]=...;</pre>

11.4.2 Multidimensional Arrays - Partitioned Memory

It is possible to allocate multidimensional array along with given dimensions after dividing it into multiple memories. Cycle count can be reduced in the automatic scheduling mode because array reference/assignment, which was divided in different memory, can be executed in parallel. Further, the wiring for memory access gets improved because the accessed memory module is divided into multiple memories.

- In the dimension which is accessed by all constant subscripts, the array related to that dimension is unrolled and synthesized in a way that multiple memories are used (However, control from Options is possible).
- In case other dimensions have to be unrolled, it is synthesized in a way that multiple memories are used by specifying attribute.

Above 2 cases are explained below.

11.4.2.1 In case array for dimension accessed by all constant subscripts is unrolled

When multi dimension array having dimensions accessed by all constant subscripts is used, then multiple memories which are unrolled by such dimensions are used.

Original description	After part dimension unrolling
<pre>mem (0:8) M[2] [3] [10] /* 8(bits) x 600(words) */ = M[0] [x] [y]; ...= M[1] [x] [y]; M[0] [x] [y] = ...; M[1] [x] [y] = ...;</pre>	<pre>mem (0:8) M_0_a_a [3] [10]; mem (0:8) M_1_a_a [3] [10]; /* 8(bits) x 30(words) x 2(nos.) */ = M_0_a_a [x] [y]; ...= M_1_a_a [x] [y]; M_0_a_a [x] [y] = ...; M_1_a_a [x] [y] = ...; /* M_0_a_a(a) and M_1_a_a will be different memory*/</pre>

The function of partitioning arrays into multiple memories can be controlled by the following options. However, it cannot be applied in *shared* memory, as it is necessary to support in the method explained below when division of *shared* memory is done.

Option	Description
-Zarray2multi_mem	Multidimensional arrays are partitioned for allocation to multiple memories. (default)
-Zarray2multi_mem=NO	Multi-dimensional arrays are not partitioned.

11.4.2.2 In case array related to other dimensions are unrolled

When the functionality of dividing the memory related to other dimension more than once is used, it is necessary to specify the below expand_dim attribute.

In this attribute, the dimension value intended for unrolling is specified.

For every dimension, the specified dimension value is in the format where right most dimension of array is 1 and thereafter 2,3,... in the left. Further, when this attribute is specified, similar attribute value must be specified in all the processes using the corresponding shared memory.

Attribute	Description	Target setting
/* Cyber expand_dim = #[:#...]* /	Unroll multi-dimensional by # number from right	Array

In case of below example, subscript of 2nd and 3rd dimension from right during assignment of array M is variable. Memory division for these dimensions is possible. Here, if attribute and /*Cyber expand_dim=2:3*/ is specified for array M, memory division is specified in 2nd and 3rd dimensions from right and changes to 6 arrays after it is unrolled by these subscripts. It is allocated to separate memory.

Original description

```
mem(0:8) M[2][3][10]
/*Cyber expand_dim=2:3*/
/* 8(bits) x 60(words) */

M[x][y][z] = ...;
```

After memory division

```
mem(0:8) M_0_0_a[10];
mem(0:8) M_0_1_a[10];
mem(0:8) M_0_2_a[10];
mem(0:8) M_1_0_a[10];
mem(0:8) M_1_1_a[10];
mem(0:8) M_1_2_a[10];
/* 8(bits) x 10(words) x 6(Unit)

switch (x) {
case 0 :
    switch (y) {
        case 0 :
            M_0_0_a[z] = ...;
            break;
        case 1 :
            M_0_1_a[z] = ...;
            break;
        case 2 :
            M_0_2_a[z] = ...;
            break;
    }
    break;
case 1 :
    switch (y) {
        case 0 :
            M_1_0_a[z] = ...;
            break;
        case 1 :
            M_1_1_a[z] = ...;
            break;
        case 2 :
            M_1_2_a[z] = ...;
            break;
    }
    break;
}
```

11.4.3 Converting to One-Dimensional Arrays - Subscript Specification

Option	Description
-Zaddress_calc_mult = <i>kind</i> [: <i>kind</i> ...]	The address evaluation is done using the multiplication expression at the time of converting multi-dimensional array into one dimensional array.
-Zaddress_calc_concat = <i>kind</i> [: <i>kind</i> ...]	The address evaluation is done by the ::(concatenation operation) expression at the time of converting multi-dimensional array into one dimensional array.

Keywords to be specified in kind	Responding array type
io	Input output array variable
inside_mem	Internal memory
shared_mem	Shared memory
outside_mem	Outside memory
reg_file	Internal register array
shared_reg	Shared register array

Attribute	Description	Settings target
/*Cyber address_calc=MULT*/	The address evaluation at the time of converting multi-dimensional array into one dimensional array is done by multiplication.	Array variable
/*Cyber address_calc=CONCAT*/	The address evaluation at the time of converting the multidimensional array into	Array variable

	one dimensional array is done by :: (concatenation operation).	
--	---	--

Synthesis is done after the conversion of a multidimensional array into one-dimensional array.

At the time of conversion, the correspondence of the array elements of a multidimensional array to the one-dimensional array is similar to the C language. This means that when the subscript of the converted one-dimensional array is increased by one then the subscript at the extreme right of the original multidimensional array also increases by one.

Original description	After conversion into one dimension
-----------------------------	--

```

mem(0:16) a[3][100]; → mem(0:16) a[300];
    a[0][0] → a[0]
    a[0][1] → a[1]
    .
    .
    .
    a[0][99] → a[99]
    a[1][0] → a[100]
    a[1][1] → a[101]
    .
    .
    .
    a[2][99] → a[299]
```

There are two methods to convert multidimensional arrays into one-dimensional arrays. First method is given above. In the first method, the subscript evaluation at the time of array reference or assignment is changed into description using the constant multiplication as given below. In this method, element number after conversion to one-dimensional becomes same as the one in 'C' language, but at the same time subscript calculation will require a constant multiplication and addition, so it may require some calculation time and area.

Original description	After conversion into one dimension
-----------------------------	--

```

mem(0:16) a[3][100];   → mem(0:16) a[300];
var(0:2) x;           → var(0:2) x;
var(0:7) y;           → var(0:7) y;

.. = a[x][y];   → .. = a[ x*100 + y ];
a[x][1] = ...;   → a[ x*100 + 1 ] = ...;
```

The second method is to allocate a bit in each subscript. In this method, each subscript other than the extreme left subscript is rounded to the nearest power of 2. Therefore, if number of data elements of each subscript is not a power of 2, some unused elements will be allocated.

Original description	After conversion into one dimension
-----------------------------	--

```

mem(0:16) a[3][100];      →      mem(0:16) a[384];
(100 is rounded up to 128 )
    a[0][0]      →      a[0]
    a[0][1]      →      a[1]
    .
    .
    a[0][99]     →      a[99]
    a[1][0]      →      a[128]
    a[1][1]      →      a[129]
    .
    .
    a[1][99]     →      a[227]
    a[2][0]      →      a[256]
    .
    .
    a[2][99]     →      a[355]

```

During the evaluation of the subscript, each subscript will be linked using :: (concatenation operation). Therefore, the operation time or area is not increased during the subscript evaluation.

Original description	After conversion into one dimension
-----------------------------	--

```

mem(0:16) a[3][100];      →      mem(0:16) a[384];
var(0:2)  x;               →      var(0:2)  x;
var(0:7)  y;               →      var(0:7)  y;

.. = a[x][y];             →      .. = a[ x(0:2)::y(0:7)];
a[x][1] = ...;            →      a[ x(0:2)::1(0:7)] ...;

```

In the multidimensional array, which is not specified by attribute, the subscript evaluation is done according to the array type as given below.

However, when the data element has second power, it is necessarily evaluated using :: (concatenation operation).

The subscript evaluation of each array type can be changed using the “-Zaddress_calc_concat” option or the “-Zaddress_calc_mult” option. For example, “-Zaddress_calc_concat = inside_mem” is specified in order to change the subscript evaluation of the internal memory using the ::

(concatenation operation). Multiple specifications can be done by putting ":" while specifying the multiple memory types. For example, "-Zaddress_calc_concat = inside_mem:outside_mem".

Types of arrays	Subscript evaluation method when there is no specification	
	Method using multiplication	Method using ::
input output array variable	O	—
internal memory	O	—
shared memory	O	—
outside memory	O	—
internal array register	—	O
shared array register	O	—

11.4.4 Subscript Specification - Array Reference/ Assignment

If an array is referenced when an assignment is in progress, there is possibility that some data elements may be referenced while the array is being assigned. Therefore, referencing is disabled until the assignment is finished.

On the other hand, when the assignment is to be done, it should be ensured that all references are complete before the assignment process starts. Therefore, the assignment should be disabled until the referencing is finished.

Moreover, concurrent assignments or substitutions to the array are not allowed. If an assignment is in progress, subsequent assignments shall be disabled until the previous ones are complete.

However, concurrent references to the array are allowed because the result of an array references is not affected by the simultaneous references of same elements of the array.

In other words, there exists an interdependence relationship between the assignment and referencing as given below.

Order of Array reference, assignment	Interdependent relation (exists or does not exist)
reference → assignment	Yes
assignment → reference	Yes
assignment → assignment	Yes
reference → reference	No

However, it is clear from the following example that in case the value of the subscript at the time of array reference or assignment is different, there is no need to support the order relation. In other words, array reference can be done to M[x+1] without waiting for the result of the assignment to the array M[x].

Example 1:

```
M[x]= in 1;
out1=M[x+1];
```

Even if it is difficult to decide if the value of subscripts of an array is different in the description, then there are cases when the subscripts of the array are different. In such cases, subscript exclusion specification can be specified using the following attributes.

Attribute	Description	Settings target
/*Cyber ex_group=# [:#...] */	Specifies the exclusion of subscript at the time of array reference, assignment into group specified ('#').	Array reference, assignment.

The array reference of the same group number:

In the example 2 given below, it is considered that the value of the subscript of the assignment to the array m[b] and array reference m[c] is necessarily exclusive. (Group no. 1). Also, the subscript value of the array assignment m[a] and array reference m[d] is necessarily exclusive (Group no. 2). Multiple groups can be specified in a single attribute using " ".

Example 2 :

```

reg(0:8) m[5] /*Cyber rw_port=RW3*/
m[a] /* Cyber ex_group = 2 */= in 1;
m[b] /* Cyber ex_group = 1 */= in 2;
out1 = m[c] /* Cyber ex_group=1*/;
out1 = m[d] /* Cyber ex_group=2*/;
```

11.4.5 Exclusion Specification - Array Reference/ Assignment in Loop Folding

Attribute	Description	Settings target
/* Cyber folding_ ex_group= #[:#]*/	Specifies exclusion of subscript in loop folding.	Array reference, assignment.

Loop folding capability is explained in section **17**, and attribute mentioned above is explained in **section 17.3.8**.

11.4.6 Merging Multiple Arrays

At present, one memory is allocated to one array.

In order to allocate multiple arrays to one memory, multiple arrays are merged into one by specifying the attributes mentioned in the following table. Then, the merged array is allocated to a single memory.

Attribute	Description	Settings Target
/*Cyber array_merge = <i>name</i> */	Merges the multiple arrays with the name " <i>name</i> " as they occur in the declaration order.	Array variable
/*Cyber array_merge = <i>name:#</i> */	Merges in the array name " <i>name</i> " so that it begins with the data element number #.	Array variable
/*Cyber array_merge = <i>name@#</i> */	Merge multiple arrays with	Array variable

Attribute	Description	Settings Target
	array name "name" as #th numbered array.	

There are three methods to specify merging of multiple arrays. However, it is mandatory that merging method specified for original arrays should be same.

The first method is to specify the array name of the merged array. In this case, the array to be merged is allocated in order of declaration of original arrays starting from data element number 0 in the merged array.

This is illustrated in the figure below.

Description	After merging
<pre>mem(0:16) a[50] /* Cyber array_merge = M */; mem(0:16) b[70] /* Cyber array_merge = M */; mem(0:16) c[100]/* Cyber array_merge = M */; a[x] = ... b[y] = ... c[z] = ...</pre>	<pre>→ mem(0:16) M[220]; M[x] = ... M[y+50] = ... M[z+120] = ...</pre>

The second method is to specify the array name and the start position of the array in the merged array. In this case, the final merged array is allocated in such a way so that it starts from the element number specified in the merged array. The following figure illustrates this case.

Description	After merging
<pre>mem(0:16) a[50] /* Cyber array_merge = M:256 */; mem(0:16) b[70] /* Cyber array_merge = M:0 */; mem(0:16) c[100]/* Cyber array_merge = M:512 */; a[x] = ... b[y] = ... c[z] = ...</pre>	<pre>→ mem(0:16) M[612]; M[x+256] = ... M[y] = ... M[z+512] = ...</pre>

The third method is to specify the array name and the merging sequence. In this case, the final merged array is allocated in such a way that the component arrays occur in the sequence specified in the array_merge attribute of the array declaration. This case is illustrated in the figure below.

Description

```

mem(0:16) a[50] /* Cyber array_merge = M@2 */;
mem(0:16) b[70] /* Cyber array_merge = M@3 */;
mem(0:16) c[100] /* Cyber array_merge = M@1 */;
    a[x] = ...
    b[y] = ...
    c[z] = ...

```

→ mem(0:16) M[220];

```

M[x+100] = ...
M[y+150] = ...
M[z] = ...

```

While merging multiple multidimensional arrays into one array, the merging is done after converting them into one-dimensional array.

Description

```

mem(0:16) d[2][4][50]      /* Cyber array_merge = M */;
mem(0:16) e[2][4][70]      /* Cyber array_merge = M */;
mem(0:16) f[2][4][100]     /* Cyber array_merge = M */;
    d[x1][x2][x3] = ...
    e[y1][y2][y3] = ...
    f[z1][z2][z3] = ...

```

↓

After conversion into 1 dimension

```

mem(0:16) d[400] /* Cyber array_merge = M */;
mem(0:16) e[560] /* Cyber array_merge = M */;
mem(0:16) f[800] /* Cyber array_merge = M */;
    d[ x1*200 + x2*50 + x3 ] = ...
    e[ y1*280 + y2*70 + y3 ] = ...
    f[ z1*400 + z2*100 + z3 ] = ...

```

↓

After merging

```

mem(0:16) M[1760];
    M[ x1*200 + x2*50 + x3 ] = ...
    M[ y1*280 + y2*70 + y3 + 400 ] = ...
    M[ z1*400 + z2*100 + z3 + 960 ] = ...

```

11.4.7 Speculative Execution Specification

Speculative execution is to perform an operation or memory access that may not be needed if the determining conditions evaluates FALSE prior to checking the condition, in order to utilize empty cycles. The result is used only when the condition has been evaluated as TRUE. Otherwise, the result is discarded (**Section 19.3**).

In case the condition to perform speculative execution of array assignment is false then it would involve a lot of cost to remove that assignment therefore it is not supported at present. By default speculative execution of some part of array reference is not performed in order to refer out of range of memory elements.

- Array reference with speculative execution
 - Register array

- Array allocated in memory specified by element count with power of 2 in memory restrictions.
- Array allocated in memory unspecified by element count in memory restrictions.
- Array reference with controlled speculative execution
 - Array allocated in memory specified by element count other than power of 2 in memory restrictions.

Example:

```
mem(0:16) M[200];
if (i<200) {
    y= M[i];
}
```

However, it is possible to refer an array allocated in memory specified by element count other than power of 2 by specifying -Zmem_read_speculation option.

Option	Description
-Zmem_read_speculation	Performs speculative execution of memory reference at the time of automatic scheduling.
-Zmem_read_speculation = NO	Does not perform the speculative execution of memory reference at the time of automatic scheduling. (default)

Furthermore, it is also possible to explicitly specify permission or control of speculative execution of array reference or array by using attributes.

Attribute	Description	Settings target
/*Cyber speculation = YES*/	Performs the speculative execution of array reference.	Array variable, expression element
/*Cyber speculation = NO*/	Does not perform the speculative execution of array reference.	Array variable, expression element

Example:

```
mem(0:16) M[200] /*Cyber speculation= YES*/;
if(i<200){
    Y= M[i];
}
```

Example:

```
mem(0:16) M[200];
if(i<200){
    Y= M[i] /*Cyber speculation= YES*/;
}
```

11.4.8 Converting Array Accesses Using Switch

In order to achieve the I/O array, they must be completely unrolled. Ideally, it is required that all the subscripts at the time of array reference or assignment are constants for variable unrolling. However, even in case the subscript is a variable it is possible to make an equivalent constant expression using the switch statement and results in complete array being unrolled.

Attribute	Description	Settings Target
/* Cyber array_index=const*/	Converts the array reference/assignment into description which uses switch.	Array, statement, expression element, assignment point

Example:

```
in ter(0:16) M1[4] /* Cyber array_index = const */;
y = M1[i];
```



```
in ter(0:16) M1[4];
switch(i){
    case 0: y = M1[0]; break;
    case 1: y = M1[1]; break;
    case 2: y = M1[2]; break;
    case 3: y = M1[3]; break;
}
```

Example:

```
out ter(0:16) M2[4];
/* Cyber array_index = const */
M2[i] = z;
```



```
out ter(0:16) M2[4];
→ switch(i){
    case 0: M2[0] = z; break;
    case 1: M2[1] = z; break;
    case 2: M2[2] = z; break;
    case 3: M2[3] = z; break;
}
```

Setting targets according to conversion locations of attributes is given below.

- In case of array, all the locations where the array is used with expression element / assignment point.
- In case of statement, array of expression element / assignment point within the statement.
- Expression element of an array.
- Assignment point of an array.

After assigning the value in all elements as shown below, in case of array reference by array subscript, it takes 1 cycle after assignment till the reference because when synthesis is done as array then register array is used.

If array is developed after adding array_index = const attribute, then synthesis is possible only by multiplexer and register array is not used.

```

var(0:16) value[4] /* Cyber array_index = const */;
var(0:3) i;
var(0:3) s;

/* Cyber unroll_times = all */
for(i=0;i<3;i++) {
    value[i] = ..;
}
y =value[s];

↓

var(0:16) value_a00,value_a01;
var(0:16) value_a02,value_a03;
var(0:3) i;
var(0:3) s;
var(0:16) tmp;

value_a00 = ....;
value_a01 = ....;
value_a02 = ....;
value_a03 = ....;

switch(i) {
    case 0: y = value_a00; break;
    case 1: y = value_a01; break;
    case 2: y = value_a02; break;
    case 3: y = value_a03; break;
}

```

Caution Point

In case reference is out of scope of array subscript when power of element count of array is not 2 then return value of array is taken as 0.

Example:

```

mem(0:16) M3[3];
y = M3[i]/* Cyber array_index = const */;

```



```

mem(0:16) M3[3];
switch(i){
    case 0: y = M3[0]; break;
    case 1: y = M3[1]; break;
    case 2: y = M3[2]; break;
    default: y = 0; break;
}

```

11.4.9 Initial Value of an Array.

Option	Description
-Zno_aryinit	Ignores the initial value of array during synthesis.

This option is applicable to the synthesis phase, and initial value of the array will only be required during simulation phase (with behavioral description). With this option, behavioral description will be handled equivalently to the case when initial values are not specified.

11.4.10 Variable Name after Unrolling the Array

Attribute	Description	Settings Target
/* Cyber array_index_suffix=word*/	Specifies the character string of suffix added to variable that unrolls the array.	Array variable

By default, for array being unrolled, each element name is formed by adding a suffix “_a” followed by element number to the array name

For example, when array variable is Array[128], it is unrolled in the following manner: Array_a00, Array_a01,Array_a127 etc.

By default, for register array being converted, each element name is formed by adding a suffix “_rg” followed by element number to the register name

For example, in case of array variable regA[128], registers are generated with register names such as regA_rg00,regA_rg01, ...,regA_rg127etc.

An attribute “array_index_suffix” is available to explicitly specify first part of suffix.

```
var (0:8) val[8] /* Cyber array_index_suffix = _p*/;
      ↓
var(0:8)  val_p00;
var(0:8)  val_p01;
.
.
.
var(0:8)  val_p07;
```

```

reg (0:8) regB [8] /* Cyber array_index_suffix = _p*/;
↓
reg(0:8) regB_p00;
reg(0:8) regB_p01;
.
.
.
reg(0:8) regB_p07;

```

This attribute has a limitation. The limitation is that in the case of multidimensional arrays there are cases when only part dimensions of the multidimensional arrays undergo unrolling (refer to **section 11.4.2**). After partial dimensions are unrolled by naming convention, a suffix is added to them.

```

reg (0:8) regC [2] [8] /* Cyber array_index_suffix = _p*/;
↓
reg(0:8) regC_0_a[8]/* Cyber array_index_suffix = _p*/;
reg(0:8) regC_1_a[8] /* Cyber array_index_suffix = _p*/;
↓
reg(0:8) regC_0_a_p00;
reg(0:8) regC_0_a_p01;
.
.
.
reg(0:8) regC_0_a_p07;
reg(0:8) regC_1_a_p00;
reg(0:8) regC_1_a_p01;
.
.
.
reg(0:8) regC_1_a_p07;

```

11.5 Detailed Memory Specification

This section explains three out of five items that are used while specifying a memory. The rest two items shall be described in other sections. The items are as follows:

1. Port specification of memory.
2. Specification related to memory access timings.
3. Specification related to memory access chain.
4. Specification of circuits which write the initial value in memory (refer to **section 10.2.3**).
5. Specification of memory allocation type of arrays (refer to **section 20.9**).

11.5.1 Port Specification of Memory

The port to connect the memory is generated in accordance with the port information specified by the memory library (MLIB). In case the options “-Zmem_re”, “-Zmem_cs”, “-Zmem_be”, “-Zmem_clocking” etc. are specified, the memory allocation is done based on detailed counts provided in memory count file (MCNT). The memory that fulfills the specification according to the

option used is targeted for allocation. In case the attributes "mem_re","mem_cs","mem_be", "mem_clocking" etc. are specified in the array signal, then the memory that fulfills the specification of that attribute becomes the target of allocation to the aforesaid array.

In case the port information is not specified by the MLIB, the port to connect memory is generated. This is according to the generation rule of the Cyber internal. In the memory port where Cyber can be treated, there are three types of ports generated: read-write dual port, read-only port, and write-only port. Moreover, the following ports are generated in respect to each port by default.

1. read-write dual port
 - when it is external memory
 - read-write data port
 - read-write address port
 - write-enable port
 - when it is internal memory
 - read data port
 - write data port
 - read-write address port
 - write-enable port
2. read-only port
 - read data port
 - read address port
 - read-enable port
3. write-only port
 - write data port
 - write address port
 - write-enable port

11.5.1.1 Read-Write Port – Data Port

Option	Description
-Zrw1=1port	Generates the read write terminals in bidirectional read write ports of external memory. (default)
-Zrw1=2port	Generates the write data output terminal and read data input terminal in bidirectional read write ports of external memory.

In case port information is not specified in memory library (MLIB), then the read-write dual port of external memory is generated as two ports such as: write data output port and read data input port. When -Zrw1=1port option is specified, then a single bidirectional port is generated as a data port for read-write dual port of external memory.

11.5.1.2 Read-Write Port – Read-Enable Port

Option	Description
-Zmem_re	Generates the read enable terminal and write enable terminal in bidirectional read write port.
-Zmem_re=NO	Generates the write enable terminal in bidirectional read write port. (default)

Attribute	Description	Settings Target
/* Cyber mem_re= create */	Generates the read enable terminal in the bidirectional read write port of array memory.	Array variable
/* Cyber mem_re= ignore */	Do not generate the read enable port in the bidirectional read write port of array memory.	Array variable

If the port information is not specified by the MLIB, only the write-enable port is generated in the bidirectional read-write port by default. However, in case the memory having the read-enable port is used, then both the write-enable port and read-enable port can be generated using the following two methods:

- The first method is to specify “-Zmem_re” option. The read-enable port and write-enable port are generated in all the two way read-write ports.
- The second method is to specify the “mem_re” attribute in the array variable. The write-enable port and the read-enable port are generated in the two way read-write ports of the memory allocated to the array.

In case the port information is specified by the MLIB, then these options and attributes can be used to specify whether to use the memory with the read-enable port.

11.5.1.3 Chip-Select Port

Option	Description
-Zmem_cs	Generates the chip select terminal in the port.
-Zmem_cs=NO	Does not generate the chip select terminal in the port.

Attribute	Description	Settings Target
/* Cyber mem_cs= create */	Generates the chip select port in the memory ports.	Array variable
/* Cyber mem_cs= ignore */	Do not generate the chip select port in the memory ports.	Array variable

In case the port information of memory in the MLIB is not specified and the "-Zmem_cs" option is specified, then the chip-select port is generated in all the memory ports as given below. When the "mem_cs" attribute is specified in an array variable, the chip select ports as given below are generated in the allocated memory.

1. two way read-write port
 - chip-select port
2. read-only port
 - read chip-select port
3. Write-only port
 - write chip-select port

In case the memory port information is specified in the MLIB, then these options and attributes can be used to specify whether to use the memory with the chip-select port.

11.5.1.4 Byte-Enable Port

Option	Description
-Zmem_be	Allocates all the arrays whose data bit width is the multiple of 8 in the memory with byte enable port.
-Zmem_be=NO	Does not allocate all the arrays in the memory with byte enable port. (default)

Attribute	Description	Settings Target
/* Cyber mem_be= create*/	Allocates in the memory having byte enable port (Byte boundary is taken as 8 bit).	Array variable
/* Cyber mem_be= ignore*/	Does not allocate in the memory having byte enable port.	Array variable

There are cases where the byte-enable port can be used depending on the memory.

In case the port information is not specified by the MLIB, the byte-enable port is generated for all the memory by specifying the “-Zmem_be” option. Also, by adding “mem_be” attribute to the array, byte enable port can be generated in memory without specifying the port information allocated by array. However, in case port information is not specified clearly with MLIB, the byte-enable memory can be used only when the data bit width is multiple of 8. Here, the byte enable signal can be generated by 1 bit allocated for 8 bit of memory bit width.

In case byte enable port is specified with port information in MLIB, the array allocated for the byte enable memory with -Zmem_be option or mem_be attribute are assumed as an option for allocation.

However, except the case where data bit width of memory is a multiplier of 8 and 8 bit delimiter enable signal, byte enable pattern specification (ENABLEPTN) is specified besides the port information in MLIB and it is necessary to specify the memory allocated by INSTANCE in MCNT file or sig2mu_port attribute or sig2mu attribute in the allocated array.

In operation description, it is necessary to mention that assignment to array is carried out by partial bit assignment.

However, please note that byte enable is not used for an assignment in which the range of bit assigned this time does not match with the byte enable pattern. In such cases, the value of array assigned is acquired by all bit reference. Update the value of modified bit position and it is treated as behavior description which performs the bit assignment.

Operation description

```
mem(15..0) M[256] /* Cyber mem_be = create, sig2mu=MEMB16W256 */;
switch ( c ){
    case 0: M[v](15..12) = il; break;
    case 1: M[v](11..8) = il; break;
    case 2: M[v](7..4) = il; break;
    case 3: M[v](3..0) = il; break;
}
```

Memory library file (MLIB)

```
#@VERSION { 1.00 }
#@LIB { foo }
@MLIB {
    NAME      MEMB16W256
    KIND      R1,W1
    BITWIDTH 16
    DELAY    x2
    WORD     256
    ENABLEPTN 4+
    DEFMOD {
        in ter (7..0) RA1 /* ra:1 */;
        out ter (15..0) RD1 /* rd:1 */;
        in ter (0..0) RE1 /* re:1 */;
        in ter (0..0) RCLK1 /* rclk:1 */;
        in ter (7..0) WA2 /* wa:2 */;
        in ter (15..0) WD2 /* wd:2 */;
        in ter (0..0) WE2 /* we:2 */;
        in ter (3..0) BE2 /* be:2 */;
        in ter (0..0) WCLK2 /* wclk:2 */;
    }
}
#@END { foo }
```

Memory count file (MCNT)

```
#@VERSION { 1.00 }
#@CNT { foo }
@MCNT {
    NAME      MEMB16W256
    LIMIT    1
}
#@END { foo }
```

11.5.1.5 Clock Port for Synchronous SRAM

Option	Description
-Zmem_clocking = none	Does not generate memory clock port for synchronous SRAM. (default)
-Zmem_clocking = enable	Generates memory clock port for synchronous SRAM and outputs memory clock signal only at the time of memory access. (corresponds to the -syncRAM option of the earlier Cyber 3.6)
-Zmem_clocking= always	Generates memory clock port for Synchronous SRAM and outputs a steady memory clock signal. (corresponds to the -syncRAM2 option of the earlier Cyber 3.6)

/*Cyber mem_clocking = enable*/	Generates the memory clock port for synchronous SRAM and outputs the memory clock signal only during memory access.	Array variable
/*Cyber mem_clocking = always*/	Generates the memory clock port for synchronous SRAM and outputs memory clock signal on a steady basis.	Array variable

In case the port information of memory is not specified in the MLIB, then by specifying the " -Zmem_clocking" option, the following clock ports are generated. If "mem_clocking" attribute is specified in array, then the following clock ports are generated in memory allocated in array:

1. bidirectional read-write port
 - memory clock port
2. read-only port
 - read-clock port
3. write-only port
 - write-clock port

11.5.1.6 Treatment of Unused Ports

Option	Description
-apA:o	Does not generate the port used by external memory.
-api:o	"
-ap#:o	"

Depending on the condition of reference or assignment of an allocated array, the port for external memory is not generated if the port is not used by the external memory. By default, even unused ports are generated. They are generated to behave in following ways:

- Output port is fixed to '0'
- Input port is left floating
- Bidirectional port has a high impedance output

If after the options for external memory specification (-apA,-api,-ap#) ":o" is specified (-apA:o,-api:o,-ap#:o), the generation of the port that is not used by the external memory is suppressed.

However, in case the port of the memory port is specified by the port information (DEFMOD) of the MLIB, the port for the external memory is generated even if the port is unused. In this case, the options (-apA:o,-api:o,-ap#:o) are not valid.

11.5.2 Specification related to chaining of memory access

Option	Description
-mc	Allows chaining the memory reference in assigning a memory.
-mcD	Allows chaining of memory reference to a different memory in assigning a memory.
-mcN	Does not chain the memory reference in assigning a memory.

In automatic scheduling mode, in case memory reference is done during the first half of the cycle and memory assignment is done in the later half of the cycle, scheduling is done in such a way that the data referenced from memory in the same step is assigned to the memory by default.

In case the "-mcN" option is specified, and if data referenced belongs to same memory, scheduling is done in such a way that referencing is performed in a different cycle by placing a register in the path.

In case the “-mcD” option is specified and the data referenced by a memory is assigned to a different memory, the scheduling is done in the same cycle. And in case the assignment is done to a different port by the same memory, the scheduling is done in a different step.

11.6 Access Timings Specifications

Three types of memories are supported in CyberWorkBench.

- Synchronous read and write memory(pipeline memory)
- Asynchronous read and synchronous write memory
- Asynchronous read and asynchronous write memory

“Synchronous read and write memory” is defined as “pipeline memory” in the manual, In this section, access timing and specification methods required for each type of memory have been explained.

Option	Description
-mr?w?	Specifies the memory access timings.

Attribute	Description	Settings Target
/*Cyber rw_timing=r?w?*/	Specifies the timings of array reference, assignment.	Array variable

- Manual scheduling

Timing type specifying in r? or w?	
Symbol	Description
rewe	Uses pipeline memory.
rawe	Uses asynchronous read memory and synchronous write memory.
rf	Outputs the read enable signal in the first half of the clock cycle during memory reference.
rf0	Outputs read enable signal in the first half of the clock cycle during memory reference.
rl	Outputs the read enable signal in the later half of the clock

	cycle during memory reference.
rl0	Outputs the read enable signal in the later half of the clock cycle during memory reference.
wf	Outputs the write enable signal in the first half of the clock cycle during memory assignment.
wf0	Outputs the write enable signal in the first half of the clock cycle during memory assignment.
wl	Outputs the write enable signal in the later half of the clock cycle during memory assignment.
wl0	Outputs the write enable signal in the later half of the clock cycle during memory assignment.

- Automatic scheduling

Timing type specifying in r? or w?	
Symbol	Description
rf	Outputs the read enable signal in the first half of the executable cycle, and does not perform address calculation in the cycle similar to executable cycle.
rf0	Outputs the read enable signal in the first half of the executable cycle, and does not perform address calculation apart from the ones which can be done at a short delay.
rl	Outputs the read enable signal in the later half of the executable cycle, and does not use the reference data in calculation in the cycle similar to the executable cycle.
rl0	Outputs the read enable signal in the later half of the executable cycle, and does not use the reference data in the calculations except the ones which can be done at a short delay in the cycle similar to the executable cycle.

wf	Outputs the write enable signal in the first half of the executable cycle and does not perform the address calculation, data calculation in the cycle similar to the executable cycle.
wf0	Outputs the write enable signal in the first half of the executable cycle, and does not perform address calculation, data calculation apart from the ones which can be done at a short delay.
wl	Outputs the write enable signal in the later half of the executable cycle.
wl0	Outputs the write enable signal in the later half of the executable cycle.
rewe	Uses the pipeline memory.

11.6.1 Synchronous read and write memory (Pipeline memory)

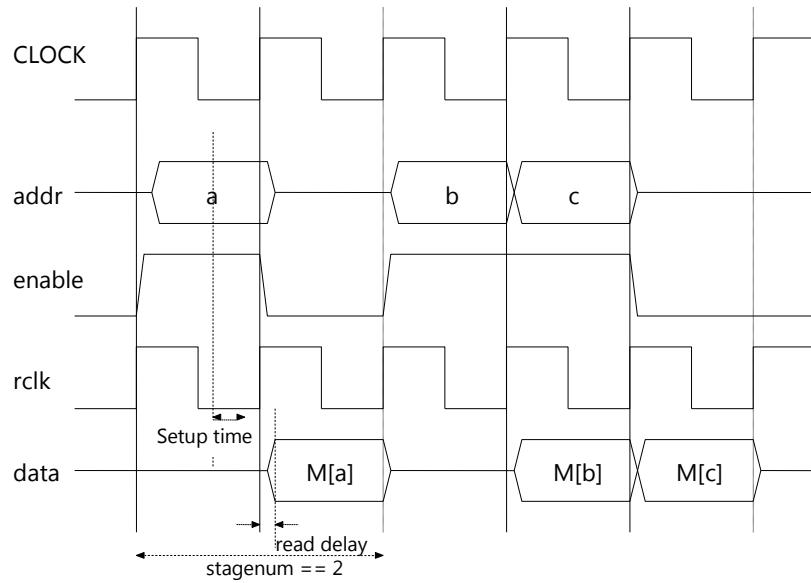
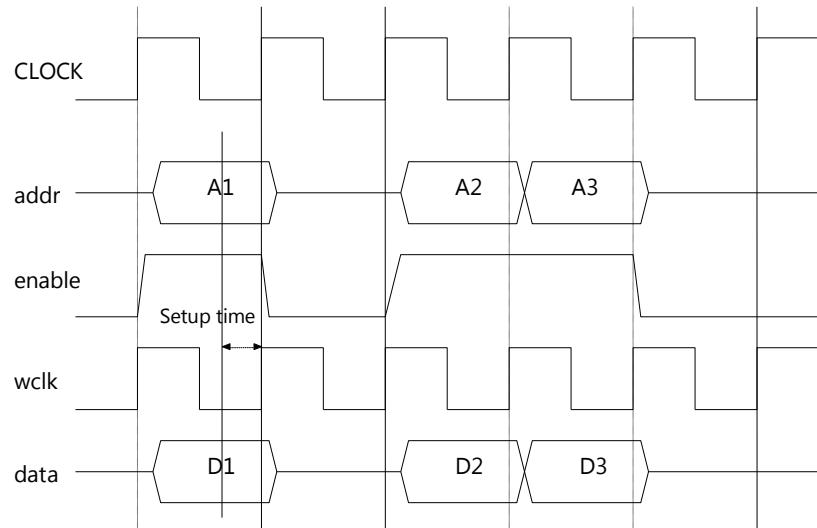
A pipeline memory operates in synchronization with system clock, allowing for the use of an entire clock cycle for the calculation of the address. In comparison, memory clocks in non-pipelined memories utilize the negative phase of the system clock and need to complete the address calculation within half clock cycle of the system clock. Either of the two memories may be deployed as required.

In order to use the pipeline memory function, the following settings are required:

- To specify the read, delay, and the number of stages (stage num) in the DELAY field of the MLMT and MLIB.
- Number of stages for array variable is determined uniquely. For example, number of stages is specified in attribute (pipemem=x#).

Figure 29 depicts the timing chart of memory read operation. The data is being received by the clock next to the clock which fixes the address. Such timing are called that "the number of stages (stagenum) are 2" (this is specified by delay field of MLMT/MLIB). The time till the read data is valid from the rising of memory clock, is called as (read delay) or (delay). (Similarly, it is specified by the DELAY field of MLMT/MLIB).

Figure 30 depicts the timing chart of memory write operation. Writing is actually done to the memory by the next clock, which fixes the address and data.

**Figure 29:** read timings of pipeline memory**Figure 30:** Write timings of pipeline memory

11.6.1.1 Specification Method of Pipeline Memory

Option	Description
-mrewe	Schedules the array as synchronous memory.

Attribute	Description	Setting Target
/* Cyber rw_timing = rewe */	Schedules the array as synchronous memory.	Array variable

Attribute	Description	Settings Target
/* Cyber pipemem = x# */	Specifies the number of stages of pipeline memory	Array variable

In order to allocate array in pipeline memory, it is necessary to determine the number of stages for array variable. Number of stages is determined by any of the following method.

1. Allocated memory module is specified by attribute (sig2mu or sig2mu_port) and number of stages is determined from MLMT or MLIB. Description of MLMT, MLIB is given later.
2. Number of stages is specified by attribute (pipemem).
3. Number of stages in MLMT or MLIB registers only similar pipeline memory. Description of MLMT, MLIB is given later.

However, when number of stages is determined by 1, 3 method, since number of stages cannot be determined during the auto generation (-Zmlib_mcmt_out) of MLIB, option -mrewe is specified or attribute rw_timing = rewe is specified in array variable. It is necessary to specify that array variable is pipeline memory. In such case, number of stages for auto generated MLIB is always 2.

Also, it is necessary to register the information of memory that can be used as pipeline memory in the MLMT or the MLIB. In order to specify that the memory can be used as a pipeline memory, read delay and stagenum need to be specified in the following format in the DELAY field of MLIB/MCNT file.

N x M

x M

(N: Read delay. N is a natural number greater than or equal to 0. This value is set to 1 if not specified.)

(M: Stage num. M is a natural no. greater than 2)

Read delay must be smaller than the clock cycle. The memory in which read delay is bigger than the clock cycle, read delay cannot be allocated even if it is specified in the memory constrained

file. In case the memory whose read delay is smaller than a clock cycle is specified by sig2mu attribute, it will result in an error.

Setup time can be specified using port information (DEFMOD) of memory library file (MLIB) in the pipe line memory.

Below is the example of MLMT and MLIB.

[MLMT]

# IDX	NAME	LIMIT	KIND	BITW [xWORD]	DELAY
1	mem01A	1	RW1	8x128	100x2
2	mem02A	1	R1,W1	8x128	100x2

[MLIB]

```

#@VERSION { 1.00 }
#@LIB { MLIB_GENERATED_BY_MSUM2MLIB }
@MLIB {
    NAME      RAM_R1_32x250
    KIND      R1
    BITWIDTH 32
    WORD      250
    DELAY    100x2 # Select for pipeline memory
use.
    DEFMOD {
        # i/o t/r bitw name /* kind:port,pol */;
        in   ter   (0:8) RA    /* ra:1 */;
        in   ter   (0:1)  CS    /* rs:1,neg */;
        in   ter   (0:1)  BEA   /* rclk:1 */;
        out  ter   (0:32) DOA   /* rd:1 */;
        in   ter   (0:8)  test1 /* other */;
    }
}
#@END { MLIB_GENERATED_BY_MSUM2MLIB }
```

11.6.1.2 Limitations

Here, there are certain limitations as given below.

- While using external memory of the RWn type, the bidirectional port cannot be used.
- While using external memory of the RWn type, It is necessary that either the option – Zrw1=2port is specified or read data (rd) and write data (wd) port are specified at defmod part of MLIB file.

11.6.1.3 Pipeline Memory Access by Automatic Control

Address input, Data output timing is controlled automatically depending on the number of stages determined uniquely for items with description similar to normal array access.

11.6.1.3.1 Pipeline Memory Access by Automatic Control during Manual Scheduling

During manual scheduling, address is input when memory reference is described and data is output after the cycle for number of stages passes. That value cannot be used until data is output after address input.

BDL Example

```

mem char ary[256]/* Cyber pipemem = x3 */;
char addr;
char data, data2;
char v;
process entry(){
    v = ary[addr]; // Address input
    $  

    // data = v;    // Error during reference
    $           // Make it such that state from address
    data = v;    // Data output input till data output changes to 3 or more
    $  

    data2 = v;   // Can be referred here also
}

```

SystemC Example

```

SC_MODULE(foo) {
    cwb::cwb_mem<sc_int<8>, CWB_1D(256)> ary/* Cyber pipemem = x3 */;
    void entry();
    SC_CTOR(foo) {
        SC_CTHREAD(entry, clk.pos());
        :
    }
};

void
foo::entry() {
    sc_uint<8> addr;
    sc_int<8> data, data2;
    sc_int<8> v;
    wait();
    while(true) {
        v = ary[addr]; // Address input
        wait();
        // data = v;    // Error during reference
        wait();        // Make it such that state from address
        data = v;    // Data output input till data output changes to 3 or more
        wait();
        data2 = v;   // Can be referred here also
    }
}

```

Following limitations are there in the assignment point variable of pipeline memory reference.

- Scalar variable of sign, bit width same as pipeline memory variable is required.
- In case of BDL description, it is necessary to change physical type with var type or without it.

11.6.1.3.2 Pipeline Memory Access by Auto Control during Automatic Scheduling

During automatic scheduling, address is input when memory reference is described and data is output after the cycle for number of stages passes. Timing control from address input till data output is done automatically.

BDL Example

```
mem char ary[256]/* Cyber pipemem = x3 */;
out char out1;
char addr;
process entry() {
    out1 = ary[addr]; //State from address input till data output
                      // changes automatically to 3 or more
                      //After that, it is assigned to out1
}
```

SystemC Example

```
SC_MODULE(foo) {
    cwb::cwb_mem<sc_int<8>, CWB_1D(256)> ary/* Cyber pipemem = x3 */;
    sc_out<sc_int<8>> out1;
    void entry();
    SC_CTOR(foo) {
        SC_CTHREAD(entry, clk.pos());
        :
    }
};

void
foo::entry() {
    sc_uint<8> addr;
    wait();
    while(true) {
        out1.write(ary[addr]); // State from address input till data
                              // output changes automatically to 3 or more
                              // After that, it is assigned to out1
        wait();
    }
}
```

When scheduling block is used, it is described with a method (**11.6.1.3.1**) same as manual scheduling.

BDL Example

```

mem char ary[256]/* Cyber pipemem = x2 */;
out char out1;
char addr;
char v;
process entry() {
    /* Cyber scheduling_block */
    {
        v = ary[addr];    // Make it such that state from address
        $                  // input till data output changes to 2 or more
        out1 = v;
    }
}

```

SystemC Example

```

SC_MODULE(foo) {
    cwb::cwb_mem<sc_int<8>, CWB_1D(256)> ary/* Cyber pipemem = x2 */;
    sc_out<sc_int<8>> out1;
    void entry();
    SC_CTOR(foo) {
        SC_CTHREAD(entry, clk.pos());
        :
    }
};
void
foo::entry() {
    sc_uint<8> addr;
    sc_int<8> v;
    wait();
    while(true) {
        /* Cyber scheduling_block */
        {
            v = ary[addr]; // Make it such that state from address input
            wait();        // till data output changes to 2 or more
            out1.write(v);
        }
        wait();
    }
}

```

11.6.1.4 Pipeline Memory Access by Manual Control

Attributes	Description	Setting Target
/* Cyber pipemem_io */	Used as pipeline memory input output variable	Signal

Assignment in Pipeline memory input/output variable from Pipeline memory is considered as Address input and Pipeline memory input/output variable reference is considered as data output and timing can be controlled manually.

Manual control is used in following cases.

- Address input is done but data is not read (Dispose the data after reading)

```
in unsigned char in1;
in char c1;
out char out1;
mem char ary[256]/* Cyber pipemem = x2 */;
char v/* Cyber pipemem_io */;
process entry() {
    do {
        v = ary[in1]; // Outputs the value when c1 become false.
    }while(c1);
    out1 = v;
}
```

- Read data without address input (Read empty data)

```
in unsigned char in1;
in char c1;
out char out1;
mem char ary[256]/* Cyber pipemem = x2 */;
char v/* Cyber pipemem_io */;
process entry() {
    if (c1) {
        v = ary[in1];
    }
    out1 = v; // Outputs valid value when c1 is true
}
```

- Address input, Data output between different loop iteration is done.

```
in unsigned char in1;
in char c1;
out char out1;
mem char ary[256]/* Cyber pipemem = x2 */;
char v/* Cyber pipemem_io */;
process entry() {
    v = ary[in1];
    do {
        out1 = v; // Outputs value corresponding to address
                   // input before 1 iteration

        v = ary[in1];
    }while(c1);
}
```

Pipeline Memory Input/Output Variable consists of following characteristics.

- Writing in pipeline memory Input/Output variable only allows simple assignment from pipeline memory.
- It is considered that address input is done in pipeline memory when written in pipeline memory input output variable.
- Data is considered to be output from pipeline memory when Pipeline Memory Input/Output Variable is referred.

- Pipeline Memory Input/Output Variable must be pipeline memory which becomes assignment source and bit width, signal, physical type are same.
- In Pipeline Memory Input/Output Variable, not only scalar are used but arrays are also usable.

11.6.1.4.1 Pipeline Memory Access by Manual Control during Manual Scheduling

In manual scheduling, both address input, data input can be specified at any time. However, whether correct value is obtained or not is user's responsibility.

BDL Example

```

mem char ary[256]/* Cyber pipemem = x3 */;
char addr;
char data;
char v/* Cyber pipemem_io */;
process entry(){
    v = ary[addr]; // Address input
    $  

    $           // Make it such that state from
    data = v;   // Data output address input till data output changes to
                3
    $  

}

```

SystemC Example

```

SC_MODULE(foo) {
    cwb::cwb_mem<sc_int<8>, CWB_1D(256)> ary/* Cyber pipemem = x3 */;
    void entry();
    SC_CTOR(foo) {
        SC_CTHREAD(entry, clk.pos());
        :
    }
};
void
foo::entry() {
    sc_uint<8> addr;
    sc_int<8> data;
    sc_int<8> v/* Cyber pipemem_io */;
    wait();
    while(true) {
        v = ary[addr]; // Address input
        wait();
        wait(); // Make it such that state from
        data = v; // Data output address input till data output changes to 3
        wait();
    }
}

```

11.6.1.4.2 Pipeline Memory Access by Manual Control during Automatic Scheduling

In automatic scheduling, the time constraint function (interval_id attribute, path_interval_id attribute, latency_loop attribute) of address input and data output or using the scheduling block, etc. is controlled by user.

However, there is no divergence between address input and data output and in case the correspondence relation of one to one address input, data output is known, then scheduling is done in a way that cycle for number of stages desegregated.

BDL Example

```

in char c1;
mem char ary[256]/* Cyber pipemem = x2 */;
char addr;
char data;
char v/* Cyber pipemem_io */;
process entry(){
    // Outputs data at
    // x2 time
    /* Cyber latency_loop=1T */      // as loop body is
    do {                           // scheduled in 1 cycle.
        data = v;                  // Data output
        v = ary[addr];            // Address input
    }while(c1);
}

```

SystemC Example

```

SC_MODULE(foo) {
    sc_in<sc_int<8> > c1;
    cwb::cwb_mem<sc_int<8>, CWB_1D(256)> ary/* Cyber pipemem = x2 */;
    void entry();
    SC_CTOR(foo) {
        SC_CTHREAD(entry, clk.pos());
        :
    }
};

void
foo::entry() {
    sc_uint<8> addr;
    sc_int<8> data;
    sc_int<8> v/* Cyber pipemem_io */;
    wait();
    while(true) {
        // Outputs data at
        // x2 time
        /* Cyber latency_loop=1T */      // as loop body is
        do {                           // scheduled in 1 cycle.
            data = v;                  // Data output
            v = ary[addr];            // Address input
            wait();
        }while(c1.read());
        wait();
    }
}

```

11.6.1.4.3 Example in case Behavior Level Verification and Cycle/ RTL Verification differs

When timing control for pipeline memory access is done manually, exercise caution when Behavior level verification and cycle/ RTL verification values does not match.

Example is shown below.

- In case the number of stages specified does not match with the timing of data output from address input, even if the state from address input till data output is specified by 3 through pipemem attribute. If it is accessed by state 2 in behavioral description, then value of in1 address comes out in out1 in behavior level verification, but the value of in1 address does not come out in out1 in cycle/RTL verification.

```

in unsigned char in1;
in char c1;
out char out1;
mem char ary[256]/* Cyber pipemem = x3 */; //State from address input
                                              //till data output is
                                              //expected to be 3
char v/* Cyber pipemem_io */;
process entry() {
  v = ary[in1];
  $  

  out1 = v; // State from address input till data output is 2
}

```

- In case of data output received corresponding to address input before multiple iteration, it is a description expecting access to pipeline memory during loop and receiving data output corresponding to address input of 2 iteration. In behavior level verification, value before 1 iteration is output to out1 but value before 2 iteration is output in cycle/RTL verification.

```

in unsigned char in1;
in char c1;
out char out1;
mem char ary[256]/* Cyber pipemem = x3 */; // State from address input
                                              // till data output is
                                              //expected to be 3
char v/* Cyber pipemem_io */;
process entry() {
  int i;
  /* Cyber latency_loop = 1T */
  for (i=0; i<c1; i++) {
    out1 = v; // Outputs data corresponding to address input before 2
              // iteration
    v = ary[in1];
  }
}

```

In such description, whether address input/data output or the mechanism for entering both in the conditional clause is required for matching behavior level verification and cycle/ RTL verification. Further, as shown in below example, using arrays in pipeline memory input/Output variable is valid.

```

in unsigned char in1;
in char c1;
out char out1;
mem char ary[256]/* Cyber pipemem = x3 */; // State from address
                                              // input till data output //is
                                              // expected to be 3
char v[2]/* Cyber pipemem_io */;
process entry() {
int i;
/* Cyber latency_loop = 1T */
for (i=0; i<c1; i++) {
    out1 = v[i%2]; // Outputs data corresponding to address input before 2
iteration
    v[i%2] = ary[in1];
}
}

```

11.6.1.5 Pipeline Memory of Manual Scheduling Mode of CWB5.1.2 and prior

Synthesis is possible by carrying out the below points for pipeline memory access through partition description prior to CWB5.1.2.

- Regarding memory read behavior, BDL description (Memory partition description) for pipeline memory is provided.
- Option -Zpipemem1=1 is specified.
- During behavioral synthesis, specification option (-mrewe) related to timing of memory access or timing of pipeline memory in attribute (rw_timing=rewe) is specified. (Described earlier).
- Pipeline memory in MLMT file, MLIB file is arranged (Described earlier).

11.6.1.5.1 Memory Partition Description

In case pipeline memory is used in manual scheduling mode, it is necessary to describe an address input and a data output separately for memory read operation.

The address from which data has to be fetched/read has to be described as array expression not having the left part, and the data can be assigned to the output similarly as done in usual array reference expression. Pairing of address input and data output results into read memory once. This is called the memory partition description. For the array that is described by memory partition, only the pipeline memory is allocated. Also, it is necessary that the interval of the state describing the address input and the state describing the data output must match the state of memory to be used. If it does not match, then the synthesis shows an error. The memory partition description in the pipeline memory of 2 cycles by stage num is given below.

Note that memory partition description is not required for memory write operation.

```

in ter(0:8) i_x ;
out reg(0:8) o_x ;
mem char ary[256];
process main(){
    char addr ;
    addr = i_x ;
    ary[addr];      // Address Input
    $              // from address input till data output
    o_x = ary[addr];// set state count of data output as 2
}

```

11.6.1.5.2 Memory Partition Description - Array Subscript

For the array subscript of memory partition description, the array subscript of data output needs to be described as same element as the array subscription of address input. The array subscript of data output is only used to get the correspondence of data output and it is omitted from the synthesis target. There are structural constructs in array subscript which can be specified or which cannot be specified.

Table 1 shows the structural constructs which can be specified, and **Table 2** shows the structural constructs which cannot be specified. Now in case the variables which cannot be specified need to be used in the subscripts, it is possible to specify them indirectly: store values of those variables in the middle variables once and then specify those middle variables in array subscripts.

Structural constructs that can be specified	Specify example
Constant	ary[10]
ter,reg variable	ary[x]
Operation expression	ary[x+y]
Array	ary1[ary2[x]]

Table 1: Structural constructs Things that can be specified as array subscript of memory partition description

Structural constructs which cannot be specified
in,out variables
Functions

Table 2: Structural constructs Things that cannot be specified as subscripts of memory partition description

11.6.1.5.3 Memory Partition Description - Cautions

In case of using memory partition description, please be careful about the following points

- In case of using local variable of function in a subscript, be careful about its name space (scope). Even if the names of two variables are the same, the function to which those variables belong is different and then they will not be considered as the same subscript. For instance, in the following example, a variable of same name "addr" has been used along with the address input and data output expression. However, it shall not be recognized as the same subscript because each belongs to a different function.

```
void Ad(char addr) {  
    ary[addr];  
}  
ter char Rd(char addr) {  
    ter char tmp;  
    tmp = ary[addr];  
    return tmp;  
}  
process main() {  
    Ad(t) ;  
    $  
    Rd(t) ;  
    $  
}
```

- In the memory partition description, you must be careful that the correspondence of address input expression and data output expression is being judged by having the uniformity of the expression of that subscript. And it is not being judged on the base that it has the same value.

For example, in the following description, since the physical type of the array subscript is "ter", the value of array subscript in address input expression and that in data output expression actually differs from each other. The value of array subscript "addr" in the address input expression takes the value of input signal in0. However, since the value of array subscript "addr" in the output expression is not defined by anything, it becomes 0.

Despite that it is judged that data output expression (2) corresponding to address is same as address input expression (1).

```

in char (0:8) in0 ;
out char (0:8) o_x ;
mem char ary[256];
process main(){
char addr ;
    addr = in0 ;
    ary[addr];      // Address input expression (1)
    $               //
    o_x = ary[addr];// Data output expression (2)
}

```

On the contrary, in the following description, you will realize that the value of array subscript in address input expression and the value of subscript in data output expression become same if you see closely. However, since the signal being used in array subscript differs, these expressions are judged to be not corresponding.

```

void Ad(char addr) {
    ary[addr];
}
ter char Rd(char addr) {
    ter char tmp;
    tmp = ary[addr];
    return tmp;
}
process main(){
    reg(0:8) r0 ;
    r0 = in0 ;
    $
    Ad(r0) ;
    $
    Rd(r0) ;
    $
}

```

11.6.1.5.4 Memory Partition Description - Limitations

There are following limitations in the memory partition description:

1. In a loop whose count (upper bound) is not specified or loop that is not unrolled, describing only one part of memory partition description (address input, data output) is not allowed. In such a case, there is a possibility that Cyber may misunderstand the response of address input and data output. It is important to be careful.

```

in ter i_x, i_y ;
out reg o_x ;
mem(0:8) ary[4];
process main(){
    reg char cnt1 = 0 ;
    reg char cnt2 = 0 ;
    reg char addr, max ;

    addr = i_x ;
    max = i_y ;
    $
    for(;cnt1 <=max;++cnt1$) {
        ary[addr]; // In loop with unknown number of iterations
                    // only address input can be described
    }
    $
    for(;cnt2 <=max;++cnt2$) {
        o_x = ary[addr]; // In loop with unknown number of iterations
                           // only data output can be described
    }
    $
}

```

2. The physical type of the array performing the memory partition description should be "mem".
In case it is not "mem", the following synthesis error message occurs:

```
F_BT7954: Type of array isn't "mem".
[Action] Declare array with "mem" type.
```

3. In one BDL description, the memory partition description and continuous assignment for array cannot be mixed. If they are mixed in one BDL description, synthesis is interrupted as the following error:

```
F_BT7958: Can't refer the array (%s) assigned pipeline memory and
described always connect.
```

Even if the target of memory partition description and the target of "always connect" description is different like the description shown below, mixing these description in one BDL is limited.

```
mem(0:8) a_ary[32]/* Cyber rw_timing = rlwl */;
const var(0:2) CONST_TABLE[3] = {3, 2, 1};

process main()
{
    in0 :> ter(0:8) t0 ;
    out1 ::= CONST_TABLE[in1]; /* Array that performs total
                                combination and*/
    t0 = in0 ;
    p_ary[t0]; /* memory that describes the memory allocation
                  have restrictions even if they are different */
    $
    out0 = p_ary[t0];
    return ;
    $
}
```

4. As shown below, the multiple data output cannot be described for one address input.

```
ary[addr];
$
o_x = ary[addr];
o_y = ary[addr];
$
```

In such a case, this limitation can be avoided by assigning data output to an intermediate variable first.

```
ary[addr];
$
tmp = ary[addr];
o_x = tmp;
o_y = tmp;
$
```

5. Memory partition description cannot be used in parallel with pointers

In case Memory partition description and pointer description are used in parallel, Synthesis is interrupted with the following message.

```
F_BT4837 No left side found with pointer reference,
or pointer to pipeline memory exist
[Action] Check bdl description.
```

6. In case data output expression is referred by partial bit then address input expression should also be described so that it is referred similarly by partial bit.

```
in ter(0:8) in0 ;
out reg(0:8) o_x ;
mem char ary[256];
process main(){
    char addr ;
    addr = in0 ;
    ary[addr](4:4);      // Address input expression also
                          // requires partial bit reference
                          // expression
    $                   //
    o_x = ary[addr](4:4); // Refer data output expression with
                          // partial bit
}
```

7. It does not support partial bit assignment to perform Read Modify Write.
When partial bit assignment of array is described then normally it is implemented with one of the following methods.

- In case byte enable (or bit enable) port of memory is ready for use then it is used.
- In case byte enable port of memory cannot be used then synthesis is done to do Read Modify Write

However, in case pipeline memory is to be used with manual scheduling then synthesis to carry out Read Modify Write cannot be done due to restriction.

- Read Modify Write is a method to implement partial bit assignment in following order.
- Read out specified element from memory. Process the read data. Modify only the value which should be assigned and use the value output from memory in bit which should not be assigned.
- Write the processed value in specified element of memory.

When Partial bit assignment is described and byte enable port cannot be used (Read Modify Write is required) then following Error Message is output.

F_BT7953: Memory address input to support memory data output of 2nd state cannot be found.

[Action] Describe memory address input 1 state before the 2nd state.

And please note that the above error is not resolved even if memory address input required in Read Modify Write is described as per the error message.

In order to carry out partial bit assignment that requires Read Modify Write, then it is necessary to explicitly describe the Read Modify Write process. Following is an example to explicitly describe Read Modify Write.

```

in ter(0:8) in0 ;
out reg(0:8) o_x ;
mem char ary[256];
process main(){
    char addr ;
    char M_01 ;
    char M_02 ;
    addr = in0 ;
    ary[addr];
    $
    M_01 = ary[addr] ;      // Read value of target element
    M_02 = M_01(3..0)::in0 ;// Process the value
    ary[addr] = M_02;      // Write processed value in
                           // target element
}

```

11.6.1.5.5 Memory Partition Description - Description Example

- In case the arithmetic operation is used in subscript, the expression of subscript of the data output does not become the target of synthesis.

```

ary[addr+1];
$
rd=ary[addr+1]; // the adder of subscript is not synthesized
$
```

Note that, the space included in the subscript is ignored.

- Continuous reading of memory (pipeline access) (part 1)

```

ary[addr+1];
$
dout = ary[addr+1];
ary[dout];    // Next address input possible to describe
$              // with same state as first data output
dout = ary[dout];

```

- Continuous reading of memory (pipeline access) (part 2)

```

cnt = 0 ;
$
addr = cnt;
ary[addr];
$
for(; cnt < num; cnt++){
    addr = ary[addr] + 1;
    ary[addr];
}
dout = ary[addr] ;
$
```

- Example of using two port memory

```

ary[addr+1];
ary[addr+2];
$
o_x = ary[addr+1];
o_y = ary[addr+2];
$
```

- Two-dimensional arrays

In case of multidimensional arrays, it is important to make all the subscripts same as address input.

```

ad_x = i_x ;
ad_y = i_y ;
ary[ad_x][ad_y];
$
o_x = ary[ad_x][ad_y];
$
```

- Concatenation on the left side

```

in ter(0:8) i_x ;
out reg(0:8) o_x, o_y ;
short ary[256];
process main()
{
    char addr;
    addr = i_x ;
    ary[addr];
    $
    o_x::o_y = ary[addr]; // Converts upper level 8 bit of read data
                           // into o_x, lower level 8 bit in o_y
}

```

- `typedef struct _tag {`

```

    char a;
    char b;
    char c;
    char d;
} RULE;

in ter(0:8) i_x ;
out reg(0:32) o_x ;
RULE ary[256];
process main(){
    char addr;
    RULE dout ;
    addr = i_x ;
    ary[addr];
    $
    dout = ary[addr];
    o_x = (dout.a + dout.b) + (dout.c + dout.d);
}

```

- When the subscripts are arrays

```

k[addr];                                //address input of array k
$
ary[k[addr]];                            // data output of array k and
                                         // address input of array ary
$
dout= ary[k[addr]];                     // data output of array ary

```

In order to make correspondence between address input and data output of array "ary", it is necessary to describe array "k" similar to address input in the subscript of data output of array "ary". This description is just used for making correspondence of the partition description of "ary", and "k" in the subscript does not become a target of the synthesis.

11.6.2 Asynchronous read and synchronous write memory

In asynchronous read and synchronous write memory, asynchronous read data access is possible and data can be read from memory in the clock set by read address. Timing of write access is same as timing of pipeline memory and data can be written in memory with clock next to the clock set as write address.

In FPGA, resource can be used effectively as mapping to the LUT memory is possible.

11.6.2.1 Specification method of asynchronous read and synchronous write memory

Option	Description
-mrawe	Array is scheduled as asynchronous read synchronous write memory.

Attribute	Description	Settings Target
/* Cyber rw_timing = rawe */	Array is scheduled as asynchronous read synchronous write memory.	Array

Below specifications are required to allocate the array as asynchronous read synchronous write memory.

- 1) Set attribute (rw_timing=rawe) in the array or, you can also specify -mrawe as synthesis option.

```
mem(0:16) ary[32] /* Cyber rw_timing = rawe */;
```

- 2) Synchronous write (WRITE_SYNC YES) and absolute delay (or, specifying the cycle) must be specified in memory library file. (see **39.10.**for specification of memory library file)

```
#@VERSION { 1.00 }
#@LIB { main }
@MLIB {
    NAME MEMB8W32
    KIND R1,W1
    BITWIDTH 8
    DELAY 150
    WORD 32
    MEM_SYNTH LUT
    WRITE_SYNC YES
    DEFMOD {
        in ter (0:5) RA1 /* ra:1 */;
        out ter (0:8) RD1 /* rd:1 */;
        in ter (0:5) WA2 /* wa:2 */;
        in ter (0:8) WD2 /* wd:2, setup=200 */;
        in ter (0:1) WE2 /* we:2 */;
        in ter (0:1) WCLK2 /* wclk:2 */;
    }
}
#@END { main }
```

11.6.3 Asynchronous read asynchronous write memory

11.6.3.1 Specification of memory access timing of manual scheduling

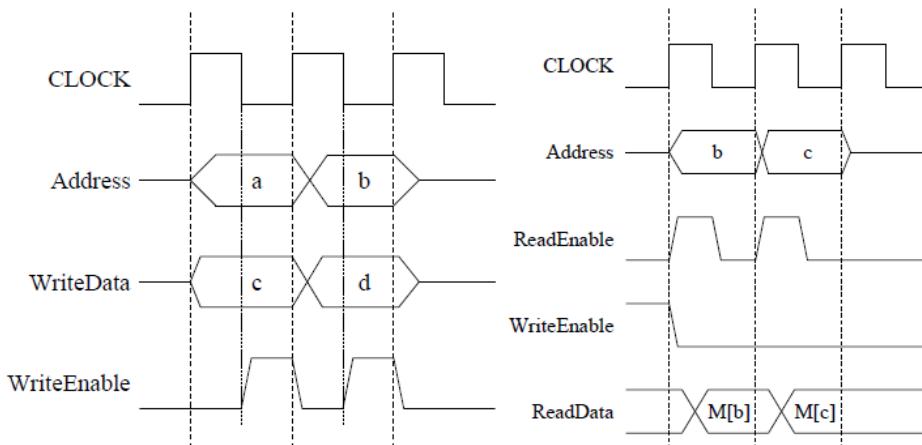
In manual scheduling, circuit is synthesized as memory of one cycle. Memory of multiple cycles must be described explicitly. (See section **26.3**)

By default, it is synthesized as memory of asynchronous SRAM type and enable signal becomes active as shown below.

Data and address assigning to memory till later half of cycle are fixed when assigning to the memory. And a circuit is generated where write enable signal during the second half of cycle becomes active.

Memory is referred in first half of the cycle hence, circuit is generated where read enable signal during first half of cycle becomes active.

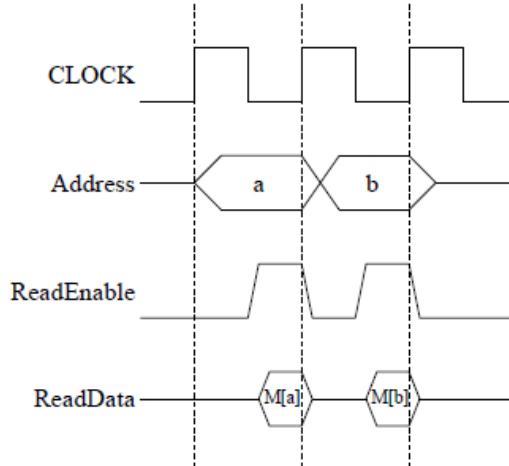
Therefore, in manual scheduling, any delay in time in address computation and allocating data calculation must be adjusted in half cycle.



If you do not activate read enable signal, in case of asynchronous SRAM type memory then read data is not output. It can be done by adding the circuit which maintains read data in later half of the cycle to the output port of memory or, by connecting write enable signal to the read enable port.

Read enable signal can be activated in later half of the cycle as shown below. It does take time for address computation but, take care of the delay of calculation using referred data.

Moreover, write enable can be activated in first half of the cycle but, it is of no significance.



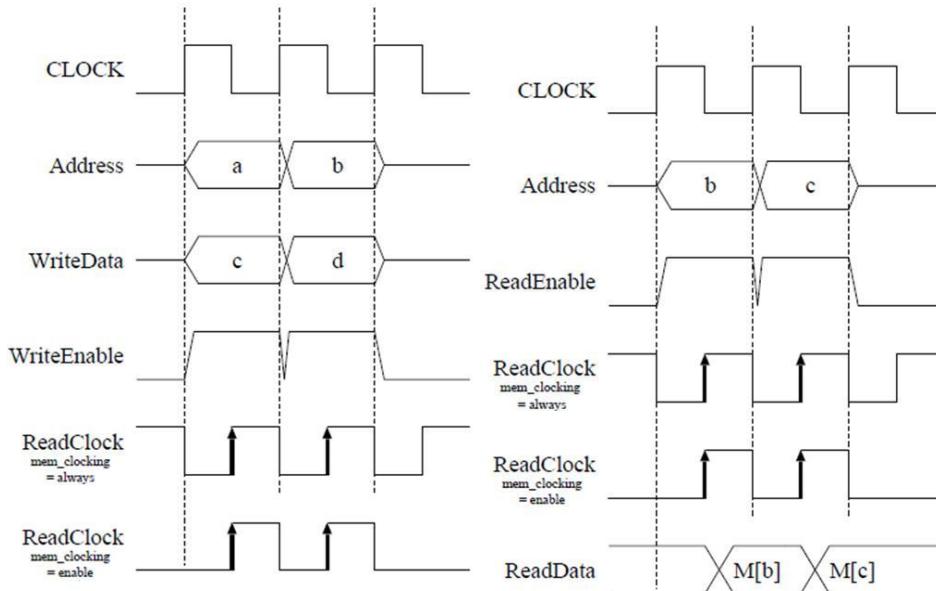
There are option, attribute which generates enable signal and clock signals as memory of synchronous SRAM type of 1 cycle. Clock port information is specified in memory library file (MLIB) and it is automatically assumed as memory of synchronized SRAM type in case of non-pipeline memory.

In order to use this option and attribute, in synthesis option both read enable signal and write enable signal must be set to get activated in second half of the cycle.

Data and address assigning to memory in first half of cycle are fixed when assigning to the memory. Write enable signal becomes active for one cycle and circuit is generated where memory clock signal starts in start of the second half of the cycle.

In case of `-Zmem_clocking=enable` option and `mem_clocking=enable` attribute, it is synthesized as memory clock signal starts up in the usage step apart from the referring and assigning.

In case of `-Zmem_clocking=always` option and `mem_clocking=always` attribute, it is synthesized that always memory clock signal starts up apart from the referring and assigning.



11.6.3.2 Specification of memory access timing of automatic scheduling

In automatic scheduling, memory that operates with multi cycle is explained.

First of all, we explain about operation in one cycle. Operation is considered to be of one cycle when delay of the memory specified in memory constraint file is within one cycle.

In automatic scheduling, similar to manual scheduling when type is not specified, memory is synthesized as memory of asynchronized SRAM type. At the time of allocating to the memory, data and address are scheduled as fixed till the second half of the cycle and write enable signal becomes active during second half of the cycle. Memory reference is executed in first half of the cycle hence cycle is generated where read enable signal will be active in first half of the cycle.

By default in automatic scheduling, data is read in first half of the cycle at the time of reference. It is schedule in such way that only the small delay operation such as & can do address calculation in the same cycle. (**Figure 31**)

However, if it is identified that delay is not managed even if address calculation is done in the same cycle, you can specify such that address computation using small delay calculation by option -mrfw? and attribute rw_timing = rlw?, is not executed in the same cycle. (**Figure 32**)

You can specify with option mrl0w? and attribute rw_timng = rl0w? to read the data in second half of the cycle. And for small delay operation such as &, referred data is scheduled to be calculate in the same cycle. (**Figure 33**)

However, if it is identified that delay is not managed even if referred data is used for calculation in the same cycle, you can specify that calculation where referred data is used in small delay calculation is not executed in the same cycle. (**Figure 34**)

You can specify the same at the time when allocating to the memory.

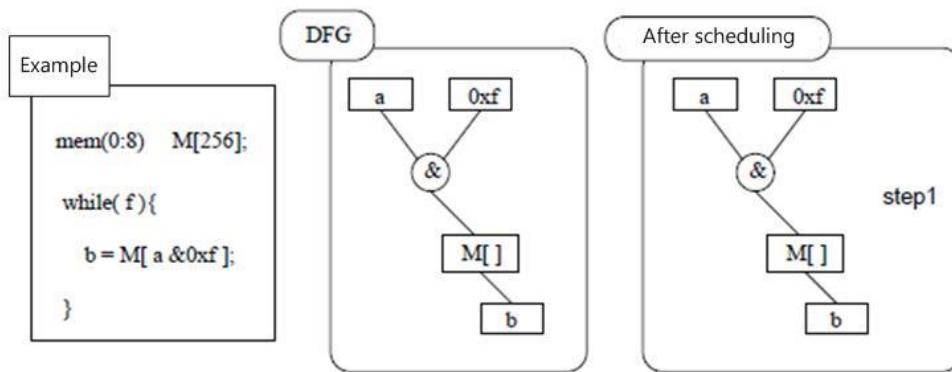
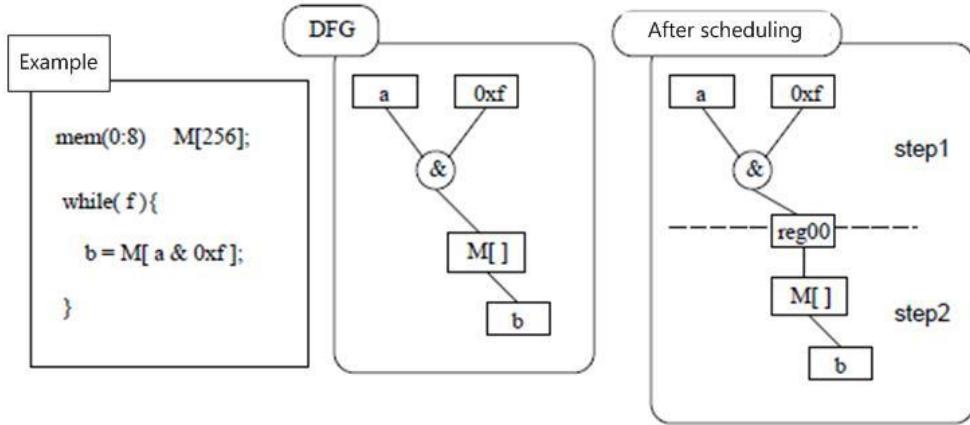
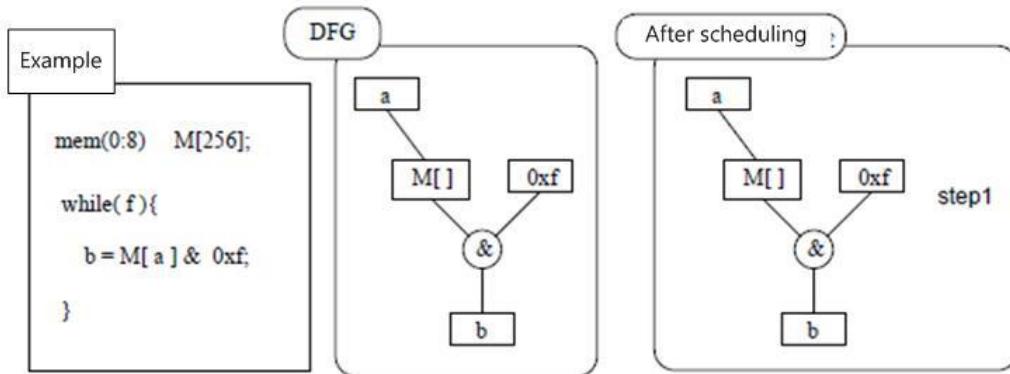
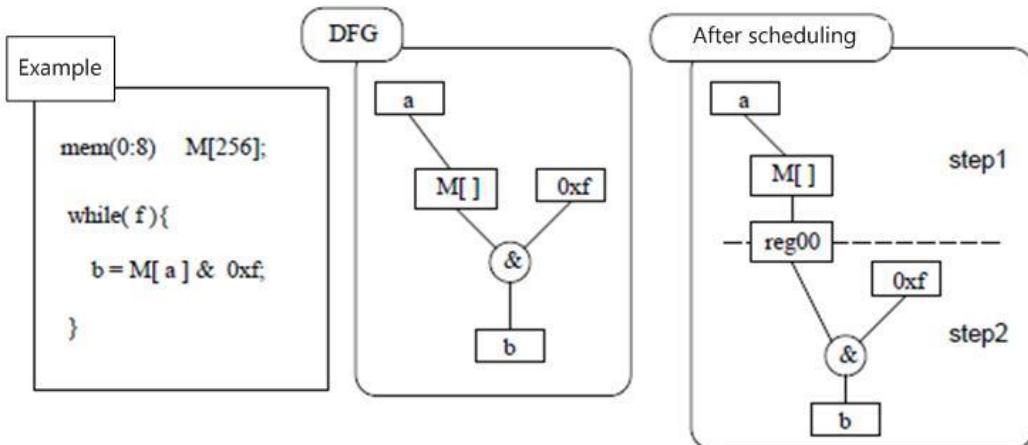


Figure 31: Scheduling when memory is referred (rf0w?)

**Figure 32:** Scheduling when memory is referred (rfw?)**Figure 33:** Scheduling when memory is referred (rl0w?)**Figure 34:** Scheduling when memory is referred (rlw?)

You can specify the type of memory at the time of synthesis. When using synchronous SRAM type memory, reference and allocation to the memory are executed in second half of the cycle same as in case of manual scheduling. You can specify -Zmem_clocking option and mem_clocking attribute.

Next, it explains about multi cycle memory.

If clock cycle is smaller than the delay value of memory specified in memory constraint file, it is synthesized as multiple cycle memory. For instance, when delay value 5000 (50ns) is specified in memory constraint file and memory is processed at 2000 (20ns) clock cycle, it will be synthesized as memory processed in 3 cycles.

In this case, write and read enable signal will be active in half of the operating cycle. However, fraction cycle is omitted. In other words, in case of memory which operates in 3 cycle, enable signal of one cycle will be active.

mem.MLIB

```
#@VERSION { 1.00 }
#@LIB { entry }
@MLIB {
    NAME mem08A
    KIND RW1
    BITWIDTH 8
    DELAY 5000
    WORD 200
    ...
}
@MLIB {
    NAME mem08B
    KIND R1
    BITWIDTH 8
    DELAY 5000
    WORD 200
    ...
}
#@END { entry }
```

mem.MCNT

```
#@VERSION { 1.00 }
#@CNT { entry }
@MCNT {
    NAME mem08A
    LIMIT 1
}
@MCNT {
    NAME mem08B
    LIMIT 1
}
#@END { entry }
```

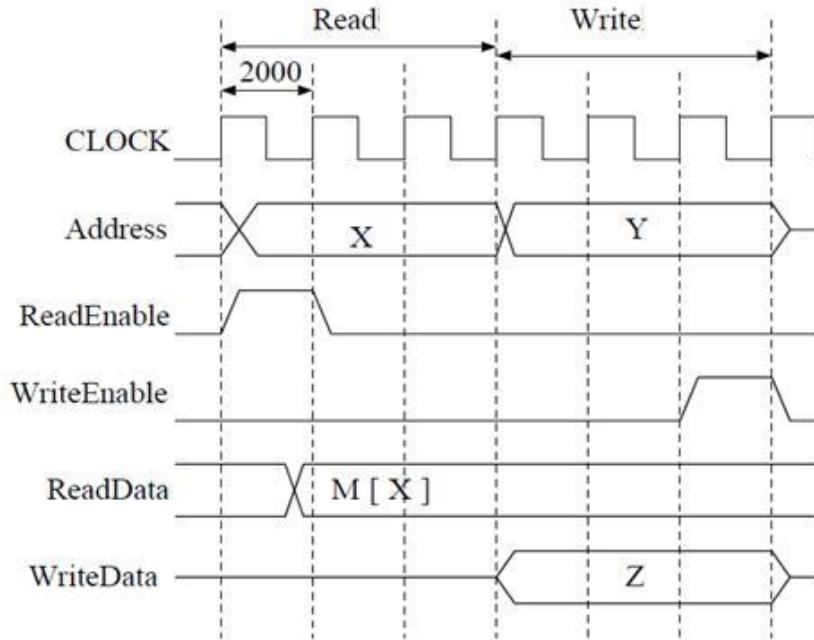


Figure 35: Timings of multi cycle memory

11.7 Insert Register in Pipeline Memory Port

Option	Description
-Zmem_reg=none	Do not insert register in Input/Output side access of Pipeline memory (Default).
-Zmem_reg=in	Insert register in Input side access of Pipeline memory.
-Zmem_reg=out	Insert register in Output side access of Pipeline memory.
-Zmem_reg=in:out	Insert register in Input/Output side access of Pipeline memory.
-Zoutside_mem_reg=outside	Synthesize register of Input/Output side of external memory as external (Default).
-Zoutside_mem_reg=inside	Insert Input/Output side register of external memory as internal.

Attribute	Description	Settings Target
/* Cyber mem_reg = none */	Do not insert register in Input/Output side access of Pipeline memory (Default).	Array variable
/* Cyber mem_reg = in */	Insert register in Input side access of Pipeline memory.	Array variable
/* Cyber mem_reg = out */	Insert register in Output side access of Pipeline memory.	Array variable
/* Cyber mem_reg = in:out */	Insert register in Input/Output side access of Pipeline memory.	Array variable

Insert register before/after memory access as a counter measure in case delay around memory is strict (**Figure 36**). There are 3 options available to insert the register. These are Input only, Output only & Input-Output.

In case Auto Scheduling Mode or Auto Pipeline Mode is used then there is no need to modify the behavioral description because access timing is adjusted automatically.

In case Manual Scheduling Mode used then it is necessary to specify the timing description assuming that register is inserted. For example, in case of using memory of x2 delay, the description must be specified as following.

Before modification :

```
mem chr x[128];

x[adr] ; /* Address Input Expression */
$

dat = x[adr]; /* Data output Expression */
/* Delay is x2 therefore data is output
   with clock after Address Input Expression.
```

After modification :

```
mem chr x[128] /* cyber mem_reg = in:out */;

x[adr] ; /* Address Input Expression */
$
$
$

dat = x[adr]; /* Data output Expression */
/* As register is inserted in both input side, output side,
   data is output after 3 clock*/
```

And the behavior at the time of specifying option would be different depending upon how memory entity implements (Internal memory, External memory, Shared memory).

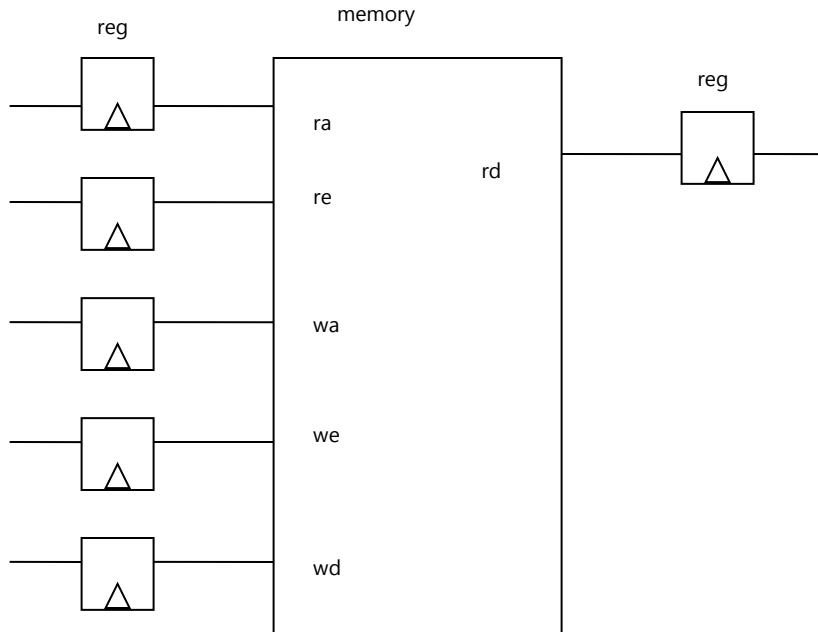
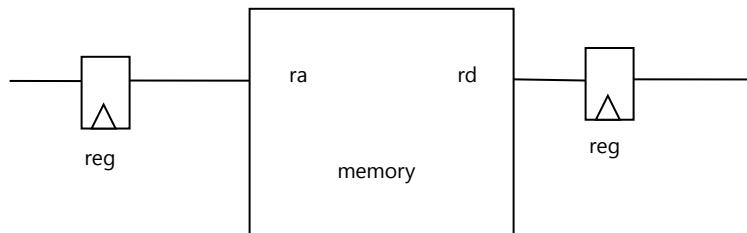
- In case memory is implemented as internal memory.

The memory entity undergoes synthesis and register are inserted before/after memory access as per attribute and option (**Figure 37**).

- In case memory is implemented as external memory or shared memory.

The memory entity does not undergo synthesis and synthesis with timing assuming that register is included. Hence, register is not created during synthesis stage (**Figure 38**).

In case register is to be created during synthesis (**Figure 39**) then option "-Zoutside_mem_reg=inside" must be specified.

**Figure 36:** Insert Register in Memory Port**Figure 37:** Insert Register in Internal Memory

11.7.1 Note

- In case `clock_sig/reset_sig` is specified in `memory` signal then inserted register would also be driven with that clock & reset.
- Memory with register inserted before/after memory access and memory without register insertion are not allocated in same type of memory.

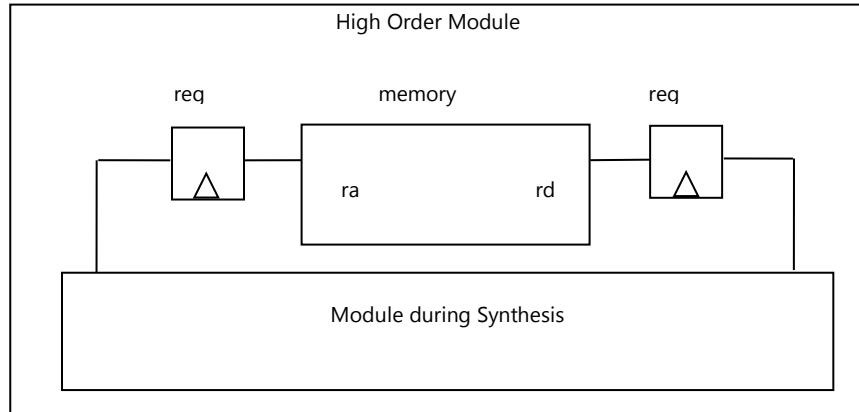


Figure 38: Insert register in high order (-Zoutside_mem_reg=outside)

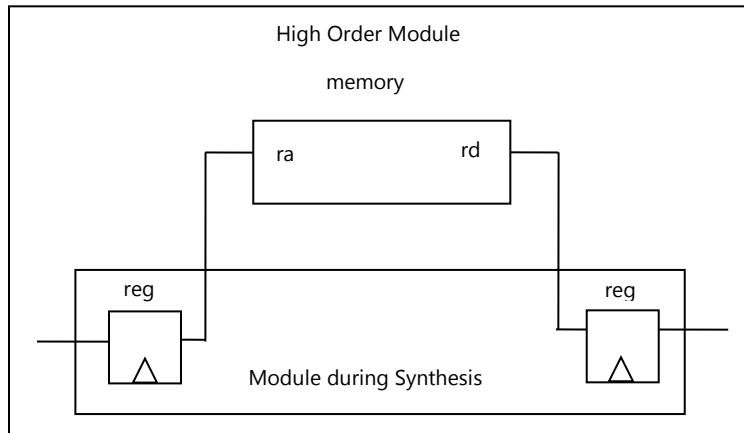


Figure 39: Insert register in low order (-Zoutside_mem_reg=inside)

11.8 Synthesis function for shared memory without read enable and shared register array

11.8.1 Overview

In this section, Synthesis function of shared memory without read enable and shared register array are explained.

11.8.2 Related option

Option	Description
-Zshared_mem_re_gen [=YES]	Read enable signal is generated that controls the multiplexers address of shared memory. (default)
-Zshared_mem_re_gen =NO	Read enable signal is not generated that controls the multiplexer address of shared memory (memory without read enable and chip select).
-Zshared_reg_lutram_re_gen [=YES] -Zshared_reg_lutram_re_gen =NO	Read enable signal is generated that controls multiplexer address of shared register array (LUT memory format). Read enable signal is not generated that controls multiplexer address of shared register array (LUT memory format). (default)

11.8.3 Description

In case read enable is not included in shared memory and shared register array having read/write dual use port, exclusive access control is necessary when access is done in single port from multiple processes in top module.

If the option “-Zshared_mem_re_gen” is specified in case of shared memory, port corresponding to read enable is auto generated when array is referred in module, declared as shared (option “-Zshared_reg_lutram_re_gen” is specified in case of shared register array). Based on the port generated automatically, exclusive access control of port access is carried out in top module.

In the following example, access to memory of read/write dual use port from 2 modules is carried out. Reading of memory is done in module 1 and writing to memory is done in module 2. If this option is enabled in synthesis of module 1, port (rel_auto in **Figure 40**) corresponding to read enable is auto generated. In synthesis of top module, exclusive access control of logic address is carried out by using the port corresponding to read enable generated automatically.

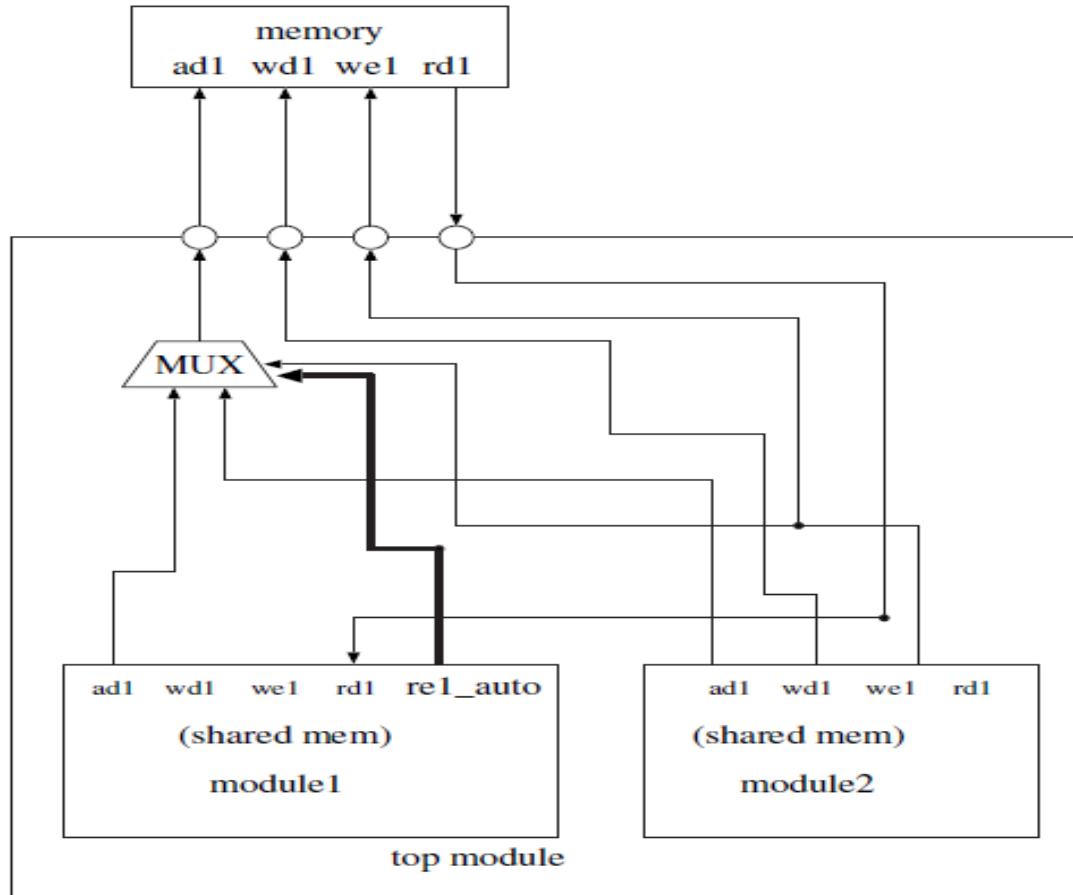


Figure 40 : Connection diagram of auto generated read equivalent port

11.9 Connection specification of control signal, reset, clock of memory

Clock, reset of circuit are connected to the clock port, reset port of the non shared memory module. On the other hand,, clock reset of circuit clock of same hierarchy where writing is executed, can be specified in reset port/ clock port of the shared memory. However, if -Zset_lower_clock_reset=NO is specied, clock, reset of module declaring shared memory, will be connected.

Option	Description
-Zset_lower_clock_reset=ALL	Sets reset clock of lower clock reset which carries out writing, to the clock reset signal of register and shared memory. (Default)
-Zset_lower_clock_reset=SHARED_MEM	Sets same reset clock of lower clock reset carries out writing, to the clock reset signal of shared memory. Sets clock reset signal of current level to the clock reset of registered and shared memory.
-Zset_lower_clock_reset=NO	

You can specify port of the control signal by setting port type as "Other" in "Port information (DEFMOD)of memory library file (MLIB)" as control signal of the memory module. Inactive value is connected to the input "Other" port of the memory module. Constant value 0 and 1 is connected when polarity is HIGH and LOW ctive respectively. Output other port of memory module will bot be connected to any of them.

If there are multiple clocks in hierarchical design, clock other than the circuit clock can be specified in clock port of the memory. In this case, if clock_sig attribute is specified to the array assigned in memory module, signal specified with attribute will be connected with clock port of the memory.

There are cases when signal for each clock port is desired to be connected in case of multiple clock ports in the memory module. And External input port is desired to be connected if control port used for testing in the memory module.

As mentioned above, if connection of clock port, reset port and control port of the memory module is desired to specified seperately, they can be specified through port connection (PORTCONNECT) in memory count file (MCNT).

In port connection (PORTCONNECT), one connection information is specified in one line. Connection information consist connection variable (first field) and connection expression (second field). It will be connected, if signal of under operation description and port of memory module is specified.

At this time, it is necessary to put '@' before the port name in order to differentiate from signal of operation description in the port of memory module.

Apart from signal constant value 0 or 1 and expression having operation &, |, ^, ~ can be specified in the connection expression(second field).

Example of connection of external input and output with control port of memory by combining separate input to each clock port of dual port memory is given below.

Description

```

in ter(0..0) t;
out ter(0..0) s;

in ter(0..0) clkA;
in ter(0..0) clkB;

in ter(7..0) a;
out reg(7..0) b;

mem(7..0) M[256];

defmod modA {
    in ter(7..0) i1;
    out ter(7..0) o1;
    shared mem(7..0) memA[256];
} instA;
process test()
{
    allstates{
        assign(M,instA.memA);
    }
    instA.i1 ::= a;
    b ::= instA.o1;
}

```

Memory library file (MLIB)

```

#@VERSION { 1.00 }
#@LIB { test }
@MLIB {
    NAME MEMB8W256
    KIND R1,W1
    BITWIDTH 8
    DELAY x2
    WORD 256
    DEFMOD {
        in ter (7..0) RA1 /* ra:1 */;
        out ter (7..0) RD1 /* rd:1 */;
        in ter (0..0) RE1 /* re:1 */;
        in ter (0..0) RCLK1 /* rclk:1 */;
        in ter (7..0) WA2 /* wa:2 */;
        in ter (7..0) WD2 /* wd:2 */;
        in ter (0..0) WE2 /* we:2 */;
        in ter (0..0) WCLK2 /* wclk:2 */;
        in ter (0..0) TEST /* other */;
        out ter (0..0) STATUS /* other */;
    }
}
#@END { test }

```

Memory count file(MCNT)

```

#@VERSION { 1.00 }
#@CNT { test }
@MCNT {
    NAME MEMB8W256
    INSTANCE      M
    RESOURCE      MEM

    PORTCONNECT {
        @RCLK1      clkA
        @WCLK2      clkB
        @TEST       t
        @STATUS     s
    }
}
#@END { test }

```

Limitations:

- Port connection specification (PORTCONNECT) is not possible other than memory used for defmod abstract interface as it is not supported.
- Signal, array and structure variables other than I/O variables cannot be specified as it is not supported.
- Port and signal of poly bit cannot be specified as it is not supported.

11.10 Decoder enabled Register Array - Detailed Specification

In this section the following 4 items shall be described that can be specified while an array is synthesized as a decoder enabled register array:

- Specification of the number of decoders of decoder enabled register array
- Specification of the decoder number used in reference or assignment of decoder enabled register array.
- Specification of decode delay of decoder enabled register array.
- Specification of the function of dividing the decoder of decoder enabled register array into small decoder and multiplexer.

11.10.1 Specification of the Maximum Number of Decoders

Option	Description
-au	Does not limit the no. of decoder enabled register array at the time of manual scheduling (default).
-al	Limits the no. of decoders of decoder enabled register array at the time of manual scheduling.

Attribute	Description	Settings Target
/*Cyber rw_port= RW# */	Specifies the maximum number of common decoders for reference & assignment of decoder enabled register array in #.	Array variable
/*Cyber rw_port== R#.W# */	Specifies the maximum number of reference decoders and assignment decoders of decoder enabled register array respectively in #.	Array variable
/*Cyber rw_port== R# */	Specifies the maximum number of reference decoder of decoder enabled register array in #.	Array variable
/*Cyber rw_port== W# */	Specifies the maximum number of assignment decoder of decoder enabled register array in #.	Array variable

This subsection describes about the maximum number of decoders used in register array provided with decoder. Depending on the number of decoders, the number of reference or assignment within one cycle from / to a decoder enabled register array is decided. (However, the array referencing or assignment with a constant subscript is not included.)

In manual scheduling, as the number of decoders required for each step is clear in description, synthesis is performed by providing required number of decoders. If the number of decoder has to be limited, the “-al” option should be specified. In addition, the maximum number of decoders in the array should be specified by using the “rw_port” attribute. In case the number of decoders are not enough then there is an allocation error. When option -al is specified but the number of decoders has not been specified in the “rw_port” attribute, then the number of decoders that can be used in both referencing and assignment is restricted to one decoder.

In automatic scheduling mode, Scheduling will be done based on the number of decoders specified by rw_port attribute which is taken as maximum. If the specification by rw_port attribute does not exist, then one decoder for reference and one decoder for assignment are assumed as maximum during synthesis. However in case of loop folding & auto pipeline scheduling, synthesis is done considering the number of reference and assignment respectively within the loop as maximum.

There are four methods of specifying maximum number of decoders using the “rw_port” attribute.

- The first method is to specify the maximum number of decoders which can be used in both reference and assignment, specified as “RW2” in the attribute value.
- Second method is to specify maximum number of decoders which can be used either in reference only or assignment only, specified as “R2.W1” in the attribute value.
- Third method is to specify the maximum number of decoders which are only used in reference, specified as “R2” in the attribute value.
- Last method is to specify the maximum number of decoders which are only used in assignment, specified as “W1” in the attribute value.

Example:

```
reg(0:8) x[10]/* Cyber rw_port = RW2 */;
reg(0:8) y[10]/* Cyber rw_port = R2.W1 */;
reg(0:8) r[10]/* Cyber rw_port = R2 */;
reg(0:8) w[10]/* Cyber rw_port = W1 */;
```

11.10.2 Decoder Number Specification

Attribute	Description	Settings Target
/*Cyber sig2dec_port = r#w#*/	Specifies the default decoder number used in the array reference or assignment of decoder enabled register array	Array variable
/*Cyber dec_port=#*/	Specifies the decoder number used in the array reference or assignment of register array provided with decoder individually.	Array reference, array assignment

This section will explain about the specification method of decoder used in the reference or assignment of the register array provided with decoder.

If nothing is specified, decoders used in each reference or assignments are automatically decided. However, user can directly specify the decoder only among which the `sig2dec_port` attribute is specified.

Following are the 4 types of methods of description in the "sig2dec_port" attribute.

- (a) The first method is shown as (a). This method is for specifying both, the decoder number for reference and decoder number meant for assignment.
- (b) The second method is shown as (b). This method is used to specify the decoder number only for reference.
- (c) The third method is shown as (c). This method is used to specify the decoder number only for assignment.
- (d) The fourth method is shown as (d) not to specify anything.

Example:

```
reg(0:8) w[10]/* Cyber rw_port = RW2,sig2dec_port = r1w2 */; /* (a) */
reg(0:8) x[10]/* Cyber rw_port = RW2,sig2dec_port = r1 */; /* (b) */
reg(0:8) y[10]/* Cyber rw_port = RW2,sig2dec_port = w2 */; /* (c) */
reg(0:8) z[10]/* Cyber rw_port = RW2,sig2dec_port */; /* (d) */
```

The "dec_port" attribute is used to specify a decoder which will be used in each array reference or assignment by specifying to them. While using the "dec_port" attribute, it is necessary that the "sig2dec_port" attribute has been specified in the array signal of the "dec_port" attribute.

As a value of "dec_port" attribute, the decoder number is specified. Decoder number starts with 1. In case the decoder for reference and decoder for assignment are specified separately, such as

/*Cyber rw_port= R1.W2*/", numbering is started from the decoder used for referencing. The decoder used for referencing becomes decoder number 1. Subsequently, the decoder used for assignment becomes decoder number 2 and then decoder number 3.

In the following example, register array "foo" has been specified to have 3 decoders. The assignment to foo[a] uses the third decoder in accordance with the dec_port attribute. The assignment "foo[b]" doesn't have any specification. Therefore, it is decided automatically. The reference of foo[e] uses the third decoder in accordance with the dec_port attribute. The reference of foo[g] follows the sig2dec_port attribute and uses the first decoder.

Example:

```
reg(0:8) foo[10]/* Cyber rw_port = RW3,sig2dec_port = r1 */;
foo[a] /* Cyber dec_port = 3 */ = b;
foo[b] = c;
d = foo[e] /* Cyber dec_port = 3 */;
f = foo[g];
```

Limitations:

- In case of specifying the "sig2dec_port" attribute, if the number of decoders is not specified by the "rw_port" attribute, then synthesis will be performed with one decoder. In other words, in the array variable where the "sig2dec_port" attribute is specified, the "-au" option is not valid.
- Currently, in the automatic scheduling, there are cases such that decoders cannot be allocated according to the specification and results in an error.

11.10.3 Decode Delay of decoder enabled register array - Specification

This section will explain about the delay value of decoder of decoder enabled register array.

The delay value of decoder of decoder enabled register array is decided on the basis of delay value of Basic Library File (refer **section 39.6**). When the delay value of decoder is not specified, it is treated as short delay value. It can be explicitly specified by adding the "rw_delay" attribute to the array variable as shown below.

A positive delay value should be specified as a delay value. The unit of delay can be specified as ns (nano second), ps (pico second). If the unit is not specified, this attribute follows the time unit specified by –cu option (**Refer to section 8**).

Attribute	Description	Settings Target
/*Cyber rw_delay= #*/	Converts the delay of register array provided with decoder into #.	Array variable

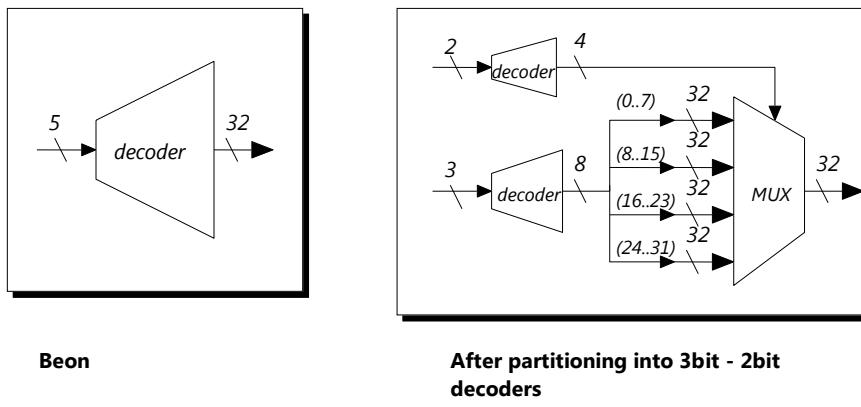
Example:

```
reg(0:8) x[10]/* Cyber rw_delay = 100 */;
reg(0:8) y[10]/* Cyber rw_delay = 1ns */;
reg(0:8) z[10]/* Cyber rw_delay = 1000ps
```

11.10.4 Partitions of the Decoder Register Array - Function Specification

This section will explain about the partition of decoder of register array is provided with decoder. Dividing a decoder means partitioning a big decoder into small decoders and multiplexers.

In the example provided below, a decoder of 5-bit input and 32-bit output can be divided into a decoder of 3-bit input and 8-bit output, decoder of 2-bit input and 4-bit output, and a multiplexer of 32-bit 4 way.



Attribute	Description	Settings Target
/*cyber decoder_div=#:#[:#...]*/	Partition the decoder of register array provided with decoder into small decoders and multiplexer.	Array variable

In order to specify this by an attribute "decoder_div", an input of the decoder should be specified in the following sequence.

At this time, it is necessary to specify the small decoders in a manner so that the sum total comes out to be equal to the input bit width of the original decoder.

Example:

```
reg(0:8) x[32] /*Cyber_decoder_div=3:2*/;
```

Option	Description
-Zdecoder_div	Divide the decoder of decoder enabled register array into small decoders and multiplexer (average method/mode: 4 bit unit and input more than 8 bit).
-Zdecoder_div= no	Does not divide the decoder of register array (default).
-Zdecoder_div=ave	Divides the decoder of register array by average method.
-Zdecoder_div=pre	Divides the decoder of register array by left justified method.
-Zdecoder_div_unit=#	Divides the decoder of register array by# bit unit.
Zdecoder_div_over=#	Divides the decoder whose input bit is more than #bit.

Also, above option can be used to specify partitioning of array for whole circuit. As a method of division, the left justified and average method is used.

In the left justified method, the decoder is divided sequentially by bit unit specified in the "-Zdecoder_div_unit=#".

For example, when 8-bit input and 256-bit output decoder is divided by 5-bit unit, the decoder is divided in pieces of 5-bits each from the front/beginning. The decoder is finally divided into decoders of 5-bit input and 32-bit output and decoders of 3-bit input and 8-bit output.

In the average method, once the number of divisions have been decided after dividing a decoder by bit unit specified in the "-Zdecoder_div_unit=#" option, the decoder is divided in such a way that the bit width becomes average by the number of divisions.

For example, when 8-bit input 256-bit output decoder is divided by 5-bit unit, the number of divisions becomes 2.

Then, 8bits are equally divided into two, i.e. 4-bit input 16-bit output decoder.

11.11 Universal Memory Interface Specification

11.11.1 Overview

The port to connect the memory is generated in accordance with the port information specified by the memory library (MLIB). This functionality enables connection with all types of memory I/F or Chip Device by adding a Handshake port in memory access.

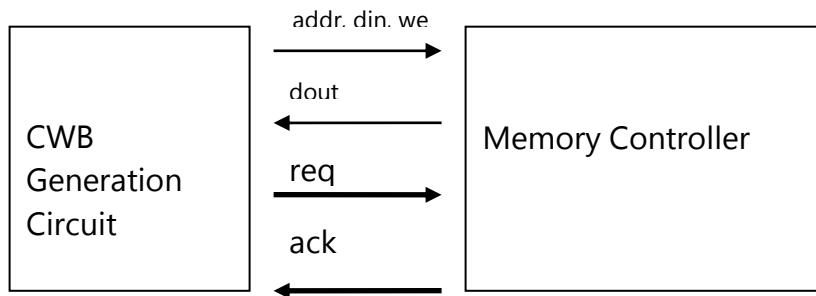


Figure 41: Universal Memory Interface Connection Image

11.11.2 Related Attributes

Attribute	Description	Settings Target
/* Cyber univ_mem_if = create */	Generates Universal Memory Interface in memory.	Array variable
/* Cyber univ_mem_if = ignore */	Does not generate Universal Memory Interface in memory.	Array variable

11.11.3 Port types of MLIB

MLIB can be used as Universal Memory Interface by adding Request and Acknowledge Port types.

It is necessary to specify both Request and Acknowledge for each port in port specification. Port types supported by each type of memory are given below.

	RWn type memory	Rn,Wn type memory	
	read/write port	read port	write port
Request port	req	rreq	wreq
Acknowledge Port	ack	rack	wack

Examples of MLIB file are given below.

- **RWn type memory**

```
@MLIB {
    # mem08A
    NAME      mem08A
    KIND      RW1
    BITWIDTH  8
    WORD      4
    DELAY     x2
    DEFMOD {
        # i/o      t/r bitw   name /* kind:port,pol */ ;
        in       ter (1..0) A    /* ad:1 */;
        out      ter (7..0) DO   /* rd:1 */;
        in       ter (7..0) DI   /* wd:1 */;
        in       ter (0..0) WEB  /* we:1,neg */;
        in       ter (0..0) CSB  /* cs:1,neg */;
        in       ter (0..0) BE   /* clk:1 */;
        in       ter (0..0) REQ  /* req:1 */;
        out      ter (0..0) ACK  /* ack:1 */;
    }
}
```

- **Rn,Wn type memory**

```
@MLIB {
    # mem08A
    NAME mem08A
    KIND R1,W1
    BITWIDTH 8
    WORD 4
    DELAY x2
    DEFMOD {
        # i/o      t/r bitw   name /* kind:port,pol */ ;
        in       ter (1..0) AA   /* wa:2 */;
        in       ter (7..0) DI   /* wd:2 */;
        in       ter (0..0) CSA  /* ws:2,neg */;
        in       ter (0..0) BEA  /* wclk:2 */;
        in       ter (0..0) RST2 /* wrst:2 */;
        in       ter (0..0) REQ2 /* wreq:2 */;
        out      ter (0..0) ACK2 /* wack:2 */;

        in       ter (1..0) AB   /* ra:1 */;
        out      ter (7..0) DO   /* rd:1 */;
        in       ter (0..0) CSB  /* rs:1,neg */;
        in       ter (0..0) BEB  /* rclk:1 */;
        in       ter (0..0) RST1 /* rrst:1 */;
        in       ter (0..0) REQ1 /* rreq:1 */;
        out      ter (0..0) ACK1 /* rack:1 */;
    }
}
```

11.11.4 Memory Read Timing

- CWB Generation Circuit Process (behavior)

Process explanation is given along with example of wave form in **Figure 42**. Numbers mentioned below correspond to numbers given in wave form.

- 1) Activate the Request signal in Read Access Timing Cycle.
- 2) Output value such as active Address signal, Enable signal etc along with request signal.
- 3) Request signal would remain in active state till Acknowledge signal is activated.
- 4) Address signal & Enable signal would remain inactive after activation of latest Request signal till activation of Acknowledge signal.
- 5) CWB Generation Circuit would be stalled till Acknowledge signal is activated.

- Memory Controller Process

Memory controller is expected to behave as given below in Memory Read Cycle.

- Obtain the value (address value etc.) provided by CWB Generation Circuit when latest Request signal is activated, and retain it till Acknowledge signal is activated.
- Return Read data to CWB Synthesis Circuit on timing corresponding to Memory Latency while Acknowledge Signal is active. For example, in case Memory Latency is x3 then it is expected that Read data would be returned next to next cycle (after 2 cycles) when Acknowledge is activated in Memory Controller.

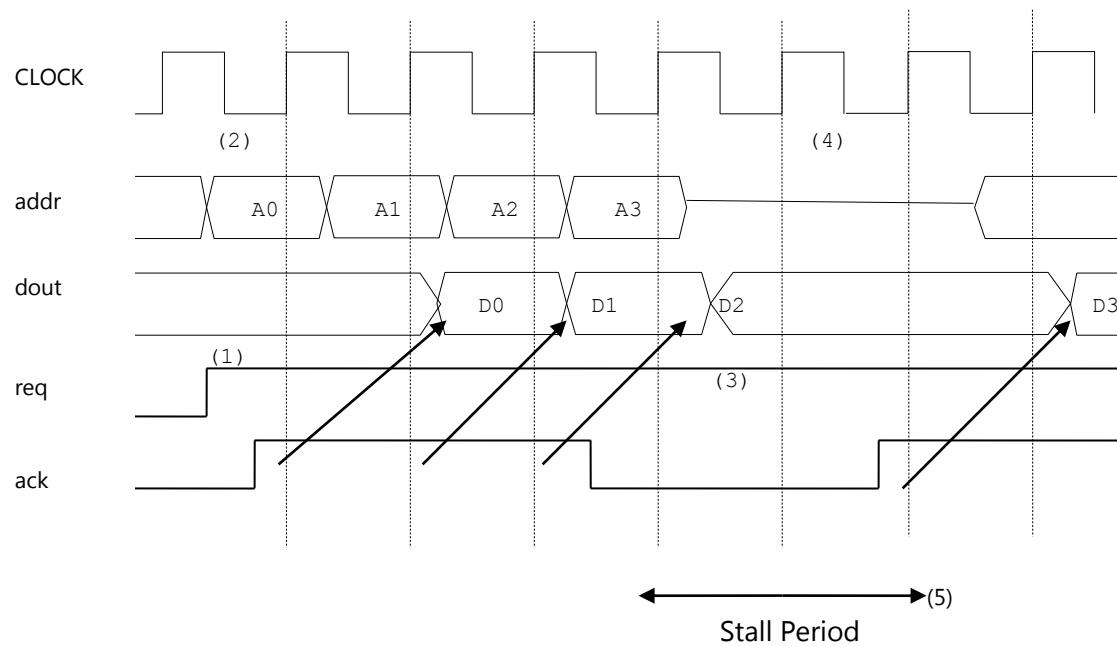


Figure 42: Memory Read Timing Waveform (x3 Memory)

11.11.5 Memory Write Timing

- CWB Generation Circuit Process

Process explanation is given along with example of wave form in **Figure 43**. Numbers mentioned below correspond to numbers given in wave form.

- 1) Activate the Request signal in Write Access Timing Cycle.
- 2) Output value such as active Address signal, Enable signal etc along with request signal.
- 3) Request signal would remain in active state till Acknowledge signal is activated.
- 4) Address signal, Enable signal & Data signal would remain inactive after activation of latest Request signal till activation of Acknowledge signal.
- 5) CWB Generation Circuit would be stalled till Acknowledge signal is activated.

- Memory Controller Process

Memory controller is expected to behave as given below in Memory Write Cycle.

- Obtain the value (address value etc.) provided by CWB Generation Circuit when latest Request signal is activated, and retain it till Acknowledge signal is activated.
- Activate the Acknowledge signal on Write Complete Cycle in Memory.

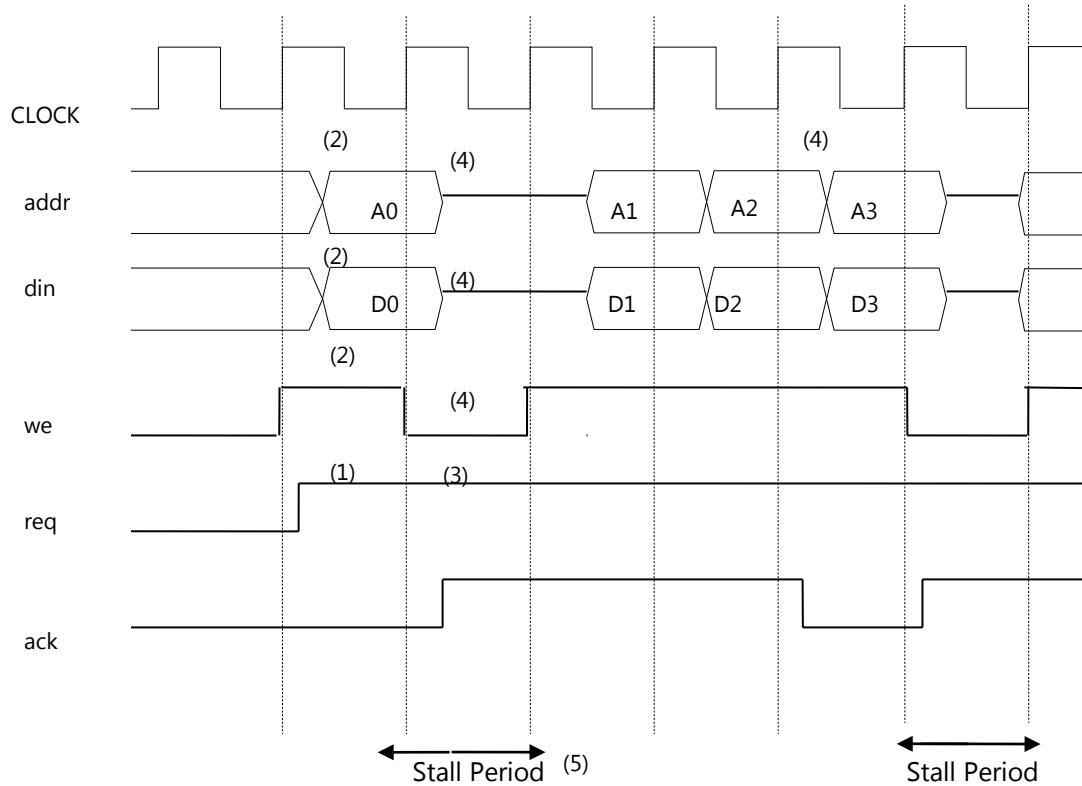


Figure 43: Memory Write Timing Waveform

11.11.6 Behavioral Specifications on using Stall Control

- When CWB Generation Circuit process is stalled by stall control input then Request signal is not activated and Acknowledge signal is not received.
- Therefore when Request signal is inactive in memory controller then Acknowledge signal must not be activated.
CWB Generation Circuit cannot receive Acknowledge signal because stall waiting for Acknowledge signal cannot be deleted.

11.11.7 Example of description by BDL of Memory Controller

Example of description of Memory Controller using RW1 Memory is given below.

In this example, when memory is accessed by one module out of two modules (module1, module2) then controller is adjusted to give preference to module1.

```
/* Access from module1 */
out ter(0..0) ACK_1 /* ack:1 */;
out ter(7..0) DO_1 /* rd:1 */;

in ter(0..0) REQ_1 /* req:1 */;
in ter(7..0) A_1 /* ad:1 */;
in ter(7..0) DI_1 /* wd:1 */;
in ter(0..0) WEB_1 /* we:1 */;
in ter(0..0) CSB_1 /* cs:1 */;
in ter(0..0) BE_1 /* clk:1 */;
in ter(0..0) RST_1 /* rst:1 */;

/* Access from module2 */
out ter(0..0) ACK_2 /* ack:1 */;
out ter(7..0) DO_2 /* rd:1 */;

in ter(0..0) REQ_2 /* req:1 */;
in ter(7..0) A_2 /* ad:1 */;
in ter(7..0) DI_2 /* wd:1 */;
in ter(0..0) WEB_2 /* we:1 */;
in ter(0..0) CSB_2 /* cs:1 */;
in ter(0..0) BE_2 /* clk:1 */;
in ter(0..0) RST_2 /* rst:1 */;

clock CLOCK;
reset RESET;

/* Memory Module Instance*/
defmod mem08A {
    in ter (7..0) A      /* ad:1 */;
    out ter (7..0) DO     /* rd:1 */;
    in ter (7..0) DI      /* wd:1 */;
    in ter (0..0) WEB    /* we:1 */;
    in ter (0..0) CSB    /* cs:1 */;
    in ter (0..0) BE     /* clk:1 */;
    in ter (0..0) RST    /* rst:1 */;
}inst_m;

var(0..0) we1, cs1;
var(0..0) we2, cs2;
var(7..0) adr1, adr2;
var(7..0) wd1, wd2;
var(0..0) module2_update = 1;
```

```

process mem08A_controller() {
    ter(1..0) lack;

    /* Input clock, reset signal normally in memory */
    inst_m.BE ::= CLOCK;
    inst_m.RST ::= RESET;

    /* Direct connection between memory output and memory controller output
     */
    DO_1 ::= inst_m.DO;
    DO_2 ::= inst_m.DO;
    ACK_1 ::= lack(0..0);
    ACK_2 ::= lack(1..1);

    /* Retain value when Request signal is activated. */
    adr1 = A_1;
    wd1 = DI_1;
    we1 = WEB_1;
    cs1 = CSB_1;

    /*
     * Retain value when Request signal is activated.
     * ACK_2 is not true for REQ_2 so value is not updated.
     */
    if (module2_update == 1) {
        adr2 = A_2;
        wd2 = DI_2;
        we2 = WEB_2;
        cs2 = CSB_2;
    }

    /* Wait till Request signal is activated*/
    if (REQ_1|REQ_2) {
        /* Adjust to give priority to REQ_1 */
        lack = mem_arbiter(REQ_1, REQ_2);
        if (lack & 0b01) { /* If lower order 1bit is true then incorporate
                           value of module1 */
            /* Input value in memory */
            inst_m.A = adr1;
            inst_m.DI = wd1;
            inst_m.WEB = we1;
            inst_m.CSB = cs1;
            if (REQ_2) {
                /* Do not update value of module2 */
                module2_update = 0;
            }
        }
        else { /* If lower order 1bit is true then incorporate value of
               module2 */
            /* Input value in memory */
            inst_m.A = adr2;
            inst_m.DI = wd2;
            inst_m.WEB = we2;
            inst_m.CSB = cs2;
            module2_update = 1;
        }
    }
}

```

```
/*
 * Adjust to give priority to req1.
 */
If lower order 1bit is 1 then activate module1.
* Upper order 1bit is 1 then activate module2.
*/
var(1..0)
mem_arbiter(
    var(0..0) req1,
    var(0..0) req2
)
{
    if (req1 == 1) {
        return 0b01;
    }
    else { /* req2 == 1 */
        return 0b10;
    }
}
```

11.11.8 Limitations

- It cannot be used in Manual Scheduling.
- In case of using Universal Memory Interface, it is necessary to specify memory delay of Pipeline memory.
- Array cannot be specified for array merge (array_merge).
- Specification cannot be done for memory with register insert specification (mem_reg) in port.
- Initial value cannot be specified in memory.

11.12 Generation of Memory Description for FPGA Logical Synthesis

11.12.1 Overview

Memory description for which logical synthesis is possible can be generated for FPGA of Altera company, Xilinx company. Automatic inference description is used in memory description and high speed simulation is possible.

When megafunction of Altera company or CORE generator of Xilinx company is used, then an option (-mem_black_box) for generating black box is specified by RTL generation (veriloggen/vhdlgen). In case of ASIC, black box is generated by default.

In memory types, memory for LUT implementation and block memory exists and implementation method can be controlled by attribute mem_synth specification in input description or specification of MEM_SYNTH information in MLIB file.

11.12.2 Related attribute

Attribute	Description	Setting Target
/* Cyber mem_synth = LUT */	Uses the memory implemented in LUT by FPGA synthesis	Array Variable
/* Cyber mem_synth = BLOCK */	Uses the block memory in FPGA synthesis	Array Variable
/* Cyber mem_synth = NONE */	Memory type is not specified in FPGA synthesis	Array Variable

11.12.3 MEM_SYNTH Entry for MLIB

In MEM_SYNTH of MLIB file, not only similar attributes can be specified but specific block memory type name (Ex M9K) can also be specified.

```

#@VERSION { 1.00 }
#@LIB { entry }
@MLIB {
    NAME mem08A
    KIND RW1
    BITWIDTH 8
    DELAY 5000
    WORD 200
    MEM_SYNTH BLOCK           <= Block memory
    ...
}
@MLIB {
    NAME mem08B
    KIND R1
    BITWIDTH 8
    DELAY 5000
    WORD 200
    MEM_SYNTH LUT            <= Memory for LUT implementation
    ...
}
#@END { entry }

```

11.12.4 Example from the Action Description till Logical Synthesis

- 1) Attribute is specified in array by behavioral description and synthesis is done after specifying FLIB/BLIB for FPGA.

```

in ter(7..0) a;
out ter(7..0) x;
mem(7..0) M[256]/* Cyber mem_synth = LUT, pipemem = x3 */ ;
process entry()
{
    x = M[a];
}

```

As attribute specification can be abbreviated, it is treated as block memory when abbreviated.

- 2) Auto generation of MLIB file.

Here, MEM_SYNTH can be modified but it is required to retain the conformity with attributes.

```

#@VERSION { 1.00 }
#@LIB { test }
@MLIB {
  NAME MEMB3W256
  KIND R1
  BITWIDTH    3
  DELAY        x3
  WORD 256
  MEM_SYNTH LUT
  DEFMOD {
    in   ter (7..0)  RA1      /* ra:1 */;
    out  ter (2..0)  RD1      /* rd:1 */;
    in   ter (0..0)  RCLK1    /* rclk:1 */;
  }
}
#@END { test }

```

3) Memory allocation by synthesis

Memory type of attribute and MLIB file can be allocated by below combination.

mem synth Attribute	MLIB MEM SYNTH
LUT	LUT
BLOCK	BLOCK, NONE, OTHERS (Except LUT, NONE)
NONE	NONE
Not specified	All available

4) Generates memory description by RTL generation tool (veriloggen/vhdlgen)

```

module MEMB3W256 ( RA1 ,RD1 ,RCLK1 );
input [7:0] RA1 ;
output [2:0] RD1 ;
input RCLK1 ;
wire Ren1_wire ;
reg [0:0] Ren1_sr [0:0] ;
reg [2:0] Rd1_sr [0:1] ;
reg [0:2] MEMB3W256_r [0:255] /* synthesis romstyle = MLAB */ ;
initial begin
    MEMB3W256_r[0] = 3'h1;
    MEMB3W256_r[1] = 3'h2;
    MEMB3W256_r[2] = 3'h3;
    MEMB3W256_r[3] = 3'h4;
    MEMB3W256_r[4] = 3'h5;
    MEMB3W256_r[5] = 3'h0;
    ...
    MEMB3W256_r[255] = 3'h0;
end
assign Ren1_wire = 1'h1 ;
assign RD1 = Rd1_sr[1] ;
always @ ( posedge RCLK1 )
begin
    if ( RCLK1 )
        begin
            Ren1_sr [0] <= Ren1_wire ;
            if ( Ren1_wire )
                begin
                    Rd1_sr [0] <= MEMB3W256_r[RA1] ;
                end
            if ( Ren1_sr[0] )           Rd1_sr [1] <= Rd1_sr[0] ;
        end
    end
endmodule

```

- 5) Generates script for Quartus II or ISE by Logical synthesis script auto generation tool (LSscrgen) (Refer to **Section A**) "Logical Synthesis Script Generation Tool")

If Quartus II (or ISE) is executed with command line, then project file is generated. Quartus II (or ISE) is started in gui mode and analysis can be done on gui based on project file specification.

11.12.5 Limitations

- Target memory supported is contingent to support range of Altera company Quartus II, Xilinx company ISE.
- Unsupported memory type does not generate memory description by RTL generation. However, it is possible to generate simulation model (*basename_E_SIM.v*) by sim_mem option during RTL generation. Attributes are specified, revised for this model when needed and it can be used in logical synthesis.

-

11.13 Summary

This section covered the following:

- There are two methods of synthesizing memory. The first method is to synthesize it as "internal memory" and the second method to synthesize it as "external memory".
- The register array is also referred to as register file.
- The condition that is necessary to carry out the variable unrolling of an array is that at the time of array reference or assignment all the subscripts are constants.
- When all the references and assignments of a multidimensional array contain a constant subscript along a fixed dimension, then that specific dimension is unrolled by the tool.
- It is possible to allocate multidimensional array along given dimensions after partitioning it into multiple memories.
- There are two methods to convert multidimensional arrays into one-dimensional arrays.
 - o In the first method, the subscript evaluation at the time of array reference or assignment is changed into description using the constant multiplication.
 - o The second method is to allocate a bit in each subscript.
- When an array is referenced when an assignment is in progress, referencing is disabled until the assignment is finished.
- When the assignment is to be done, it should be ensured that all references are complete before the assignment process starts.
- In order to allocate multiple arrays to one memory, multiple arrays are merged into one by specifying the attributes.
- Speculative execution is an operation or memory access that may not be needed if the determining conditions evaluate as FALSE later.
- The I/O array variables must be completely unrolled if they are to be converted to a switch case description style.
- The port to connect the memory is generated in accordance with the port information specified by the MLIB.
- The port to connect the memory is generated in accordance with the port information specified by the MLIB.
- For the array subscript of memory partition description, it is necessary that address input, array subscript, and element must be described at the same time in the array subscript of data output.
- In manual scheduling, as the number of decoders required for each step is clear in description, it is synthesized after providing the required number of decoders.
- In an automatic scheduling mode, the number of decoders is specified by the "rw_port" attribute, which is scheduled to the original.

12 External/ Shared Memory, Shared register / ter / var, External functional unit

12.1 Overview

This section covers the following.

- The concept of external/ shared memory, shared registers, shared ter, shared var and external functional units.
- The concept of shared registers.
- The concept of external functional unit.

12.2 External/ Shared memory

Here, 'Shared' means that resource is being shared by different processes while it exists outside all the processes. 'Outside' refers to the fact that this resource exists externally.

External memory (or functional unit) is the memory (or functional unit) that resides outside the synthesized circuit. For accessing this memory (or functional unit), I/O ports are generated in the synthesized circuit.

Shared memory/registers/ ter/ var are memory that is meant for shared use with other processes and it exists outside the synthesized circuit. I/O ports are generated in synthesized circuit for accessing this shared memory/ registers/ ter/var.

Specification method for external memory and shared memory has been explained in **section 11.3.2** of chapter **11**.

Further, it is also used while synthesizing top level through top level output tool, which is specified in **Appendix L**.

12.3 Shared register/ ter/var

12.3.1 Overview

'Shared' can be associated with arrays and register variables. 'Outside' can only be associated with an array.

Further, as generation of ports is a pre-requisite for 'outside' and 'shared', they can only be set using global variables.

Just like shared memory, a shared register / ter / var or register array resides outside the synthesized circuit in order to enable its sharing with other processes. The synthesized circuit will have I/O ports generated in order to connect it with the register / ter / var.

To specify a shared register/ ter / var or a shared register array, use the type "shared" declaration as shown below.

Description:

```
shared reg(0:32) R;
shared reg(0:32) M[2];
shared ter(0:32) TARY[2];
shared var(0:32) V;
shared var(0:32) VARY[2];
```

As shown below, it is assumed that a register / ter / var resides outside the synthesized circuit, three ports are generated for every variable of shared type (for access purpose) based on the variable name 'name' in the synthesized circuit.

Port name	Input/ output	Name of port generated
Read data port	Input	<i>name_rd</i>
Write date port	Output	<i>name_wd</i>
Write enable port	Output	<i>name_we</i>

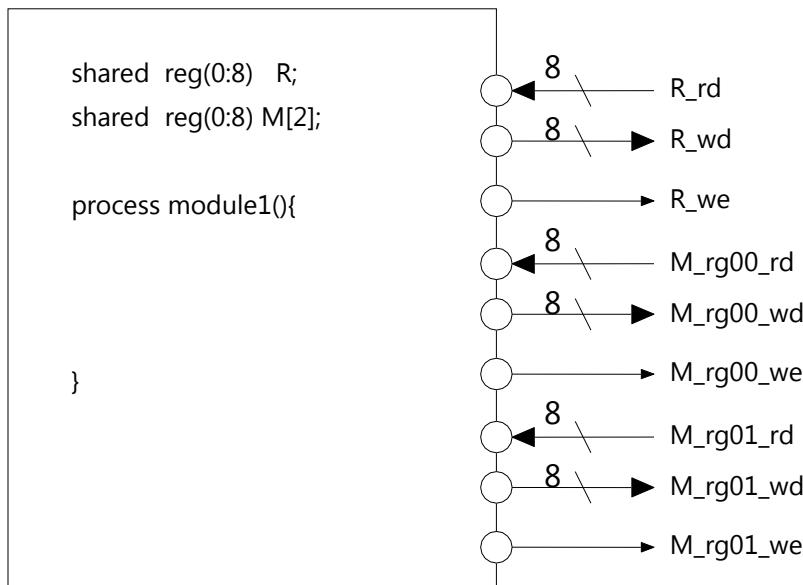
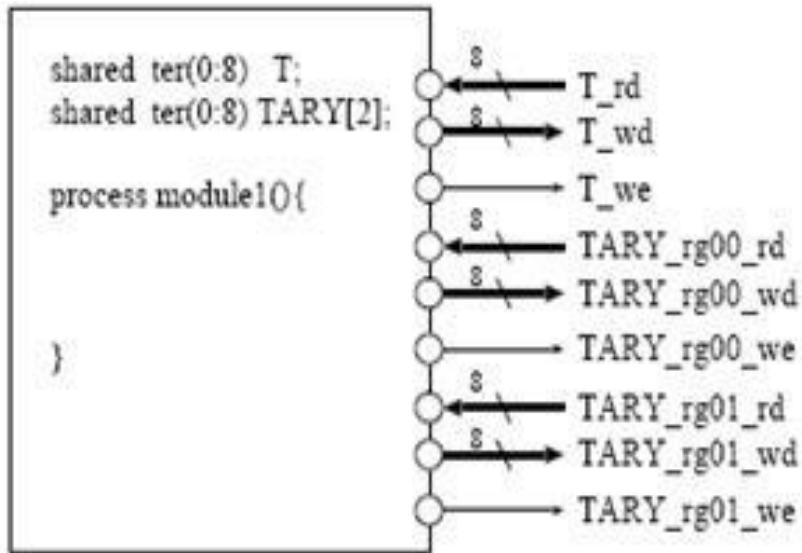
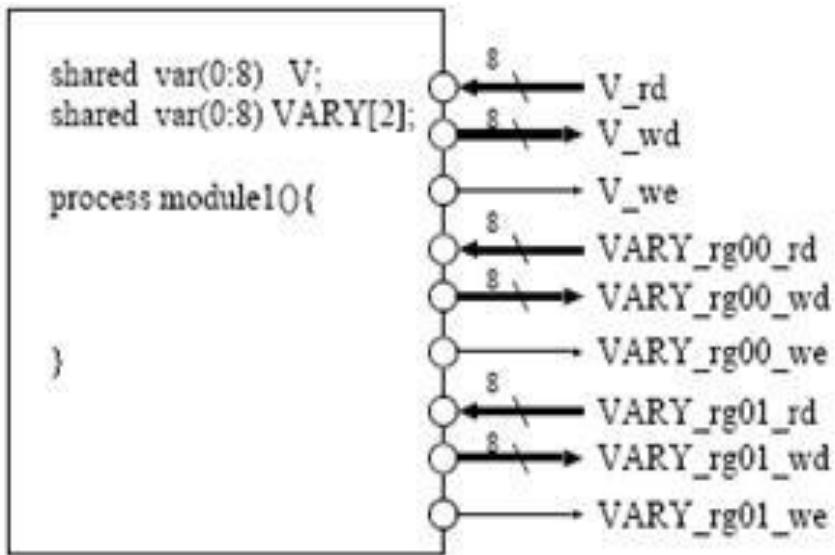


Figure 44: Shared register/ shared register array

**Figure 45:** shared ter / shared ter array**Figure 46:** shared var / shared var array

12.3.2 Initial value of shared register/var

When shared reg or shared var are initialized in the module declared, synthesized circuit changes according to the value specifying the following option -Zshared_reg_reset.

When -Zshared_reg_reset=UPPER or default is specified initialization is done in the module having the main frame of register of top hierarchy. In case -Zshared_reg_reset=HERE is specified, initialized circuit is generated in the module where shared variable is declared.

Option	Description
-Zshared_reg_reset=UPPER	Carries out initialization of shared register in top hierarchy and not in current process (default)
-Zshared_reg_reset=HERE	Carries out the initialization of shared register in current process

12.3.3 Activate Clock signal/Reset signal of shared register/var

In this section, description regarding the clock signal or reset signal which activates the register (Below, called as "main frame register") which becomes the main frame of shared variable is given.

- Clock reset of main frame register is set by clock reset of low hierarchy which carries out the writing.
- Main frame register is reset by reset signal mode of low hierarchy (Asynchronous or synchronous) which carries out the writing.
- However, when option Zset_lower_clock_reset=NO is used, clock register of main frame register is set by clock reset of the module where main frame register exists. Moreover, preference will be given to clock_sig attribute or reset_sig, reset_mode attribute if it is specified for the main frame register.

Option	Description
-Zset_lower_clock_reset=ALL	Sets reset clock of lower clock reset which carries out writing, to the clock reset signal of registered and shared memory. (Default)
-Zset_lower_clock_reset=SHARED_MEM	Sets same reset clock of lower clock reset carries out writing, to the clock reset signal of shared memory.
-Zset_lower_clock_reset=NO	Sets clock reset signal of current level to the clock reset of registered and shared memory.

For example, shared reg variable r in the following description carries out assignment in process foo.

As a result, register r is

- Activated by clock signal clk_a of process foo
- Reset (synchronous) by reset signal rst_a of process foo.

foo.bdl:

```
in ter (0:8)x;
shared reg (0:8)r;
clock clk_a;
reset rst_a
/*Cyber reset_mode=sync*/;
process foo(){
    r = x;
    $ 
    r= x;
}
```

bar.bdl:

```
out ter (0:8)y;
shared reg (0:8)r;
clock clk_b;
reset rst_b;
process bar() {
    y = r;
    $ 
    y = r;
}
```

RTL (verilog-HDL) description of main frame register r

```
always @ ( posedge clk_a )
if ( rst_a )
    r <= 8'h00 ;
else
    r <= ... ;
```

12.4 Value of array/shared register when not used

When array / memory / shared register are not used, value of address signal or data signal can be arbitrarily specified in any one of the Don't Care or 0.

- Options

Option	Description
-Zarray_port_other_DC[=YES]	Value of address signal, right data signal for arrays changes to Don't Care (default) when not used.
-Zarray_port_other_DC=NO	Value of address signal, right data signal for arrays changes to 0 when not used.
-Zsharedreg_port_other_DC[=YES]	Value of right data signal for shared reg variable (excluding arrays) changes to Don't Care (default) when not assigned.
-Zsharedreg_port_other_DC=NO	Value of right data signal for shared reg variable (excluding arrays) changes to 0 when not assigned.

- Synthesis example of shared register
 - Zsharedreg_port_other_DC=YES when specified (default)

```
r1_wd(0:1) ::= nmux {
    when (U_02(0:1)) : a2(0:1);
    when (U_01(0:1)) : a1(0:1);
};
```

- -Zsharedreg_port_other_DC=NO when specified

```
r1_wd(0:1) ::= nmux {
    when (U_02(0:1)) : a2(0:1);
    when (U_01(0:1)) : a1(0:1);
    default : 0;
};
```

- Array/Memory (Including shared register array) is controlled with -Zarray_port_other_DC.
- When synthesized with Don't Care, then default clause of 0 value can be added in nmux even by specifying -DO during veriloggen, vhdlgen execution.

12.5 External functional unit

Like memory or register, synthesis of a functional unit can be done while assuming that it exists external to the circuit being synthesized. Functional units covered in the scope are as follows:

- General functional unit.
- Function that is converted into a functional unit.
- Array generated by combinational circuit.

To generate functional unit implementation external to the circuit, YES should be specified for the OUTSIDE entry in the functional unit count file (refer to **Section 39.8**).

Following is an example of the required entry in functional unit count file:

```

@FCNT {
    NAME      mul32_32s
    LIMIT     1
    OUTSIDE   YES
}
@OPERATOR {
    NAME      func
    KIND      @func
    LIMIT     1
    BITWIDTH (32) ,32
    DELAY    1t
    OUTSIDE   YES
}
@OPERATOR {
    NAME      a
    KIND      @a
    LIMIT     1
    BITWIDTH (8), 32
    DELAY    1t
    SIGN     UNSIGNED
    OUTSIDE   YES
}

```

While implementing functional unit externally to the circuit, I/O ports for functional unit are generated in the synthesized circuit. By default, the name of I/O ports generated will be like (process_name)_ (port_name). Using the following options, it can be specified whether or not to add process name as prefix to port names:

Option	Description
-Zprefix_outside_fu[= YES]	Add process name to port name for external functional unit. (default)
-Zprefix_outside_fu=NO	Do not add process name to port name for external functional unit.

In case of user defined functional unit, external implementation can be done by specifying func_outside attribute along with func attribute in the function changed into user defined functional unit.

Attribute	Description	Target setting
/* Cyber func_outside = YES */	Change the user defined functional unit to external functional unit	Function definition

12.6 Summary

This section covered following:

- External memory is the memory that resides outside the synthesized circuit and accessed using the I/O ports.
- Shared memory / registers are memory that is meant for shared use with other processes, exists outside the synthesized circuit, and is accessed using I/O ports.
- Synthesis of a functional unit can be done while assuming that it exists external to the circuit being synthesized.

13 Pointer

13.1 Overview

This section covers the following:

- Pointer description and explanation.
- Pointer error message.

13.2 Related option

Option	Description
-Zpointer_analyze_effort=#	Specify the Pointer analysis level

13.3 Description

A pointer in CWB is a reference to a signal. At the time of synthesis, a pointer is substituted with its reference's contents. As this substitution process may take lots of time, the process is programmed to be aborted after some fixed duration by default.

In a description, lots of syntax can be used, such as the "if" statement, loops, etc. In such cases, if the substitution process of pointers does not finish within fixed time duration, the following synthesis error message is displayed.

```
F_BT4838: Since control syntax is complex, analysis of
the pointer can't be done
[counter measure] Increase '-Zpointer_analyze_effort=#'
```

In case of such an error message as shown in the figure above is displayed, synthesis can be enabled by increasing the value specified for the "-Zpointer_analyze_effort" option. However, increasing the value will lead to increase in the time taken for synthesis.

13.4 Restrictions

In CWB, there are restrictions in pointer description which can be synthesized. For details, please refer the section related to pointer in "BDL reference manual".

13.5 Summary

This section covered the following:

- A pointer in CWB is a reference to a signal.

- If the substitution process of pointers does not finish within fixed time duration, a synthesis error message is displayed.

Synthesis of a process, aborted after some fixed duration by default, can be enabled by increasing the value specified for the “–Zpointer_analyze_effort” option.

14 Structure

14.1 Overview

The following functionalities for synthesis related to structure have been explained in this chapter.

- Variable name after unrolling structured variable
- packed synthesis of structured variable

Refer to "BDL Reference Manual" for description of BDL specific types which can be added in structure as there is a chapter related to this in "BDL Reference Manual". Further, refer to the chapter related to Attributes of "BDL Reference Manual" for the method specifying attribute in structure.

14.1.1 Related Options

Option	Description
-Zstruct_name_length=#	Change the variable name after unrolling structured variable to # character or below. (Default value 127)
-Zmember_name_length=#	Add the top # character of member variable in the variable name after unrolling structured variable.
-Zmember_name_length	Add all the member variable name in the variable name after unrolling structured variable (Default)

14.1.2 Related Attributes

Attribute	Description	Target setting
/* Cyber packed */	Synthesize as a single signal having all the members of structured variable	Structured variable

14.2 Variable Name after unrolling Structured Variable

Option	Description
-Zstruct_name_length=#	Change the variable name after unrolling structured variable to # character or below (Default value 127)
-Zmember_name_length=#	Add top # character of member variable in the variable name after unrolling structured variable.
-Zmember_name_length	Add all member variables in variable name after unrolling structured variable (Default)

Structured variables are unrolled with variable name such as ““Structured variable name” – “Member variable name” for each member variable.

```
struct st1 {
    int m1;
    int m2;
};

Structured variable      Variable after unrolling
struct st1 s1;           →   int s1_m1;
                                int s1_m2;
```

By default variable name after unrolling is unrolled within 127 characters so that it does not exceed the signal name limit which is 128 characters. Option -Zstruct_name_length is being used that modifies the number of characters of function that manages the length of variable name after unrolling. However, variable name for circuit I/O is out of scope where behavioral description does not change variable name and an error will be generated if it exceeds 128 characters.

```
struct st2 {
    int hijklmn;
    int opqrstu;
};
Structured variable      Variable after unrolling
struct st2 abcdefg; →   int abcdefg_hijklmn;
                                int abcdefg_opqrstu;
Variable after unrolling when -Zstruct_name_length=12 is
specified

struct st2 abcdefg; →   int abcdefg_hijk;
                                int abcdefg_opqr;
```

Option -Zmember_name_length specifying the character count of added member variable name is prepared for checking the length of variable name when structure is nested. However, variable name of circuit I/O is out of scope as behavioral description is not changed.

```

struct st3 {
    int efg;
    struct st4 {
        int mnop;
        int qrst;
    }ijkl;
};

Structured variable           Variable after unrolling
struct st3 abcd;      →   int abcd_efgh;
                           int abcd_ijkl_mnop;
                           int abcd_ijkl_qrst;

Variable after unrolling when -Zmember_name_length=2 is specified
struct st3 abcd;      →   int abcd_ef;
                           int abcd_ij_mn;
                           int abcd_ij_qr;

```

14.3 Attribute packed Synthesis of Structured Variable

Attribute	Description	Target setting
/* Cyber packed */	Synthesize as single signal having all the members of structured variable	Structured variable

Structured variable is unrolled for each member and synthesized. However, in case of member etc, it is not unrolled for member and synthesized as a single variable having all structured variable.

For instance, in case of synthesizing below structured variable color, three 8 bit memory are required when unrolling is done for each member, and 24 bit memory can be synthesized 1 when all structured variable can be synthesized as single variable.

Synthesizing all structured variables as single variable is known as packed synthesis. Specified by adding attribute packed to structured variable.

```

struct color {
    char r, g, b;
};
struct color M[256];

struct color M[256]/* Cyber packed */;

```

Since structure variable having packed synthesis specification is treated as single variable for all structure variables and allocated to memory etc., reference and assignment are restricted as per the structure variable unit and there will be an error when reference and assignment are done according to the member units. Hence, structured variable having packed synthesis specification should perform reference and assignment through the structured variable having no packed synthesis specification.

```

struct color M[256] /* Cyber packed */;
struct color foo;
foo = M[adr];           /* OK */
out1 = foo.r;
out2 = foo.g;
out3 = foo.b;
foo.r = in1;
foo.g = in2;
foo.b = in3;
M[adr] = foo;           /* OK */
out1 = M[adr].r;        /* Error */
M[adr].r = in1;         /* Error */

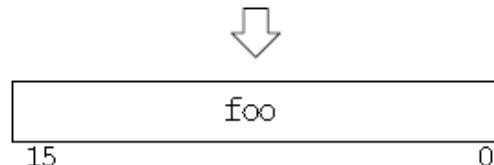
```

Structured variable having packed synthesis specification is synthesized as variable concatenated from top bit in the declaration order of the member. When including array in member, all elements are synthesized as variable concatenated from top bit.

```

struct ST1 {
    var(7..0) x;
    var(3..0) y[2];
};
struct ST1 foo;

```



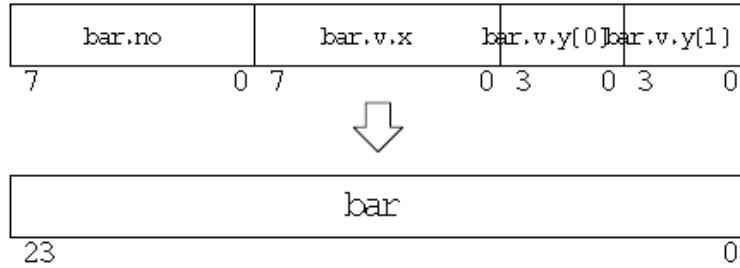
In case structure variable exists as the member of structure, member of structured variable is treated as packed synthesis when packed synthesis specification is done in external structured variable.

```

struct ST1 {
    char          no;
    struct color v;
};

struct ST1 bar/* Cyber packed */;

```



14.3.1 Notes in Hierarchical Description

When structure variable specifying packed synthesis is used in defmod abstract interface or in I/O variable, packed synthesis specification is required even in member of defmod declaration in top hierarchy and in lower hierarchy also.

In the below example, as packed synthesis specification exists in the structured variable foo declared as shared in lower hierarchy sub, packed synthesis specification is also required for member foo inside the defmod declaration of lower hierarchy sub described in top hierarchy top. In case packed synthesis specification differs, it result in a conflict error when information (.LMSPEC) of lower hierarchy is specified during top hierarchy synthesis.

Further, packed synthesis specification is required in structured variable bar in assign function of defmod abstract interface.

Lower hierarchy (sub)

```
shared mem struct color foo[256]/* Cyber packed */;
process sub() {
    ...
}
```

Top hierarchy (top)

```
defmod sub {
    ...
    shared mem struct color foo[256]/* Cyber packed */;
    ...
} inst_sub
mem struct color bar[256]/* Cyber packed */;
process top() {
    allstates {
        assign(bar, inst_sub.foo);
    };
}
```

14.3.2 Notes in User-Defined Functional Unit

packed synthesis can be done in structured variable as parameters of functional unit conversion function. In the below example, three 8 bit input are generated when there is no packed synthesis specification, and single 24 bit input is generated when pack synthesis specification exists.

```
/* Cyber func = operator */
char func1(struct color foo/* Cyber packed */) {
    ...
}
process proc1() {
    struct color bar;

    out1 =fncl(bar);
}
```

It is ignored when packed synthesis is specified to pointer variable of structure and in case the parameters of functional unit conversion function is the pointer variable of structure, packed synthesis specification can be done for generating I/O of functional unit conversion function along with the pointer variable type. However, there is a restriction in which packed synthesis specification becomes necessary in the structure variable specified in actual parameter. An error occurs when existence or nonexistence of packed synthesis specification differs in variable of parameter and actual parameter respectively.

```
/* Cyber func = operator */
void func2(struct color *p/* Cyber packed */) {
    ...
}

/* Cyber func = operator */
void func3(struct color *p) {
    ...
}

process proc2() {
    struct ST1 foo/* Cyber packed */;
    func2(&foo);
    func3(&foo); /* Error */
}
```

14.3.3 Restrictions

- Attribute packed cannot be specified in variables besides structure.
- Attribute packed cannot be specified in structure variable of structure having any of the pointer variable, memory, I/O in member.

```
struct st1 {
    in char il;
} st1pc/* Cyber packed */; /* Error*/
```

- Attribute packed in structure variable of structure having member of different physical type cannot be specified.

```
struct st1 {
    reg(0:8) r1;
    ter(0:8) t1;
} st1pc/* Cyber packed */; /*Error*/
```

- Since specification of reset value in `reset_block` and initial value during declaration of structure variable having packed synthesis specification is not supported, it results in an error.

```
struct color foo[256]/* Cyber packed */
= {{0,0,0},{0,0,0},...}; /* Not supported*/
```

Initial value is set in structure variable having no packed synthesis specification and assigned to variable having packed synthesis specification in the reset block when reset value is specified structure variable having packed synthesis specification.

In case of SystemC, initial value is set in structure variable having no packed synthesis specification and assigned to variable having packed synthesis specification in the reset state of SC_CTHREAD or constructor when reset value is specified structure variable having packed synthesis specification.

```
struct color bar[256]/* Cyber packed */;
struct color foo[256] = {{0,0,0},{0,0,0},...};

process moduleA() {
    int i;

    /* Cyber reset_block */
    {
        /* Cyber unroll_times = all */
        for(i=0;i<256;i++){
            bar[i] = foo[i];
        }
    }
    $  

    ...
}
```

- Since packed structured variable are not supported in structured dmux always connect statement and nmux always connect statement, it results in an error.

```
struct ST1 s1, s2;
struct ST1 s0 /* Cyber packed */;

s0 ::= nmux {
    when(c1): s1;
    default: s2;
};
```

- Attribute specification for member variable is ignored when structured variable specifying packed synthesis.

```
struct color {
    char r,g;
    char b /* Cyber array = RAM */;
    ↑
    /* Ignored for foo.b*/
};
struct color foo[256] /* Cyber packed ,.g={ array=REG }*/;
↑
foo.g attribute is ignored
```

- “packed” synthesis can be specified in structure of structure member but packed synthesis in separate member specification of structure variable is not supported, it results in an error.

```
struct ST1 {  
    struct color M[640]/* Cyber packed */; /* OK */  
    bool flag;  
};  
  
struct ST1 data;
```

```
struct ST1 {  
    struct color M[640];  
    bool flag;  
};  
  
struct st1 data/* Cyber .M = {packed} */; /* Not supported */
```

15 Function

15.1 Overview

This section covers the following:

- Function parameter types and return value.
- Synthesis methods for functions.
- Conversion of a function into a functional unit and types of functional unit.
- Initialization of local variable.

There are three types of synthesis methods for functions: "inline expansion", "goto conversion", and "conversion to functional unit". This section explains synthesis method, specification methods, and related restrictions for each of them.

15.1.1 Related Options

Option	Description
-fi	Synthesize function by inline expansion.
-fg	Synthesize function by goto conversion.
-fs: #	Function that has # or less statement count (Note 1) is synthesized by inline expansion
-fc:#	Function with # or less number of calls is synthesized by inline expansion.
-fx:#	Synthesize function where the number (#statements * #function calls) is less than or equal to #, by inline expansion.
-Zinit_local_sig	Initialize local variables of the function with 0
-Ztop= <i>name</i>	Synthesize the function ' <i>name</i> ' by conversion to a functional unit.

[(Note1) Regarding statement count]: Statement count refers to the number of 'command' statements within a function. Both "if" statement and "return" statement are counted as one statement. In the following description, statement count is 6.

```

var (0:10)
func ( var (0.8) a, var(0.8) b, var(0.8) c)
{
    var(0.8) x, y;
    If (c) {                                /* (1) */
        x = a + b;                          /* (2) */
        y = a - b;                          /* (3) */
    }else{
        x = a + 1;                          /* (4) */
        y = b - a;                          /* (5) */
    }
    return ( x + y ) ;                    /* (6) */
}

```

15.1.2 Related Attributes

Attributes	Description	Setting Target
/* Cyber func =inline */	Synthesize function by inline expansion.	Function definition statement, function declaration, function call
/* Cyber func =goto */	Synthesize function by 'goto' conversion	Function definition statement, function declaration, function call
/* Cyber inst_head = name */	'name' is used as signal prefix in case of inline expansion of function	Function definition statement, function declaration, function call
/* Cyber func = operator */	Convert function into a functional unit (Automatically specify port to be added)	Function definition statement, function declaration

15.1.3 Function Description Restrictions

Currently, the following functions cannot be synthesized:

- **Recursive functions cannot be used.**

A function that is called by itself or from functions that are called from this function cannot be synthesized.

- **Return value of a function without a 'return' statement can't be referred.**

For example, the following description will result in a synthesis error because the value returned by the function "func", which has no return statement, is being referenced.

```
in      ter(0:8) a;
out     reg(0:8) b;

process module () {
    b = func (a) ; /* Reference of a return value attempted for a
                      function with no "return" statement */
}
func ( var ( 0:8 ) i ){
    b = i;
    /* Function 'func' doesn't have any 'return' statement */
}
```

- **Function invocation using "continuous assignment" cannot be synthesized unless function is being converted into a functional unit.**

For example, in the following description, because the function "func" is being invoked through continuous assignment here, it will result in a synthesis error message. However, the error message does not occur if the function "func" is converted into a functional unit.

```
b : : = func ( a ) ; /* function calling by continuous assignment */
```

There are some restrictions depending upon function synthesis methods, which will be explained later in the section on synthesis method.

15.2 Function Parameter Types

It is recommended that parameters of the function are declared as "var" type. This is because even when they are declared as some other type, BDL input command converts them to "var" type. In manual scheduling, it is analyzed how a parameter has to be used. In addition, if a register is required, it is synthesized.

```

var ( 0: 32 )
func( var(0:32) m,
      var(0:32) n)
{
    reg(0:32) x,y;
    y = n;
    x = 10;
    $
    x = x + m;
    $
    return( x + y);
}

```

Parameter 'n' is used only in the first cycle within function and not in subsequent cycles. Therefore, a register is not required.

As parameter 'm' is used in second cycle inside the function, a register is required. Therefore, a register which will hold the value of 'm' will be synthesized.

15.3 Type of Function Return Value

It is recommended that a function return value is also declared as the "var" type. Moreover, the "reg" type or "mem" type declaration is not allowed. However, if it is declared as "ter" type, it is automatically converted to the "var" type.

15.4 Synthesis Method

There are three methods to synthesize functions.

- inline expansion
- "goto" conversion
- Conversion of a function into a functional unit

It is possible to reduce circuit area or cycle count by changing synthesis method according to the function process contents and the number of calls.

15.4.1 Inline Expansion

Inline expansion is a method of expanding the function contents at all places where function is being invoked.

As the function contents are placed and expanded where invocation is done, the overall size of the description increases. This increase in description results in possible increase in the die area. However, the execution cycle count of synthesized circuit generally decreases as compared to the "goto" conversion.

In case of inline expansion, the name of the signal automatically generated during inline expansion can be controlled to some extent. By specifying the /* Cyber inst_head = *name* */

attribute with the expression invoking the function, prefixes can be specified for signals generated during inline expansion.

For example, the prefix "ab_" is specified for the first invocation of function "func". Moreover, "ac_" is specified as a prefix for the second invocation.

```

in ter(0:8) a;
in ter(0:8) b;
in ter(0:8) c;
out ter(0:8) o1;
out ter(0:8) o2;

process main()
{
    o1 = func(a, b)      /* Cyber inst_head=ab_ */;
    o2 = func(a, c)      /* Cyber inst_head=ac_ */;
}
/* Cyber func=inline */
var(0:8) func(
    var(0:8) x,
    var(0:8) y
)
{
    if (x < y) {
        return y - x;
    } else {
        return x - y;
    }
}

```

In a regular inline expansion, the synthesized result (_C.BDL) is as specified below:

```

ST1_01 :

    x_1(0:8) := a(0:8);
    y_1(0:8) := b(0:8);
    if (x_1(0:8) < y_1(0:8)) {
        C_func_1(0:8) := y_1(0:8) - x_1(0:8);
    }
    else {
        C_func_1(0:8) := x_1(0:8) - y_1(0:8);
    }
    o1(0:8) = C_func_1(0:8);
    x(0:8) := a(0:8);
    y(0:8) := c(0:8);
    if (x(0:8) < y(0:8)) {
        C_func(0:8) := y(0:8) - x(0:8);
    }
    else {
        C_func(0:8) := x(0:8) - y(0:8);
    }
    o2(0:8) = C_func(0:8);
    goto ST1_01;
}

```

However, when the prefixes are specified using 'inst_head' attribute, the synthesized result (_C.BDL) is as specified below:

```

ST1_01:

/* invocation of first function func is supported from here
   and, prefix ab_ is added*/
ab_x_1(0:8) := a(0:8);
ab_y_1(0:8) := b(0:8);
if (ab_x_1(0:8) < ab_y_1(0:8)) {
    ab_C_func_1(0:8) := ab_y_1(0:8) - ab_x_1(0:8);
}
else {
    ab_C_func_1(0:8) := ab_x_1(0:8) - ab_y_1(0:8);
}
o1(0:8) = ab_C_func_1(0:8);
/* invocation of 2nd function func is supported from here
   and, prefix ac_ is added */
ac_x(0:8) := a(0:8);
ac_y(0:8) := c(0:8);
if (ac_x(0:8) < ac_y(0:8)) {
    ac_C_func(0:8) := ac_y(0:8) - ac_x(0:8);
}
else {
    ac_C_func(0:8) := ac_x(0:8) - ac_y(0:8);
}
o2(0:8) = ac_C_func(0:8);
goto ST1_01;
}

```

15.4.2 Goto Conversion

The "goto" conversion is a synthesis method in which the processes of functions are consolidated into a single location. This consolidation is such that all the function process invocations are executed from this single location only.

If the same function is to be invoked from multiple locations, consolidation of the processing to a single location leads to a smaller circuit area in comparison to inline expansion. However, as the invoking of the function requires one cycle for function call, this method tends to results in increased execution cycle count as compared to inline expansion.

15.4.3 Conversion of a Function into a Functional Unit

This is a synthesis method in which the target function is first synthesized as a separate process (separate module). Then, this module is invoked as a functional unit.

For details refer to **section 15.5**.

15.4.4 Function Synthesis Method Specification

The synthesis method of a function is determined in the following order:

1. If the attribute is specified at the function invocation, synthesis is carried out according to the attribute. Error is generated when synthesis cannot be done by specified synthesis method.

inline expansion attribute

```
o1 = func1(a, b, c) /* Cyber func=inline */;
```

goto conversion attribute

```
o2 = func2(a, b, c) /* Cyber func=goto */;
```

2. When an attribute is specified in the definition or declaration part of a function, synthesis is done according to the attribute. In case the function cannot be synthesized as per the method specified by the attribute, it will result in an error.

Inline expansion attribute

```
/* Cyber func = inline */
var(0:8)
func1( var(0:8) a,
       var(0:8) b,
       var(0:8) c )
{
:
:
}
```

goto conversion attribute

```
/* Cyber func = goto */
var(0:8)
func2( var(0:8) a,
       var(0:8) b,
       var(0:8) c )
{
:
:
}
```

**Attribute to convert function to
a functional unit**

```
/*Cyber func =operator */
var(0:8)
func3( var(0:8) a,
       var(0:8) b,
       var(0:8) c )
{
:
:
}
```

3. For a function with no attribute specification, synthesis method is decided based on options, size of the respective function, and the number of invocations.

Option	Description
-fi	Synthesize function by inline expansion.
-fg	Synthesize function by goto conversion.
-fs: #	Function that has # or less statement count is synthesized by inline expansion.
-fc:#	Function which isn't called more than # times is synthesized by inline expansion.
-fx:#	Function in which (#statements) * (#function calls) isn't more than # is synthesized by inline expansion.

- In manual scheduling, all the functions are synthesized using inline expansion if any of the above option is not specified.
Even if one of the above options is specified, but the function does not fulfill restrictions of the "goto" conversion, the function will be synthesized using inline expansion.
When one of the above options is specified, either inline expansion or "goto" conversion will be decided according to the option.
If either of the "-fi" or "-fg" option is specified, then decision to synthesize is made accordingly. In case one or more of the remaining four options (-fs, -fc, -f1, -fx) are specified, function that satisfies even one specified condition will be synthesized using

inline expansion. In case it doesn't meet even one of the specified conditions, the function will be synthesized using the "goto" conversion.

- In case of automatic scheduling, either of inline expansion or "goto" conversion will be decided based upon the above options specified.

If either of "-fi" or "-fg" option is specified, then the decision to synthesize according to the scheduling mode is made accordingly. In case one or more of remaining four options (-fs, -fc, -f1, -fx) are specified, function that satisfies even one specified condition will be synthesized using inline expansion. In case it doesn't meet even one of the conditions, it will be synthesized using goto conversion.

However, if none of the above option is specified, then one of these options is decided automatically and synthesis is done accordingly.

- In case func=operator is not specified, function is not assumed to be functional unit even when the functional unit constraint exists and it is synthesized with inline expansion or goto conversion.

15.5 Conversion of User defined functional unit

15.5.1 Overview

Note: Please take a note of following terms which have been referred throughout this section.

Target function: Target function refers to function that is being synthesized by conversion of function into a functional unit. It is synthesized before the parent function specified by attribute func=operator as a separate process (separate module). And this module is synthesized and invoked as a part (functional unit). It is sometimes also referred as **child function**. Even by specifying attribute operator_block in block and not only in function, it is synthesized as a separate process depending on the function of converting into user defined functional unit and hierarchization can be done.

Parent function: Function that invokes the target function is known as parent function.

Functional Unit Count File: This is a file that specifies the resource constraints for each functional unit used in synthesis.

Functional Unit Constraint File: This is a file that specifies information and constraints regarding functional units used in synthesis.

Following are the benefits of converting a function into a functional unit:

- Doing trade off between the circuit area and execution cycle count is easier.

When a function is synthesized as a functional unit, it is possible to adjust level of parallelism of the function simultaneously by adjusting number of functional units that are required.

Therefore, it becomes easier to do a trade-off between the circuit area and execution cycle count.

- Synthesis mode, synthesis option, and functional unit constraints can be altered function wise. In this method, synthesis mode (manual / automatic / automatic pipelined) and synthesis options can be specified. In addition, the mode and options can be altered function wise to synthesize separate, respective processes for each function.
Further, only special function can be synthesized with emphasis on area or latency by altering functional unit constraints function wise.
- Layered synthesis is possible.
By synthesizing larger descriptions in layers, it is possible to control the overall flow of synthesis, which helps in cutting down synthesis time.

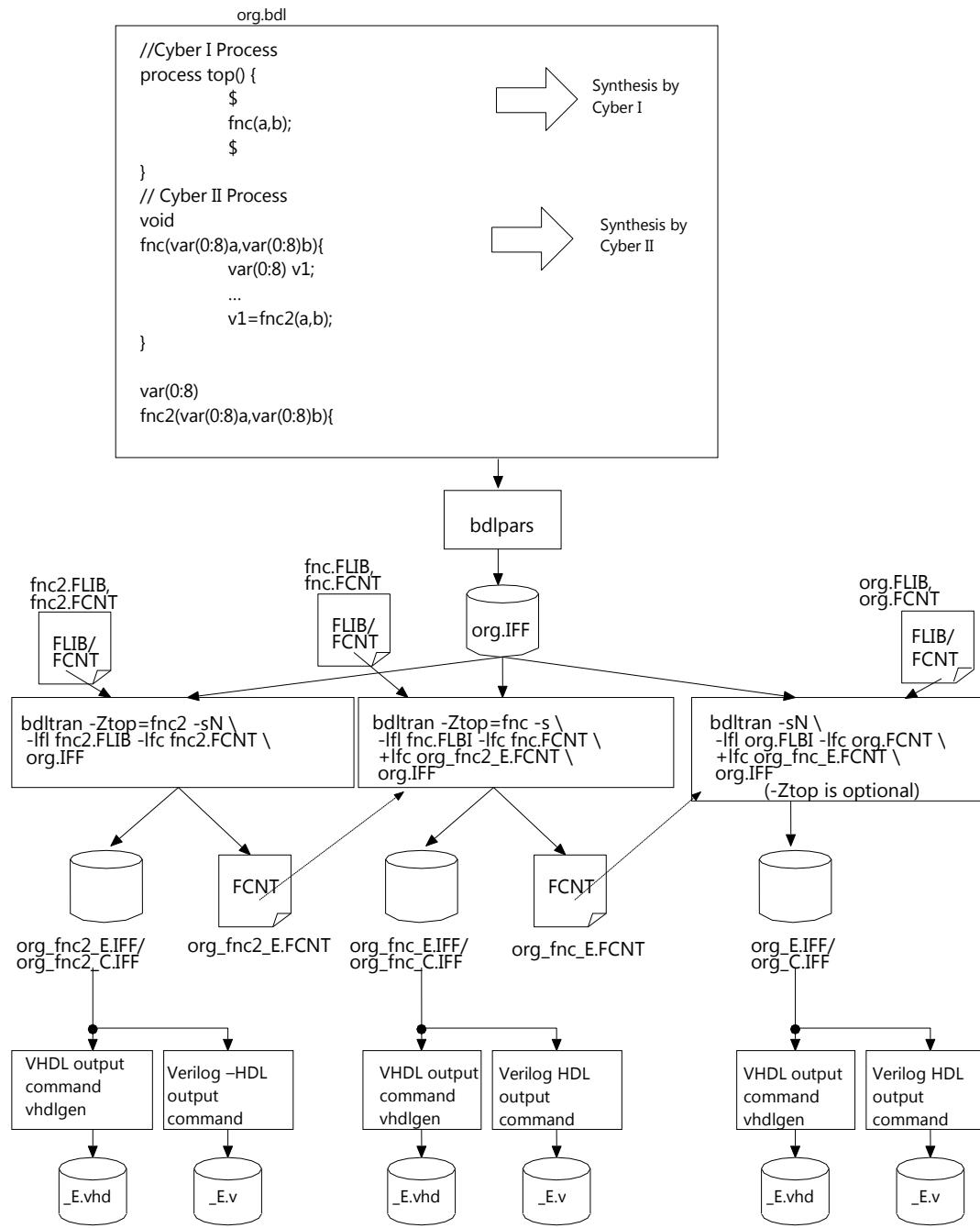
Below, function specifying the attribute func=operator and block specifying the attribute operator_block are called with function of functional unit conversion target.

15.5.2 Synthesis options

Options	Description
-Ztop = <i>name</i>	Synthesizes function ' <i>name</i> ' or the block specified by operator_block= <i>name</i> by converting it into a functional unit.

In the case of a function synthesis using target function, first of all the function that needs to be converted to functional unit needs to be synthesized. Next, the parent function is synthesized. When the function to be synthesized needs to be converted into a functional unit, the “-Ztop=*name*” option must be specified. (Here, ‘*name*’ is the name of the target function that needs to be converted into a functional unit). Simultaneously, a lower module specification file (LMSPEC file) is also generated that has information about the function module being synthesized. At the time of synthesis of the process function that invokes the target function, LMSPEC file is used for input. In case there are multiple nested functions (functions that are called in a nested order) that are to be converted to functional units, synthesis starts from the function that is at lowest level in the order of invocation (Refer to **Figure 47**). For functions that are to be converted into functional units, attribute specified in next paragraph should be used.

When synthesis is done using the “-Ztop=*name*” option, the output file name is different than the usual file name. For example, if the IFF file name is foo.IFF, generally a suffix will be added to “foo” so that the name of the output file becomes “foo_E.IFF”. However, if the “-Ztop=*name*” option is specified, the “*name*” is joined with “foo” as “foo_*name*”. Then, a suffix is added to the “foo_*name*” so that the name of the output file becomes “foo_*name*_E.IFF”. The name of the lower module specification file (LMSPEC file) will also be like “foo_*name*_E.LMSPEC.”.

**Figure 47:** Synthesis flow for conversion of a function into functional unit

	-Ztop option specified			-Ztop option not specified (synthesize process)
	Specify process -Ztop = ptop	Specify function -Ztop = func1	Specify operator_block -Ztop=blk1	
Input IFF	Foo.IFF			
Output file	Foo_ptop.SUMM Foo_ptop.err Foo_ptop_E.IFF Foo_ptop.LOG Foo_ptop-auto.LMT Foo_ptop_E.LMSPEC	Foo_func1.SUMM Foo.fun1.err Foo_func1_E.IFF Foo_func1.LOG Foo_func1-auto.LMT Foo_func1_E.LMSPEC	foo_blk1.SUMM foo_blk1.err foo_blk1_E.IFF foo_blk1.LOG foo_blk1-auto.FLIB foo_blk1_E.LMSPEC	Foo.SUMM Foo.err Foo_E.IFF Foo.Log Foo-auto.LMT Foo_E.LMSPEC *

* Please note that the specifications will be different because LMSPEC created by synthesis of process is for topmodgen.

15.5.3 Target Function Synthesis - Specification Method

Attribute	Description	Setting Target
/* Cyber func = operator */	Change function into functional unit (Auto specification of port added)	Function definition statement, function declaration part
/* Cyber operator_block= name */	Block is changed into hierarchy	Block, control statement
/* Cyber func = combinational_operator */	Converts function into functional unit as a combinational circuit	Function definition statement, function declaration part
/* Cyber func = sequential_operator */	Converts function into functional unit as a sequential circuit.	Function definition statement, function declaration part
/* Cyber func = pipeline_operator */	Converts function into functional unit as a pipelined circuit.	Function definition statement, function declaration part

/* Cyber func = operator [:c][:r][:n][:s][:e][:h]	Converts function into a functional unit(port to be added can be specified)	Function definition statement, function declaration part
/* Cyber func_out_sig = name */	Specifies that 'name' is an output port for function being converted to functional unit	Function definition statement, function declaration part
/ *Cyber func_clock_sig = name */	Specifies that 'name' is a clock port for function being converted to functional unit	Function definition statement, function declaration part
/* Cyber func_reset_sig = name */	Specifies that 'name' is a reset port for function being converted to functional unit	Function definition statement, function declaration part

15.5.3.1 Attribute specified for function synthesized as functional unit

Clock signal, reset signal or the handshake signal between the hierarchy are generated automatically as needed by specifying the attribute func=operator in function targeted for functional unit conversion. It will good if attribute operator_block is specified in block.

* Control signal must be specified till CWB5.2.2, however from CWB5.3, it is automatically determined if required, therefore, it will be fine if func=operator or operator_block is specified. Caution is required in case of synthesis is done before CWB5.2.2.

15.5.3.2 Individual specification of synthesis method of function targeted for functional unit conversion

In the function targeted for functional unit conversion, attributes func= combinational_operator, func = sequential_operator, func=pipeline_operator are specified and synthesis method can be specified separately. However, caution is required for getting the intended operation done, therefore **it is not recommended excluding the special cases.**

- Functions synthesized as combinational circuit (combinational_operator)

While synthesizing function as combinational operator, clock signal, reset signal and handshake signal will not be generated. Further, there will be an error if clock signal and reset signal is required, for example, when a register is present in the circuit to be synthesized as combinational operator, etc.

- Functions synthesized as a sequential circuit (sequential_operator)

While synthesizing function as sequential operator, clock signal, reset signal, and handshake signal will be generated.

- Function synthesized as pipelined circuit (pipeline_operator)

While synthesizing function as pipeline operator, clock signal and reset signal are generated. Synthesis should be carried out in automatic pipeline mode.

15.5.3.3 Individual specification of generated circuit

In case generation of clock signal, reset signal, handshake signal of execution start–execution end between parent-child function and availability of stall signal individually, attribute func=operator[:c][:r][:n][:s][:e][:h] can be used. We don't recommend the specification where handshake circuit is not created when the function is synthesized as a sequential circuit because there needs to be some elaboration to achieve desirable behavior.

To convert a function into a functional unit, the "func=operator" attribute should be specified. When specifying the "func=operator" attribute, requirement of 'clock' signal, 'reset' signal, execution start/stop handshake signal between parent and child function, and pipeline stall signal can be specified.

- Clock signal(c), reset signal (r)

Clock signal can be generated by specifying 'c' in the "func=operator" attribute. And reset signal can also be generated by specifying 'r'.

Clock signal and/or reset signal are automatically generated if necessary. Hence, unless there is a requirement of forced generation of clock signal or reset signal, specification of 'c' and 'r' is not required.

- Stall signal (h)

Generation of stall port can be specified by specifying "h" in attribute func = operator. Further, whether stall signal is required or not is determined automatically unless it is not specified particularly. As it is auto generated when required, it is not necessary to specify "h" except when it is to be generated compulsorily. Generation of stall port can be specified by option - Zfunc_stall. Refer to the description given later.

- Handshake signal (s, e,), No handshake signal (n)

Depending upon the requirement of execution of start/stop handshake signals between the target function process (child function) and the process invoking function (parent function), the target function can be categorized into 4 types: N type, S type, SE type, and SEA type.

- **N type functional unit (func=operator:n)**

In an N type functional unit, a handshake is not required between the target function (child) and the process invoking (parent) function process.

- **S type functional unit (func=operator:s)**

In an S type functional unit, a 'start' signal (that signifies execution start) from parent function process specifies execution start to child function process. The S type functional unit monitors start signal in a wait condition. In addition, when 'start' signal changes to 1, it accepts arguments from parent function and starts execution.

A return value is passed back to the parent function when the execution of target function process ends. After the return value is passed, the S type functional unit again starts monitoring signal in wait condition.

- **SE type functional unit (func=operator:s:e)**

In an SE type functional unit, an 'eop' signal (end of process signal) signifying the end of execution of child function. It is used along with 'start' signal. Just like S type functional unit, SE type functional unit starts monitoring 'start' signal in wait condition.

When start signal changes to 1, it accepts arguments from parent function and starts executing. When the execution of target function process is complete, the 'eop' signal is set to 1 while passing the return value to the parent function.

Again, the SE type functional unit starts monitoring for start signal in wait condition.

15.5.3.4 Port name of automatically generated port

The names of the ports automatically generated in functions converted into functional unit are as follows:

Kinds of port	Port name	Attribute to change port name
Clock port	[CLOCK] or port name of signal declared as 'clock'	<code>/* Cyber func_clock_sig = name */</code>
Reset port	[RESET] or port name of signal declared as 'reset'	<code>/* Cyber func_reset_sig = name */</code>
Start port	[function name_start]	None
'eop' port	[function name_eop]	None
Pipeline stall port	"STALL" or port name of Stall signal	None
Argument count	[argument count name]	None

Output port	[V_function name]	/* Cyber func_out_sig = name */
-------------	-------------------	---------------------------------

However, the name of clock port or reset port can be changed by synthesis options “-Fc”, “-Fr”. Moreover, attribute to change port name can be set with either function definition statement or function declaration.

15.5.4 Functional Unit Count File (lower module specification file)

While synthesizing target function, a lower module specification (LMSPEC) file is simultaneously generated by specifying “-Ztop=name” option. The functional unit count file contains module definition information of the target function. If the name of the IFF file is “foo.IFF” and function name is “name”, then the name of the file specified above will be ‘foo_name_E.LMSPEC’. While synthesizing parent process, the (generated) lower module specification file should be input. However, it is recommended that automatically generated LMSPEC file should not be edited (manually).

15.5.5 Target Function Instance specification

In case of multiple instantiation of target function process, the upper limit of the number of instances can be specified in the constraint count (LIMIT count) of functional unit count file (FCNT file). The functional unit count file (FCNT file) is automatically generated when process that calls functional unit is synthesized.

15.5.6 Target Function Global Variables - Access Mode Specification

To access global variables from target function, two kinds of synthesis methods are available: pass by value (copy mode) and pass by reference (shared mode).

- Pass by value (copy mode)

In case of this synthesis method, the value of a global variable is copied into a register inside the functional unit during the start of function execution. When the function execution ends, the value inside the register of functional unit is written back to the global variable.

In case of target function synthesis in manual scheduling mode, access can't be done using copy mode. Moreover, external input and external output ports cannot be accessed using copy mode.

- Pass by reference (share mode)

In case of this synthesis method, copying of a global variable to a register inside the functional unit is not done. This is because the direct (read and write) access to the global variable is available at the required instance.

Access to external input and external output is only possible in a ‘share’ mode.

It is automatically decided whether synthesis needs to be done using pass by value (copy mode) or pass by reference (shared mode). **Hence, in principle, it cannot be necessary to specify which mode should be used for synthesis.** Global variable can't be accessed from target function if the global variable meets any of the following conditions:

- Memory Array signal
- Pointer signal.
- External I/O (inout) signal
- Global variable of ter type (excluding external input, external output)

15.5.7 Option specification if stall signal generation of function targeted for functional unit conversion

Option	Description
-Zfunc_stall=auto	Generates stall control signal in function if necessary (default)
-Zfunc_stall=create	Generates stall control signal in function
-Zfunc_stall=ignore	Do not generate stall control signal in function

Stall signal of function targeted for functional unit conversion is auto generated and control by option -Zfunc_stall is also possible.

15.5.8 Static Variables of Target Function

Attribute	Description	Settings Target
/*Cyber func_static = asitis*/	Exclusively Maintains the static variable within a function at every instance.	Function definition statement, Function declaration part
/*Cyber func_static = asglobal*/	Shares the static variable of a function between instances	Function definition statement, Function declaration part

The static variable of a function, which is synthesized as a functional unit, is usually synthesized as global variable of process function of upper module. However, in case of function call from a process function when the child function is to be synthesized, a static variable can only be called once. At that time, the register which implements the static variable of target function is synthesized such that functional unit instance can be maintained).

Also, when the "func_static=asistis" attribute is specified in function, a static variable of the function is synthesized while maintaining it in each instance.

In this case, thus, when there are multiple instances of a functional unit, static variable's value in each functional call may not be same. Thus caution is required.

15.5.9 Calling of Function for Converting Functional Unit (in case of new function/top module)

Option	Description
-ll <i>filename_funcname_E.LMSPEC</i>	Specifies <i>filename_funcname_E.LMSPEC</i> as lower module specification file.

In case of synthesis of parent process function, the LMSPEC file is used as the constraint input file. In case the loading of multiple LMSPEC files is required, use option "+ll" (instead of "-ll"). If parent process function requires invoking of target function in multiple numbers, use the "+ll" option for every LMSPEC file and specify as follows:

```
bdltran ... +ll foo_name1_E.LMSPEC +ll foo_name2_E.LMSPEC ...
```

In addition, ensure that while specifying options, "+ll" is not missed out for every LMSPEC file specified as shown in the following example.

```
bdltran ... +ll foo_name1_E.LMSPEC     foo_name2_E.LMSPEC ...
          ^^^
          +ll is required
```

15.5.10 Input Values - Default Specification

Options	Description
-Zfunc_arg_other_DC = NO	Specifies that in case target function is not invoked, then during synthesis <ul style="list-style-type: none"> • Hold target function input port to 0, • Suppress logical optimization process
-Zfunc_arg_other_DC = YES	Specifies that in case target function is not invoked, then during synthesis:

- | | |
|--|--|
| | <ul style="list-style-type: none"> • Do not hold its input port to 0 • Do not suppress logical optimization process. |
|--|--|

Normally, when a functional unit is not invoked by any parent process, the logical optimization process is done treating input port as don't care (without holding it to '0'). Therefore, it is not clear whether the input of the functional unit is '0'. In case there is a requirement of assuring that input port is 0 when functional unit is not invoked, the "-Zfunc_arg_other_DC" option must be specified. For example, in the following description, target function ip_fnc() is called in state ST1_01 and ST1_03, but not called in state ST1_02.

```

in ter(0:8) in0,in1;
out ter(0:8) out0,out1;
clock clk;
reset rst;

process main(){
    out0 = ip_fnc(in0,1) ; /* Invocation at state ST1_01 */
    $
                           /* No invocation at state ST1_02 */
    $
    out1 = ip_fnc(in1,1) ; /* Invocation at state ST1_03 */
    return ;
    $
}
defmod ip {
    in ter(0..0) enable ;
    in ter(7..0) in_dat ;
    out ter(7..0) out_dat ;
    clock clk;
    reset rst;
} inst1 ;
/* Cyber func = operator */
var(7..0) ip_fnc(var(7..0) v0, var(0..0) en)
{
    ter(7..0) dat;
    inst1.clk ::= clk;
    inst1.rst ::= rst;
    inst1.in_dat = v0 ;
    inst1.enable = en ;
    dat = inst1.out_dat ;
    return dat;
}

```

The part of the synthesis output in Verilog-HDL format obtained above is shown below.

If input of target function ip_fnc() (low order module ip_fnc()) is noticed by logical optimization of behavioral synthesis, the value of input port 'enable' (including state ST1_02 where target function is not invoked) is fixed to '1' in all states. However, value of input port v0 is 'x' under state ST1_02 where target function is not invoked and depends on logical optimization in logical synthesis.

```

assign M_02 = 1'h1 ;
always @ ( ST1_03d or in1 or ST1_01d or in0 )
  case ( { ST1_01d , ST1_03d } )
    2'b10 :
      M_01 = in0 ;
    2'b01 :
      M_01 = in1 ;
    default :
      M_01 = 8'bx ;
  endcase
ip_fnc ip_fnc_1 (
  .v0(M_01) ,.en(M_02) ,.clk(clk) ,
  .rst(rst) ,.V_ip_fnc(M_03) );

```

By specifying the “-Zfunc_arg_other_DC” option, the Verilog-HDL output received is shown below. The value of input port is 0 in state ST1_02 that is not invoked by target function.

```

assign M_02 = ( ST1_01d | ST1_03d ) ;
always @ ( ST1_03d or in1 or ST1_01d or in0 )
  case ( { ST1_01d , ST1_03d } )
    2'b10 :
      M_01 = in0 ;
    2'b01 :
      M_01 = in1 ;
    2'b00 :
      M_01 = 8'h0 ;
    default :
      M_01 = 8'bx ;
  endcase
ip_fnc ip_fnc_1 (
  .v0(M_01) ,.en(M_02) ,.clk(clk) ,
  .rst(rst) ,.V_ip_fnc(M_03) );

```

However, the “-Zfunc_arg_other_DC” option has no effect on target functions that do not have a clock port.

15.5.11 Parent Function Conversion to a Functional Unit

During synthesis, parent function of target function can also be synthesized by the conversion of function into a functional unit. All child (target) functions are synthesized before parent functions. This is required to generate the FCNT file.

Moreover, while specifying the “-Ztop” option, the FCNT file generated earlier is specified.

15.5.12 Restrictions

There are some restrictions in synthesis of target function.

- Restrictions regarding parameters to target function.

Only array variables such as scalar variables, structure variable, that have EXPAND or register are allowed as parameters to target function for synthesis. For pointer variables, pointers to

scalar type or structure type are allowed but, pointer or multiple pointers to array are not allowed. In array, the element count should be specified explicitly.

```
OK:  
/* Cyber func = operator */  
var(7..0) fnc(var(7..0) v0) { ... }  
  
/* Cyber func = operator */  
var(7..0) fnc(var struct { var(7..0) mmbrl ; } v0) { ... }  
  
/* Cyber func = operator */  
var(7..0) fnc(char *p) {  
    *p = in0;  
    out= *p;  
}  
  
/* Cyber func = operator */  
var(7..0) fnc(var(7..0)ary[64]) { ... }
```

NG:

In case the element count is shortened in multidimensional
dummy argument)
/* Cyber func = operator */
var(7..0) fnc(var(7..0) (*ary)[64]) { ... }

In case of accessing dummy argument pointer as array)
/* Cyber func = operator */
var(7..0) fnc(var(7..0) *ary) {
 return ary[0];
}

In case dummy argument count is multi-pointer)
/* Cyber func = operator */
var(7..0) fnc(var(7..0) **ary) { ... }

Bit vector is allowed both in ascending as well as in descending order such that at least one of the initiating bit or terminating bit is 0.

```

OK (Either of the initiating or termination bit is 0)

    /* Cyber func = operator */
Descending order      var(7..0) fnc(var(7..0) v0) { ... }

    /* Cyber func = operator */
Ascending order      var(0:8) fnc(var(0:7) v0) { ... }

NG (neither of the initiating or termination bit is 0)
    /* Cyber func = operator */
Descending order      var(7..0) fnc(var(7..2) v0) { ... }
                                         ^^^^^^

    /* Cyber func = operator */
Ascending order      var(0:8) fnc(var(2:5) v0) { ... }
                                         ^^^^^^

```

It is recommended that the parameter to target function is declared as "var" type. Even if all parameters are declared as a type other than "var", such as 'ter' type and "reg" type, the synthesis results will not change.

4. Restrictions regarding return value of target function

It is recommended that the return value of the target function should be also declared as "var" type. (Declaration as 'reg' type or 'mem' type is not allowed.)

For the return value type of function, data structure that holds a pointer member cannot be used.

Bit vector is allowed in ascending and descending order such that either the initial or the last bit is 0.

```

OK (Either of the initiating or termination bit is 0)
    /* Cyber func = operator */
Descending order      var(7..0) fnc(var(7..0) v0) { ... }

    /* Cyber func = operator */
Ascending order      var(0:8) fnc(var(0:7) v0) { ... }

NG (neither of the initiating or termination bit is 0)
    /* Cyber func = operator */
Descending order      var(7..2) fnc(var(7..0) v0) { ... }
                                         ^^^

    /* Cyber func = operator */
Ascending order      var(2:8) fnc(var(0:5) v0) { ... }
                                         ^^^

```

3. Restrictions regarding global variable access from target function.

As explained in section **15.5.6**, the global variables applicable to any one of the following conditions cannot be accessed from target function:

2. Memory array signal
3. Pointer signal
4. External I/O (inout) signal
5. Global variable of ter type (excluding external input and external output)

• Restrictions regarding synthesis method of local array variables of target function.

Local arrays of target function can't be synthesized as external memory. Synthesis is done by other methods such as internal memory, register array, variable expansion etc. Refer to **section 11.3.2** for details on external memory.

• Restrictions regarding synthesis options for target functions synthesis.

While synthesizing target functional units, specification of options or attributes that automatically generate input-output ports of functional unit module is not allowed.

- Functional unit cannot be implemented outside circuit using the feature to convert a function into an external functional unit. This capability automatically generates an external port for connecting to external functional unit. Refer to **section 12.5** for details of external functional unit.
- Even if "-Zport_valid_sig_gen" option is specified, it doesn't influence the generated circuit.
- Synthesis should not be done in order to generate the value of state signal by specifying the "-Zgen_stateno_out" option. This option automatically generates output port for monitoring state signals externally. Refer to **section 7.7** for details regarding this option.
- Wrapper generation capability for internal memory bypass for logicBIST (built in self test) should not be used by specifying the "-ZZlogic_bist" option. This option automatically generates test port. (Refer to **section 31** for details regarding this option.)

• Restrictions regarding target function call positioning in parent function.

- In total coupling statement (::=) and allstates statement, the target function having output other than return value cannot be called.

In the target functions which has output other than return value, there are user defined functional unit of SE type, external input, user defined functional unit accessing the global variable synthesized by external output signal or share method, user defined functional unit for which executed cycle count differs for each call. Generated port can check the lower module specification file file (LMSPEC file).

Moreover, function target of S type cannot be called.

There is F_BT4583 error when it corresponds to this restriction.

```

NG)
process main(){
    out0 ::= fnc_SE(in0);      /* Function with output other than
                                return value can't be called in the
                                middle of total coupling statement*/
}
/* Cyber func = operator:s:e */
var(0:8) fnc_SE(var(0:8) v0)
    /*In SE type functional unit, 'eop' signal exists in output signal
     other than return value*/
{
    ...
}

```

- o User defined functional unit of SE type cannot be called inside the body and conditions of wait() statement and watch() statement.
Even if called, there will be F_BT1342 error or F_BT4582 error when it corresponds to this restriction.
- o It is recommended that the target function should call by individual assignment statement and it should not call in the middle of the expression or in the conditional expression.

```

process main(){
    Recommended)
        out0 = (in0);          /* Should call by individual assignment */

    Not recommended)
        while ( fnc(u0) == 0 ) {
            ...
                /*should not call in the middle of
                 conditional statement */
        }
        out0 = fnc_s(in0) + fnc_seq(in1) ;
                /*should not call in the middle of
                 expression*/
}

```

- Restrictions regarding synthesis mode of parent function
Target functions of output existing besides return value cannot be called from parent function that is being synthesized by automatic pipelined synthesis mode. (Even if it is called, a synthesis error will be generated).
- Restriction regarding inaccurate delay analysis of the calling process of target function
In the delay analysis of process that calls the target function, there is a limitation that the delay of path including the target function after this is inaccurate. The delay is separately analyzed inside the target function and the call side of the function, because unconnected

paths are added. For instance, sometimes the delay of operator before and after the call of target function executed by a different cycle is regarded as the same cycle and the total is done, and sometimes in the internal target function when register assignation is done, at that time also, it is regarded as combination circuit and the total is done.

Therefore, in the delay of circuit quality report of the process calling the target function, bigger delay is displayed and at the time of synthesis of process calling target function of SE type, F_BT6316 error occurs.

- Restriction of delay error when pointer to array is called in actual argument of function operation machine with variable offset

In actual argument of user defined functional unit, there is a restriction that delay error occurs when pointer to array is called with variable offset.

In actual argument of user defined functional unit of SE type, in case pointer to array is called with variable offset, F_BT6316 error occurs when the total amount of delay of those delay and user defined functional unit for calculating the offset exceeds the clock cycle.

However, as mentioned earlier there is a limitation in the delay analysis of the process which calls the target function. Even when the delay is reduced, there are chances of synthesis error because the delay analysis is done for not only the delay of the part related to access of the corresponding array but also for delay of all target functions by using the critical path.

In that case, it is recommended to take measures like not calling pointer to array with variable offset in actual argument of target function.

Moreover, when pointer to array is called with variable offset in actual argument of target function, prepare the address decoder required in attribute rw_port because only same address decoder of declared array in temporary argument of target function is necessary.

- Restriction regarding detection of access outside territory when array is used in argument of target function

When array is used in argument of target function and further offset is specified in actual argument and size of actual argument and temporary argument are different, then even if array access in target function accesses the area outside array, there is no rule that it will be automatically detected at the time of synthesis or at the time of circuit operation. (At the time of synthesis when it is clearly identified then either a warning or error is generated.) Please note that care should be taken as in case other than the area of array is accessed, the circuit does not perform the operation as intended and a different location (address) of the array may be discarded.

- Restriction at the time of SystemC input

Function having wait() statement cannot be synthesized as a pipelined functional unit inside the function. If synthesis is run, then error F_BT5063 occurs.

F_BT5063: In Auto pipeline synthesis of SystemC input, conversion into functional unit is not possible with wait() description

Synthesis cannot be done when sc_signal is accessed in a certain function of wait() statement in internal part of the function. If synthesis is run, then error F_BT5068 occurs.

F_BT5068: Limitation in the functionality of converting function into functional unit at the time of SystemC input: wait() cannot be described in target function which accesses sc_signal
[Action] Function func() is not converted into functional unit.

Function which is being accessed in sc_in and sc_out array cannot be synthesized as functional unit. If synthesis is tried then F_BT5064 error occurs.

F_BT5064: Limitation in functionality of converting function into functional unit at the time of SystemC input: Target function cannot access sc_in/sc_out array.
[Action] Function func() is not converted into functional unit.

Synthesis cannot be done when temporary argument of the function is array. If synthesis is tried, then error F_BT5069 occurs.

F_BT5069: Limitation in functionality of converting function into functional unit at the time of SystemC input: It does not correspond to array argument.
[Action] Function func() is not converted into functional unit.

In case parameters of function are pointer and structure, then synthesis is not possible. If synthesis is tried, then error F_BT5071 occurs.

F_BT5071: Restrictions for Function of converting function into functional unit: In the parameter of function func () of converting functional unit, the structure having the sc_in/sc_out pointer signal or the structure pointer having sc_in/sc_out signal cannot be used
[Action] Function func() is not converted into functional unit.

15.6 Initialization of Local Variable

Option	Description
-Zinit_local_sig	Initializes local variable of function to 0 only in automatic scheduling mode

The value of local variable before assignment is undefined just like in 'C' language. In the case of example given below, if the executable statement setting local variable 'r' of function 'func1' inside while() loop is not executed at least once, 'r' remains undefined. In such cases, it is recommended that at the time of variable declaration, an initial value should be assigned to the variable (e.g. var(0:8) r = 0;).

```

var(0:8)
func1( var(0:8) i1,
       var(0:8) i2 ){
    var(0:8) r,c;
    c = i1;
    while( c > 10 ){
        r = i2;
        c--;
    }
    return( r );
}

```

The “-Zinit_local_sig” option can be used to perform synthesis by initializing local variables within a function to '0'. However, the option “-Zinit_local_sig” is not valid for initialized variable or array variables and pointer variables.

15.7 Summary

This section covered the following:

- There are three types of synthesis methods for functions: “inline expansion”, “goto conversion”, and “conversion to functional unit”.
- It is recommended that parameters of the function are declared as “var” type.
- It is recommended that a function return value is also declared as the “var” type.
Inline expansion is a method of expanding the function contents at all places where function is being invoked.
- The “goto” conversion is a synthesis method in which the processes of functions are consolidated into a single location.
- Conversion of a function to a functional unit is a synthesis method in which the target function is first synthesized as a separate process (separate module).

- In manual scheduling, all the functions are synthesized using inline expansion if none of the six options are specified.
- In case of automatic scheduling, either of inline expansion or "goto" conversion will be decided based upon the six options specified.
- To convert a function into a functional unit, the "func=operator" attribute should be specified.
- The target function can be categorized into 4 types: N type, S type, SE type, and SEA type.
- The three categories of frequently used combinations are Synthesis as a combinational circuit, synthesis as a sequential circuit (sequential operator), and synthesis as a pipelined circuit (pipeline operator).
- While synthesizing target function, an FCNT file is simultaneously generated by specifying "-Ztop=name" option.
- The static variable of a function, which is synthesized as a functional unit, is usually synthesized as global variable of process function of upper module.
- When a functional unit is not invoked by any parent process, the logical optimization process is done treating input port as without holding it to '0'.
- The value of local variable before assignment is undefined.

16 for loop

16.1 Overview

This section covers the following:

- Loop unrolling of 'for' loop where repeat count is fixed or can be calculated
- Optimization of operations within unrolled loop
- Functionality to merge identical structures in "for" loop

16.1.1 Related Options

Option	Description
-uA	Unroll the loop according to the restrictions (default)
-u0	Do not unroll 'for' loop
-UB#	Unroll if number of statements after unrolling of 'for' loop will be # or less.
-UL#	Unroll if loop count of 'for' loop is # or less.
-UN	Do not restrict unrolling of 'for' loop
-oH	Optimize operations within unrolled loop (default)
-o-H	Do not perform optimization of operations within loop unrolled
-Zopt_loop_merge	In automatic scheduling, merge 'for' loops with identical structures into one (default)
-Zopt_loop_merge=NO	In automatic scheduling, do not merge 'for' loops with identical structures into one

Refer to **chapter 15** Note -1, for "number of statements" (or the statement count)

16.1.2 Related Attributes

Attribute	Description	Settings Target
/* Cyber unroll_times = all */ /* Cyber unroll_times = #[:#] */	Unroll 'for' loop Unroll 'for' loop #times in parts If [:#], is specified, Out of remainder loop-count, loop statements are executed partially for # times before 'for' loop execution	'for" statement 'for" statement
/* Cyber thr_unroll = YES */ /* Cyber thr_unroll = NO */	Optimize operations within unrolled loop Do not optimize operations within loop unrolled	Statement Statement
/* Cyber loop_merge_ex = name */	In automatic scheduling, aggregate all 'for' loops with identical structures labeled 'loop_merge_ex' attribute into one 'for' loop.	'for' statement
/* Cyber loop_merge_ex_top = name */	In automatic scheduling, aggregate all 'for' loops with identical structures that have value of attribute loop_merge_ex set to 'name' into one 'for' loop. (position resulting code after merging at top loop)	'for' statement
/* Cyber loop_merge_ex_bottom = name */	In automatic scheduling, aggregate all 'for' loop with identical structures that have value of attribute loop_merge_ex set to 'name' into one 'for' loop. (position resulting code after merging at bottom loop)	'for' statement

16.1.3 "for" loop with fixed repeat count

If iteration count of "for" loop is fixed, it is necessary that the following conditions are fulfilled:

```

[Initial setting part] [Condition part][Re-setting part]
↓           ↓           ↓
for ( i = 0, j = 0;      i < 10;      i++) {
:
:
← [ loop main body]
}

```

- **Initial setting part**
 - First statement is setting the counter variable to initial value.
 - When assignment destination is variable concatenation (a::b=...) or an array variable, loop will not be unrolled.
 - Assigned value is a constant value.
 - In case the assignment destination is a member of a structural variable, there is a possibility of not being unrolled.
- **Condition part**
 - Operator should be one of ">","<","<=",">=","!=".
 - Variable declared as counter variable in initial setting part is an operand in this part.
 - The other operand is a constant.
- **Re-setting part**
 - If there are 2 or more statements, loop is not unrolled.
 - In case there is a "\$", loop is not unrolled.
 - If assignment destination is variable concatenation (a::b=...), loop does not unroll
 - Variable assumed as counter variable in initial setting part is reset by a constant value.
 - Resetting is done using assignment operators '++', '-','+','=','-','=','*','=','/','=','<<','=','>>' or operators '+','-','*','/','<<','>>' or assignment statement "=".
- **Loop main body**
 - If there exists a label in main body of loop, the loop isn't unrolled

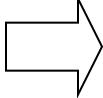
16.2 “for” Loop unrolling

16.2.1 Unrolling Types

There are two types of ‘for’ loop unrolling. One type is where the loop is unrolled for all loop iterations. Another type is where the loop is unrolled for a few iterations. The second type is also called partial loop unrolling.

1. Loop unrolling

Unroll loop processing for all iterations.

Description	After loop unrolling
<pre>/* Cyber unroll_times = all */ for(i = 0 ; i < 8 ; i++) { ary [i] = i ; }</pre>	 <pre>ary [0] = 0 ; ary [1] = 1 ; ary [2] = 2 ; ary [3] = 3 ; ary [4] = 4 ; ary [5] = 5 ; ary [6] = 6 ; ary [7] = 7 ;</pre>

Indices used in array access are transformed into a constant by loop unrolling. This, leads to the speeding up of accesses depending upon array implementation method. Again, if there is no dependency relationship among iterations, parallel implementation is possible. This parallel implementation leads to reduction in execution cycles.

However, in case the loop iteration count is very big, description size becomes large. Therefore, the loop will require more time for synthesis.

2. Partial Loop Unrolling

In this type of unrolling, process is unrolled within the loop only as per the number specified and iteration count of the loop is adjusted appropriately.

Description	After partial loop unrolling
<pre>/* Cyber unroll_times = 4 */ for(i = 0 ; i < 8 ; i++) { ary [i] = i ; }</pre>	 <pre>for(i = 0 ; i < 8 ; i = i + 4) { ary [i] = i ; ary [i+1] = i+1 ; ary [i+2] = i+2 ; ary [i+3] = i+3 ; }</pre>

16.2.2 Specification Method

Decision methods to do 'for' loop unrolling have been demonstrated below:

1. If the iteration count of "for" loop is unknown, unrolling is not done. (Refer **16.1.3**)

```
for (i = 0; i < x; i++)
```

2. Unrolling is based on the specification of attribute of the "unroll_times" option.

- o When the "all" attribute is specified, the loop is unrolled completely.

```
/* Cyber unroll_times = all */
for (i = 0; i < 12; i++)
```

- o If a number is specified, loop is unrolled partially for the number of times specified.

```
/* Cyber unroll_times = 3 */
for (i = 0; i < 12; i++)
```

- Do not unroll if number specified is 0 or 1.

```
/* Cyber unroll_times = 0 */
for (i = 0; i < 12; i++)
```

- When iteration count is not divisible by the specified number (in attribute definition), the loop body is implemented for the remainder portion (remainder of operation "loop count/specified number") after the loop.

```
/* Cyber unroll_times = 5 */
for (i = 0; i < 12; i++){
    foo[i] = i;
}
↓
for (i = 0; i < 10; i = i + 5 ){
    foo[i] = i;
    foo[i + 1] = i + 1;
    foo[i + 2] = i + 2;
    foo[i + 3] = i + 3;
    foo[i + 4] = i + 4;
}
foo[10] = 10;
foo[11] = 11;
```

When iteration count is not divisible by specified number (unroll count), it can be specified that some part of (division) remainder can be implemented before loop and rest part can be implemented after the loop. This is done by specifying ":"# after the unroll count, so that out of remainder value, loop body is implemented # times before loop and implemented "remainder-#" times after the loop as shown below.

```
/* Cyber unroll_times = 5:1 */
for (i = 0; i < 12; i++) {
    foo[i] = i;
}
↓
foo[0] = 0;

for (i = 1; i < 11; i = i + 5) {
    foo[i] = i;
    foo[i + 1] = i + 1;
    foo[i + 2] = i + 2;
    foo[i + 3] = i + 3;
    foo[i + 4] = i + 4;
}
foo[11] = 11;
```

3. When no attribute has been specified for "for" loop, loop unrolling is decided on the basis of the statement count or iteration count of loop. This occurs according to the options specified below.

If no option has been specified, unrolling is performed when the statement count after loop unrolling comes out to be less than or equal to 512.

Option	Description
-uA	Unroll the loop according to the restrictions (default)
-u0	Do not unroll 'for' loop
-UB	Unroll if the number of statements after unrolling of 'for' loop will be # or less.
-UL	Unroll if loop count of 'for' loop is # or less.
-UN	Do not restrict unrolling of 'for' loop

16.3 Optimization of Operations

This section explains optimization of operations contained in unrolled 'for' loop. This functionality is only available in the case of automatic scheduling mode.

16.3.1 Related Options

Option	Description
-oH	Optimize operations within unrolled loop (default)
-o-H	Do not perform optimization of operations within unrolled loop.

16.3.2 Related Attributes

Attributes	Description	Settings Target
/* Cyber thr_unroll = YES */ /*Cyber thr_unroll = NO*/	Optimize operations within unrolled loop	Statement
	Do not optimize operations within unrolled loop	Statement

16.3.3 Description

The optimization functionality tries to reduce the execution cycles by parallelizing assignment operations within the unrolled "for" loop statements. To reduce execution cycles, it is important to increase the number of functional units available by customizing it along with this functionality. However, if the number of usable functional units is increased, area size can also increase. Desired performance level can be attained by controlling 'usable functional units count'.

Operations that are targeted for optimization are shown below:

- Increment operator (++)
- Decrement operator (--)
- Assignment operators (+=,-=,*=,<<=,>>=,&=,|=,^=)

The following assignment operations are also targeted:

```
j = j + a[i]; /* j += a[i]; */
j = j - a[i]; /* j -= a[i]; */
j = j * a[i]; /* j *= a[i]; */
j = j << a[i]; /* j <= a[i]; */
j = j >> a[i]; /* j >= a[i]; */
j = j & a[i]; /* j &= a[i]; */
j = j | a[i]; /* j |= a[i]; */
j = j ^ a[i]; /* j ^= a[i]; */
```

The following figure demonstrates an example of conversion functionality. The description tend to be synthesized in parallel if the "-oH" option is used rather than the "-o-H" option. The "-oH" option is the default option.

```
var(0:8) j;
var(0:8) a[10];
/* Cyber unroll_times = all */
for (i = 0; i < 5; i++) {
    j += a[i];
}
↓
j += a[0];
j += a[1];
j += a[2];
j += a[3];
j += a[4];
↓
/* -OH (DEFAULT) SPECIFICATION*/
j = ((j + a[0]) + (a[1] + a[2])) + (a[3] + a[4]);
/* -o-H SPECIFICATION*/
j = (((j + a[0]) + a[1]) + a[2]) + a[3] + a[4];
```

Optimization can be controlled by specifying “thr_unroll=NO” attribute for target statement.

```
in ter(0:8) in0;
var(0:8) j;
var(0:8) v;

/* Cyber unroll_times = all */
for (i = 0; i < 5; i++) {
    v = in0;
    /* Cyber thr_unroll = NO */
    j += v;
}
```

16.4 Merge Capability of 'for' Loop

This section describes the merging functionality of "for" loops. The merging functionality enables the aggregation of multiple "for" loops with identical structure at one place. This functionality is only available in automatic scheduling mode.

16.4.1 Related Options

Options	Description
-Zopt_loop_merge	In automatic scheduling, merge 'for' loops with identical structures into one (default)
-Zopt_loop_merge=NO	In automatic scheduling, Do not merge 'for' loops with identical structures into one

16.4.2 Related Attributes

Attribute	Description	Setting Target
/*Cyber loop_merge_ex = name */	In automatic scheduling, aggregate all 'for' loop with identical structures that have value of attribute loop_merge_ex set to 'name' in one.	'for' statement
/* Cyber loop_merge_ex_top = name */	In automatic scheduling, aggregate all 'for' loop with identical structures that have value of attribute loop_merge_ex set to 'name' in one. (position resulting code after merging at top loop)	'for' statement
/* Cyber loop_merge_ex_bottom = name */	In automatic scheduling, aggregate all 'for' loop with identical structures that have value of attribute loop_merge_ex set to 'name' in one. (position resulting code after merging at bottom loop)	'for' statement
/* Cyber loop_merge_ex_force= name */	In automatic scheduling, aggregate all 'for' loop with identical structures that have value of attribute loop_merge_ex set forcefully to 'name' in one	'for' statement
/* Cyber loop_merge_ex_top_forc e= name */	In automatic scheduling, aggregate all 'for' loop with identical structures that have value of attribute loop_merge_ex set forcefully to 'name' in one (position resulting code after merging at top loop)	'for' statement

Attribute	Description	Setting Target
/* Cyber loop_merge_ex_bottom_f orce= name */	In automatic scheduling, aggregate all 'for' loop with identical structures that have value of attribute loop_merge_ex set forcefully to 'name' in one (position resulting code after merging at bottom loop)	'for' statement

16.4.3 Description

This functionality leads to the improvement of throughput by merging of multiple "for" loops into a single structure and hence trying to attain parallelization.

As illustrated in the following example, parallelization of process A and process C is aimed at by merging identical structured 'for' loops into one. However, this functionality is exercised only if dependency relationship and control relationship of operations are maintained even after merging. (Refer to **16.4.4** for details)

```

for(i=0; i<count ; i++) {
    A;
}
B;
for(i=0; i<count ; i++) {
    C;
}

```

```

for(i=0; i<count ; i++) {
    A;
}
C;
B;

```

If this conversion is performed incorrectly, the following information is shown in error (.err) file.

[Source line]

```
I_BT4945: Loops of identical structures are merged
14(foo.bdl) : (i = 0; i < count; i++) {
18(foo.bdl) : (i = 0; i < count; i++) {
```

This optimization functionality is utilized by default. However, this functionality can be switched off by specifying the "-Zopt_loop_merge=NO" option. Also, merging of specific "for" loops into one can be ensured by specifying the "loop_merge_ex" attribute with a unique *name* for the target loops.

```

/* Cyber loop_merge_ex = merge1 */
for(i=0; i<count ; i++) {
    A;
}
B;
/* Cyber loop_merge_ex = merge1 */
for(i=0; i<count ; i++) {
    C;
}
D;
for(i=0; i<count ; i++) {
    E;
}

```

→

```

for(i=0; i<count ; i++) {
    A;
    C;
}
B;
D;
for(i=0; i<count ; i++) {
    E;
}

```

In the example provided above, two ways of merging can be classified in terms of dependency relationship among A, B and C. One method is to place the merged "for" loops at process A (topmost loop). The other method is to place the merged loop at process C (last loop).

Either of the above two merging type can be particularly specified. For merging at the top loop, the "loop_merge_ex_top" attribute needs to be specified instead of "loop_merge_ex" attribute. Similarly, for merging at last loop, the "loop_merge_ex_bottom" attribute needs to be specified.

* In the case of merging to loop of process A (topmost loop)

```

/* Cyber loop_merge_ex_top = X */
for(i=0; i<count ; i++) {
    A;
}
B;
/* Cyber loop_merge_ex_top = X */
for(i=0; i<count ; i++) {
    C;
}

```

→

```

for(i=0; i<count ;
i++) {
    A;
    C;
}
B;

```

* In the case of merging to loop of process C (last loop)

```

/* Cyber loop_merge_ex_bottom = Y */
for(i=0; i<count ; i++) {
    A;
}
B;
/* Cyber loop_merge_ex_bottom = Y */
for(i=0; i<count ; i++) {
    C;
}

```

→

```

B;
for(i=0; i<count ; i++) {
    A;
    C;
}

```

Further, merging can be done in cases where data dependency exists and where merging is not possible by using any of the loop_merge_ex_force, loop_merge_ex_top_force, loop_merge_ex_bottom_force attributes.

```
/* Cyber loop_merge_ex_force = X */
for (i = 0; i < count; i++) {
    ary[i] = i;
}
A;
/* Cyber loop_merge_ex_force = X */
for (i = 0; i < count; i++) {
    ary2[i] = ary[i] + 1;
    /* Usually it is assumed that
       dependency exists in ary[i],
       and merging is impossible
*/
}
→ } A;
```

In automatic judgment, merging can be done in cases where it is not possible, but in reality, since operation might change before and after merging, it is necessary to specify while taking caution.

```
/* Cyber loop_merge_ex_force = X */
for (i = 0; i < 3; i++) {
    s = i;
}
A;
/* Cyber loop_merge_ex_force = X */
for (i = 0; i < 3; i++) {
    out0 = s;
    /* 2 is output thrice to out0
       before merging but after
       merging, 0,1,2 is output to
       out0 */
}
→ } A;
```

16.4.4 Conditions for Merging

It is necessary that the following conditions must be satisfied while merging two or more "for" loops. However, when specifying any of the loop_merge_ex_force, loop_merge_ex_top_force, and loop_merge_ex_bottom_force attributes, it is merged forcefully even if the other conditions are not met when the structure of merge target loop is same. (Removed in case other loop main frame exists in if statement).

- Structure of loop for merging is same.
- Loop format is the same.
- Control dependency of operations before and after merging is retained.
- When loop execution depends only on the loop return condition
- When one loop is executed once completely, the other loops can definitely be executed.
- Dependency relationship of data before and after merging is retained.

- Data dependency relationship within two or more "for" loops is retained.
- Data dependency relationship between two or more loops executions and within two "for" loops is retained.

16.4.4.1 Determining whether the Format is same for "for" Loops

Variable, that is defined in the initialization part, referred in the condition part, and updated in the reset part of the "for" statement is known as loop counter variable.

- Bit width of loop counter is same

```
int i;
var(0:8) j; /* NG */
for (i = 0; i < 100; i++) {
    A;
}
for (j = 0; j < 100; j++) {
    B;
}
```

```
int i, j;
for (i = 0; i < 100; i++) {
    A;
}
for (j = 0; j < 100; j++) {
    B;
}
```

```
for (i = 0; i < 100; i++) {
    A;
}
B;
j = i;
```

- Initialization of loop counter in the initial setting part is the same.
- Updated value of loop counter in the resetting part remains the same.
- No variable other than the counter is initialized in the initialization part.
- No variable other than the counter is updated in the resetting part.
- The logic of conditions, including the sequence of operands in condition part, are the same.
(However, when the loop count differs, then it is considered to be of same format)

```
for(i=0;i<100;i++) {
    A;
}
for(i=0;100>i;i++) { /* NG */
    B;
}
```

```

for (i = 0; i < 100; i++) {
    A;
}
for (i = 0; i < 80; i++) {
    B;
}
for (i = 0; i < 100; i++)
{
    A;
    if (i < 80) {
        B;
    }
}

```

- Loop counter variable is not updated inside the loop body.
- Variable other than the loop counter variables that are referred in the condition, resetting or initialization part of the "for" statement are not updated within the loops being merged
- Initialization, condition, and resetting part do not refer to "shared", input/output variable.

16.4.4.2 Determining whether loop execution depends only on the loop return condition

- 'exit', 'return', 'goto', 'goto0' and labels do not exist inside 'for' loop
- 'for' loops don't have any 'break' statement
- 'for' loops doesn't contain any 'continue' statement (except last loop, which may contain 'continue' statement)

16.4.4.3 Determining Whether the Execution Count of Entire Merged Loop is the Same

- All loops being merged should be implemented under identical conditions.

```

for(i=0;i<100;i++) {
    ...
}
if( f ){
    /* if statement,hence merging is not possible*/
    for(i=0;i<100;i++) {
        ...
    }
}

```

- Statements "continue" and "break" that change control between multiple "for" loops do not exist.

```

while(...) {
    for(i=0;i<100;i++) {
        ...
    }
    if(...){ break;} /* NG */
    for(i=0;i<100;i++) {
        ...
    }
} →
for(i=0;i<100;i++) {
    ...
}
while(...) {
    ...
    if(...){ break;} /* OK */
    ...
}
for(i=0;i<100;i++) {
    ...
}

```

- "goto", "goto0", return, exit, labels, wait, watch do not exist among two or more "for" loops.
- Attribute "folding", "unroll_times", and "unroll_folding" provided for multiple "for" loops are the same.
- Inside "for" loops and among "for" loops, time-limit attributes such as 'interval_id' and 'latency_loop' do not exist.

16.4.4.4 Deciding Condition to Save Data Dependency Relation within the Loop

- Values of variables that differ in loop iterations in leading loops are not referred by variables defined in the succeeding loops.

```

for(i=0;i<100;i++) {
    x = a + i;
}
for(i=0;i<100;i++) {
    y = x;
} →
/* data dependency
relation of x is not
retained*/
for(i=0;i<100;i++) {
    x = a + i;
    y = x;
}

```

```

for(i=0;i<100;i++) {
    x = 1;
}
for(i=0;i<100;i++) {
    y = x;
} →
/* OK */
for(i=0;i<100;i++) {
    x = 1;
    y = x;
}

```

- The succeeding loops do not update variables referred by the leading loops.

```

for(i=0;i<100;i++) {
    b = x + i;
}
for(i=0;i<100;i++) {
    x = y;
}

```



```

/* data dependency relation
of x is not retained*/
for(i=0;i<100;i++) {
    b = x + i;
    x = y;
}

```

- Succeeding loops do not reassign variables specified by the leading loops

```

for(i=0;i<100;i++) {
    if (f) {
        x = a;
    }
}
for(i=0;i<100;i++) {
    if (f) {
        x = b;
    }
}
o = x;

```



```

/*data dependency relation
of x is not retained*/
for(i=0;i<100;i++) {
    if (f) {
        x = a;
    }
    if (g) {
        x = b;
    }
}
o = x;

```

16.4.4.5 Retaining data dependency Relation between loops

- In the case of merging "for" loops at the leading "for" loop:
 - Statements between the loops do not refer to variable that has been assigned or updated by succeeding 'for' loops.

```

for(i=0;i<100;i++) {
    ...
}
a = x;
for(i=0;i<100;i++) {
    x = y;
}

```



```

/* data dependency relation
of x is not retained*/
for(i=0;i<100;i++) {
    ...
}
x = y;
a = x;

```

- Variables referred in the succeeding "for" loops have not been set or updated between loops.

```

for(i=0;i<100;i++) {
    ...
}
x = a;
for(i=0;i<100;i++) {
    y = x;
}

```



```

/* data dependency relation
of x is not retained*/
for(i=0;i<100;i++) {
    ...
}
y = x;
x = a;

```

- o Statements between "for" loops do not update variables updated in the succeeding "for" loop.

```

for(i=0;i<100;i++) {
    ...
}
x = a;
for(i=0;i<100;i++) {
    x = y;
}

```



```

/* data dependency relation
of x is not retained*/
for(i=0;i<100;i++) {
    ...
}
x = y;
x = a;

```

- In the case of merging "for" loops at the succeeding "for" statement
 - o Statements between "for" loops do not refer to a variable updated in the leading "for" loop.

```

for(i=0;i<100;i++) {
    x = a + i;
}
y = x;
for(i=0;i<100;i++) {
    ...
}

```



```

/* data dependency relation
of x is not retained*/
y = x;
for(i=0;i<100;i++) {
    x = a + i;
    ...
}

```

- o Statements between the "for" loops do not refer to a variable updated in the leading "for" loop.

```

for(i=0;i<100;i++) {
    a = x + i;
}
x = y;
for(i=0;i<100;i++) {
    ...
}

```



```

/* data dependency relation of x
is not retained*/
x = y;
for(i=0;i<100;i++) {
    a = x + i;
    ...
}

```

- o Statements between "for" loops do not update a variable that is updated in the leading "for" loop.

```

for(i=0;i<100;i++) {
    x = i;
}
x = y;
for(i=0;i<100;i++) {
    ...
}

                         → /* data dependency relation of x is not
                           retained*/
                         x = y;
                         for(i=0;i<100;i++) {
                             x = i;
                             ...
                         }

```

16.5 Summary

This section covered the following:

- There are two types of 'for' loop unrolling.
 - o One where the loop is unrolled for all loop iterations.
 - o Another is where the loop is unrolled for a few iterations.
- In loop unrolling, indices used in array access are transformed into a constant by loop unrolling leading to speeding up of the access.
- In partial loop unrolling, process is unrolled within the loop only as per the number specified and iteration count of the loop is adjusted appropriately.
- Unrolling of a loop is based on the specification of attribute "unroll_times".
- When no attribute has been specified for "for" loop, loop unrolling is decided on the basis of the statement count or iteration count of loop.
- The loop counter value is not updated after loop unrolling if the "-Zloop_counter=not_use" option is specified.
- The optimization functionality tries to reduce the execution cycles by parallelizing assignment operations within the unrolled "for" loop statements.
- The merging functionality enables the aggregation of multiple "for" loops with identical structure at one place.
- It is important that the following conditions are satisfied while merging two or more "for" loops:
 - o Loop execution count is the same for "for" loops.
 - o The execution count of entire merged loop is the same.
 - o Condition to save data dependency relation within the loop is determined.
 - o Data dependency relation is retained.

17 Loop Folding

17.1 Common Scheduling Modes

17.1.1 Overview

This section covers the following:

1. Timing description at the time of folding of 'while' loop.
2. Timing Description at the time of folding of "for" loop.
3. Timing Description at the time of folding of 'do- while' loop.
4. Array Reference - specifying the exclusivity of subscript at the time of assignment.
5. Loop folding operation of 'for' loop.
6. Loop folding operation of 'do-while' loop.
7. Loop folding operation of "break" statement.
8. "wait" statement in the loop folding.
9. Loop folding operation for multiple "wait" statements.
10. Loop folding operation for coexistence of "wait" and "break" statements.

The term "loop folding" implies the initiation of the next iteration during execution of current iteration³ after a specified number of cycles. The number of shifted cycles is called Data Initiation Interval (DII).

Loop folding specifies attribute (folding) for each loop, and specifies DII as attribute value. Further, each DII state divided from loop head state is called as stage. In stage, numbers are assigned in sequence from loop head. For example, state from loop head till DII is called as stage 1. Loop folding will consider this stage as a pipeline stage and execute the pipelining."

Figure shows the behavioral image in case loop folding is carried out on below description with DII=1. **Figure** shows the behavioral image in case loop folding is DII=2 for the same description.

³ One time processing of loop such as for, while.

```

in ter(0:8) max_in, w_in;
out ter(0:32) dout;
out ter(0:1) fin;
process main()
{
    var(0:8) v, max, w;
    var(0:32) y, y1, y2, ary[256]/* Cyber rw_port=RW4 */;
    v = 0;
    max = max_in;
    w = w_in;
    /* Cyber folding=1 */
    while(v<max){
        y1 = ary[v] + ary[v + 1];
        y2 = ary[v + w] + ary[v + w + 1];
        y = (y1 + y2)/4;
        dout = y;
        v++;
    }
    fin = 1;
}

```

In the automatic scheduling, loop can be described by conducting nesting in the loop which specified folding (Refer section **17.1.3**) about restrictions). **Figure 50** shows the operation image when following description is done with loop folding.

```

/* Cyber folding=1 */
for(i = 0; i < N1; i++) {
    Processing A;
    /* Cyber folding=2 */
    for(j = 0; j < N2; j++) {
        Processing B;
        Processing C;
        Processing D;
    }
    ProcessingE;
    /* Cyber folding=1 */
    for(k = 0; k < N3; k++) {
        Processing F;
        Processing G;
    }
}

```

2- One processing for loops such as for, while

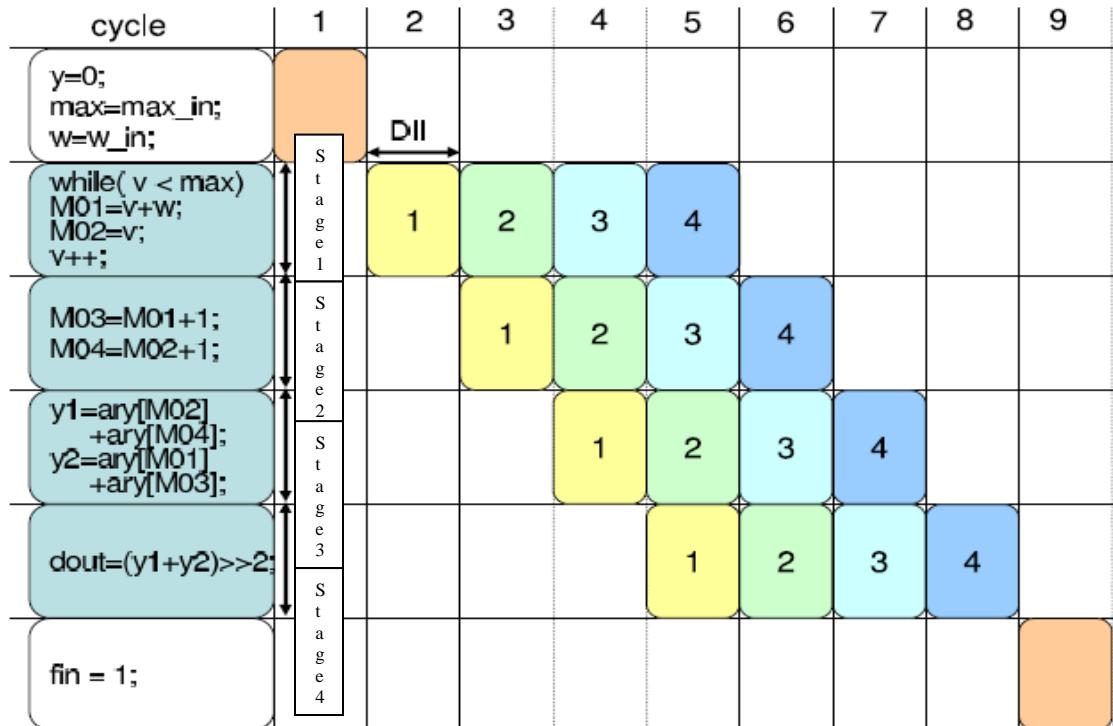


Figure 48: Behavioral image of loop folding (DII=1)

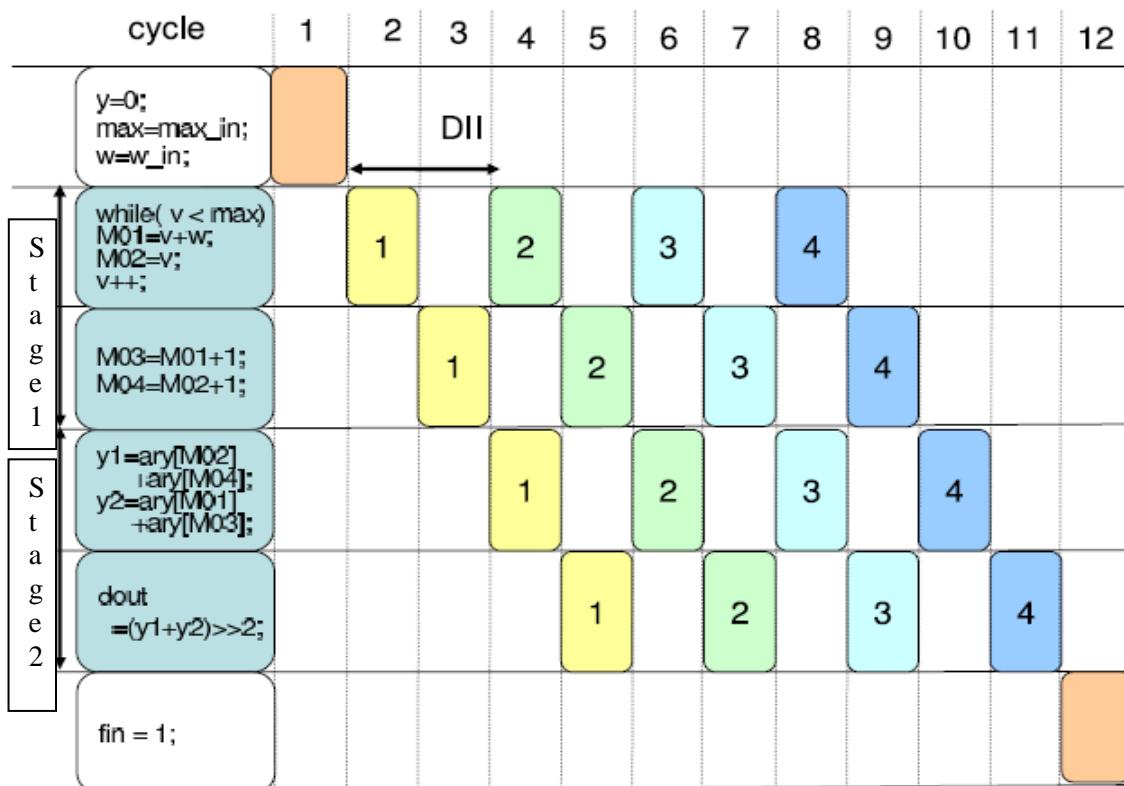
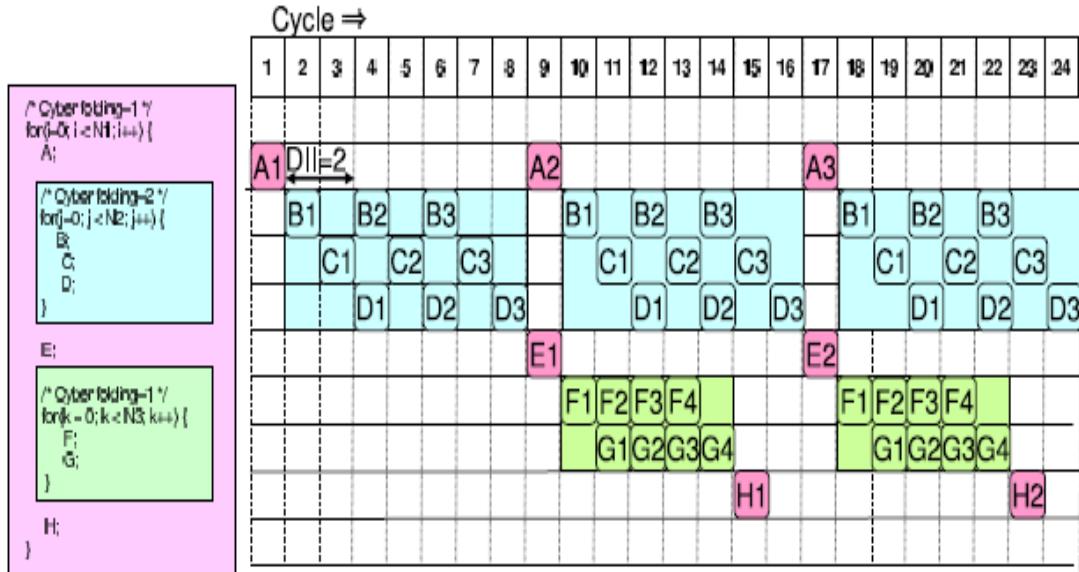
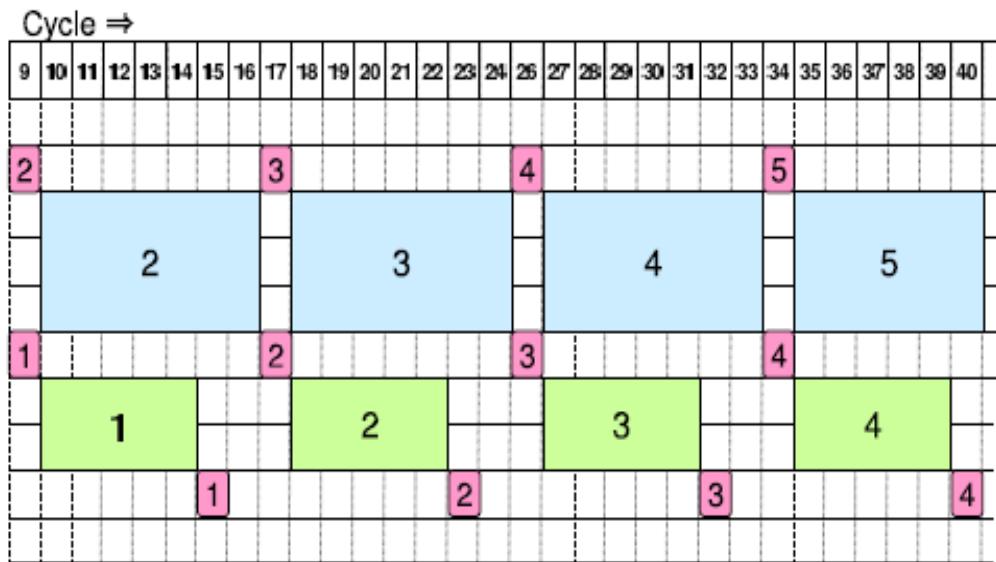


Figure 49: Behavioral image of loop folding ($DII=2$)**Figure 50:** Behavioral image of nested loop folding**Figure 51:** Image showing nested loop behavior in parallel

The nested loop operates concurrently at the same time. **Figure 51** shows that once the first loop completes second execution in 16th cycle, the following 2nd loop starts the second execution and

at the same time the first loop starts the third execution. In this way, the third ($N + 1$) execution of the first loop and second (N) execution of the second loop run parallel.

However, in each loop, it is not required to start the next execution until one execution ends. For instance, in the first loop as shown in **Figure 50**, the first execution is completed by executing D3 in the 8th cycle and the second execution is started by executing B1 again in the 10th cycle. Thus, N execution does not overlap with $N + 1$ execution.

DII of external loop does not become the value specified by the value of attribute folding and it is determined by the maximum value of required cycle count in the completion of first execution of internal loop. For instance, in **Figure 51** the blue colored loop completes one execution in 7 cycles and the green colored loop completes execution in 5 cycles. Therefore, DII of external loop becomes that maximum value for 7 cycles. DII of an external loop also changes at the same time when there is a change as per the data in which execution cycle of internal loop is processed.

Folding of nested loop is only supported for automatic scheduling mode. Moreover, folding attribute should not only be specified in external loop but in internal loop as well. For other restrictions please refer section **17.1.3**.

17.1.2 Related Attribute

Attribute	Description	Settings Target
<code>/* Cyber folding = # */</code>	The loop will be folded up by DII of # cycles.	for statement, while statement and do-while statement

17.1.3 Constraints

The constraints where loop folding cannot be applied to the loop body are enumerated below. These constraints are applicable in both manual and automatic scheduling modes.

- Loop including "goto" statement, "goto0" statement, "watch" statement, "return" statement, and exit() cannot be folded.
- Loop which contains function or functions synthesized with goto conversion cannot be folded.
- Loop in which referencing label exists cannot be folded.
- Loop folding is not possible when referencing is done in "allstates" statements.
- Loop which has "wait" statement present under a conditional statement cannot be folded.
- Loop which has "continue" statement cannot be folded. (In future releases, this constraint will be removed.)

There is an error when loop folding attribute is specified for the loop description having these restrictions.

When the loop is further nested inside the specified loop folding, there will be an error as synthesis will not be possible in the following cases:

- Manual scheduling mode
 - When internal loop cannot be unrolled
 - When internal loop can be unrolled but there is break statement in internal loop
- In case there is an error, change to any one of the following whichever is applicable
- change to loop that can be rolled
 - Use wait statement (in case of BDL)
- Automatic scheduling mode
 - When internal loop cannot be unrolled and when folding is not done
 - When internal loop can be unrolled but there is a break statement in the internal loop
 - When internal loop folding can be done but it is described under conditional statement (in "then" or "else")

In case there is an error, change to any one of the following whichever is applicable

- Folding is done by specifying folding attribute
- Change to loop that can be rolled
- Use wait statement (In case of BDL)

Constraints specific to other scheduling mode have been explained in respective sections.

17.2 Automatic Scheduling mode

17.2.1 Related Options

Options	Description
-Zarray_hazard=AVOID -Zarray_hazard=CHECK -Zarray_hazard=IGNORE	Insert the control circuit for avoiding the array hazard (default) Check the array hazard Ignore the array hazard
-Zport_hazard=AVOID -Zport_hazard=CHECK -Zport_hazard=IGNORE	Insert the control circuit for avoiding the port hazard (default) Check the port hazard Ignore the port hazard
-Zreg_hazard=AVOID -Zreg_hazard=CHECK -Zreg_hazard=IGNORE	Insert the control circuit for avoiding the register hazard (default) Check the register hazard Ignore the register hazard (shared reg variable also becomes the target)
-Zhazard_avoid_sche=ALL -Zhazard_avoid_sche=REG -Zhazard_avoid_sche=PORT -Zhazard_avoid_sche=REGARRAY -Zhazard_avoid_sche=MEM -Zhazard_avoid_sche=NO	Set the time constraint for suppressing the hazard (Valid only for variable having hazard AVOID specification.) Set the time constraint for suppressing the hazard for register (including shared reg variable) other than array (Valid only for variable having hazard AVOID specification.) Set the time constraint for suppressing hazard for Port (Valid only for variable having hazard AVOID specification.) Set the time constraint for suppressing hazard for register array (Valid only for variable having hazard AVOID specification.) Set the time constraint for suppressing hazard for memory (Valid only for variable having hazard AVOID specification.) Do not set time constraint for suppressing the hazard (Valid only for variable having hazard AVOID specification.) Default: -Zhazard_avoid_sche=REG:PORT
-Zarray_order=AUTO -Zarray_order=IGNORE	Automatically determines the array access sequence to be assured (default) Ignore the array access sequence
-Zunnesting=AUTO -Zunnesting=YES -Zunnesting=NO	Automatically determines nest deletion Does not delete nest, and there is error when it cannot be done. Does not delete nest (Default)

Options planned to be disused in future

Options	Description
-Zarray_data_hazard=AVOID	Avoid the data hazard of array
-Zarray_data_hazard=CHECK	Check the data hazard of array
-Zarray_data_hazard=NO	Do not check the data hazard of array

17.2.2 Related Attributes

Attributes	Description	Settings Target
/* Cyber array_hazard = AVOID */	Insert the control circuit for avoiding the array hazard	Array
/* Cyber array_hazard = CHECK */	Check the data hazard of array	Array
/* Cyber array_hazard = IGNORE */	Ignore the data hazard of array	Array
/* Cyber port_hazard = AVOID */	Insert the control circuit for avoiding the I/O port hazard	I/O variable
/* Cyber port_hazard = CHECK */	Check the data hazard of I/O port	I/O variable
/* Cyber port_hazard = IGNORE */	Ignore the data hazard of I/O port	I/O variable
/* Cyber reg_hazard = AVOID */	Insert the control circuit for avoiding the register hazard	Variable
/* Cyber reg_hazard = CHECK */	Check the data hazard of register	Variable
/* Cyber reg_hazard = IGNORE */	Ignore the data hazard of register (in reg_hazard attribute, shared reg variable is also used)	Variable

/* Cyber hazard_avoid_sche=YES */	Set the time constraint for suppressing the hazard (Valid only for variable having hazard AVOID specification)	Variable
/* Cyber hazard_avoid_sche=NO */	Do not set time constraint for suppressing the hazard (Valid only for variable having hazard AVOID specification)	Variable
/* Cyber array_order=AUTO */	Automatically determines the array access sequence to be assured	Array
/* Cyber array_order=IGNORE */	Ignore the array access sequence	Array
/* Cyber folding_ex_group = #[:#...] */	Specify the exclusiveness of subscript of array during loop folding	Array access
/* Cyber Unnesting=AUTO*/	Automatically determines nest removal	for statement
/* Cyber Unnesting=YES*/	Does not remove nest, and there is error when it cannot be removed	for statement
/* Cyber Unnesting=NO*/	Does not remove nest	for statement

Attributes planned to be disused in future

Attributes	Description	Settings Target
<code>/* Cyber pipeline_forwarding = NO */</code>	Do not synthesize signal as forwarding register	Variable
<code>/* Cyber group_interval_array = val */</code>	Set the time constraint specified by val for array	Array
<code>/* Cyber pipeline_interval_check = NO */</code>	Do not check the structural hazard	Signal
<code>/* Cyber array_data_hazard = AVOID */</code>	Avoid the data hazard of array	Array
<code>/* Cyber array_data_hazard = CHECK */</code>	Check the data hazard of array	Array
<code>/* Cyber array_data_hazard = NO */</code>	Do not check the data hazard of array	Array

17.2.3 Overview

In case of carrying out loop folding in auto scheduling mode, folding attribute is added to for loop, while loop and do-while loop, and DII is specified for attribute value, as given below. The data dependency is considered, and scheduling is done in a way that pipelining is possible by specified DII, and pipelining is done.

The problems during pipelining include data hazard, which occurs when original loop execution and equivalent result cannot be obtained once data flow is broken, and structural hazard, which occurs due to limited resources.

Data hazard can be categorized into the following three hazards.

- RAW (Read After Write) Hazard: It is generated when a value assigned in certain iteration is definitely referred in next iteration.
- WAR (Write After Read) Hazard: It is generated when a value referred in certain iteration is definitely assigned in next iteration.
- WAW (Write After Write) Hazard: It is generated when a value assigned in certain iteration is definitely assigned in next iteration.

By default, loop folding in auto scheduling mode will prevent most of such hazards. However, considering the cases i.e. when pipelining could not be done by specified DII, etc, the following three options and attributes are available.

- Insert control option (*_hazard = AVOID)

In order to prevent the occurrence of hazard during execution, this option will insert the control circuit, which will control the execution of next iteration. Though its performance will not be like specified DII, it will operate by smallest DII capable of avoiding the hazard through control circuit.

- Error check (*_hazard = CHECK)

This option will check the hazard occurrence during synthesis, and gives out error. It can generate circuit that will operate by specified DII for sure.

- Ignore hazard (*_hazard = IGNORE)

This option ignores the occurrence of hazard. It can be used for descriptions that assure that hazard will not occur, such as array description.

The objects, where hazard can occur, are classified as register, I/O port, memory port, and explanation regarding respective option operation is given below.

17.2.4 Register Hazard

Among register hazards, the one that requires caution is RAW hazard, generated when a value assigned in certain iteration is definitely referred in next iteration. WAR hazard is eliminated automatically by loop folding functionality. WAW hazard will not occur.

Shared reg variable is also included in the register referred here.

```
in ter(0:4) max_in, din;
out ter(0:8) dout;
process foo()
{
    var(0:4) v, max;
    var(0:8) y, ary[16];
    v = 0;
    max = max_in;
    /* Cyber folding=1 */
    while(v<max){
        y = ary[v] + din;
        if(y<16){
            v++;
        }
        dout = y;
    }
}
```

In the above case of while loop, the one which can possibly cause RAW hazard, is variable "v". Given below is the explanation regarding the operation of available options that act as measure against RAW hazard.

- Insert control circuit for avoiding hazard (reg_hazard = AVOID)

In case of selecting "Insert control circuit for avoiding hazard", a control circuit is automatically added where settings are such that unless previous iteration assigns the value of v, next iteration execution will not start. Hazard occurrence is prevented through this control circuit. For example, in case of above mentioned while loop, as there is a possibility that hazard may occur by v, the control circuit will work in a way that next loop condition will not be executed until $v++$ is executed. Refer to **section 17.2.9** for details of operation when execution wait occurs due to control circuit.

In this way, if this option is selected for descriptions generating hazard, a mechanism to avoid the hazard during execution will work, and the next iteration execution will not be started as per specified DII. As a result, there is possibility that expected performance may not be obtained, therefore, caution is required.

In case it is detected that DII value may not be fulfilled, a warning message given below will be generated, therefore, be careful about error files.

```
W_BT5651: Since there is data dependency between iteration of
variables (RG_v_05), processing of folded loop is stalled.
State 1 of stage 4 of next iteration cannot be started until
state 1 of stage 5 is completed.
[Action]Modify the description so that the related processes
are executed at the same stage (or within DII).
```

```
[Source line]
12(reg_hazard.c):    if(y<16) {
[Source line]
10(reg_hazard.c):    while(v<max) {
```

Further, in case this message is generated, normally operating circuit will be generated even without changing anything. However, by increasing the DII value until warning generation is over, sometimes, area can be shorten without any impact on process speed.

- Hazard check (reg_hazard = CHECK)

In case it needs to be assured that operation will be executed by DII value specified for attribute, a circuit, which does not insert above mentioned control circuit, can be generated by specifying the options and attributes to check the register hazard.

In case it is detected that register hazard can occur at the time of selecting this option and attribute, the following error message will be generated.

```
F_BT5651: Since there is data dependency between iteration of
variables (RG_v_05), processing of folded loop is stalled. State
1 of stage 4 of next iteration cannot be started until state 1
of stage 5 is completed.
[Action]Modify the description so that the related processes
are executed at the same stage (or within DII).
```

```
[Source line]
12(reg_hazard.c):    if(y<16) {
[Source line]
10(reg_hazard.c):    while(v<max) {
```

- Behavior while ignoring hazard (reg_hazard = IGNORE)
In case ignore register hazard is selected; control circuit will not be inserted even if variable, where hazard is occurring, is detected. Therefore, hazard will occur, and behavior as intended in the loop description will not be assured. Its usage is not recommended.

17.2.5 I/O port hazard

I/O port maintains the execution of access sequence of identical I/O ports as per loop description. Change of this execution sequence is considered as hazard.

For example, in the while loop below, din is accessed twice inside the loop, however, after din is referred for 2nd time in a certain iteration, it is mandatory that first din is referred in next iteration.

```
in ter(0:4) max_in, din;
out ter(0:8) dout;
process foo()
{
    var(0:4) v, max;
    var(0:8) x, y, z, ave;
    v = 0;
    max = max_in;
    /* Cyber folding=1 */
    while(v<max) {
        x = din;
        y = din;
        z = x + y;
        ave = z / 2;
        if(ave<16) dout = ave;
    }
}
```

During Synthesis, the following three behaviors can be selected for handling the hazards generated by din. Below is the explanation for behavior of respective options.

- Insert control circuit for avoiding hazard (port_hazard = AVOID)
In case of selecting "Insert control circuit for avoiding hazard", a control circuit is automatically added where settings are such that unless previous iteration assigns the value of din, next iteration execution will not start. Hazard occurrence is prevented through this control circuit. For example, in case of above mentioned while loop, as there is a possibility that hazard may occur by din when DII=1, the control circuit will work and operation will be executed by DII=2. Refer to **section 17.2.9** for details of operation when execution wait occurs due to control circuit.

Same as register hazard, if this option is selected, there is possibility that expected performance may not be obtained, therefore, caution is required.

In case it is detected that DII value may not be fulfilled, a warning message given below will be generated, therefore, be careful about error files.

W_BT5651: Since there is data dependency between iteration of variable (CH_din), processing of folded loop is stalled. State 1 of stage 4 of next iteration cannot be started until state 1 of stage 5 is completed.

[Action] Modify the description so that the related processes are executed at the same stage (or within DII).

```
[Source line]
12(port_hazard.c): y = din;
[Source line]
11(port_hazard.c): x = din;
```

- Hazard check (port_hazard = CHECK)

In case it needs to be assured that operation will be executed by DII value specified for attribute, a circuit, which does not insert above mentioned control circuit, but it can be generated by specifying the options and attributes to check the I/O port hazard. In case it is detected that I/O port hazard can occur at the time of selecting this option and attribute, the following error message will be generated.

F_BT5651: Since there is data dependency between iteration of variable (CH_din), processing of folded loop is stalled. State 1 of stage 4 of next iteration cannot be started until state 1 of stage 5 is completed.

[Action] Modify the description so that the related processes are executed at the same stage (or within DII).

```
[Source line]
12(port_hazard.c): y = din;
[Source line]
11(port_hazard.c): x = din;
```

- Behavior while ignoring the hazard (port_hazard = IGNORE)

In case ignore I/O port hazard is selected; operation is executed by specified DII without reporting the possibility of hazard even if detected. However, in case structural hazard, which causes resource conflict, has occurred, the behavior of respective reference and assignment of I/O port is defined against resource conflict as follows.

I/O port reference: single input port is shared for access from all stages.

Assignment to output port: assignment of iteration where execution is started later is given priority. (Assignment with larger stage number is given priority)

If it is assured that value is identical, even if referred from optional location inside loop, operation can be carried out as per the description by sharing conflicting resources.

It can be used for cases, where parameter, whose value will not change, is provided from outside during pipelining operation.

17.2.6 Array Hazard

Among array hazard, there is a possibility that RAW hazard, WAR hazard, RAW hazard will occur while accessing identical array. However, hazard will not occur in accesses having different index, even if array is identical. But by default, access of identical array is considered as a hazard, considering the possibility of identical index. However, depending upon the array access, there are cases when it can be assured that all accesses have different index. In such cases, the following types of attributes are available.

- Assure the index exclusiveness of all accesses

In case of following loop description, array ary is accessing identical index in each iteration; therefore, there is no dependency relation between array accesses even on carrying out pipelining. In such cases, it is assured that there is no dependency relation of arrays in each iteration by specifying array_order = IGNORE.

Specifications can be added to an array by specifying an attribute at the time of its declaration in the description or it can be added as an option if all the arrays are targeted in the description.

```

in ter(0:4) max_in, din;
out ter(0:8) dout;
process foo()
{
    var(0:4) v, max, i;
    var(0:8) y, ave, ary[16]/* Cyber array_order=IGNORE */;
    v = 0;
    max = max_in;
    /* Cyber folding=1 */
    for(i=0; i<max; i++){
        y = ary[i] + din;
        ave = y / 2;
        ary[i] = ave;
    }
    dout = ary[i];
}

```

- Assure the index exclusiveness of any specific access set

In case of following description, regarding assignment and reference of ary[i], it can be assured that there is identical index access in each iteration, therefore, hazard will not occur even on carrying out pipelining. However, regarding ary[j], there is a possibility that assignment and dependency relation of ary[i] will occur.

In such cases, identical ID is specified by folding_ex_group attribute against reference and assignment of ary[i], and it is specified that both array accesses are exclusive. Identical ID can be specified for 3 or more accesses, and mutual accesses inside the group with identical ID are considered exclusive.

```
in ter(0:4) max_in;
out ter(0:8) dout;
process foo()
{
    var(0:4) v, max, i, j;
    var(0:8) y, z, ary[16]/* Cyber rw_port=RW3 */;
    reg(0:8) a;
    v = 0;
    max = max_in;
    /* Cyber folding=1 */
    for(i=1, j=0; i<max; i++,j++) {
        y = ary[i]/* Cyber folding_ex_group = 1 */;
        z = ary[j];
        a = (y + z)/ 2;
        ary[i]/* Cyber folding_ex_group = 1 */ = a;
    }
    dout = ary[i];
}
```

Regarding the identical array accesses, which are not assured as exclusive, similar to I/Oport, it is considered that hazard will occur, and the following three option behavior can be selected during operation.

- Insert control circuit for avoiding hazard (array_hazard = AVOID)

In case "insert control circuit for avoiding hazard" is selected, in the above mentioned for loop example, a control circuit is automatically added, which does not start the referencing of ary[j] in next iteration until value of ary[i] is assigned in previous iteration. Hazard is prevented by this control circuit. For example, in the above mentioned for loop, in case assignment ary[i] is scheduled after ary[j] referencing in state 1, execution will be carried out so that ary[j] is referenced after waiting for assignment of ary[i] by DII=1. **Figure 52** shows the execution image. Refer to **section 17.2.9** for details regarding operation, in case execution wait occurs due to control circuit.

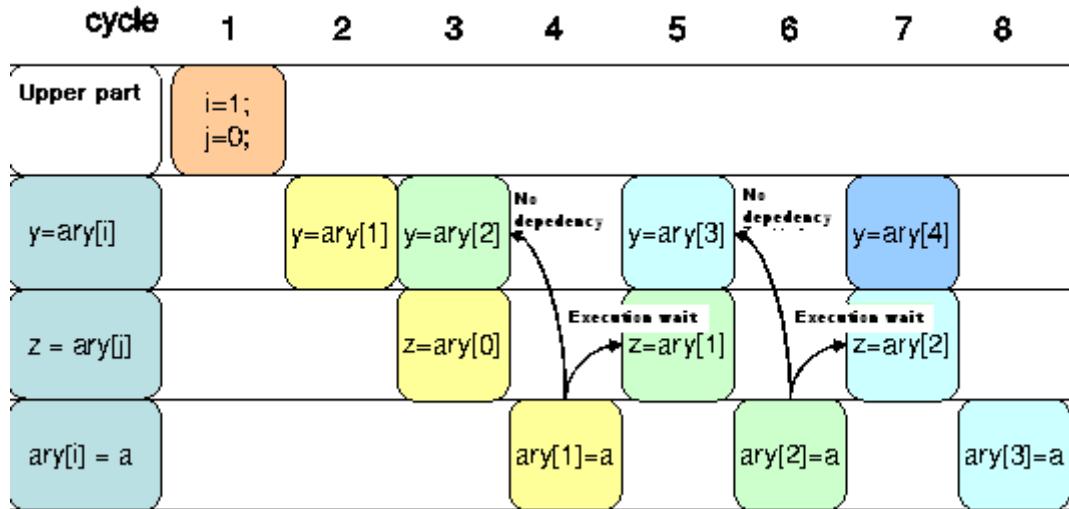


Figure 52: Hazard avoiding operation of array

Same as other hazards, if insert control circuit option is selected, there is a possibility that expected performance may not be obtained, therefore, caution is required.

In case it is detected that DII value may not be fulfilled, a warning as given below will be generated. At this time, decoder of register array and memory port is not sufficient. When pipeline operation cannot be done with the specified DII, the warning given below is generated.

In case this message is generated confirm whether the port count and decoder count specified in attribute and library file is sufficient for the specified DII or not.

```

W_BT5655: Processing of folded loop is stalled because of resource
conflict of array (ary).
Allocated resource: ary and Port:1
State that causes conflict: State 1 of stage 2 and state 1 of stage
3
[Action] Modify the description so that there is no conflict. Solve
it on priority because it can cause problems in pipeline processing
[Source line]
15(mem_hazard.c): z = ary[j]/* Cyber id = R1 */;
[Source line]
14(mem_hazard.c): y = ary[i]/* Cyber folding_ex_group =1 */;
```

Moreover, in case the decoder count of register array and memory port is sufficient but the possibility of data dependency between arrays is detected and pipeline operation cannot be done by the specified DII, the following warning is generated.

```

W_BT5652: Processing of folded loop is stalled because of data
dependency between iteration of array (ary)
State 1 of stage 1 of next iteration cannot be started till state
1 of stage 2 is completed.
[Action] Modify the description so that the related processes are
executed at the same stage (or within DII). In case it is ok to
change the order of access, specify option -Zarray_order = IGNORE
or attribute array_order = IGNORE.
[Source line]
17(mem_hazard.c):           ary[i] = ave;
[Source line]
15(mem_hazard.c):           z = ary[j];

```

- Hazard check (array_hazard = CHECK)

In case it needs to be assured that operation will be executed by DII value specified for attribute, a circuit, which does not insert above mentioned control circuit, can be generated by specifying the options and attributes to check the array hazard.

When it is detected that there is a possibility that the specified DII value cannot be satisfied, an error message is generated and synthesis is stopped. At this time, in case when pipeline operation cannot be done by the specified DII and decoder of register array, memory port are not sufficient, the following error is generated.

When this message is generated, it is confirmed whether the port count and decoder count specified in attribute and library file against the specified DII is sufficient or not.

```

F_BT5655: Processing of folded loop is stalled because of resource
conflict of array (ary).
Allocated resource: ary, and Port:1
State that causes conflict: State 1 of stage 2 and state 1 of stage
3
[Action] Modify the description so that there is no conflict. Solve
it on priority because it can cause problems in pipeline processing
[Source line]
15(mem_hazard.c): z = ary[j]/* Cyber id = R1 */;
[Source line]
14(mem_hazard.c): y = ary[i]/* Cyber folding_ex_group =1 */;

```

Moreover, in case the decoder count of register array and memory port is sufficient but the possibility of data dependency between arrays is detected and pipeline operation cannot be done by the specified DII, the following error is generated.

F_BT5652: Since there is data dependency between iteration of array (ary), processing of folded loop is stalled. State 1 of stage 1 of next iteration cannot be started till state 1 of stage 2 is completed.

[Action] Modify the description so that the related processes are executed at the same stage (or within DII). In case it is ok to change order of access, specify option -Zarray_order = IGNORE or attribute array_order = IGNORE.

```
[Source line]
17(mem_hazard.c):           ary[i] = ave;
[Source line]
15(mem_hazard.c):           z = ary[j];
```

- Behavior while ignoring the hazard (array_hazard = IGNORE)

In case ignore array hazard is selected; operation is executed by specified DII without reporting the possibility of hazard even if detected. However, in case structural hazard, which causes resource conflict such as decoder of register array and port of memory, has occurred, the behavior is defined against resource conflict as follows.

Access of iteration where execution is started later, is given priority. The value that refers to non priority access side is not assured. In case of assignment, assignment itself is ignored.

17.2.7 Scheduling constraint setting during specification of AVOID

In case option or attribute for inserting the control circuit for avoiding hazard in register, I/O port and array is used, time constraint within DII is set for each access and scheduling is done. Option in which this time constraint is not set is prepared. In command line option, time constraint is not set by specifying Zhazard_avoid_sche=NO.

In case user wants to set up time constraint against only a specific expression which may cause hazard, two methods are prepared. One method is to specify setting target type with option value and the other method is to specify by attribute value in a particular variable.

Option values are as following:

- -Zhazard_avoid_sche=REG : Specifies time constraint against register other than array (including shared reg variable)
- -Zhazard_avoid_sche=PORT : Specifies time constraint against port
- -Zhazard_avoid_sche=REGARRAY : Specifies time constraint against register array
- -Zhazard_avoid_sche=MEM : Specifies time constraint against memory
- -Zhazard_avoid_sche=ALL : The option -Zhazard_avoid_sche=YES which can be specified in the version (old) that specifies time constraint against all the above mentioned options is explained to be replaced with ALL.

In case it is required to set up time constraint against multiple types, the multiple options are linked with a colon. When it is required to set up time constraint against port and register, then -

Zhazard_avoid_sche=REG:PORT is set up. Default value is -Zhazard_avoid_sche=REG:PORT. These options and attributes are valid only for the variables in which AVOID is specified.

At the time of specifying AVOID, DII+1 pipelining operation is carried out even if DII time constraint is not satisfied, therefore, it is not necessary to strictly follow this time constraint. In case loop latency is increased due to increase in DII value specified for loop folding attribute, latency can be reduced by trying this option.

17.2.8 Loop folding break operation

If a break statement occurs in the folded loop, then the iteration which does not contain break statement executes till the loop end and the iteration where break statement occurs, halts its execution after the cycle next to break.

Below, **Figure 53** shows the behavioral image of loop description including break.

```
in ter(0:8) max_in, w_in;
out ter(0:8) dout;
out ter(0:1) fin;
process foo()
{
    var(0:8) v, max, w;
    var(0:8) y, y1, y2, a[256]/* Cyber rw_port=RW4 */;
    v = 0;
    max = max_in;
    w = w_in;
    /* Cyber folding=1 */
    while(v<max) {
        y1 = a[v] + a[v+1];
        y2 = a[v+w] + a[v+w+1];
        if(y2<16) break;
        y = (y1 + y2)/4;
        dout = y;
        v++;
    }
    fin = 1;
}
```

cycle	1	2	3	4	5	6	7	8	9
<code>y=0; max=max_in; w=w_in;</code>									
<code>while(v < max) M1=v+w; M2=v; v++;</code>		1	2	3	4	5	6	7	
<code>M3=M1+1; M4=M2+1;</code>		1	2	3	4	5	6		
<code>y1=a[M2]+a[M4]; y2=a[M1]+a[M3]; if(y2<16) break;</code>		1	2	3	4	5	6		
<code>dout=(y1+y2)>>2;</code>		1	2	3	4				
<code>fin = 1;</code>									Break Application

Figure 53: Basic operation of break

At this time, in the iteration that started the execution later on, the description which is not originally executed is already executed at the stage where break exists, but for this execution it returns to the same state as the one when it is cancelled internally and it is not executed during loop break out (In **Figure 53**, part with the oblique).

Operation such as access to shared memory, assignment of memory and assignment of value to output port gets executed in such stages before break condition is applied then it will be impossible to cancel the impact or to change the values of memory which have been updated in such execution. Therefore caution is required.

In automatic scheduling, in case of loop folding description if execution of statement such as assignment to output port or assignment to memory is found before break condition is applied, the following error will be generated.

W_BT5651: Since there is data dependency between iterations of variable (CH_din), processing of folded loop is stalled. State 1 of stage 1 of next iteration cannot be started till state 1 of stage 2 is completed.

[Action] Modify the description so that the related process are executed at the same stage (or within DII)

[Source line]
12(test.c): dout1 = i;

Since control circuit is inserted and execution of operation whose impact cannot be reversed or cancelled are made to wait till the break execution is done, warning will be reported that operation will not be carried out as per specified DII.

17.2.9 Loop folding wait operation

Iteration in which wait is applied will operate same as in case loop folding is not done. When *wait* statement is encountered in an iteration, execution of operation in other stages will be as explained below:

All the stages which are greater than the stage with wait (or Iteration which can be executed before a stage having wait statement) will continue its execution.

All the stage which are smaller than the stage with wait (or iteration which can be executed only after wait state is released) will stop further execution and wait till wait stage is complete.

The behavioral image in case wait is scheduled at state 2 of stage 2 is shown in **Figure 54**. If wait condition of stage 2 is applied in 3rd iteration, the 3rd stage (stage with number larger than stage having wait) will continue the execution of 2nd iteration. Stage 1 (Stage with number smaller than stage having wait) will continue the execution till state 3 of stage 1, and from 7th cycle onwards, it will wait for lower stage to finish the execution by state 3.⁴

⁴ -Zhazard_avoid_sche=YES which can be specified in previous version can be explained by substituting with ALL

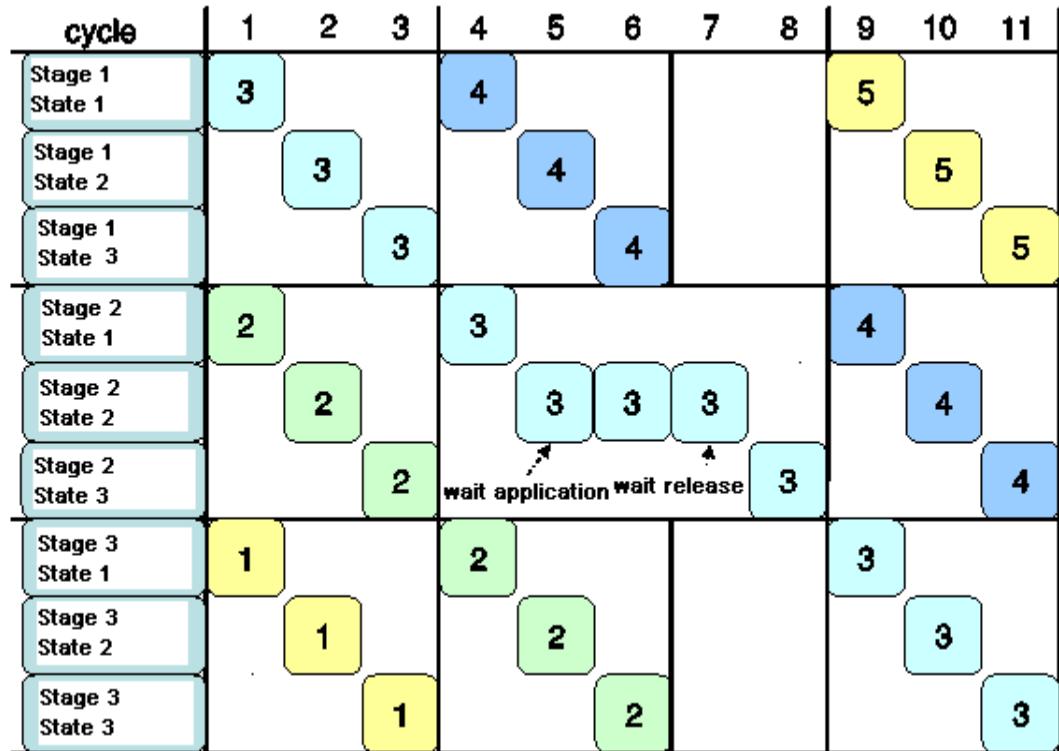


Figure 54: Basic operation of wait

This operation can also be applied when control circuit is inserted and execution is stopped through control circuit.

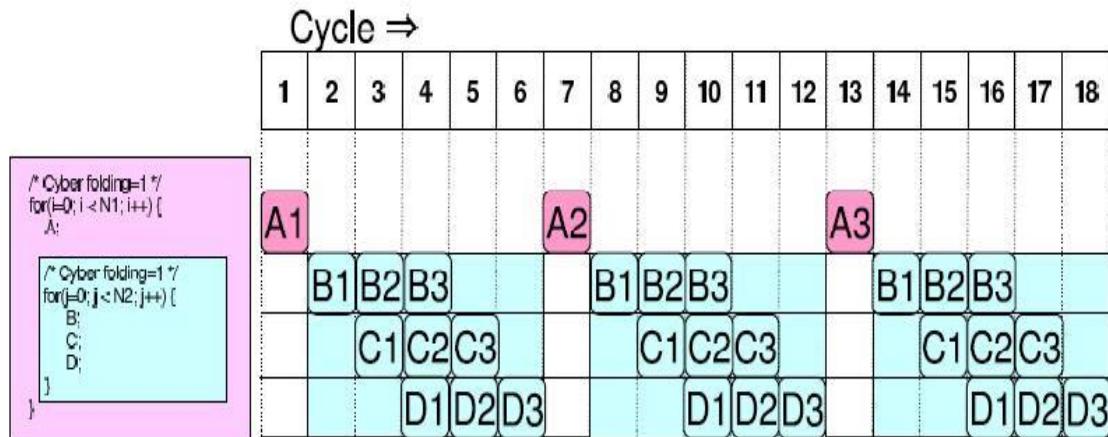
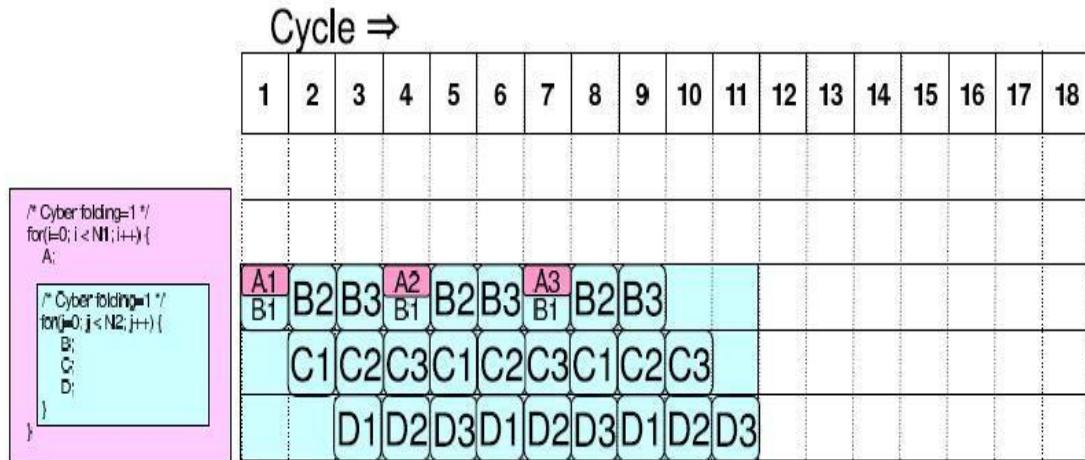
17.2.10 Release of Nested loop folding

In the nested loop folding the Nth execution of each loop does not overlap with N+1 execution. The process of loop can be linked and executed by unnesting from the attribute.

```

/* Cyber folding = 1 */
for (i = 0; i < N1; i++) {
    Processing A;
    /* Cyber folding = 1, unnesting = AUTO */
    for (j = 0; j < N2; j++) {
        Processing B;
        Processing C;
        Processing D;
    }
}

```

**Figure 55:** Behavioral image before Unnesting**Figure 56:** Behavioral image after Unnesting

Attributes and options to unnest loop folding are explained here. In the 2 loops for unnesting, the attribute will be specified in the internal loop.

- Automatic determination of Unnesting

It can be automatically determined whether unnesting can be done or not by specifying unnesting=AUTO attribute, and –Zunnesting =AUTO option. In case there is a restriction in this, then unnesting is not conducted.

- Enforced unnesting

Unnesting can be enforced by specifying unnesting=YES attribute and –Zunnesting=YES option. Unnesting is done even if there is a possibility of dependency. The error ends when it belongs to restrictions other than these.

- Prohibition of unnesting

Unnesting is prohibited by specifying unnesting=NO attribute and -Zunnesting=NO option.

Characteristics and restrictions of the loop for which unnesting is possible are shown below. Here the 2 loops for unnesting are mentioned as "Internal loop" for the loop which is inside and "external loop" for loop which is outside.

Loop which can be unnested has the following characteristics

- It is possible to statically determine that internal loop is executed one or more times.
- In the body of the external loop, there is no folding loop other than the internal loop.

```
/* Cyber folding = 1 */
for (i = 0; i < 128; i++) {
    /* Cyber folding = 1, unnesting = AUTO */
    for (j = 0; j < 128; j++) {
        /* Cyber folding = 1, unnesting = AUTO */
        for (k = 0; k < 128; k++) {
            Processing A;
        }
    }
}
```

Moreover, there are restrictions when:

- the characteristics of loop for which unnesting can be done are not present
- performance deteriorates even if unnesting is done
- Options or description which do not correspond to unnesting are specified

Detailed restrictions are given below:

The following loop does not do unnesting even when unnesting=AUTO is specified but if unnesting=YES is specified, enforced unnesting is conducted. However, the performance may deteriorate after this.

- When there is a possibility of resource conflict or data conflict after unnesting

In the body of external loop, there is a description to cause resource conflict or data conflict with the body of the internal loop. When that total exceeds the respective DII value, it is judged that hazards can occur and the performance may deteriorate. The description that can cause resource conflict or data conflict is as following:

- Input output terminal
- Memory
- Shared register, shared memory
- When there is a variable that refers in top side of internal loop inside external loop by writing in the internal loop

Loop synthesized by any one of the following conditions does not conduct unnesting even if unnesting=YES is specified.

- Conditions that become restrictions of internal loop
 - in case of loop statements other than "for" statement
 - when it cannot be determined after body is executed one or more times.

In the following example, the number of executions is not clear because loop break condition of the internal loop is input terminal. Therefore, unnesting cannot be done.

```
in ter(0:8) i1;
/* Cyber folding = 1 */
for (i = 0; i < N1; i++) {
    Processing A;
    /* Cyber folding = 1, unnesting = AUTO */
    for (j = 0; j < i1; j++) {
        Processing B;
    }
}
```

- When loop counter is changed in the part other than resetting part of loop
- When there is break statement in the body of loop
- When there is an unnesting folding loop in the body

```
/* Cyber folding = 1 */
for (i = 0; i < N1; i++) {
    /* Cyber folding = 1, unnesting = AUTO */
    for (j = 0; j < N2; j++) { /* (1) */
        /* Cyber folding = 1 */
        for (k = 0; k < N3; k++) { /* (2) */
            Processing A;
        }
    }
}
```

In this example, in the body of loop of (1), as there is a folding loop (2) that does not do unnesting, unnesting of (1) cannot be done.

- Conditions for restrictions of external loop
 - When there is wait(), watch() and \$ in scheduling_block in the body of external loop
 - When there is folding loop other than internal loop in the body of external loop

```

/* Cyber folding = 1 */
for (i = 0; i < N1; i++) { /* (1) */
    /* Cyber folding = 1, unnesting = AUTO */
    for (j = 0; j < N2; j++) { /* (2) */
        Processing A;
    }
    Processing B;
    /* Cyber folding = 1 */
    for (k = 0; k < N3; k++) { /* (3) */
        Processing C;
    }
}

```

In this example, (1) is external loop and (2) is internal loop. Here, unnesting cannot be done as there is folding loop (3).

- Conditions for other restrictions
 - When DII value is different in Internal loop and external loop
 - In case of pipeline synthesis
 - In case of SystemC description

17.2.11 Restrictions in auto scheduling mode

This section explains about the restrictions in auto scheduling mode.

1. Multicycle functional units

Loop which has multi-cycle functional units or operators cannot be folded. In case there are functional units or operators whose execution is not possible in single cycle; Cyber will give an error.

2. "for" loop folding

Normally while using "for" loop, resetting part is scheduled in the last block, however, if loop folding is specified, resetting part has to be scheduled in the starting or head block. This is because optimization is done as per the type which used loop counter of "for" statement.

In the resetting part of loop like the one in the following example, when the assigned variable (d) in the loop is referred, depending on the description, a variable other than the one in the loop counter causes the hazard. When there is a complicated description in the resetting part of "for" statement, caution is required as the following warning or error is generated. When the warning and error cannot be avoided, "while" statement should be described which is equivalent to "for" statement.

```
in ter(0:8) din;
out ter(0:8) dout;
process main() {
    var(0:8) i, sum, ary[10];
    var(0:2) d;
    reg(0:8) x, z;
    x = din;
    sum=0;
    /* Cyber folding=1 */
    for(i=0; i<16; i+=d) {
        z = x + ary[i];
        sum = z + ary[i+1];
        d = sum % 4;
        dout = sum;
    }
    dout = sum;
    dout = i;
}
```

FBT5651: Since there is data dependency between iterations of variable (RG_d_13), the processing of folded loop is stalled. State 1 of stage 11 of next iteration cannot be started till state 1 of stage 13 is completed.

[Action] Modify description so that the related processes are executed at the same stage (or within DII)

```
[Source line]
23 (test_for.c) :           d = sum % 4;
[Source line]
20 (test_for.c) :           for(i=0; i<16; i+=d){
```

3. "do-while" loop folding

Sometimes it is difficult to understand the cause from the source line or variable name but as it corresponds to the case where register hazard occurs, it is required to use a greater DII value same as the action for the register hazard.

In case, assignment is done inside loop for variable referred by do-while conditional expression, loop is finished with following error or warning, depending upon the specified DII value. Sometimes it is difficult to understand the cause from the source line or variable name but as it corresponds to the case where register hazard occurs, it is required to use a greater DII value same as the action for register hazard

F_BT5651: Since there is data dependency between iterations of variable (RG_03_14), the processing of folded loop is stalled. State 1 of stage 12 of next iteration cannot be started till state 1 of stage 14 is completed.

[Action] Modify description so that the related processes are executed at the same stage (or within DII)

```
[Source line]
26(test_do.c):      }while(sum<i);
[Source line]
26(test_do.c):      }while(sum<i);
```

17.3 Manual Scheduling Mode

17.3.1 Related Attributes

Attributes	Description	Settings Target
/*Cyber unroll_folding = # */	It does the folding by #	for statement
/*Cyber folding_ex_group = # [:#...]*/	It specifies the exclusiveness of additional character of array at the time of loop folding.	Array access

17.3.2 Overview

In order to use loop folding in "for" statement, "while" statement, and 'do-while' statement, the cyber attribute "folding" needs to be added with these statements.

The attribute value is taken as the DII. The difference of operations between the executions of loop when it is folded verses execution of loop when it is not folded is explained in Figure 57. The example illustrates loop folding in manual scheduling mode.

```
in ter(0:8)    a,b;
out ter(0:8)   c;
process folding_module(){
    reg(0:8)  i,m,r;
    m = 0;
    /* Cyber folding = 2 */
    for( i = 0 $; i< 10; i+$){
        r = a + m;
        $
        m = b + i;
        $
        $
        c = r;

    }
    $
}
```

In **Figure 57**, the "for" loop will be executed in 40 cycles (4 cycles×10 times) if the loop is not folded. On the other hand, If the loop is folded with a DII of 2 cycles, then it will get executed in 22 cycles (2 cycles+2 cycles× (10-1) times+2 cycles=22 cycles).

In this case, the following points (related to loop carried dependencies) must be taken into account:

- Variables referenced after an assignment in loop process/body (variable "r" in example of **Figure 57**).

While executing the loop folding, if a value is used in current iteration but gets modified in the next iteration, care must be taken to use the current iteration value of the variable. As shown in the example, the value used for variable "r" in step 4 should be of the first iteration because in step 3 the next LOOP value has been assigned to the variable "r".

In this case, the register variable will be allocated so that the loop-wise values are stored in separate registers. Therefore, the number of register will increase as result of synthesis.

- Variable referred before the assignment in loop process (Variable "m" in example of **Figure 57**).

The assigned value in certain LOOP will be referred in the next LOOP. When loop folding is executed, if the reference is in the previous step or the same step of the assignment, then the data dependency occurs. Therefore, loop folding cannot be done thereby resulting in an error.

The operations shown in Figure 57 is further illustrated in Figure 58 with DII = 1. In this case, the variable "m" will not fulfill the dependency relationship. Therefore, it will result in the F_BT4665 error as shown below.

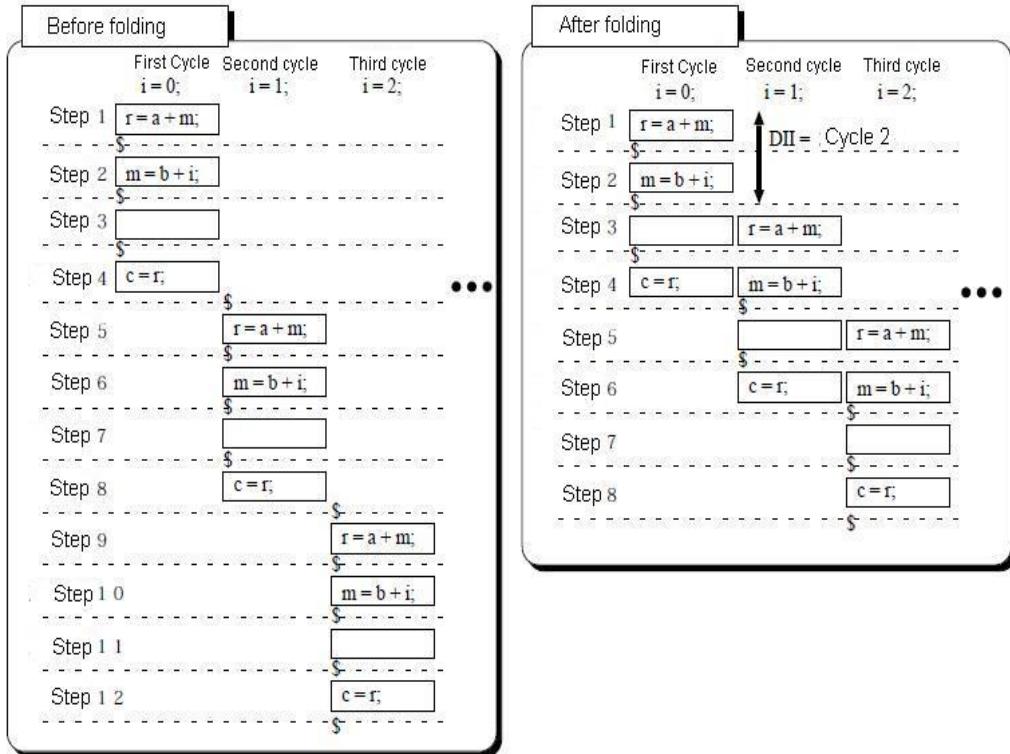


Figure 57: Loop folding of 'for' statement, Example No 1

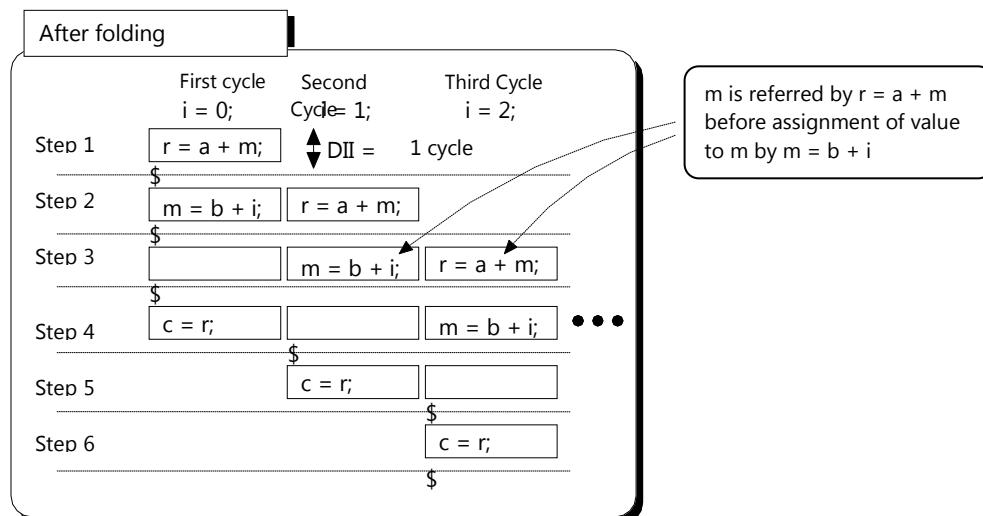


Figure 58: Loop folding of 'for' statement, Example No 2.

F_BT4665: register m reference is above previous DII than the reading [Counter Measure] In the loop, the reference will not execute more or before the DII than the reading.

[Source line]
 9(fold1.bdl): r = a + m;
 11(fold1.bdl): m = b + i;

- Input (output) variable that are referred (assigned) in multiple cycles in the loop.

There are input variables that are referred in multiple cycles in the loop. If these variables are referred multiple times in the same cycle because of loop folding, it will result in an error

Similarly, there are output variables which are assigned in multiple cycles in the loop. If they are assigned in the same cycle by loop folding, it will also result in an error.

In this case, the error can be avoided either by increasing the DII value or by splitting the I/O variable into multiple I/O variables.

In **Figure 57**, when the input variables "a" and "b" are made into the same variable "a" then the input variable "a" is input twice in two cycles for two different stages. When synthesis is performed with loop folding (with DII=1), input variable "a" will be input twice in one cycle for two different stages. This will result in an error as shown in Figure 59.

In this case, the error can be avoided either by dividing the input variable "a" into two variable as "a" and "b" or by making the DII as two cycles.

F_BT4910: The access of I/O port 'a' in the loop to be folded is not executed in 1 stage.

[Counter Measure] The I/O port not to be shared, reduction of the I/O access frequency.

[Source line]
 9(fold1.bdl): r = a + m;
 11(fold1.bdl): m = a + i;

Figure 59: Error due to multiple cycle reference of input variable.

17.3.3 Loop Folding of “while” Statement

At the time of loop folding of “while” statement, even though the \$ can be placed at any of the specified positions, it is mandatory to place it at the 4th and 5th position.

```

process A;
$ ←(1)
While ( c1) $ { ← (2)
    $ ← (3)
    process B;
    $ ← (4)
}
$ ← (5)
process C;

```

At the time of loop folding, the operation will be different from the steps whose explanations are provided in the BDL manual. Following is the explanation regarding the steps at the time of loop folding of "while" statement along with the example description of \$.

The loop folding operation of "while" will be like the basic operation as mentioned in the loop folding overview.

In the normal step operation, the operation of LOOP 2 starts after the completion of LOOP1. However, in the loop folding operation, the operation of LOOP 1 starts followed by the DII step, and then the operation of LOOP 2 begins.

	while loop example	cycle image(normal)	cycle image(folding)																																								
1	<pre> A; \$ B /* Cyber folding = 1 */ while(C{ D; \$ E; \$ F; \$ } \$ G; </pre>	<p style="text-align: center;">t</p>	<table style="width: 100%; text-align: center;"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr> <td>A</td><td></td><td></td><td></td></tr> <tr> <td>B,C,D</td><td></td><td></td><td></td></tr> <tr> <td>E</td><td>C,D</td><td></td><td></td></tr> <tr> <td>F</td><td>E</td><td>C,D</td><td></td></tr> <tr> <td>C,D</td><td>F</td><td>E</td><td>C</td></tr> <tr> <td>E</td><td></td><td></td><td></td></tr> <tr> <td>F</td><td></td><td></td><td></td></tr> <tr> <td>C</td><td></td><td></td><td></td></tr> <tr> <td>G</td><td></td><td></td><td></td></tr> </table>	1	2	3	4	A				B,C,D				E	C,D			F	E	C,D		C,D	F	E	C	E				F				C				G			
1	2	3	4																																								
A																																											
B,C,D																																											
E	C,D																																										
F	E	C,D																																									
C,D	F	E	C																																								
E																																											
F																																											
C																																											
G																																											

In the table shown above, the "while" loop is executed three times. The loop pulls out of the operation at the fourth conditional statement where the difference between the previous and the next operation of loop folding is DII=1.

In the loop folding operation at DII=1, the next second LOOP operation starts after the starting of

the first LOOP.

	while loop folding	cycle image(normal)	cycle image(folding)																																																												
2	<pre>A; \$; B /* Cyber folding = 2 */ while(C){ D; \$; E; \$; F; \$ } \$; G;</pre>	<p style="text-align: center;">t</p>	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;">1</td> <td style="width: 15%;">2</td> <td style="width: 15%;">3</td> <td style="width: 15%;"></td> <td style="width: 15%;"></td> <td style="width: 15%;"></td> </tr> <tr> <td>A</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td>B, C, D</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td>E</td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td>F</td> <td>C, D</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td>E</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td>F</td> <td>C, D</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td>E</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td>F</td> <td>C</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td>G</td> </tr> </table>	1	2	3				A							B, C, D							E							F	C, D						E					F	C, D						E					F	C							G
1	2	3																																																													
A																																																															
	B, C, D																																																														
		E																																																													
			F	C, D																																																											
				E																																																											
			F	C, D																																																											
				E																																																											
			F	C																																																											
					G																																																										

The table shown above illustrates the difference in the operation when DII=1 in the loop is replaced with DII=2. In the loop folding step operation of DII=2, LOOP 1 starts first and then after two steps LOOP 2 starts.

17.3.4 Loop Folding of "for" Statement

The example shown below illustrates the "for" statement that specifies loop folding. In the "for" statement, \$ can be described in the spaces marked with boxes (□) as shown in the example below. At the time of loop folding, '\$' is required to be described at the boxes in position (2). Further, in order to cut the cycle at the end of the loop execution, there is a need to specify the \$ in any of (1) or (4). Besides, if '\$' is not placed at position (3), initialization timing of loop counter can be incorrect and thus, a warning will be generated.

```
process A;
  □
  for(i=0, □ j=0 □ (3) ; i < x; i++, j++ □ (4)) □ {
    □
    process B;
      □ ← (1)
    }
    □ ← (2)
  process C;
```

At the time of loop folding, the operation will be different from the steps whose explanation is provided in the BDL manual.

In the table provided below, the operation step at the time of loop folding of 'for' statement has been explained with the \$ description example. In addition, the table shows the comparison between the normal step operation and loop folding step operation.

In case of 'for' loop, the resetting portion D will be executed at the last step of the loop in normal step operation. However, in loop folding operation, it will be executed at the DII step from the starting of the loop. When it executes resetting portion at DII step from the beginning of the loop and the result differs from normal operation, it will be an error.

	for loop example	cycle image(normal)	cycle image(folding)																																
1	<pre>A; /* Cyber folding = 1 */ for(B\$; C; D\$){ E; \$ F; \$ G; \$ } \$; H;</pre>	<p style="text-align: center;">t</p>	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>A, B</td><td></td><td></td><td></td></tr> <tr><td>C, E, D</td><td></td><td></td><td></td></tr> <tr><td>F</td><td>C, E, D</td><td></td><td></td></tr> <tr><td>G</td><td>F</td><td>C, E, D</td><td></td></tr> <tr><td>G</td><td>F</td><td>C</td><td></td></tr> <tr><td>G</td><td></td><td></td><td>G</td></tr> <tr><td>H</td><td></td><td></td><td></td></tr> </table>	1	2	3	4	A, B				C, E, D				F	C, E, D			G	F	C, E, D		G	F	C		G			G	H			
1	2	3	4																																
A, B																																			
C, E, D																																			
F	C, E, D																																		
G	F	C, E, D																																	
G	F	C																																	
G			G																																
H																																			

The above specified table shows a case where:

- 'for' loop takes 3 cycles for execution
- Loop pulls out at 4th conditional judgment
- Difference between consecutive operations of folded loop is DII=1.

As explained earlier, in the loop folding operation, resetting portion 'D' is executed at step DII=1 from loop beginning, and in case of result differing from normal operation, it results in an error. In such cases, DII value can be set to a larger value to make synthesis possible.

	for loop example	cycle image(normal)	cycle image(folding)																																																				
2	<pre>A; /* Cyber folding = 2 */ for(B\$; C; D\$){ E; \$; F; \$; G; \$ } \$; H;</pre>	<p style="text-align: center;">t</p>	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> </tr> <tr> <td style="border: 1px solid black; padding: 5px; text-align: center;">A, B</td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">C,E</td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">F,D</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">G</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">C,E</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">F,D</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">G</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">C,E</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">F,D</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">G</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">C</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="border: 1px solid black; padding: 5px; text-align: center;">H</td> </tr> </table>	1	2	3	4	A, B					C,E					F,D					G				C,E				F,D				G				C,E				F,D				G				C				H
1	2	3	4																																																				
A, B																																																							
	C,E																																																						
		F,D																																																					
			G																																																				
			C,E																																																				
			F,D																																																				
			G																																																				
			C,E																																																				
			F,D																																																				
			G																																																				
			C																																																				
			H																																																				

The table shown above illustrates the difference between consecutive operations of loop folding if the DII=2 instead DII=1.

In loop folding operation when DII=2, the resetting portion D is executed at 2nd step from the loop beginning. Compared with the case of DII=1, the operation of resetting portion D is delayed for 1 cycle and there is more possibility of avoiding the loop folding error.

17.3.5 Loop Folding of 'do- while' Statement - 1

In a "do- while" statement, it is necessary to describe \$ in positions (4) and (5) during "do-while" statement loop folding.

```
process A;
□ ← (1)
Do {
    □ ← (2)
    process B;
    □ ← (3)
} while ( c2) □ ; ← (4)
□ ← (5)
process C;
```

At the time of loop folding, the step action that takes place is different from what has been explained in the manual of BDL language. Here, the step action during loop folding of "do-while" statement has been explained along with the \$ description example.

In a "do- while" statement, the execution of condition determination takes place at the end of the loop. Consequently during loop folding, before it is decided that loop should proceed with condition determination, the next loop starts executing.

The execution image has been displayed in **Figure 60** below. In the figure, the cycle image (folding) shows that loop folding should be implemented in such a way that the execution of C, D statements in 4th iteration and C statement in 5th iteration should not get executed.

Even if they are executed, circuit should be operated in such a way that action of these statements should not corrupt behavior w.r.t normal operation. However, if output to external port and writing on memory exist in C, D part, that operation cannot be stopped therefore to carry out the loop folding of do-while loop, the procedure discussed in the next section must be followed.

	do-while example	cycle image(normal)	cycle image(folding)																																													
1	<pre>A; \$; B; /* Cyber folding=1 */ do { C; \$; D; \$ } while (E) \$; \$; F;</pre>	<p style="text-align: center;">t</p>	<table style="width: 100%; text-align: center;"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr> <tr> <td>A</td><td></td><td></td><td></td><td></td></tr> <tr> <td>B,C</td><td></td><td></td><td></td><td></td></tr> <tr> <td>D</td><td>C</td><td></td><td></td><td></td></tr> <tr> <td>E</td><td>D</td><td>C</td><td></td><td></td></tr> <tr> <td>C</td><td>E</td><td>D</td><td>C</td><td></td></tr> <tr> <td>D</td><td></td><td>E</td><td>D</td><td>C</td></tr> <tr> <td>E</td><td></td><td></td><td>E</td><td>D</td></tr> <tr> <td>F</td><td></td><td></td><td></td><td>F</td></tr> </table>	1	2	3	4	5	A					B,C					D	C				E	D	C			C	E	D	C		D		E	D	C	E			E	D	F				F
1	2	3	4	5																																												
A																																																
B,C																																																
D	C																																															
E	D	C																																														
C	E	D	C																																													
D		E	D	C																																												
E			E	D																																												
F				F																																												

Figure 60: Difference of do-while loop operation (Example of DII=1)

17.3.6 Loop Folding of 'do- while' Statement - 2

The "do- while" loop folding that has been explained in this section, makes it necessary to describe \$ in position (3) in addition to the description of \$ in positions 4 and 5.

In the normal step operation, the conditional judgment E is executed at the last step of the loop. However, at the time of loop folding, the conditional judgment E is executed in advance at the DII step from the loop beginning. Therefore, the operation is different from the normal step operation, an error will occur.

Condition determination E is executed at the last step of loop in general step operation. In the step operation during loop folding, execution is done in advance in the DII step from the

beginning of loop. At this time, an error occurs when general step operation and operations changes by the execution of condition determination in advance.

	do-while example	cycle image(normal)	cycle image(folding)
1	<pre> A; \$; B; /* Cyber folding=1 */ do { C; \$; D; \$ } while (E) \$; \$; F; </pre>	<p style="text-align: center;">t</p>	<p style="text-align: center;">1 2 3 4</p>

Figure 61: Difference of 'do-while' loop operation (Example of DII=1)

The above mentioned **Figure 61** shows the case of the 'do- while" loop, which takes 3 cycles for execution in the loop, is executed 3 times. Moreover, the loop terminates at the third conditional judgment where the difference between the previous and next operation of loop folding is DII=1.

The conditional judgment E is executed one step before the beginning of step C of next repetition. An error will occur in case the result obtained by the execution of the conditional judgment statement, "E", one step prior to the execution of next iteration, is different from the normal operation. In such cases, synthesis can be done by setting the DII to a larger value.

If the folding of "do- while" statement is done with DII=1 as shown in the Figure 61, the conditional judgment E is executed at the same step with the first executed statement C of the loop.

Therefore, assignment in the loop will not be possible due to data dependency relationship.

	do-while example	cycle image(normal)	cycle image(folding)
2	<pre>A; \$; B; /* Cyber folding=2 */ t do { C; \$ - \$ } while (E) \$; \$; F;</pre>		

Figure 62: Difference of 'do-while' loop operation (Example of DII=2

When compared with behavior of DII=1, the execution of conditional judgment statement E will be delayed by one clock cycle in case of DII=2. So there are less chances of error occurrence due to lesser data dependency problems in DII=2 in comparison to DII=1.

17.3.7 Loop Folding and Loop Unrolling of 'for' loop

Even if loop folding is executed, loop unrolling can be executed, it is present in the above description of loop folding.

In case unrolling is possible in "for" loop of fixed cycle, it is possible to execute loop folding along with loop unrolling as the array access subscript is constant. In that case, the "unroll_folding" attribute should be specified above the "for" statement and DII should be specified in the attribute value. This attribute can only be used at the time of manual scheduling mode. Moreover, it does not support break statement and wait statement .

```
/* Cyber unroll_folding = 2 */
For ( i = 0 ; i < 10; i++ )
```

17.3.8 Array Reference - Specifying the Exclusivity of Subscript

At the time of array reference, assignment is done similarly as other variables, in the loop which is executed using folding. If the referencing is done before assignment, the following error will occur due to occurrence of data dependency.

```
F_BT4664: Array M reference is above and before DII than the reading.  
[Counter Measure] The reference will not be executed above and before DII than  
the reading in the loop.  
10 (file2.bd1) : r ( 0:8) = (unsigned ter (0:8) ) (M [ I (1:7)] (0:8) + m  
(0:8) );  
16 (file2.bd1) : M [ i_1 ( 1:7) ] (0:8) = r ( 0:8) ;
```

```
in ter(0:8) b;  
outside mem(0:8) M[100];  
  
process folding_mem(){  
    reg(0:8) i,m,r;  
  
    m =0;  
    /* Cyber folding = 2 */  
    for( i =0 ; i <100; i++){  
        r = M[i] + m;  
        $;  
        m = b + i;  
        $;  
        $;  
        M[i] = r;  
        $;  
    }  
    $;  
    $;  
}  
$;
```

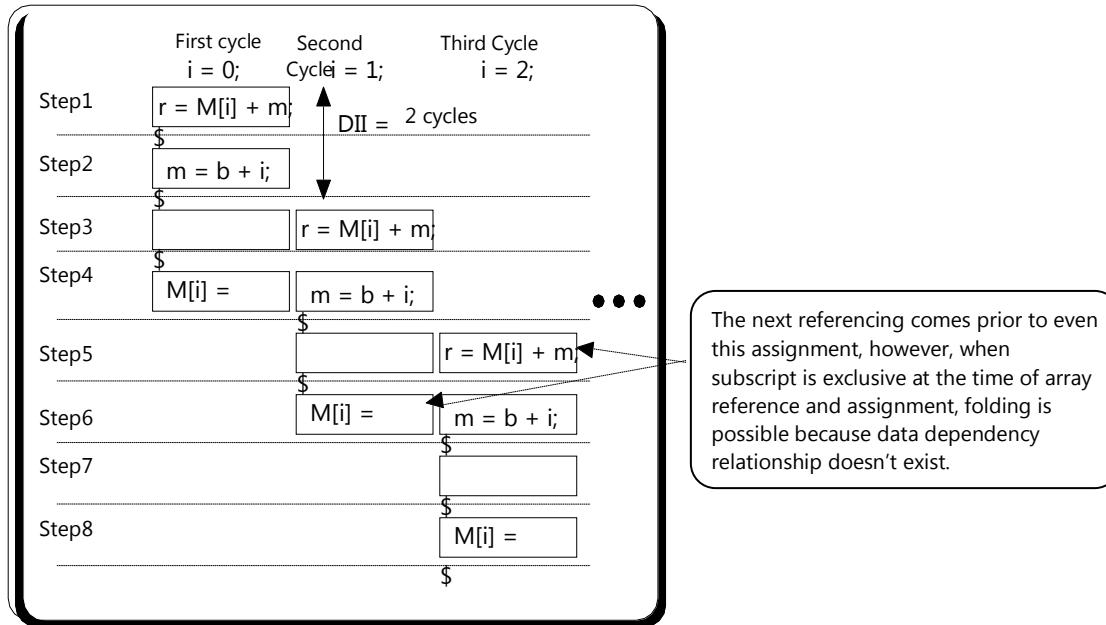


Figure 63: Usage example of the "folding_ex_group" attribute

In case it is assured that subscript value is exclusive at the time of array reference and assignment, the data dependency relationship gets eliminated. Therefore, folding can be executed even in the case as mentioned above.

Regarding the above mentioned reference ($r = M[i] + m$) of array M and assignment ($M[i] = r$) to array M, folding can be possible. This is because array subscript at the time of assignment to array M of i th cycle and array subscript at the time of reference of array M of $(i + 1)$ th are exclusive.

However, there is a need of specifying it by using the "folding_ex_group" attribute because it is difficult to analyze the exclusivity of the array subscript statically. Therefore, the group number needs to be specified in the attribute value. The array subscript is considered to be exclusive to the array reference and assignment for which the same group number is specified.

```

in      ter(0:8)    b;
outside   mem(0:8)  M[100];

process folding_mem() {
    reg(0:8) i,m,r;
    m =0;
    /* Cyber folding = 2*/
    for( i =0 ; i<100; i++) {
        r = M[i]/* Cyber folding_ex_group = 1*/ +m;
        $;
        m = b + i;
        $;

        $;
        M[i] /* Cyber folding_ex_group = 1*/ = r;
        $;
    }
    $;
    $;
}

```

In the case of specifying multiple groups in one array reference and assignment, then multiple group names can be specified by putting ":" as shown below.

```

R = M[i]/* Cyber folding_ex_group = 1:3 */ + m;
:
:
:
If(flag) {
    M[i]/* Cyber folding_ex_group = 1 */ = r;
} else{
    M[i]/* Cyber folding_ex_group = 3 */ = a;
}

```

17.3.9 Restrictions in manual scheduling mode

At present, a number of restrictions exist in the loop folding in manual scheduling mode. Given below are points to be noted while using loop folding in manual scheduling mode.

- Loop without break

Loop folding functionality can be used without any problem in manual scheduling mode, if the loop is infinite and with no "break" statement.

Example showing possible synthesis

```
in ter(0:8) in1;
out ter(0:8) out1;

process main()
{
    unsigned char param1, param2;
    unsigned char x, y, z;

    param1 = in1;
$;
    param2 = in1;
$;
    /* Cyber folding=1*/
    while(1){
        x = in1;
$;
        y = param1 + x;
$;
        z = param2 + y;
$;
        out1 = z;
$;
    }
}
```

- Loop with break

In case loop has "break" statement, error can be avoided if only simple operation is described in the first state after break and the description should be such that it is always executed. Loop statements that describes \$ in body should avoid the descriptions that complicate the transfer of state immediately after exiting the loop i.e. if loop, switch-case loop.

Example showing possible synthesis

```
in ter(0:8) in1;
out ter(0:8) out1;

process main()
{
    param1 = in1;
$;
    param2 = in1;
$;
    /* Cyber folding=1*/
    while(c){
        x = in1;
$;
        y = param1 + x;
$;
        z = param2 + y;
$;
        out1 = z;
$;
    }
$;
//Example where state after loop break is
considered for simple operation
    out1 = z + y;
$;
}
```

- for loop

In case DII applies 2 or more loop folding in "for loop", the variable, which is used as loop counter, holds incorrect value after loop break. Therefore, there is no need to reference the loop counter variable after loop break.

Example showing incorrect synthesis

```

in ter(0:8) in1;
out ter(0:8) out1;

process main()
{
    param1 = in1;
$;
    param2 = in1;
$;
    /* Cyber folding=2*/
    for (i=0$; i<c; i++) {
        x = in1;
$;
        y = param1 + x;
$;
        z = param2 + y;
$;
        out1 = z;
$;
    }
$;
    out1 = i; //Referencing of loop counter "i"
    will give out incorrect result
}

```

Besides this, in case array and I/O port are described in initialization part of "for loop", resetting part, conditional part of do-while statement, there are cases when error may occur. Refer to release notes for such errors.

17.3.10 “break” statement in the loop folding

When a break appears in loop, breakout of the loop is the most basic operation as at that time the processing is immediately stopped. However, when multiple break is specified in the loop, the break statement that breaks the loop when folding is not conducted is considered to be the statement that breaks the loop is folding as well. In any case, "break" statement operation in the folding loop, is consistent with the processing order while unfolding. Concrete example is as given below:

- When there is single description for "break" in the loop

In **Figure 64** (DII=1) given below, when break occurs at stage 3 (5th cycle) loop will be terminated immediately and LOOP3 execution of process will be discarded. During this time, execution of LOOP1 and LOOP2 process has been completed and accepted. This process sequence is similar to 'break' behavior of unfolded loop.

- When multiple 'break' statements have been specified in the loop

In **Figure 65** (DII=2), 2 breaks happen at different timings (6th cycle and 7th cycle respectively) but the break of LOOP2 is applicable in this case. So when there are multiple breaks in the loop, Cyber

breaks LOOP according to 'break' occurring in an earlier loop and is not related to timing sequence of breaks.

17.3.11 Loop folding break statement limitations

- Positional limitations of output statements etc.

Specification of following statements at a place which is DII cycles prior (or earlier) to "break" statement is not allowed.

1. Output Statement
2. mem assignment, shared reg assignment

Consider the example (DII=2) as shown in **Figure**. "Break" statement in the seventh cycle has been executed in LOOP2. In this case, the output statement has already been processed in LOOP3 in the sixth cycle.

This implies that the processes, which are after the loop where break statement execution occurred, are executed thereby losing the actual purpose of the break statement. This results in an error in Cyber as shown below.

Moreover, in this example, the only possible position where output statement can be specified is the later half of stage 2.

<pre>process main(){ /*Cyber folding=1*/ while(i<10) { i++; /*stg1*/ a=in1+b; \$ /*stg2*/ out2=a; d=in2; \$ if(c){/*stg3*/ e=a+d; break; }else{ e=d+4; } \$ f=e+3; /*stg4*/ \$ g=f+in3; /*stg5*/ } \$ out1 = g; /*out of LOOP*/ }</pre>		cycle	1	2	3	4	5	6	7
		c	0	0	0	0	1		
	stage1	LOOP1	LOOP2	LOOP3	LOOP4 (X)			LOOP4 execution is canceled	
	stage2		LOOP1	LOOP2	LOOP3			break statement is executed	
	stage3			LOOP1	LOOP2	LOOP3 (X)		and terminates the while loop	
	stage4				LOOP1	LOOP2			
	stage5					LOOP1	LOOP2		
	out of LOOP								

Figure 64: Operation for break of loop folding

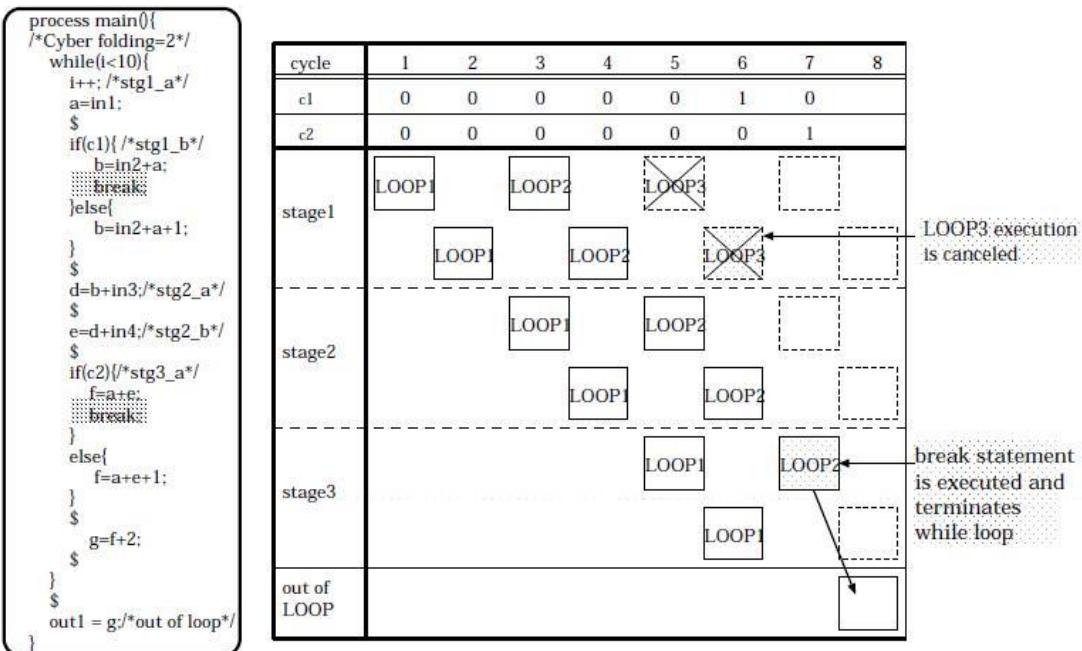


Figure 65: Operation for break of loop folding (in case of multiple breaks)

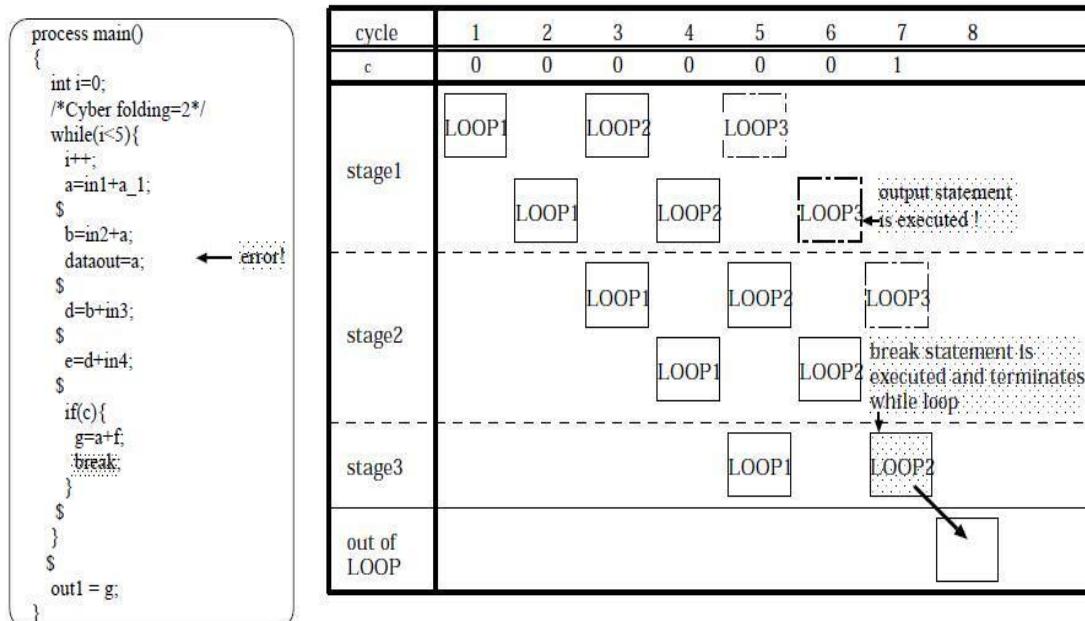


Figure 66: Positional Limitation of output statement (Example of DII=2)

F_BT4914: The output signal `dataout` has been described before `break` and `DII` in the loop.

[Counter Measure] Reset the `DII` to a larger value.

[Source line]

```

25(ex2.bdl): while(i<5){
30(ex2.bdl): dataout=a;

```

17.3.12 "wait" Operation in loop folding

This section covers the operations when "wait" statement is described in the loop where folding is to be executed.

When DII=1

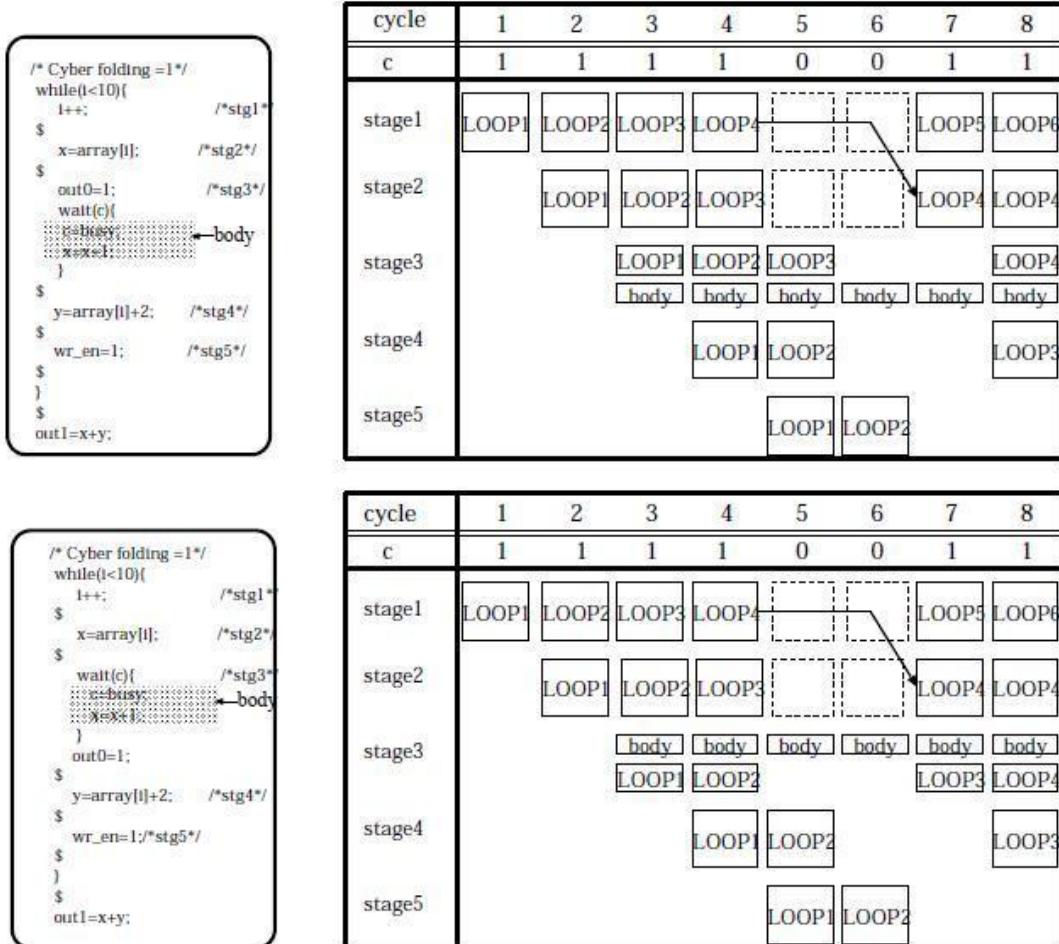
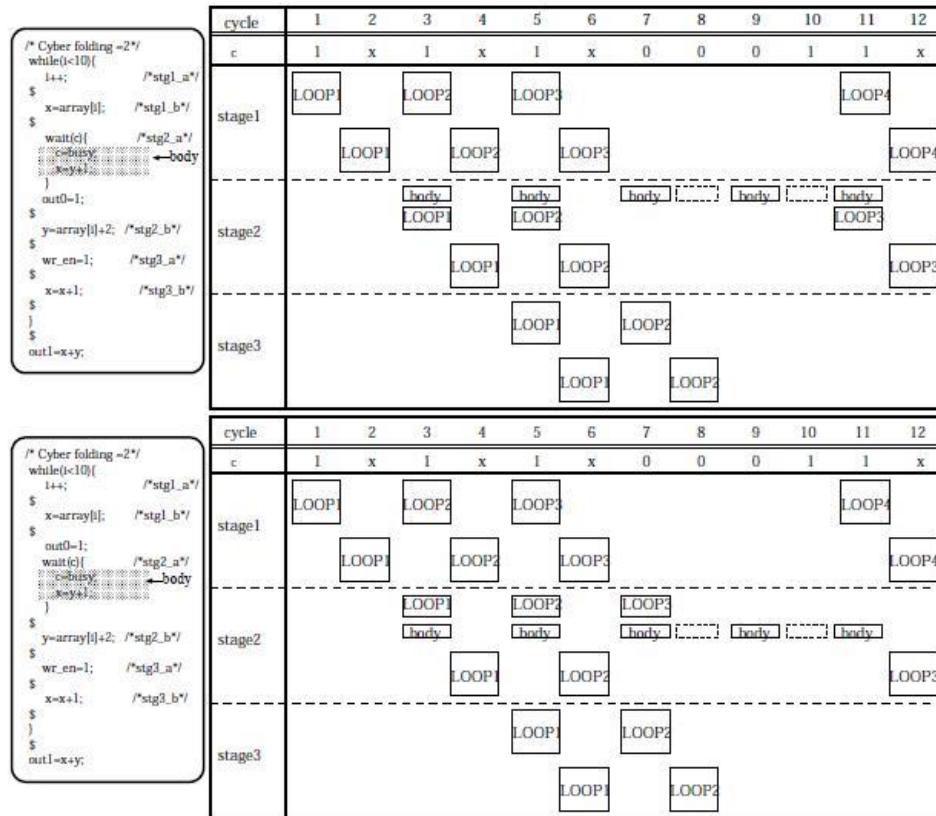


Figure 67: Operation for wait loop folding (DII=1)

When DII=1, suppose "wait" statement in the loop is executed at certain time, at that time, prior LOOP iteration continue running and body of 'wait' statement of that LOOP continues running. The later LOOP iteration is not started. This is illustrated in

Figure . In both the programs, the "wait" statement has been described in the third stage. When in the fifth cycle 'c' becomes '0' and *wait* is executed, the explanation of the operations is provided below.

**Figure 68:**Operation for *wait* loop folding (DII=2)

When *wait* is executed, LOOP1 and LOOP2 are already moving to the fourth stage. Therefore, the process will continue as it is. On the other hand, LOOP3 will execute the *body* process of *wait* statement of the third stage until *wait* is pulled out (in the figure, *wait* will be terminated in the seventh stage). As LOOP3 is under execution, so LOOP4 will not move to the next stage and will remain as it is.

- When DII is more than 2

From now onwards, DII=N ($N \geq 2$).

At certain point of time in the loop when *wait* is executed, previous loops continue the process and the next loop's starting point of process is same with the process of DII=1. However, the difference with the case having DII=1 is that the execution of the body part is **once in N cycle** when the *wait* state is being executed in the folding loop.

In both the examples of DII=2 shown in **Figure 68**, *wait* statement is described in the former half of stage 2. The explanation about the operation of each loop when *wait* is executed, i.e. in the seventh cycle when c=0, is given below.

Both LOOP1 and LOOP2 will continue the processing as it is similar to the case when DII=1. On the other hand, LOOP3 will execute the process of *wait statement body* of stage 3 until *wait* is pulled out. Here, while *wait* is executed, the execution of body part is done **once in 2 cycles** (at every even numbered cycle in the example), and not in every cycle.

As a result in this example, "wait" will be pulled out in the eleventh cycle. Moreover, LOOP3 is executing the "wait". Therefore, LOOP 4 cannot start the process in the seventh cycle (LOOP3 will pull out wait), and it will start the process only at the eleventh cycle.

Further, wait operation of DII= 2 or more in manual scheduling is different, at present, from auto scheduling mode. In future release, it is planned to match with specifications of auto scheduling.

17.3.13 Multiple "wait" Loop Folding Operation

Explanations regarding the operation in the case of multiple description of "wait" statement in the loop are given in this section. When the condition is set in one wait statement in certain timing, then the operation will be the same as the explanation made in the previous section. On the other hand, in case of simultaneous set of multiple "wait" condition at certain timing, then only the body part is executed for the earliest LOOP, not the body of other LOOPS. This operation is the same as DII =N ($N \geq 2$).

Let us consider the example illustrated in **Figure 69** (DII=1).

cycle	1	2	3	4	5	6	7	8	9	10
c1	1	1	1	1	0	0	0	0	1	1
c2	1	1	1	1	1	0	0	1	1	1
stage1	LOOP1	LOOP2	LOOP3	LOOP4					LOOP5	LOOP6
stage2	LOOP1	LOOP2	LOOP3		body	body	body	body	body	body
stage3	LOOP1	LOOP2	LOOP3						LOOP4	
stage4				body	body	body	body	body		LOOP3
stage5				LOOP1	LOOP2					LOOP3

```
/* Cyber folding =1*/
while(i<10){
    i++;
    /*stg1*/
    $ 
    x=array[i];
    /*stg2*/
    $ 
    wait(c1);
    /*stg3*/
    if(wait1)
        x=x+1;
    body
}
out0=1;
$ 
wait(c2);
/*stg4*/
x2=x+2;
$ 
array[1];
$ 
out1=y;
$ 
wr_en=1;
/*stg5*/
$ }
```

Figure 69: Operation for multiple "wait" loop folding(DII=1)

Here, the "wait" statement has been described in the second and fourth stage. In addition, the condition signals c1 and c2 are given in the pattern as shown in figure. First of all, c1 becomes 0 in the fifth cycle. Then, in LOOP4, wait of stage2 is set and the body part is executed.

In the sixth cycle, the "wait" condition of the third stage in LOOP3 is set and the body part is executed. Here, **during the set of the "wait" condition of LOOP3, the wait body part of LOOP4 is not executed.**

17.3.14 Operation for Coexistence of "wait" and "break" Loop Folding

This section covers the operation when both "wait" and "break" statements are described in the loop where folding is to be executed.

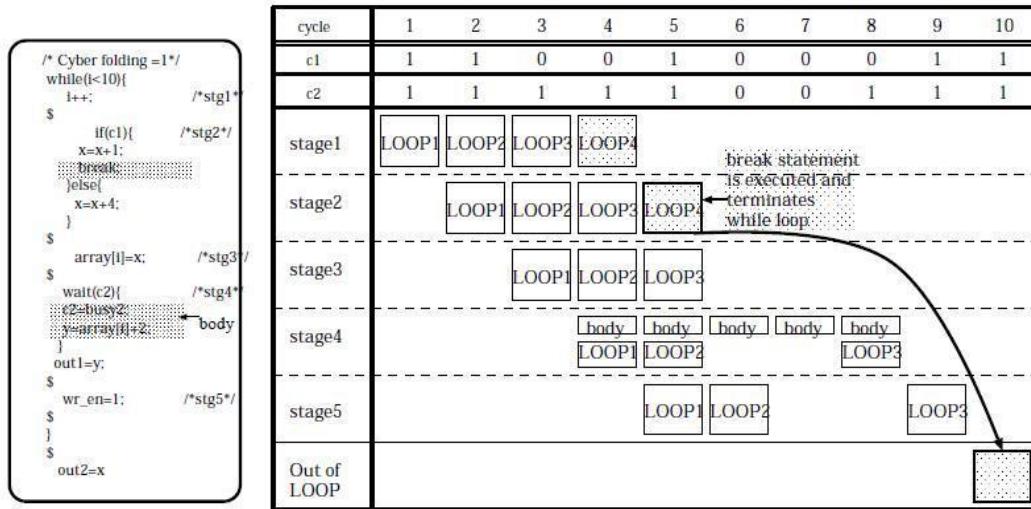


Figure 70: Mixed operations of "wait" and "break" statement of loop folding (DII=1)

Figure 70 illustrates an example where:

- At the fifth cycle in LOOP4, "break" is executed.
- At the sixth cycle in LOOP3, "wait" condition is turned ON.

In this case, the cycle in which LOOP4 gets terminated by the "break" statement is after the cycle in which the "wait" is completed for process in LOOP3.

In addition, **Figure 71** illustrates an example where:

- At the fifth cycle in LOOP3,"break" is executed.
- At the seventh cycle in LOOP4, wait has been set.

This time, since the *break* is set in LOOP3, the processes from LOOP4 onwards will be invalid. Accordingly, *wait* process will also be invalid.

As shown above, even when the *wait* statement is coexisting with *break* statement, it:

- Let all the processes (that are before the loop where 'break' is set) to be completed.
- Invalidates all the processes after the loop where *break* is set.

And it corresponds to the process order when folding is not executed.

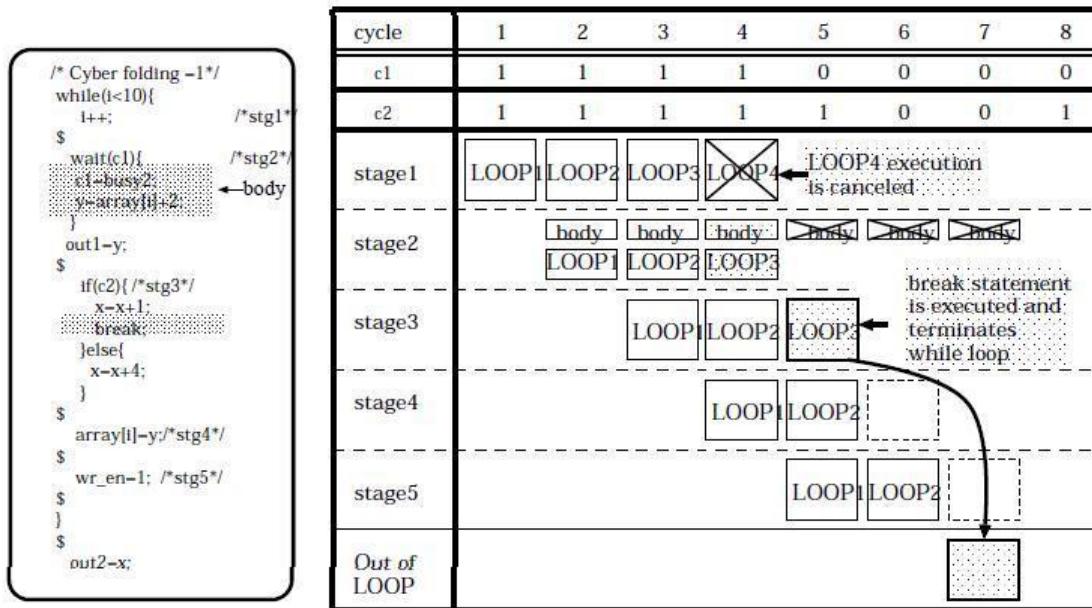


Figure 71:Mixed operations of "wait" and "break" statement of loop folding (DII=1)

17.3.15 Limitations for Loop Folding "wait"

- When the assigned register is referred in the same condition as "wait" is placed
As it can be seen in the description of **Figure 72**, the value of "r1" register, which has to be assigned to the output port 'out1', differs depending on the "wait" statement execution frequency. In other words, when the execution frequency is one, the value of the register is "in1". Moreover, the value of the register is "in2" when the execution frequency is two or more.
- In other words, if the conditions listed below are satisfied, Register assignment statement is done before one cycle of wait statement.
- In case there is a statement in same cycle as that of 'wait' process and specified before wait, and which is same as assignment statement in point 1, and further there is an assignment in wait statement which refers to same register , it will result in following error:

 - In case referencing and assignment is carried out for identical variable inside wait statement, or when multiple wait statements are described inside loop

In **Figure 73**, the contents of register x, which is referenced in variable assignment processing $x=x+1$ inside wait loop, are differing depending upon the execution frequency of wait statement. In other words, in case execution frequency is 1, it is $array[i]$, and in case execution frequency is 2 or more, it is x as assigned in previous wait loop. Now, as given below the **Figure 73** if wait is simultaneously applied at LOOP2, 3 in 7th cycle, wait processing of LOOP3 will get stopped, due to operation of multiple wait (Refer to **section 17.3.13**). As a result,

register x, which is assignment destination in process $x=x+1$ inside wait loop, will become unstable, therefore, when multiple wait loop are specified, description of assignment process to carry out self referencing in wait loop is the limitation.

Therefore, when multiple "wait" statements are described, description of the assigned process where self reference is done in the "wait" statement is not allowed.

Moreover, even if reference and assignment are executed with respect to same variable "var" in the *wait* statement, due to same reason it will be a description limitation.

But when the "wait" statement description is made at one place in the loop, then that will not be a limitation.

2. Pipeline Memory and Pipeline Function

Folding where pipeline memory and pipeline function co-exist with a 'wait' statement is restricted.

```
/* Cyber folding = 1 */
while(cond){
    r1 = in1;
    $
    r1 = in2;
    wait(c){
        out1 = r1;
    }
    out2 = r1;
    $
    out1 = r1;
    $
}
```

Figure 72: When the assigned register is referred in same state as *wait* is placed

F_BT4896: Since the register to be referred differs depending on the *wait* statement condition, so it can not do the parallelization of signal r1 by folding.

[Counter Measure] Either modify the DII of folding or in the same condition; do not execute the reference with the assignment which placed the *wait* statement.

```
[Source line]
18(a.bdl): r1 = in2;
22(a.bdl): out2 = r1;
```

```

in ter c,c1;
in ter (0:8)in0;
out ter (0:8)out0,out1;
reg(0:8) w,x,y,z;
reg(0:8) array[10];
process main(){
    int i;
    /* Cyber folding =2*/
    while(i<10){
        i++;
        x=array[i];
        $ 
        out0=x;
        wait(c){
            x=x+1;
        }
        $ 
        w=in0+1;
        $ 
        y=array[i]+w;
        $ 
        wait(c1){
            z=y+2;
        }
        x=array[i]+x;
        $ 
    }
    out1=x+z;
}

```

cycle	1	2	3	3	4	5	6	7	8	9	10	11
c	1	1	1	1	1	0	0	0	0	1	1	1
c1	1	1	1	1	1	1	0	0	1	1	1	1
stage1	LOOP1	LOOP2	LOOP3							LOOP4		
	body	body	body						body	body		
stage2		LOOP1	LOOP2						LOOP3			
		body	body						body	body		
stage3				body	body				body	body		
				LOOP1	LOOP2				LOOP2	LOOP2		

Figure 73: At the time of multiple description of *wait* statement, assignment processing to self-reference is done.

17.4 Foldable loop in SystemC description.

17.4.1 Foldable loop in auto scheduling mode

As values of sc_in, sc_out and sc_signal are updated for wait() in SystemC description, so there are following limitations in folding loop.

- wait() is necessary at the end of loop body
- wait() is necessary in front of loop

(However, in case sc_in and sc_signal are not referenced from loop head till the first wait(), then it is ok even if wait() is not in front of loop)

There will be following error when above limitations are not fulfilled.

F_BT4390: wait() is not in the end of folding loop/automatic pipeline in SystemC input
[Counter measure] Enter wait() in the end of automatic pipeline/folding loop

F_BT4933: : wait() is not in upper part of folding loop in SystemC input
[Counter measure] Enter wait() in the upper part of folding loop

Example of description of loop folding of automatic scheduling:

```
#include <systemc.h>
SC_MODULE (foo) {
    sc_in<sc_int<32> > a,b;
    sc_out<sc_int<32> > o1,o2;
    sc_in_clk CLOCK;
    sc_in<bool> RESET;
    void entry();
    SC_CTOR (foo) {
        SC_CTHREAD(entry, CLOCK.pos());
        watching(RESET.delayed() == true);
    }
};

void foo::entry(){
    sc_int<32> x, i;
    i = 0;
    wait(); /* wait() before while statement*/
    /* Cyber folding = 1 */
    while(true) {
        i++;
        x = a.read() + b.read();
        wait();
        o1.write(i);
        o2.write(x);
        wait(); /* wait() at the bottom of loop body */
    }
}
```

There is similar limitation at the time of describing folding loop further inside the folding loop. However, the above mentioned limitation is not applicable for do-while statement corresponding to the "Loop folding stall description" given below.

17.4.2 Loop folding stall description in auto scheduling mode

When loop is executed in 1 cycle by describing loop in folding loop for stall operation, description (refer **section 17.2.9**) is done by using wait in BDL but in SystemC when description is done as given below, "do-while" statement is treated same as "wait" in BDL. The following information is sent and synthesis is done for the loop moving in 1 cycle.

I_BT4923: Converted to a format which can synthesize do-while statement in folding loop

- do-while statement in folding loop
- folding attribute is not added in do-while statement
- wait() is at the beginning of the body of do-while.
- wait() is only at the beginning of the body of do-while statement
- wait() is at the end of do-while statement

(However, in case DII of folding is 2 or above, or else there is no assignment in sc_out and sc_signal in body of do-while statement then it is ok even if wait() is not after do-while statement)

However, in case of pipelined memory and pipelined functional unit is used in upper part of do-while loop, BDL input does not handle wait() as it is handled here.

Folding stall description example

```
#include <systemc.h>
SC_MODULE (foo) {
    sc_in<sc_uint<2> > inf;
    sc_in<sc_int<32> > ind1,ind2;
    sc_in<sc_uint<3> > ind3;
    sc_out<sc_int<32> > outd1,outd2,outd3;
    sc_in_clk CLOCK;
    sc_in<bool> RESET;
    void entry();
        SC_CTOR (foo) {
            SC_CTHREAD(entry, CLOCK.pos());
            watching(RESET.delayed() == true);
        }
    };
void foo::entry(){
    sc_int<32> x, i, j;
    wait();
    while(true) {
        i = ind3.read();
        wait();
        /* Cyber folding=1*/
        while(i<15){
            i++;
            x = ind1.read() + ind2.read();
            j=0;
            do{
                wait(); /* wait() in the beginning of body*/
                j++;
                outd1.write(x);
                outd2.write(j);
            }while(!inf.read());
            wait(); /* wait() after do-while statement*/
            outd3.write(x);
            wait();
        }
        wait();
    }
}
```

However, synthesis error will occur when delay is not lessened in 1 cycle like wait of BDL or if the operation cannot be done in 1 cycle due to functional unit constraint.

17.4.3 Foldable loop in manual scheduling mode

For manual scheduling mode, wait() should be described in the below mentioned location same as the restriction (**section 17.3**) in BDL (However, there is an error because it is not supported in do-while statement).

- wait() is required at the end of loop body
- wait() is required after termination of loop※

(However, in case of loop without break (infinite loop), wait is not required immediately after loop termination)

When the above mentioned restrictions are not met then the following error occurs

```
F_BT4930: wait() is not in the end of folding loop/automatic pipeline in
SystemC input
[Action] Write wait() in the end of folding loop/automatic pipeline

F_BT4916: When loop is folded, it is required to describe wait() in the
end of loop breakout
[Action] Describe wait() in after loop breakout
```

Example of loop folding of manual scheduling

```
#include <systemc.h>
SC_MODULE (foo) {
    sc_in<sc_uint<3> > f;
    sc_in<sc_int<32> > a,b;
    sc_out<sc_int<32> > o1,o2,o3;
    sc_in_clk CLOCK;
    sc_in<bool> RESET;
    void entry();
        SC_CTOR (foo) {
            SC_CTHREAD(entry, CLOCK.pos());
            watching(RESET.delayed() == true);
        }
    };
    void foo::entry(){
        sc_int<32> x, i;
        wait();
        while(true) {
            o3.write(0);
            i = 0;
            /* Cyber folding = 1 */
            while( f.read() ){
                i++;
                x = a.read() + b.read();
                wait();
                o1.write(i);
                o2.write(x);
                wait(); /* wait() at the end of loop body*/
            }
            wait(); /* wait() after loop break*/
            o3.write(1);
            wait();
        }
    } ( ...
```

17.5 Summary

This section covered the following points:

3. While executing the loop folding, if a variable's value used in the current iteration gets modified in the next iteration, care must be taken while using the variable in the current iteration.
4. When loop folding is executed, if the reference is in the previous step or the same step of the assignment, then the data dependency relationship is finished.
5. If input/output variables are referred/assigned in the same cycle due to loop folding, an error will generated.
6. In the loop folding operation of "while" statement, the operation of loop 1 starts followed by the DII step, and then the operation of loop 2 begins.
7. In the loop folding operation of "for" statement, it is mandatory to place \$ at the specified positions.
8. During loop folding in a "do- while" statement, before it is decide that loop should proceed with condition determination, the next loop starts executing.
9. In the case of "for" loop of fixed time where unrolling is possible, execution of loop folding along with loop unrolling is possible by keeping the array access subscript constant.
10. When executing the loop folding in automatic scheduling mode, the "folding" attribute is specified to the "for" , "while", and "do- while" statements and DII is specified as the attribute value.
11. The "for" statement has the function to avoid the data dependency relationship of the variable assigned in the reset part.
12. If 'break' occurs at a certain time in a loop, then the elementary behavior will be termination of the loop and later loops are discarded immediately.

18 watch statement

18.1 Overview

This section explains about the usage of attribute for establishing a dependency relationship between I/O variable, array or shared reg variable while using watch statement, besides specifying the time constraint.

18.1.1 Related Attributes

Attributes	Explanations	Setting Target
<code>/* Cyber chain_group = #[:#..] */</code>	Specify dependency relationship between I/O variables, array, shared reg variables, and a <i>watch</i> statement.	I/O Variable, array, shared reg variable, <i>watch</i> statement
<code>/* Cyber chain_group = all */</code>		

18.1.2 “watch” statement and I/O variable - dependency

```

in  ter(0:8)    a;
in  ter(0:1)    c;
out ter(0:8)   x;
out ter(0:8)   y;

process main() {
    var(0:8)    s;
    s = a;        /* ...(1) */
    watch(c);    /* ...(2) */
    x = s;        /* ...(3) */
    y = s;
}

```

In above example, there is no dependency relationship between “watch” statement and I/O variables. Thus, there might be cases when value of ‘s’ will be written to output port ‘x’ without waiting for ‘c’ to become true.

As shown below, by specifying a chain_group with the same number, a dependency will be maintained in the order of (1), (2) & (3), and data will be written to ‘x’ only after ‘c’ becomes true.

Note: As no dependency relationship is specified for ‘y’, there are cases when data will be written to ‘y’ without waiting for ‘c’ to become true.

```
in ter (0:8)  a          /* Cyber chain_group = 0 */;
in ter(0:1)   c;
out ter(0:8)  x          /* Cyber chain_group = 0 */;
out ter (0:8) y;

process main () {
    var (0:8)  s;
    s = a;           /* ... (1) */
    /* Cyber chain_group = 0 */
    watch (c);      /* ... (2) */
    x = s;           /* ... (3) */
    y = s;           /* ... (4) */
}
```

18.2 Summary

This section covered the following:

A dependency between I/O variables, array, shared reg variables, and a watch statement "can be maintained by specifying the chain_group attribute with the same number.

19 Automatic Scheduling

19.1 Overview

This section covers the following:

1. The options for timing specifications of input/output and related attributes.
2. The options and attributes related to speculative execution.
3. The option and attributes related to comprehensive parallelization of 'if' statement.
4. Conditional Branch Scheduling and register based scheduling.
5. Description of scheduling using basic library delay considerations and its restrictions.
6. The options and attributes related to simplified time constrained scheduling and its restrictions.
7. The options and attributes related to scheduling block and its restrictions.
8. The "hv" mode.

The options used for timing adjustments in automatic scheduling mode are discussed in this section.

19.2 Input /output timing specification

19.2.1 Overview

Input-output timing specification is a capability that controls the steps where the input/output operations are performed in automatic scheduling mode.

19.2.2 Related Options

Option	Description
-S0	Schedules input as early as possible and output as early as possible
-S1	Schedules input as early as possible and output as late as possible
-S2	Schedules input as late as possible and output as early as possible(default)
-S3	Schedules input as late as possible and output as late as possible.

19.2.3 Related Attribute

Attribute	Description	Settings Target
<code>/*Cyber read_timing = ASAP*/</code>	Schedules the reference of specified input output variable, shared variable and array as early as possible.	Input output variable Shared variable Array
<code>/*Cyber read_timing = ALAP */</code>	Schedules the reference of specified input output variable, shared variable and array as late as possible.	Input output variable Shared variable Array
<code>/*Cyber write_timing = ASAP*/</code>	Schedules the assignment for specified input output variable, shared variable and array as early as possible.	Input output variable Shared variable Array
<code>/*Cyber write_timing = ALAP*/</code>	Schedules the assignment for specified input output variable, shared variable and array as late as possible.	Input output variable Shared variable Array

19.2.4 Description

In automatic scheduling mode, scheduling of various input and output timings scenarios are possible as illustrated in **Figure 74**.

In the example shown in **Figure 68**, it does not matter if input i3 and output o1 are implemented/executed by either step 1 or step 2. The implementations are equivalent description wise and action wise. In the default automatic scheduling mode, circuit is generated to minimize the number of storage registers. Therefore, input is done at a later step as far as possible, and output is done at an earlier step as far as possible.

The default behavior can be altered by specifying the options from -S0 to -S3, as shown in **Figure 75**. These options may be used when there is a necessity to read all the input ports in the initial step or when there is a necessity to write on the output port in the last step according to the circuit specification/requirements.

The “read_timing” and “write_timing” attributes are available to be individually specified to the respective I/O signal, shared variable and array.

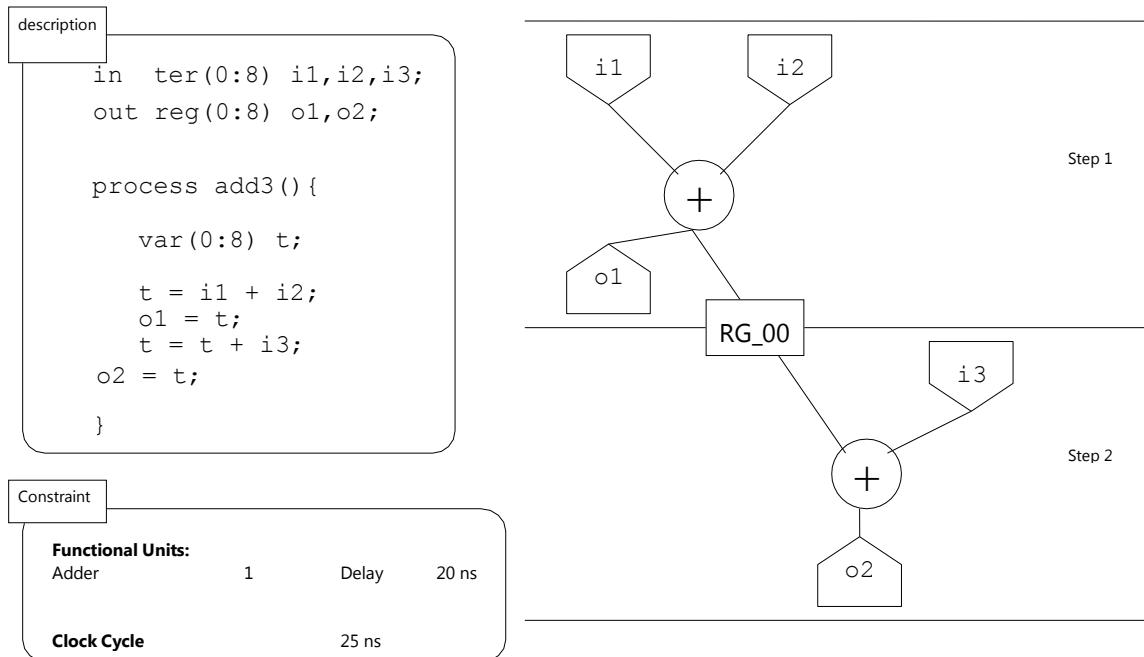


Figure 74: Input output timing (default)

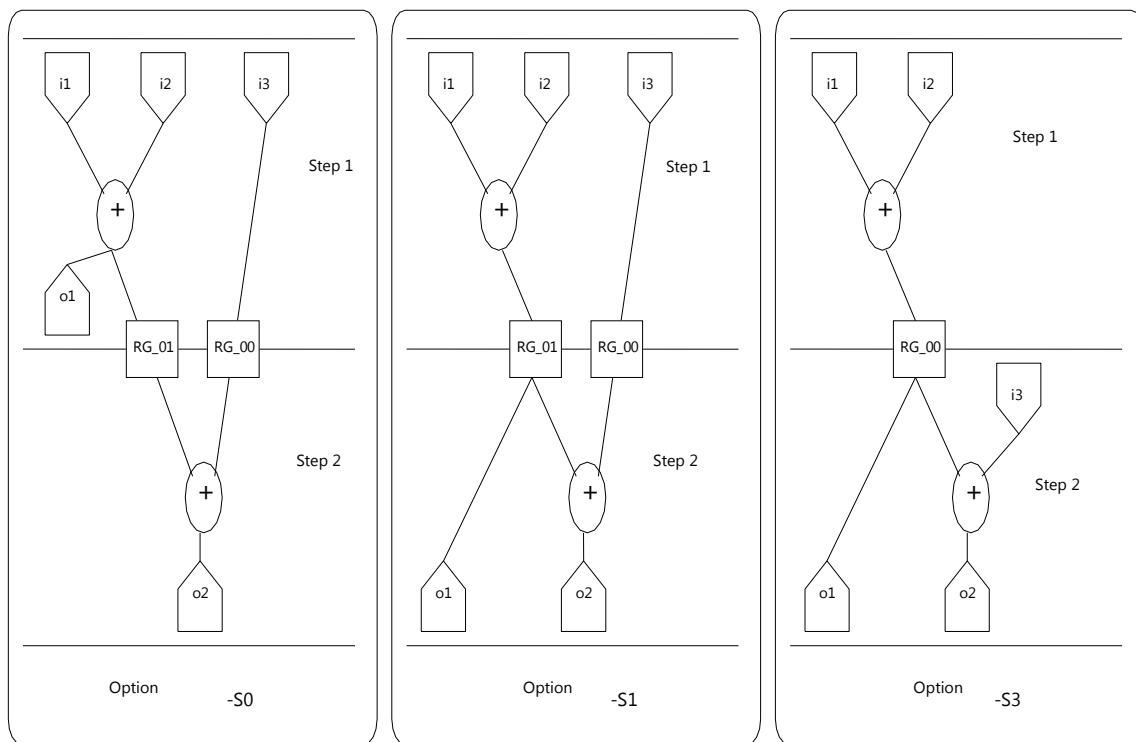


Figure 75: Change/ alteration in input output timing

19.3 Speculative Execution based Scheduling

19.3.1 Overview

Speculative execution means executing the operations in all branches in parallel before the respective branch conditions are evaluated. Later, the results of only those branches are used where conditions are evaluated as true, and the rest of the branch operations are discarded.

Circuits with fewer cycle counts can be generated using this technique resulting in better performance. However, the resulting circuit requires larger area because many functional units are required to execute operations speculatively.

Scheduling using speculative execution is performed by default.

19.3.2 Related options

Option	Description
-SS	Performs scheduling that does not use speculative execution.

19.3.3 Related attributes

Attribute	Description	Settings Target
/* Cyber speculation = YES */	Performs speculative execution	Array, Element
/* Cyber speculation = NO */	Do not perform speculative execution	Array, Element

19.3.4 Description

Figure 76 and **Figure 77** illustrate the difference of result generated in automatic scheduling due to existence or non- existence of speculative execution.

Figure 76 illustrates the case where speculative execution is performed. While the "if" statement condition $i1 > i2 + i3$ is executed in two cycles, there are extra functional units given for remaining constraints. So both 'then' side's statement $t=v+3$ and 'else' side statement $t=v-3$ are executed in parallel in first cycle, and once the condition is evaluated in second cycle, one of the results will be selected and the other will be discarded.

Figure 77 illustrates the case where speculative execution is not performed. Here, the condition of "if" statement $i1 > i2 + i3$ is executed in two cycles. After the condition has been evaluated, either of $t = v+3$ or $t=v-3$ of the "else" side of "if" statement is executed.

If both of above scenarios are compared, the case with speculative execution will take 2 cycles to execute, while the case without speculative execution will take 3 cycles to execute. Therefore, the number of process cycles required is reduced by using speculative execution. On the other hand two adders are needed in case speculative execution is performed. On the other hand one adder is sufficient when speculative execution is not performed. Therefore, the circuit area can be less if speculative execution is not performed.

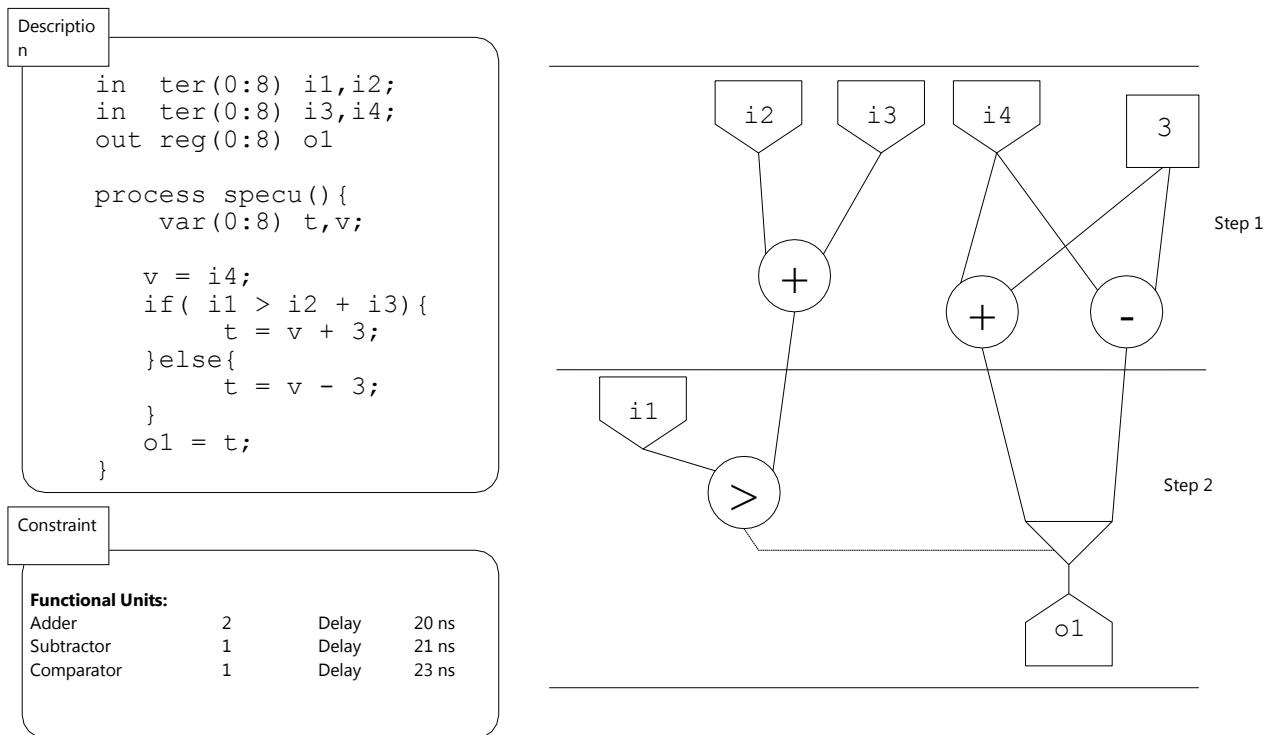


Figure 76: Example when speculative execution is performed

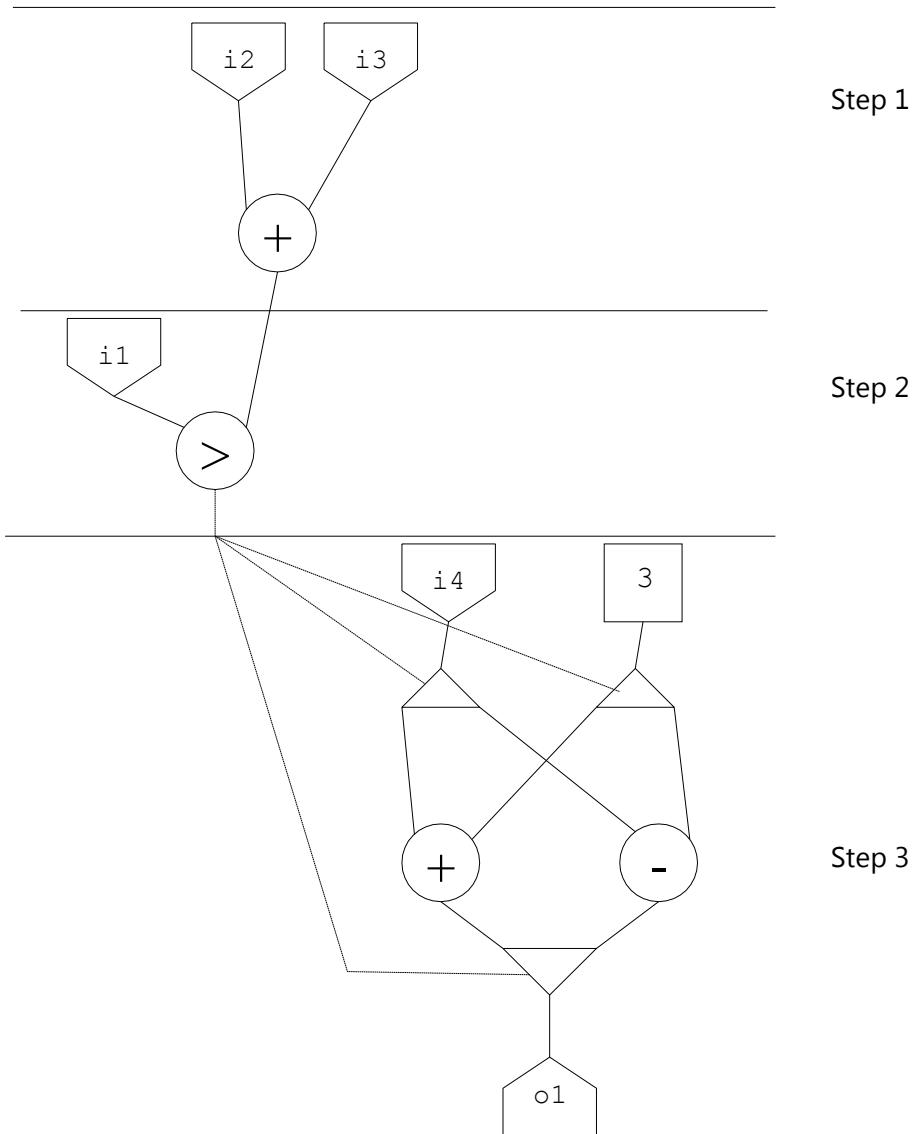


Figure 77: Example when speculative execution is not performed.

In case specific operations and array references are not to be speculatively executed, it can be specified for variable by using speculation = NO attribute.

However, in case option -SS is specified, this attribute is ignored and any speculative execution is not performed.

19.3.5 Restrictions

The speculative execution is not done in the case where branch operations have side effects. This is because it is necessary to roll back (or to discard the values of) the operations with side effects if the respective branch condition evaluates as False. This rolling back will cause a burden on the generated circuit. Hence, speculative scheduling is not performed for these cases. The examples of operations with side effects are access to array, input/output, etc.

19.4 Conditional exclusion based on data dependency

19.4.1 Related attributes

Attribute	Description	Settings Target
<code>/* Cyber implicit_exclusion = YES */</code>	Permits the conditional exclusion based on data dependency of operation and array reference	Element
<code>/* Cyber implicit_exclusion = NO */</code>	Does not permit conditional exclusion based on data dependency of operation and array reference	Element

19.4.2 Description

In case result of an operation and array reference is used only by a certain condition, sharing of functional unit and memory port can be done by carrying out operation and array reference in that condition only.

In the following example, 2 addition operations are carried out in the same condition therefore, it may be assumed that functional unit cannot be shared. However, if data dependency is considered, functional unit can be shared. Specifically, v1 is referenced only when the if condition is true and v2 is referenced only when the if condition is false, i.e. in the else block. As a result, sharing is possible by allowing 2 addition operations only in the condition when either "if" or "else" block is executed. This functionality is carried out by default in automatic scheduling mode. To stop the consideration of conditional exclusion based on data dependency for specific operations or array references, execution can be suppressed by specifying attribute `implicit_exclusion = NO` for the element.

Example:

```
in ter(0:1) f;
in ter(0:8) i1,i2,i3,i4;
out reg(0:8) o1;

process exclusive() {

    var(0:8) v1,v2,t;

    v1 = i1 + i2;
    v2 = i3 + i4;
    if( f ){
        t = v1;
    }else {
        t = v2;
    }
    o1 = t;
}
```

Example:

```
in ter(0:1) f;
in ter(0:8) i1,i2,i3,i4;
out reg(0:8) o1;

process exclusive() {

    var(0:8) v1,v2,t;

    v1 = i1 + /* Cyber implicit_exclusion = NO */ i2;
    v2 = i3 + /* Cyber implicit_exclusion = NO */ i4;
    if( f ){
        t = v1;
    }else {
        t = v2;
    }
    o1 = t;
}
```

19.5 Comprehensive Parallelization of 'if' Statement

19.5.1 Overview

Comprehensive parallelization of "if" statement means executing multiple conditional branches in parallel. For example, the operations in two "if" statements are executed in parallel.

When comprehensive parallelization of "if" statement is not performed, the next subsequent operation in "if" statement is not performed until all the operations in the preceding "if" statements are executed. This is simply a serial processing of the BDL specification.

The comprehensive parallelization of "if" statement is performed by default and circuit requiring fewer cycle counts is generated. However, parallel execution of the operations requires many functional units and it may affect the larger circuit area.

19.5.2 Related Options

Option	Description
-SP	Performs scheduling that does not apply comprehensive parallelization of 'if' statement.

19.5.3 Description

The description given below explains the difference in the synthesis when comprehensive parallelization is done and when it is not done. The description assumes that functional unit library file is synthesized where clock cycle is 5ns (500).

The difference of synthesis is explained depending on the existence or nonexistence of comprehensive parallelization of "if" statement, **Figure 78** and **Figure 79** illustrates the difference in scheduling using CDFG.

Example:

```

in ter(0:1) f1,f2;
in ter(0:8) i1,i2,i3,i4,i5;
out reg(0:8) o1,o2;
process if_parallel(){
    var(0:8) a,b;
    if( f1 ){
        a = i1 + i2;
        o1 = i3 - a;
    }else{
        a = 0;
    }
    if( f2 ){
        b = i4 * i5;
        o2 = a + b;
    }
}

```

Functional unit library file (gcd1.FLIB)

```

#@LIB { if_parallel }
@FLIB{
    NAME      add08
    KIND      +
    BITWIDTH  8
    DELAY    200
    AREA     100
}
@FLIB{
    NAME      sub08
    KIND      -
    BITWIDTH  8
    DELAY    200
    AREA     100
}
@FLIB{
    NAME      mul08
    KIND      *
    BITWIDTH  8
    DELAY    450
    AREA     400
}
#@END { if_parallel }

```

Functional unit count file (gcd1.FCNT)

```

#@CNT { if_parallel }
@FCNT {
    NAME      add08
    LIMIT    1
}
@FCNT {
    NAME      sub08
    LIMIT    1
}
@FCNT {
    NAME      mul08
    LIMIT    1
}
#@END { if_parallel }

```

Figure 78 illustrates the case where comprehensive parallelization of "if" statement is performed. The respective operations in 'if' statement have data dependency. Hence, the total calculation is performed in two cycles.

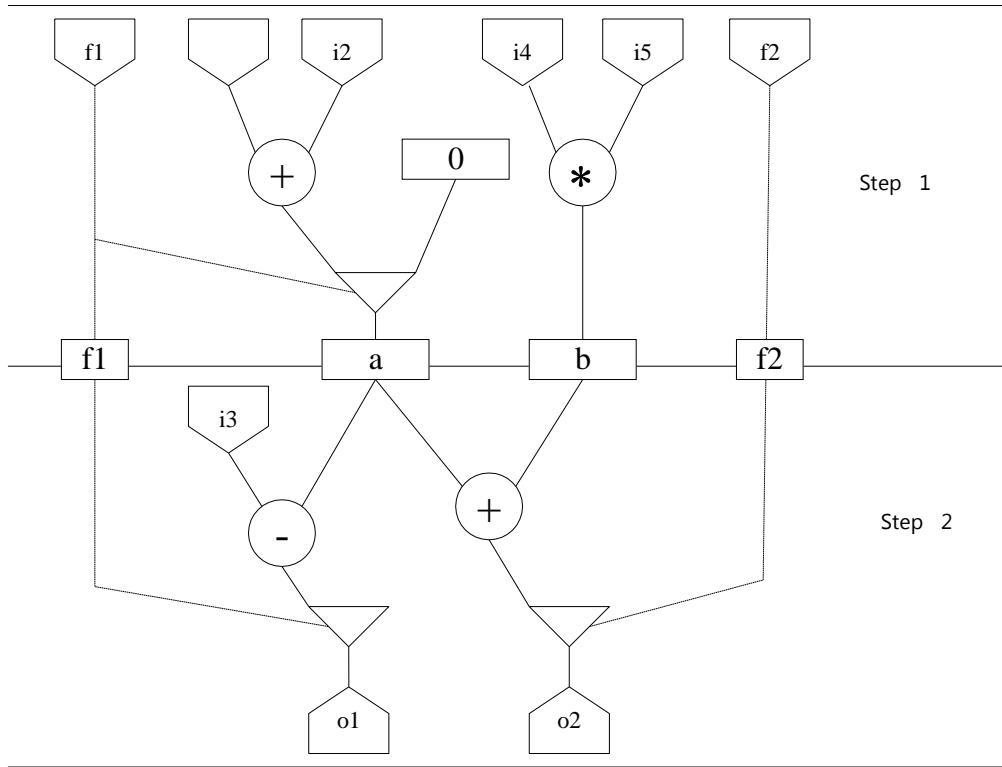


Figure 78: Example of comprehensive parallelization of "if" statement.

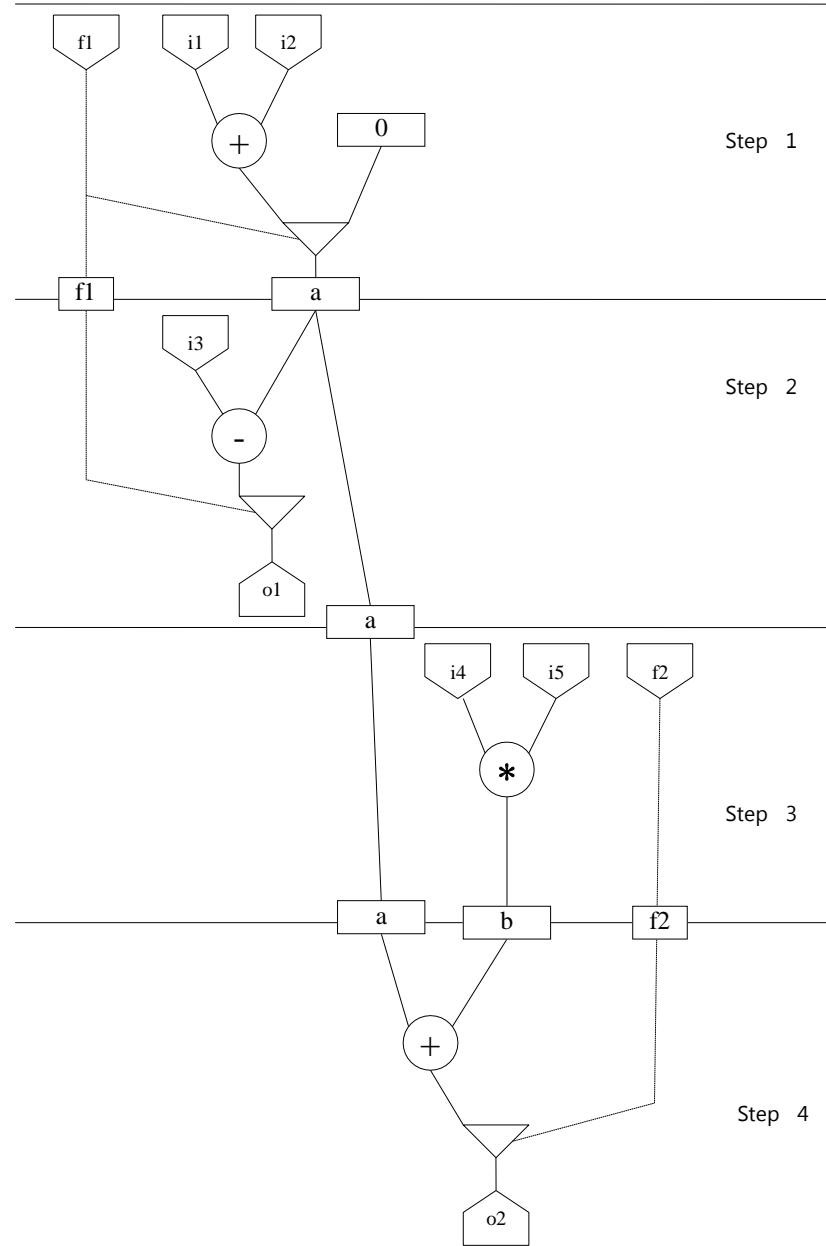


Figure 79: Example where comprehensive parallelization of "if" statement is not performed.

Figure 79 shows the case when comprehensive parallelization of "if" statement is not performed. In such cases, operations in the next "if" statement are not performed until all the operations within the first "if" statements are not completed. As illustrated in the example shown in **Figure** , two cycles are necessary to perform operations within each "if" statement. Therefore, the overall operation is performed in a total of four cycles.

Hence, the process cycle can be reduced by performing comprehensive parallelization of 'if' statement.

However, the necessary functional unit count may increase by performing comprehensive parallelization of 'if' statements.

19.6 Conditional Branch Scheduling

19.6.1 Overview

In automatic scheduling, circuits with different cycle counts are generated for every branch. This is because the overall cycle count may increase if the cycle counts of all the branches are clubbed with the maximum cycle count among the branches.

However, there are cases where cycle counts of all branches are desired to be made equal (for example, a module that is converted to a functional unit or a pipeline circuit). In such cases, the cycle counts of every branch can be the same by using the -Zsame_cycles option.

19.6.2 Related Options

Option	Description
-Zsame_cycles	Aligns cycle count of all paths of conditional branching

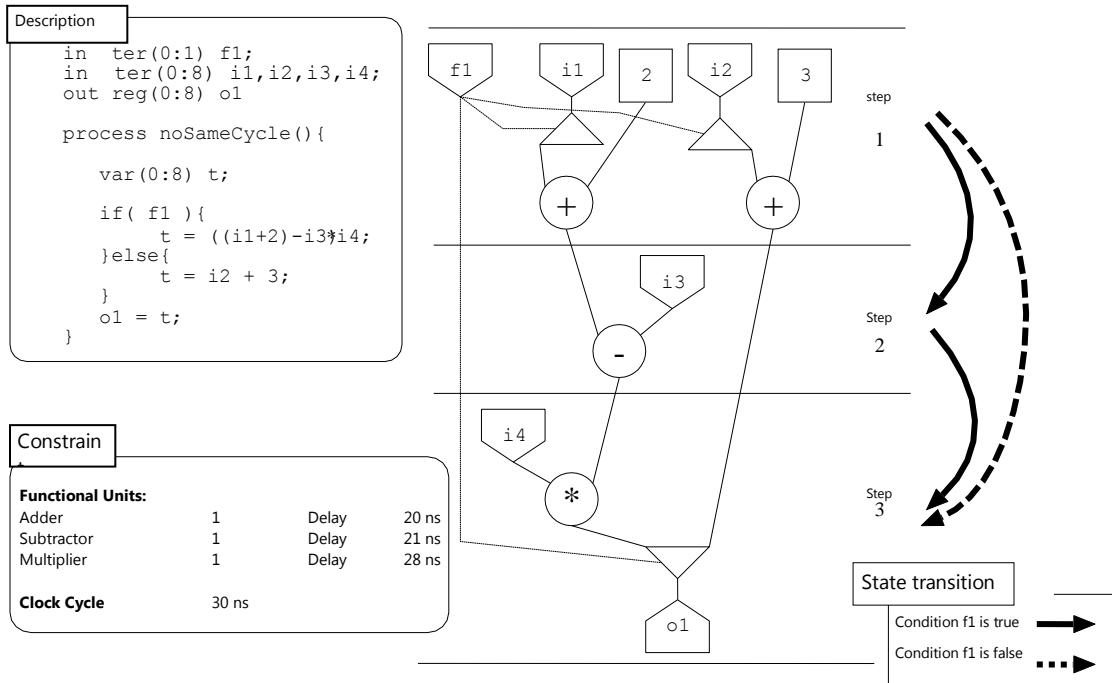
19.6.3 Description

Figure 80 illustrates that the cycle counts needed for process execution differs depending upon conditional branching.

When signal *f1* is True, then calculation is done in three cycles: step 1→step 2→step 3. Otherwise, when *f1* is False, then calculation is performed in two cycles: step1 →step 3.

By default, a circuit where cycle counts of every conditional branch are different is generated. Hence a circuit is generated which takes three cycles when *f1* is true and takes two cycles when *f1* is false.

However, the "-Zsame_cycles" option should be specified at the time of synthesis if the cycle count of every branches are required to be the same. As shown in the illustrated example, if the "-Zsame_cycles" option is specified, then a circuit is generated which takes three cycles regardless of the fact whether *f1* is true or false.

**Figure 80:** Process cycle count differs depending on conditional branching

19.7 Register Based Scheduling

19.7.1 Overview

Register Consideration Scheduling refers to a scheduling method where following aspect is taken care of:

- o When a variable declared as "reg" or specified as register with an attribute specification is required, the reference following an assignment for the same variable is always scheduled at a later step than the step where the assignment is scheduled.

There are two methods of specification

- o Declare variable as "reg" type
- o Specify relevant attributes (as described in **section 19.7.3**)

19.7.2 Related Options

Option	Description
-SR	Performs scheduling considering declaration of register variable. (Default)
-S-R	Performs scheduling without considering declaration of register variable

19.7.3 Related Attributes

Attribute	Description	Settings Target
/* Cyber reg_bind */	Do not allocate specified variable to a wire but to a register (automatic scheduling only)	Variable

19.7.4 Description

Figure 81 and **Figure 82** illustrates a simple example. **Figure 81** shows specification method using the "reg" declaration.

In automatic scheduling, the "ter" type variable which is not an array is overridden by the "var" type variable. However, the "reg" type variable which is not an array is treated as "reg" type.

In scheduling that considers the "reg" declared variable, the assignment of target variable shows the assignment to a register. In addition, a circuit is generated where the reference is done at a later step instead of the step where the assignment is scheduled.

In the example shown below, variable "t" is declared as "reg" type. Hence, the statement "t = i1", is executed in one cycle. The next reference, "o1 = t + i2", is performed in step 2.

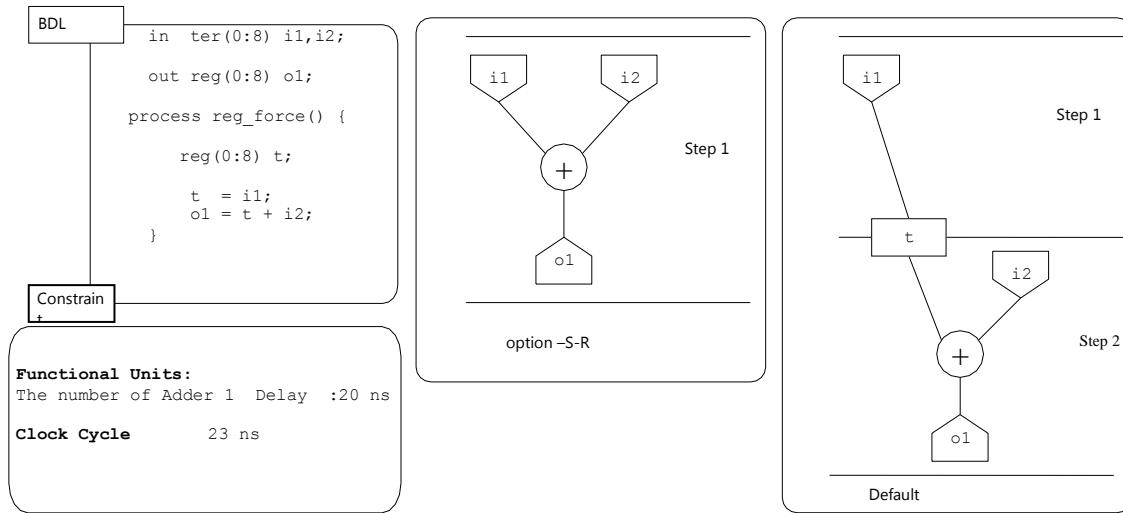


Figure 81: Scheduling considering "reg" declared variable

Figure 82 shows the specification method using the attribute.

Variable that can be specified with the attributes are "var" type variables that are not arrays. In automatic scheduling mode, "ter" type variable is converted to a "var" type. Hence, the attributes can be specified for "ter" type variables.

Variable specified as register is scheduled so as to introduce at least one cycle delay between reference and assignment. In the description shown in **Figure 82**, out of the "var" type variables t_1 and t_2 , only t_1 is specified using the "sig2reg_force = RG00" attribute. Hence, synthesis is done so that t_1 is assigned at step 1 and referred in step 2. Finally, it is allocated to the register RG00. In case the "sig2reg_ex_force = RG01" attribute is specified, a circuit is generated with exclusive allocation of specified variable to RG01, and no other variable is allocated to register RG01.

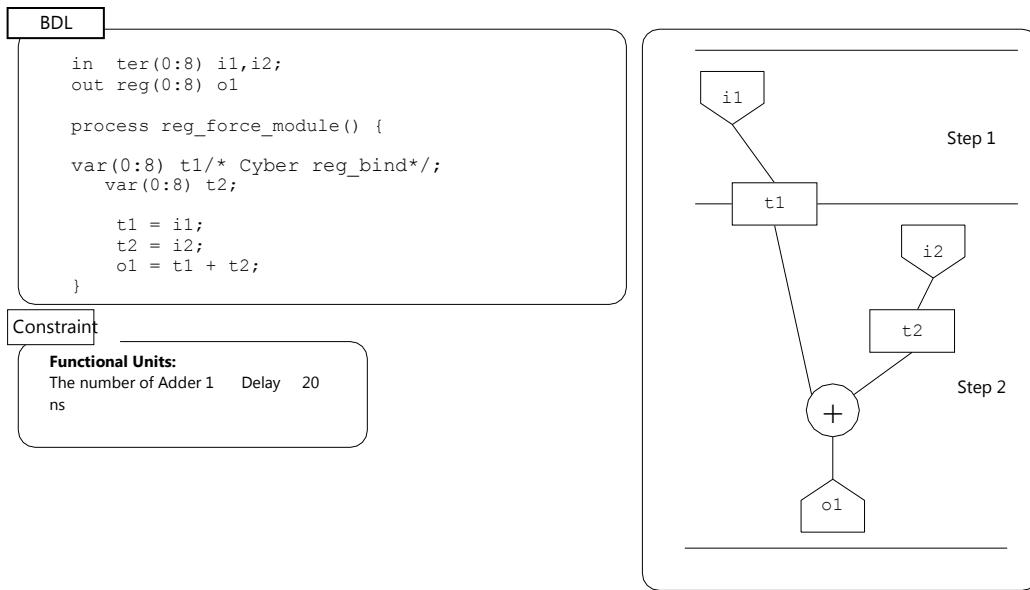


Figure 82: Scheduling considering "reg" declared variable (with attribute)

19.8 Scheduling Considering Basic Library component Delay

19.8.1 Overview

This section explains the scheduling mechanism considering the delay of basic library components, such as multiplexer (MUX) and decoder (state decoder, attribute decoder etc).

19.8.2 Related Options

Option	Description
-SM	Performs Scheduling considering delay of basic library components (default)
-S-M	Performs scheduling without considering delay of basic library components.

19.8.3 Description

Following delays are considered when scheduling is performed using the basic library component delay:

2. Delay of multiplexer (MUX) connecting to data path
3. Delay of decoder used to decode FSM states

Note: If clock uncertainty is specified using option “-cd #”, this specification is given higher priority.

4. Delay of address decoder of decoder enabled register array.

Note: If attribute rw_delay of array is specified, this specification is given higher priority.

19.8.4 Restrictions

There are following limitations while using basic library file (BLIB).

- Delay of basic library components is not considered if basic library component delay consideration scheduling option is specified and BLIB file is not specified.
- If the functional unit delay is specified in terms of cycles (1T for example) in functional unit constraint file (FCNT) or functional unit library file (FLIB) then the delay of basic library components is not considered. Delay of functional unit, delay due to state decoder of FSM, delay of multiplexer connecting to data path are interpreted as delays in specified cycles for scheduling.
- Even if the delay of functional unit becomes more than one cycle with considering the delays of basic library components, the functional unit is not synthesized as multi-cycle functional unit.
- In the latest Cyber version 5.2.2, the delay of multiplexers connected to the inputs of functional units and the delay of array address decoders are considered. For example, the delay of multiplexers connected to the inputs of registers is not considered.

19.9 Dependency relation of array and I/O specification

19.9.1 Overview

Scheduling is done after considering (emphasis) delay of logical operations (AND, NOR, RAND, etc.)

19.9.2 Related Options

Option	Description
-Zsche_logic_delay[=YES]	Scheduling is done after emphasizing the delay of logical operations (default)
-Zsche_logic_delay=NO	Scheduling is done without emphasizing the delay of logical operations

19.9.3 Description

In scheduling based on delay of logical operations, scheduling is done after emphasizing the below delay estimation.

- Delay of logical operations (AND, NOR, RAND, etc.) having large number of stages
- Delay of reduction operation having large bit width
- Delay of conditional branching (MUX) having large number of branches

Delay violation is unlikely to occur in case large number of logical operations is included or even in case the number of branches of if, switch statement is many.

19.9.4 Restrictions

There are following restrictions in scheduling based on delay in logical operations.

- In case basic library file (BLIB file) is not specified, consideration (emphasis) is not given on delay of conditional operations even if option of scheduling based on the delay of logical operations is specified.
- As there is a tendency of latency stretch, when scheduling is done with a similar method prior to CyberWorkBench 5.1.1, then -Zsche_logic_delay=NO is specified.

19.10 Dependency relationship of array, I/O Specification

19.10.1 Related attributes

Attribute	Description	Settings Target
/* Cyber chain_group= #[:#...]*/ /* Cyber chain_group= all*/	Specify the dependency relation between I/O variable, array, shared reg variable and watch statement.	I/O variable, array, shared reg variable, watch statement
/* Cyber self_chain = YES */ /* Cyber self_chain = NO*/	Specify the dependency relation between the same array signals set by chain_group Do not Specify the dependency relation between the same array signals set by chain_group (default)	Array

19.10.2 Description

In auto scheduling mode, dependency relation can be specified for watch statement, shared reg variable, array variable and I/O variable that does not have data dependency relation.

Section 18 describes about watch statement with dependency specification. Similar dependency can be specified for shared reg variable, array variable and I/O variable.

Multiple groups can also be described for a single variable and watch statement. In that case, multiple group numbers can be specified by using ":" as separator for attribute.

```

in          ter(0:8) a;
in          ter(0:8) b;
out         reg(0:8) x;
out         reg(0:8) y;
out         reg(0:1) flag;
mem(0:8) M[256];

process foo() {
    v = a;
    M[v] = b;
    x = v + b;
    flag = 1;
    x = 0;
    flag = 0;
    y = M[v];
}

```

In the above example, the assignment flag = 1 of output variable flag does not have data dependency therefore, there are cases when it gets executed even before x = v+b of output variable x. Further, similarly, referencing of array M that is assigned to output variable y also does

not have data dependency with output variable y, therefore, there is a possibility that it will get executed even before flag = 0.

In order to carry out such execution sequence as per description, attribute chain_group with the same group number is specified for respective variables that are required for specifying dependency relation as follows. The following example shows the dependency relation between output variable x and output variable flag and between output variable flag and array M. Access to variable specified by this is scheduled in way that execution is done as per described sequence in the same cycle or a later cycle.

In case the user wants to specify chain_group attribute of all group numbers, then the value of "chain_group = all" can be specified. For instance, in the illustration shown below, same dependency relation can be specified when chain_group = all is specified instead of chain_group =1:2.

```
in          ter(0:8) a;
in          ter(0:8) b;
out         reg(0:8) x; /*Cyber chain_group = 1*/
out         reg(0:8) y;
out         reg(0:1) flag; /*Cyber chain_group = 1:2*/
mem(0:8)   M[256]; /*Cyber chain_group = 2*/

process foo() {
    v = a;
    M[v] = b;
    x = v + b;
    flag = 1;
    x = 0;
    flag = 0;
    y = M[v];
}
```

Attribute self_chain can specify the dependency relation in the access processes for each element of array signal specified by the attribute chain_group.

```

in      ter(0:8) a, b, c;
out     ter(0:8) x, y, z;
out     reg(0:1) flag /* Cyber chain_group = 1 */;
mem(0:8) M[256] /* Cyber chain_group = 1 */;

process foo() {
    flag = 0;
    M[0] = a;
    M[1] = b;
    M[2] = c;
    flag = 1;
    x = M[ 2];
    y = M[ 1];
    z = M[ 0];
}

```

In the above-mentioned example (when port type of memory and count is 'R1 and W1'), the assignation is done for the 3 elements of array M after flag = 0 is assigned for output variable flag by specifying chain_group. After that assignation is done for 3 elements of array M after flag = 1 is assigned, however there is a possibility that the assignation to x, y, and z is not as per the description as there is no dependency relation between the reference processing and in the assignation for each element of array M.

In this way, for executing the access processing for same signal as per the description, self_chain attribute is specified along with chain_group attribute as shown below.

In the following example, access processing for array M is executed as per the description.

```

in      ter(0:8) a, b, c;
out     ter(0:8) x, y, z;
out     reg(0:1) flag /* Cyber chain_group = 1 */;
mem(0:8) M[256] /* Cyber chain_group = 1, self_chain = YES
*/;

process foo() {
    flag = 0;
    M[0] = a;
    M[1] = b;
    M[2] = c;
    flag = 1;
    x = M[ 2];
    y = M[ 1];
    z = M[ 0];
}

```

19.11 Summary

This section covered the following points:

- In the default automatic scheduling mode, circuit is generated to minimize the number of storage registers.
- The default behavior can be altered by specifying the options from -S0 to -S3.
- The "io_timing" attribute is available to individually specify the signal-wise input /output.
- The number of process cycles required is reduced by using speculative execution.
- The speculative execution is not performed when any of the branch operations have side effects.
- The process cycle can be reduced by performing comprehensive parallelization of 'if' statement. However, the necessary functional unit count can increase by performing comprehensive parallelization of 'if' statement.
- The "-Zsame_cycles" option should be specified at the time of synthesis if the process step of every branch is required to be the same.
- In the automatic scheduling, there is "hv" mode that reinforces the exclusion check of condition. By default, the "hv" mode is already validated.

20 Resource Allocation

20.1 Overview

This section explains the following features:

- Allocation of signed and unsigned variables to registers.
- Allocation of functional units based on the signs of the variables or operands.
- Register sharing during manual scheduling.
- Register sharing during automatic scheduling.
- The two types of allocation algorithms.
- Naming of registers.
- Allocation of signal to register with the specified name.
- Allocation of arrays to particular memory types.

This section explains the resource allocation method for signals and operators during synthesis. The allocation is done to specific registers, memory, and functional units.

20.1.1 Related Options

Option	Description
-oR	A Register is shared in case of manual scheduling (Default)
-o-R	A register is not shared in case of manual scheduling.
-ZZfu_noshare_states	In entire operations functional unit is not shared with functional unit of specified states.
-ZZfu_noshare_states=all	In entire operations a functional unit is not shared with the functional units of all the states
-ZZfu_noshare_states= #[#...]	Functional units allocated to the operations in the # states are not shared with the operations in other states
-Zmix_sign_share=ALL	Share the register amongst both the signed and unsigned variables.

Option	Description
<p>-Zmix_sign_share=REG</p>	<p>The operator can be allocated with functional units of different type (both signed and unsigned (default)).</p> <p>Share the register amongst both the signed and unsigned variables.</p> <p>The operator is not allocated with functional units of different type, i.e. both signed and unsigned functional units cannot be allocated</p>
<p>-Zmix_sign_share=OPE</p>	<p>Do not share the register amongst both the signed and unsigned variables.</p> <p>The operator can be allocated with functional units of different type (both signed and unsigned).</p>
<p>-Zmix_sign_share=NO</p>	<p>Do not share the register amongst both the signed and unsigned variables.</p> <p>The operator is not allocated with functional units of different type, i.e. both signed and unsigned functional units cannot be allocated.</p>
<p>-Zmix_bitw_reg_share</p> <p>-Zmix_bitw_reg_share=NO</p> <p>-Zmix_bitw_reg_share =0_EXPAND_ONLY</p>	<p>Share the register among variables of different bit width (ASIC: Default)</p> <p>Do not share the register among the variables of different bit width (FPGA: default)</p> <p>unsigned variable share the variable with register of different bit width signed variable does not share the variable with register of large bit width</p>
<p>-Zassign=1</p> <p>-Zassign=2</p>	<p>Uses allocation algorithm 1</p> <p>Uses allocation algorithm 2</p>

Option for allocation algorithm 1

Option	Description
-AF	Register and functional units are quickly allocated by reducing the number of candidates.

Option for allocation algorithm 2

Option	Description
-Aa	Allocation is optimized for the area by the sharing of functional units (Default)
-Ad	Allocation is optimized for the number of multiplexers by suppressing the sharing of functional units.
-Af	false loops are avoided in the resource allocation
-A-f	Do not avoid false loops in the resource allocation (Default)
-At	Allocation is optimized for the circuit delays (FPGA: Default)
-A-t	Allocation is not optimized for the circuit delays (ASIC: Default)
-Zreg_orig_name	The name of register reflects the name of the variable to which the register is allocated (Default)
-Zreg_orig_name=#	The name of register reflects the names of utmost # variables to which the register is allocated.
-Zreg_orig_name=NO	Register name is made same as per β3.4.x

	version or version prior to it.
-Zreg_name_func[=YES]	Function name is appended to the register name
-Zreg_name_func=NO	Function name is not appended to the register name (Default)
-Zreg_name_len=#	Number of characters is limited up to # for the register name (Default value 32 characters)
-Zreg_head_len=ALL	Entire variable names and the function names are reflected (Default)
-Zreg_head_len=#	Only initial # characters of variable name is reflected
-Zreg_head_len=#1:#2	Only the initial #1 characters of function name and #2 of the variable name are reflected

20.1.2 Related Attributes

Attribute	Description	Target Object
/* Cyber share_name = name */ /* Cyber no_share */	Change the register name which is the sharing target to "name". Specified variable does not share register with other variables.	Variable Variable
/* Cyber reg_bind */	In automatic scheduling the variable is allocated to the register. The variable is not allocated to wire.	Variable

/* Cyber sig2mu = name */	The array is allocated to the memory "name"	Array
/* Cyber sig2mu_port = name.r#w# */	The array is allocated to the memory with the specified ports and name	Array
/* Cyber mu_port = # */	Specifies the port to be used in array accesses.	Array access
/* Cyber ope2fu = name */	Allocate the specified operation to the functional unit <i>name</i> .	Expression element
/* Cyber ope2fu_ex = name */	Allocate the specified operator name to functional unit <i>without</i> sharing with other functional unit.	Expression element

20.2 Signed, unsigned in Resource Allocation

Option	Explanation
-Zmix_sign_share=ALL	Share the register amongst both the signed and unsigned variables. Also allocate the operators in both signed and unsigned functional unit (default).
-Zmix_sign_share=REG	Share the register amongst both the signed and unsigned variables. Do not allocate the operators in both signed and unsigned functional unit.
-Zmix_sign_share=OPE	Do not share the register amongst both the signed and unsigned variables. And allocate the operators in both signed and unsigned functional unit.
-Zmix_sign_share=NO	Do not share the register amongst both the signed and unsigned variables. And do not allocate the operators in both signed and

Option	Explanation
	unsigned functional unit.

20.2.1 Signed and Unsigned Variables in the Register Sharing

Variables with different sign-type (unsigned/signed) are assigned to different registers regardless of the availability of registers for sharing, when “-Zmix_sign_share=OPE” or “-Zmix_sign_share=NO” option is specified. When default, “-Zmix_sign_share=ALL” or “-Zmix_sign_share=REG” are specified, the registers are shared regardless of the sign of the variables.

In the following example, “sr” is a signed “reg” variable and “ur” is an unsigned “reg” variable. These variables, “sr” and “ur” can share one register because their life time does not overlap.

If -Zmix_sign_share=NO or “-Zmix_sign_share=OPE” is specified the variables are allocated to different signed and unsigned registers.

In case default, “-Zmix_sign_share=REG” or “-Zmix_sign_share=ALL” option is specified, the variables can share a register regardless of whether they are signed or unsigned. Hence, the two variables share a single register.

```

in ter(0:8) a,b;
out reg(0:8) c;
process module_s() {
    signed reg(0:8) sr;
    unsigned reg(0:8) ur;
    while (1) {
        sr = a;
        $
        c = sr;
        $
        ur = b;
        $
        c=ur;
        $
    }
}

```

20.2.2 Signed and Unsigned Variables in Functional Unit Allocation

An operator performs signed or unsigned operation depending upon the sign of the variables (or operands).

Depending upon the type of operations, three different types of functional units are used.

- Units to perform unsigned operations
- Units to perform signed operations
- Based on control signal, units to perform signed or unsigned operations

Depending on the type of operator used, functional units for unsigned operations are capable of performing signed operations and vice versa. For example in the multiplication of 8-bit unsigned variable, unsigned multiplier of 8-bit or more can be used for the operation. In addition to that, the operation can also be executed by using signed multiplier of at least 9-bit.

```
unsigned var(0:8) v1;
unsigned var(0:8) v2;
unsigned var(0:16) v;
v = v1 * v2;
```

When default, “-Zmix_sign_share=ALL” or “-Zmix_sign_share=OPE” option is specified and the operation can be performed using both signed and unsigned functional units, then such functional units should be considered as potential candidate for allocation.

Allocation of functional units with different sign type does not take place when option -Zmix_sign_share=REG or -Zmix_sign_share=NO is specified.

Notes:-

- If option -Zmix_sign_share=REG or -Zmix_sign_share=NO is specified, the operators can be assigned to functional units with different sign type (signed/unsigned) when no functional units with the same sign are found in functional unit library.
- Allocation of an operator to a functional unit of different sign depends on the kind of operation, sign of operator, and functional unit's requirement/non-requirement of carry.

20.3 Sharing of Register during Manual Scheduling

Option	Description
-oR	Shares the register during manual scheduling (Default)
-o-R	Registers are not shared in manual scheduling

In manual scheduling, multiple “reg” variables can share one register by analyzing the life time of different “reg” variables. In the following example, the “reg” variables “r1” and “r2” don't have overlapped lifetimes. Therefore, they can share one register.

However, the “-o-R” option can be used to simplify debugging and testing where register sharing is not desirable.

Input Description

```

in ter(0:8)  a,b;
out reg(0:8) c;
process module_1() {
  reg(0:8) r1,r2;
  while (1) {
    r1 = a;
    $
    c = r1;
    $
    r2 = b;
    $
    c = r2;
    $
  }
}

```

After Register Sharing

```

in ter(0:8)  a,b;
out reg(0:8) c;
process module_1() {
  reg(0:8) RG_01;
  while (1) {
    RG_01 = a;
    $
    c = RG_01;
    $
    RG_01 = b;
    $
    c = RG_01;
    $
  }
}

```

20.4 Register Sharing During Auto Scheduling

The options corresponding to -oR or for register sharing during auto scheduling have not been provided.

20.5 Register Sharing among Variable with different bit width

Option	Description
-Zmix_bitw_reg_share	Share the register with variables of different bit width (ASIC: default)
-Zmix_bitw_reg_share=NO	Variable with different bit width does not share the register (FPGA:default)
-Zmix_bitw_reg_share =0_EXPAND_ONLY	unsigned variable shares the register with variable of different bit width signed variable does not share the register with variable of large bit width

In automatic scheduling, register sharing is done among variable of different bit width.

Register count can be controlled by register sharing among variable of different bit width, but there is a possibility of increase in multiplexer and deterioration of wiring by sharing.

By specifying -Zmix_bitw_reg_share=NO, register sharing can be controlled after prohibiting register sharing among variables of different bit width.

By specifying -Zmix_bitw_reg_share=0_EXPAND_ONLY, register sharing can be controlled after prohibiting register sharing with signed variable and variable having large bit width. Cases of

deterioration of wiring after high level bit supplementation is done can be prevented if signed variable shares the register with variable having large bit width.

Note

- In manual scheduling, register sharing among the variable of different bit width is not supported. Because of that, it is treated as -Zmix_bitw_reg_share=NO.
- Even in Automatic scheduling, register sharing among the variable of 1 bit and variable of 2 bit or more is not supported.

20.6 Register Name

Option	Description
-Zreg_orig_name	The original signal name of variable is allocated to the register "name" (default)
-Zreg_orig_name[=#]	The original signal names are reflected to the allocated register name up to #-th signal
-Zreg_orig_name=NO	Change the register name to same as prior to β3.4x
-Zreg_name_func[=YES]	The function name is appended to the register's name
-Zreg_name_func=NO	The function name is not added to the register's name (default)
-Zreg_name_len=#	Restricts the number of characters for the register's name up to # (default value 32 characters)
-Zreg_head_len=ALL	Reflects all the variable and functional name to the register name (default)
-Zreg_head_len=#	Reflects only the initial # characters of the variable name to the register name.
-Zreg_head_len=#1:#2	Reflects only the initial #1 characters of functional name and initial #2 characters of the variable name to the register name.

The names of some registers are decided on the basis of name of variable allocated to that register.

In case of 1 bit register, "FF_" is added before the allocated variable name joined by "_", whereas for multiple bit register "RG_" is added e.g. for an 8 bit variable, foo8, the name of the allocated register becomes RG_foo8 whereas for a 1 bit variable, foo1, the name becomes FF_foo1.

Register, to which variable 'a' and variable 'b' are allocated, becomes RG_a_b. However, when the register is allocated by only one variable in manual scheduling, a variable name becomes a register name as in β3.4.x version or version prior to it. As a result, the names of the registers above are foo8 and foo1 respectively.

If -Zreg_orig_name=NO is specified, the register name become same as prior to β 3.4.x version.

Some of the variables allocated to registers were named as (FF_01,FF_02, ..., RG_01, RG_02, ...) prior to β 3.4.x version, however, from β 3.5 onwards, changes are made to reflect the variable name allocated to register.

If -Zreg_orig_name=# is specified, variable name is reflected in register allocated with # or less number of variables. The register allocated with more than # number of variables is named same as prior to β 3.4.x (FF_01,FF_02,..., RG_01,RG_02, ...).

If -Zreg_name_func[=YES] is specified, function name is appended to register name. In case -Zreg_name_func[=YES] is specified, the register that is allocated with 8 bit local variable foo8 of function func1 becomes RG_func1_foo8.

If -Zreg_name_len=# is specified, the length for the register name is limited to # characters. If the limit is not specified, 32 is the maximum limit. If the variable name is larger than the specified limit, the name is not appended to the register's name.

Register's name starts with RL or FL instead of RG or FG if the character limit is exceeded.

In case -Zreg_name_len=10 and -Zreg_orig_name=3 are specified and the register's to be allocated to three 8-bit global variables foo1, foo2 and foo3, the register name becomes RL_foo1.

Note: - Character limit can be specified between 4 to 64 characters

If the variable and function names are long, only the first few characters of the variable name and function name are specified by using "-Zreg_head_len" option. When the "-Zreg_head_len=#" option is specified, the function name and first # characters of the variable name are assigned to the register name. However if the "-Zreg_head_len=#1:#2" option is specified, the first#1 characters of the function name and the first#2 characters of variable name are reflected.

Note:

- There are instances where _# is appended like "foo_1" in order to avoid overlapping of the register name.
- Some of the registers such as Input / Output registers are not included in this naming convention.
- In auto-pipeline scheduling, the pipeline stage number is appended to the register name that begins with RP, FP, or RL, FL.

20.7 Sharing of Functional Unit and Memory allocation

Option	Description
-Zassign=1	Use allocation algorithm 1
-Zassign=2	Use allocation algorithm 2

There are two methods of resource allocation of memory port and functional unit.

1. Algorithm 1

It considers optimization but depending on the circuit the allocation candidates can be huge in number and so in this algorithm the synthesis may take time.

2. Algorithm 2

It is very fast however in this allocation algorithm optimization is compromised to some extent. Moreover, functionality such as avoiding false loop can be specified.

Resource allocation method of functional unit and memory support of manual scheduling is carried out with algorithm 1 by default. Even if option "-Zassgin=1" is specified, it is carried out with same algorithm. When option "-Zassign=2" is specified, it is carried out with algorithm 2.

Resource allocation method of functional unit and memory port of automatic pipeline scheduling as well as automatic scheduling can be performed with the same algorithm as that of option - Zassign=2. Even if -Zassign=1 is specified, it is ignored.

Resource allocation algorithm used in each option and mode is shown below:

Option	Manual	Automatic	Automatic Pipeline
Default	Algorithm 1	Algorithm 2	Algorithm 2
-Zassign=1	Algorithm 1	Algorithm 2	Algorithm 2
-Zassign=2	Algorithm 2	Algorithm 2	Algorithm 2

Options to be used in allocation algorithm 1

Option	Description
-AF	Register and functional unit allocation is optimized for synthesis time by reducing the number of allocation candidates.

The above mentioned option is used in the allocation algorithm 1,

-AF : Allocation is done by considering the evaluating allocation candidate count. Generally, the allocation time can be shortened to a large extent.

Options to be used during allocation algorithm 2

Option	Description
-Aa	Allocation minimizes the area by sharing functional units (default)
-Ad	Allocation reduces the multiplexing by avoiding functional unit sharing
-Af	Allocation is done by carrying out the false loop detection
-A-f	Allocation is done without the false loop detection (default)
-At	Allocation to prioritize the delay reduction (FPGA: default)
-A-t	Allocation that does not prioritize the delay reduction (ASIC: default)

The above options are used in the allocation algorithm 2 and are invalid in the allocation algorithm 1.

-Aa: Resource allocation where reduction of area has priority

-Ad: Resource allocation where multiplexer delay reduction has priority. It reduces the multiplexer's selection lines by the usage of the maximum number of functional units as specified under functional unit constraints.

(-Aa and -Ad are exclusive options).

-Af: Resource allocation that avoids the false loops which could be generated during functional unit sharing.

3. To avoid the loop there is an error when functional units are not sufficient in the constraint count.

-A-f: Do not avoid false loops while functional unit allocation.

-At: Resource allocation which prioritizes the reduction of critical path delay

1. It is valid only when basic library (BLIB) file is specified by using the "-lb" option.
2. In order to reduce the delay, more number of functional units and decoders of register arrays will be used.
3. It might take long time to synthesis.

-A-t: Resource allocation which doesn't prioritize the reduction of critical path delay

Option	Description
-ZZfu_noshare_states	A functional unit is not shared by the operations in different states.
-ZZfu_noshare_states=all	A functional unit is not shared by the operations in different states.
-ZZfu_noshare_states=#[,#...]	A functional unit allocated by the operation in the state # is not shared by the operations in the other states.

Sometimes, sharing of functional units may lead to generation of False loops/False paths in a circuit. If the above mentioned options are specified, the functional units are not shared to avoid the false loop generation. Multiple states can be specified by using the delimiter",," e.g., "-ZZfu_noshare_states=3,4".

Input Description

```

in ter flag1;
in ter(0:8) a,b,c;
in ter(0:8) d,e,f;
out reg(0:8) g;

process example1(){
    /* step1 */
    if( flag1 ){
        g = (a + b) - 1;
    }else{
        g = (c + d) - 2;
    }
    $

    /* step 2 */
    g = ( e - f ) + 3 ;
    $
}

```

Functional unit library file

```

#@LIB { example1 }
@FLIB{
    NAME      add08
    KIND      +
    BITWIDTH  8
    DELAY     400
    AREA      200
}
@FLIB{
    NAME      sub08
    KIND      -
    BITWIDTH  8
    DELAY     420
    AREA      210
}
#@END { example1 }

```

Functional unit count file

```

#@CNT { example1 }
@FCNT {
    NAME      add08
    LIMIT    1
}
@FCNT {
    NAME      sub08
    LIMIT    1
}
#@END { example1 }

```

In above example, if the “-ZZfu_noshare_states” option is specified, the addition of “a + b” and “c + d” in step 1 are allocated to the same adder. However, the addition of “(e - f) + 3” in step 2 is allocated to a separate adder. Hence, altogether two adders will be used.

Limitations

- Because of restriction on functional unit number specified in the functional unit count file, the constraint is not specified for the entire design but only for a designated state. Hence, there are cases when the number of functional unit generated during synthesis of the circuit becomes more than the constraint number specified in the functional unit count file.
- Functional units can be shared among conditionally exclusive operations even in same state.

20.8 Destination Registers Specification for Variables

Attribute	Description	Target Object
/* Cyber reg_bind */	In automatic scheduling, specified variable are not allocated to wiring but to register	Variable
/* Cyber no_share */ /* Cyber share_name = <i>name</i> */	Specified variable does not share register with other variables Change the register name of sharing object to "name"	Variable

20.8.1 Basic Function

When specified with share_name attribute, then all the signals specifying the same attribute value are allocated to similar register. Register sharing is not done with signal in which attribute is not specified or signal in which attribute value is different.

For example, below r1 and r2 are allocated to register of similar name (R5) and register sharing is not done with any other signal.

If no_share attribute is specified in signal, then register sharing is not done with other signal.

```
reg (0:8) r1 /* Cyber share_name = R5 */;
reg (0:8) r2 /* Cyber share_name = R5 */;
reg (0:8) r3 /* Cyber share_name = R4 */;
reg (0:8) r4 /* Cyber no_share */;
```

In the case of manual scheduling, the variable declared as "reg" is definitely allocated to a register. However, in auto scheduling, it is not fixed that variable would be allocated to the register; it might be allocated to a wire. In that case, share_name attribute, no_share attribute is ignored. However, even in auto scheduling, specified variable can be allocated to a register in auto scheduling mode by adding reg_bind attribute (For more details refer **section 19.7**).

Moreover, multiple registers might be required against 1 variable when folding is specified and synthesis is done with automatic scheduling. In that case, multiple registers are created having the name specified by the attribute with an underscore and number.

20.8.2 Function available when specified in Local Variable within Function

This section explains about the available function when share_name attribute is specified in local variable within inline function.

In local variable of inline function, if ? is specified in attribute value like /*Cyber share_name=?/, then all the internal variables generated when function is called multiple times can be allocated to the similar register.

Below, function func is called multiple times and local variable x generated respectively after functional expansion is allocated to similar register. Further regarding function func2, local variable x generated respectively after functional expansion is allocated to similar register but check whether variable x in func is allocated to a different register. In case x in func, func2 is to be allocated to similar register, then it will be good if similar attribute value (ex. /*Cyber share_name=xyz*/) is specified in both.

Input description

```

...
process main() {
    ...
    func();//(1)
    func();//(2)
    //By calling (1) and (2),
    //internal variable x generated respectively is
    //allocated to similar register.
    ...
    func2();//(3)
    func2();//(4)
    //By calling (3) and (4),
    //internal variable x generated respectively is
    //allocated to similar register.
    //Register generated with (1), (2) becomes
    //different register.
    ...
}
void func(void) {
    int x/*Cyber share_name=?, reg_bind*/;
    ...
}
void func2(void) {
    int x/*Cyber share_name=?, reg_bind*/;
    ...
}
```

20.9 Specifying Destination Memory for Array

Attribute	Description	Target Object
/* Cyber sig2mu= name*/	The array is allocated to the memory "name"	Array
/* Cyber sig2mu_port= name.r#w#/*/	The array is allocated to the memory "name" with the specified ports to be used in array accesses.	Array

/* Cyber mu_port = #*/	Specifies the port to be used in array accesses.	Array Access
------------------------	--	--------------

Memory library file and memory count file is needed while converting arrays into memories. An array automatically gets allocated to the memory with respect to its bit-width, the number of the elements, array references, assignments, etc. The array can be allocated to a particular memory type by specifying the "sig2mu" and "sig2mu_port" attribute at the time of array declaration.

Memory name written in the second field in the memory count file is specified to the attribute value of "sig2mu" attribute..

Memory ports to be used for reading and writing of arrays may be specified using the "sig2mu_port" attribute. It is needed to specify the respective port numbers, such as "R1W1". These ports are used for reading and writing. The port numbers are referenced by specifying them by putting a "." after memory name in the attribute value for the "sig2mu_port" attribute.

For memories using separate ports for reading and writing, the port number are designated like "R1.W1", with the port 1 specified as the read port followed by write port as port 2, port 3 and so on.

When a memory uses the same port for reading and writing, such as in case of "RW3", the port numbers are allocated on a sequential basis.

Regarding the array in which the "sig2mu_port" attribute is added, it is possible to specify a separate port number by specifying the "mu_port" attribute. This can be specified in the case of referring and assigning the port other than the default port.

In the description example given below, array "bar" is allocated to a memory type "memB" and array "foo" is allocated to a memory type "memC". For array "foo" it is specified to use port no. 1 for reference, and port no. 3 for writing. In the description it is specified to use port no. 4 in the individual write command "foo[x] = y" and port no. 2 in individual reference "z = foo[x]".

Memory Library file (MLIB)

```
#@VERSION { 1.00 }
#@LIB { func }
@MLIB {
    NAME      memA
    KIND      R1,W1
    BITWIDTH  8
    DELAY     100
    ...
}
@MLIB {
    NAME      memB
    KIND      RW1
    BITWIDTH  8
    DELAY     120
    ...
}
@MLIB {
    NAME      memC
    KIND      R2,W2
    BITWIDTH  16
    DELAY     200
    ...
}
#@END { func }
```

Memory Count file (MCNT)

```
#@VERSION { 1.00 }
#@CNT { func }
@MCNT {
    NAME      memA
    LIMIT    1
}
@MCNT {
    NAME      memB
    LIMIT    1
}
@MCNT {
    NAME      memC
    LIMIT    1
}
#@END { func }
```

Description

```
mem(0:8)  bar[100] /* Cyber sig2mu = memB */;
mem(0:16) foo[200] /* Cyber sig2mu_port = memC.r1w3 */;

process func(){

    foo[ x ]/* Cyber mu_port = 4 */= y ;

    z = foo[ x ] /* Cyber mu_port = 2 */;

}
```

20.10 Specification of functional unit type in operator allocation

20.10.1 Detailed specification method of allocated functional unit Detailed specification method o

Attribute	Description	Target Object
/* Cyber ope2fu = name */	Allocates the specified operation to the functional unit type name	Expression
/* Cyber ope2fu_ex = name */	Allocates the specified operation to the functional unit type name, other operations are not allocated to this functional unit type	Expression

An operator is automatically allocated to the functional unit specified in functional unit constraint file based on the bit width, input resources etc. An operation can be allocated to a particular functional unit type by appending ope2fu or ope2fu_ex attribute.

The attribute value of ope2fu and ope2fu_ex attribute must be one of the functional unit type names in the 2nd field of the functional unit constraint files.

The difference between ope2fu and ope2fu_ex is the functional unit type specified by ope2fu can be shared with other operations , whereas the functional unit specified by ope2fu_ex is not shared with other operations.

In the example described below, the operator "+" of "b + c" and the operator "+" of "a + ..." has functional unit type "add08" specified in the ope2fu_ex attribute. As a result, no operations are allocated to the functional unit except for these two operations "add08". On the other hand, the operator "+" of "d + e" has functional unit type"addsub08" specified in the ope2fu attribute. Therefore, this operation is allocated to functional unit type addsub08.

Functional unit library file (FLIB)

```
#@LIB { example1 }
@FLIB{
    NAME      fadd08
    KIND      +
    BITWIDTH  8
    DELAY    220
    AREA     400
}
@FLIB{
    NAME      addsub08
    KIND      +,-
    BITWIDTH  8
    DELAY    420
    AREA     200
}
#@END { example1 }
```

Functional unit count file (FCNT)

```
#@CNT { example1 }
@FCNT {
    NAME      fadd08
    LIMIT    2
}
@FCNT {
    NAME      addsub08
    LIMIT    2
}
#@END { example1 }
```

Description

```
process module1(){
    out1 = a /* Cyber ope2fu_ex = fadd08 */
            ( b /* Cyber ope2fu_ex = fadd08 */ c );
    out2 = d /* Cyber ope2fu = addsub08 */ e ;
}
```

20.11 Summary

This section covered the following:

- When the “-Zmix_share=OPE” or “-Zmix_share=NO” option is specified, the variables of different signs are allocated to different registers even if these variables can share a register.
- There are three different types of functional units.
- Units to perform unsigned operations
- Units to perform signed operations
- Based on control signal, units to perform signed or unsigned operations
- In manual scheduling, multiple “reg” variables can share one register by analyzing the scope of different “reg” variables.
- The “-o=R” option is used to simplify debugging and testing where register sharing is not desirable.
- The two algorithms used for resource allocation in auto scheduling are allocation algorithm 1 and allocation algorithm 2.
- The allocation algorithm 1 is implemented by specifying the “-Zassign=1” option.
- The allocation algorithm 2 is implemented by specifying the “-Zassign=2” option and it is the default algorithm.
- In the Cyber version β3.5 and onwards, the name of variable is reflected to allocated register.
- On specifying the “-Zreg_org_name=NO” option, the register name becomes same as that with the Cyber version β3.4 or earlier versions.
- With “sig2reg”, the allocation is done to share the registers among different variables.
- With “sig2reg_ex”, register is not shared with other variables.
- In the case of manual scheduling, the variable declared as “reg” is definitely allocated to a register.
- In auto scheduling, a register variable might be allocated to a wire.
- An array can be allocated to a particular memory type by specifying the “sig2mu” and “sig2mu_port” attributes at the time of array declaration.

21 Time Constraints

21.1 Overview

The limitation of execution timing in automatic scheduling is called time constraints. Following are the time constraints which can be specified in Behavioral Synthesis System of Cyber.

- Specific control block is synthesized as manual scheduling mode.
- Execution timing of input-output in input port and output port is specified.
- Latency of a specific loop (for statement, while statement, do while statement) is specified as 10 clock.
- Specify in such a way that scheduling is done when at least more than 1 cycle from assignment to reference has gone.
- Specify the timing in a way so that 2 operations are executed when more than 1 cycle has gone.

However, in present conditions the specified constraints may not be fulfilled. This chapter explains the details of various time constraints and limitations, and the messages generated at the time when constraints are not fulfilled.

21.2 Scheduling Block

21.2.1 Overview

Scheduling block is a method to specify and synthesize clock cycle explicitly in automatic scheduling mode, which is similar to specific control block in manual scheduling mode. The following items describe how synthesis is done in the block that specified scheduling block.

- BDL Input
 - The cycle ends at the point where \$ is present and does not end besides that. (However, as for the wait, watch statement the cycle ends when a cycle similar to manual scheduling moves around a loop)
 - The cycle ends with goto and does not end with goto0
- SystemC
 - Cycle ends at the point where wait is present and does not end besides that.
 - Cycle does not end with goto

In the following description, since the added for statement of scheduling_block attribute is synthesized in a way similar to manual scheduling therefore, the external input is saved to ary1[] and ary2[] alternately and this for statement is executed in 32 cycles in total.

```

in ter(0:8) i1,i2;
out reg(0:8) o1,o2;
reg(0:8) ary1[16];
reg(0:8) ary2[16];
process main(){
    char i;
    /* Cyber scheduling_block */
    for(i = 0; i < 16; i++ $ ){
        ary1[i] = i1;
        $ 
        ary2[i] = i2;
    }
    ....
}

```

With this function, the circuit including the part for which limitation of timing is important such as external and interface part etc. can be synthesized easily by using the automatic scheduling mode. Further, since the output is done through register in case of assignment to out reg type, sc_out, sc_signal type of SystemC, therefore the timing which can be observed externally is after 1 cycle of assignment.

21.2.2 Related Options

Option	Description
-Zscheduling_block=transparent	Parallelize with other processing when the value of scheduling_block attribute is not there (default)
-Zscheduling_block=non-transparent	Does not parallelize with other processing when the value of scheduling_block attribute is not there

21.2.3 Related Attributes

Attribute	Description	Setting target
/* Cyber scheduling_block */	Change the specified block to scheduling block (Parallelizing with other processing is not possible. Change is possible with option -Zscheduling_block)	Block Control statement Function definition
/* Cyber scheduling_block = transparent */	Change the specified block to scheduling block (Parallelizing with other processing is possible.)	Block Control statement Function definition

<pre style="margin: 0;">/* Cyber scheduling_block = non-transparent */</pre>	<p>Change the specified block to scheduling block (Parallelizing with other processing is not possible.)</p>	<p>Block Control statement Function definition</p>
---	--	--

21.2.4 Overview

Scheduling_block attribute can be added only in block, function definition and control statement (if/switch/while/dowhile/for). In case it is added in if statement then the else clause is also considered to be in the scheduling block. In case it is added in function definition then all the functions are considered to be in the scheduling block. Further, the scheduling_block attribute is ignored in manual scheduling mode.

When scheduling_block is specified as non-transparent, the description till the first \$ inside the scheduling block is executed in cycle similar to the upper description of scheduling block if possible. Similarly, the description till the last \$ inside the scheduling block is executed in cycle similar to the lower description of scheduling block if possible. Moreover, in case \$ is not included in scheduling block, then the internal part of scheduling block is executed in 1 cycle and its external part is parallelized to the extent possible.

```

in ter(0:8) i1,i2,i3,i4,i5,i6;
out reg(0:8) o1,o2,o3,o4,o5,o6;
process main(){
    o1 = i1; // Outside the scheduling block
    /* Cyber scheduling_block = non-transparent */
    {
        o2 = i2; // Executed parallel to o1 = i1
        $
        o3 = i3; // These 2 statements are executed
        o4 = i4; // in the cycle next to o2 = i2
        $
        o5 = i5 // Executed parallel to o6 = i6
    }
    o6 = i6; // Outside the scheduling block
}
```

When scheduling_block is specified as transparent, irrespective of whether \$ is there/not there inside the block the internal timing is retained and is parallelized to external to the extent possible.

```

in ter(0:8) i1,i2,i3,i4,i5,i6;
out reg(0:8) o1,o2,o3,o4,o5,o6;
process main(){
    o1 = i1; // Each processing inside the scheduling block
              // can be executed in parallel
    /* Cyber scheduling_block = transparent */
    {
        o2 = i2; // Start timing of this processing is not
specified
        $
        o3 = i3; // These 2 statements are executed in
        o4 = i4; // the cycle next to o2 = i2
        $
        o5 = i5 // Executed in the cycle next to o3 = i3; o4 = i4;
    }
    o6 = i6; // Each processing inside the scheduling block
              // can be executed in parallel
}

```

As regard to the function called inside the scheduling block, the internal part of that function becomes the target of scheduling block. However, that part will not become the target of scheduling block even if same function is called outside the scheduling block.

```

process main() {
    /* Cyber scheduling_block */
    {
        o1 = func(i1); // $ inside the func() is valid

    }
    o2 = func(i2); // $ inside the func() is not valid
}
int
func(int x) {
    $
    return x;
}

```

There will be an error if synthesis cannot be done between \$ inside scheduling block and \$ in 1 cycle. This occurs because of using the same input-output variable multiple times, reference after assignment to reg type variable in same cycle, multiple assignments to same array where subscript is not excluded, insufficient functional unit count, excess of delays etc.

21.2.5 Limitations

Following items cannot be described in the scheduling block.

- Functional call converted to "goto"
- Loop to be folded
- Variables of "ter" type except in/out
- Split description of pipeline memory
- Label that will act as jump destination from the external part of scheduling block

(however, the jump is possible to a level that is immediate next to the \$)

- Using Goto0 to jump inside other scheduling blocks
- Assignment description using concatenation statement
- Time constraint settings by attribute "latency_loop"
- Scheduling block specification for the function definition of the process function or the function specified as the process function with -Ztop.

At the time of SystemC input, in case there exists a read() till the first wait() inside a scheduling block, input of that value from outside is actually done at the timing of wait() present in the above mentioned description of scheduling block, however, input itself is considered to be inside scheduling block. Accordingly, error will be generated when input of value is not in same cycle. Similarly, in case there is write() after last wait() inside a scheduling block, output of value for external part is done at the timing of wait() present under scheduling block, however, that output itself also is considered to be inside scheduling block.

In the following example, if it is considered that "ack" will not output until wait() is executed, there is a possibility that cycle different than input of data_in will be formed when multiplication takes 2 or more cycles. In order to prevent the occurrence of such problems, synthesis is done in such a way that input and output are carried out in that cycle when read() and write() are included in a scheduling block.

```
// Cyber scheduling_block
{
    do {
        wait();
    } while(! valid.read());
    data = data_in.read();
    ack.write(1);
    // Ensure that it is written to ack in cycle same to input of
    data_in
    // (ack is observed 1 cycle after assignment because ack is
    registered output )
}
square = data * data;
wait();
```

21.2.6 Usage Example

- Handshake through scheduling block

After the start signal, 10 times the data will be entered from input signal i1 in each cycle. It will be mentioned as per below in case of calculating that value.

```

char buf[10];
/* Cyber scheduling_block = transparent */
{
    wait(start);
    /* Cyber unroll_times = all */
    for(i=0;i<10;i++) {
        buf[i] = i1;
        $
    }
}
/* Cyber unroll_times = all */
for(i=0;i<10;i++) {
    ret+=buf[i];
}
o1 = ret ;

```

At that time, it is possible to start addition processing without waiting for the completion of 10 times input by making the scheduling block transparent.

If it is made non-transparent, then the additional processing is executed after waiting for completion of input for i1.

- Parallelization in scheduling blocks
- transparent is enabled even when parallelization is possible among scheduling blocks.

In the following example, it has been decided as a protocol that the expression of (1)(2) should be at a distance of 1 cycle and the expression of (3)(4) should be at a distance of 1 cycle. However transparent function is used and each scheduling block is executed at the right position in case flow/sequence of each scheduling block is not specified.

```

/* Cyber scheduling_block = transparent */
{
    o1 = i1; /* (1) */
    $
    o2 = i2; /* (2) */
}
/* Cyber scheduling_block = transparent */
{
    o3 = i3; /* (3) */
    $
    o4 = i4; /* (4) */
}

```

If scheduling blocks becomes parallel after synthesizing then the result will be same as below. Please note that though there are 2 \$ in the description, however after synthesizing there will be only 1 \$.

```
/* Cyber scheduling_block = transparent */
{
    o1 = i1; /* (1) */
    o3 = i3; /* (3) */
    $
    o2 = i2; /* (2) */
    o4 = i4; /* (4) */
}
```

On the other hand, in case scheduling block is specified with non-transparent then there can be an order relation in which scheduling block of (3)(4) is executed after the execution of scheduling block of (1)(2).

```
/* Cyber scheduling_block = non-transparent */
{
    o1 = i1; /* (1) */
    $
    o2 = i2; /* (2) */
}
/* Cyber scheduling_block = non-transparent */
{
    o3 = i3; /* (3) */
    $
    o4 = i4; /* (4) */
}
```

A result same as below will be there after synthesizing. In that case the numbers of \$ will be 2 as mentioned in the description.

```
/* Cyber scheduling_block = non-transparent */
{
    o1 = i1; /* (1) */
    $
    o2 = i2; /* (2) */
    o3 = i3; /* (3) */
    $
    o4 = i4; /* (4) */
}
```

21.3 Time Constraint Specification for Port

21.3.1 Overview

The execution timing (count of latency desired to be input) can be specified for respective input port (including in port; input array), output port (including out port; output array), and I/O port (inout port; I/O array), by specifying attribute latency_set in respective ports.

Attribute	Description	Settings target
<code>/* Cyber latency_set = num */</code>	Specify the execution timing of signal for which time constraint is to be specified	Input signal, output signal
<code>/* Cyber latency_set = num:num */</code>	Specify the execution timing of signal for which time constraint is to be specified	Bi-directional I/O signal

21.3.2 Description

Input and output timing of signal, whose value of attribute latency_set is 1, is taken as reference, and from there I/O timing can be specified. In case of specifying timing for bi-directional I/O signals, input timing and output timing are joined by colon (":") respectively. In case either input timing or output timing needs to be specified for a bi-directional I/O signal, there will be following description.

```
Specification of input timing only
inout unsigned char inout1 /* Cyber latency_set = 1 */;
```

```
Specification of output timing only
inout unsigned char inout2 /* Cyber latency_set = :2 */;
```

Figure 83 shows the example of BDL description specifying latency_set. If this description is synthesized, synthesis will be carried out in the following manner.

- Input signal in0, in1, in2 where latency_set = 1 is specified is taken as reference (0 clock: simultaneously)
- Fetch in input from input signal in3 (latency_set = 3) after 2 clocks.
- Fetch in input from I/O signal inout1 (latency_set = 5) after 4 clocks.
- Output to I/O signal inout1 (latency_set = 6) after 5 clocks
- Output to output signal out1 (latency_set = 7) after 6 clocks

Further, synthesis error will be generated when specified timing specification is not fulfilled.

21.3.3 Points to be noted

- As a register is inserted to an in reg signal, if latency_set is specified for in reg signal, be careful as when circuit actually fetches the data, the timing becomes faster by 1 clock than the specified timing.
- As a register is inserted to an out reg signal, if latency_set is specified for out reg, be careful as it can be observed that actually output value becomes delayed by 1 clock than the specified timing. The same is true also in case attribute port_type = REG is specified for out var signal.
- Attribute latency_set can be specified only for input signal, output signal, bi-directional I/O signal. It cannot be specified for signals such as global variable, local variable, signal/array that is declared as shared (In case specified, error is generated and attribute is ignored). Further, it cannot be specified for port (defmod module port) of subordinate hierarchy also.
- Signal, where latency_set is specified, cannot be used in allstates block. However, in case it is used, error will be generated and synthesis will be terminated.
- In case pipelined or folded loop is (dynamically) stalled, be careful about its impact. In other words, when stalled input and output operations will not be carried out and the I/O timing will shift backwards.

```

in unsigned char in0/* Cyber latency_set = 1 */;
in unsigned char in1/* Cyber latency_set = 1 */;
in unsigned char in2/* Cyber latency_set = 1 */;
in unsigned char in3/* Cyber latency_set = 3 */;
inout unsigned char inout1/* Cyber latency_set = 5:6 */;
out unsigned char out0 /* Cyber latency_set = 7 */;

process foo(){
    unsigned char v0,v1,v2;
    /* Cyber folding = 2 */
    for(;;) {
        input(in0); input(in1); input(in2); input(in3); input(inout1);
        v0 = in0 + in1 + in2 ;
        v1 = v0 - in3 ;
        v2 = v1 + inout1 ;
        out0 = v2 ;
        inout1 = v2 ;
        output(out0); output(inout1);
    }
}

```

Figure 83: latency_set attribute

21.3.4 Limitations

- Synthesis is supported only when following synthesis mode or option are specified.
 - In case of automatic pipelined synthesis, or
 - In case entire process is folded, or
 - In case option -Zsame_cycles is specified

In case synthesis is done in manual scheduling mode, latency_set attribute is ignored and synthesis is continued. At this time, message conveying that attribute is ignored is generated.

- In case input() statement and output() statement of signal, where latency_set attribute is specified, are deleted during optimization etc, warning or synthesis error will be generated..
- Signal for which attribute latency_set is specified should appear only once in BDL description. If it appears at multiple locations, an error message is generated and synthesis is terminated.
 - If subscript of input array, output array, and bi-directional I/O array is different, it will be considered as different signal.
 - If the signal is copied by unrolling or function inline expansion, an error will be generated.

Further, if it does not appear even once, a warning will be generated but no synthesis error will be generated.

- Signal, in which attribute latency_set is specified can be used inside scheduling block (scheduling_block). However, clock specifier (\$) cannot be used inside scheduling block. In case it is used, scheduling may get failed.

Further, signal, in which latency_set is specified, cannot be used inside successive scheduling block (scheduling_block). In case it is used, scheduling may get failed.

21.4 Specifying the latency constraints of loop

21.4.1 Overview

The cycle of loop latency can be controlled by specifying attribute latency_loop in loop, and specifying attribute latency_loop_ext in continue statement and break statement.

21.4.2 Related attributes

- Following attributes are used for specifying latency control for loop.

Attribute	Description	Settings target
<code>/* Cyber latency_loop = range */</code>	Specify the time constraint related to loop latency in <i>range</i>	Loop statement
<code>/*Cyber latency_loop_ext = range */</code>	Specify the time constraint related to loop latency at the time of breaking, continuing the loop, in <i>range</i>	Break statement Continue statement

- The following range specifications can be specified in the above range.

Range specification	Description
<i>nT</i>	<i>n</i> cycle (fixed constraint) (many times the constraint is not fulfilled)
<i>nT-</i>	at least <i>n</i> cycle (minimum constraint)
<i>-nT</i>	within <i>n</i> cycle (maximum constraint)
<i>nT-mT</i>	more than <i>n</i> cycle and within <i>m</i> cycle (interval constraint)

- Number of cycles of loop latency at the time of continuing or breaking the loop with the methods mentioned above can be specified individually in the range of latency_loop_ext attribute also.
- In latency_loop_ext attribute, ASAP can be specified which means as soon as possible.
`(*Cyber latency_loop_ext = ASAP*)`
- In case latency_loop_ext attribute is not specified, then during continue the cycle similar to latency_loop attribute will be specified and at the time of break it will be treated as the case when `(* Cyber latency_loop_ext = ASAP*)` has been specified.
- The latency_loop_ext attribute can only be specified in the break statement, continue statement of loop specified by latency_loop attribute.

21.4.3 Description

In case of specifying the latency constraint of loop (for, while, do-while), cycle is specified for loop statements by using "latency_loop" attribute in number of cycles.

In case of specifying latency constraint at the time of break and continue inside a loop, cycle is specified for loop statements in break or continue statements by using latency_loop_ext" attribute in number of cycles.

```

/* cyber latency_loop = 2T */
for ( ; ) { // latency of loop is 2 cycles
    ...
    if( ){
        break; // Since there is no specification of
        //latency_loop_ext the loop latency at this break will be
        // as soon as possible
    }
    if( ){
        continue; // Since there is no specification of
        //latency_loop_ext the loop latency at the time of this //
        // continue will be two cycles complying the loop latency
    }
}
/* cyber latency_loop = 2T- */
while ( ) { // loop latency is 2 cycles or more
    ...
    if( ) {
        /* Cyber latency_loop_ext = 3T */
        break; // loop latency is 3 cycles
        // at the time of this break
    }
    if( ) {
        /* Cyber latency_loop_ext = ASAP */
        continue; //the loop latency at the time of this continue
        //will be as soon as possible
    }
}

```

21.4.4 Limitations

- If latency constraint is added to "for" statement, which can be unrolled, an error will occur. To avoid the occurrence of the error, the "unroll_times=0" attribute should be added.
- In case of nested loops, this constraint is only applicable for innermost loops.
- In case there are functions other than goto, labels or inline functions In processing within, sometimes the constraints are not applicable.
- Specified latency constraint will not be applicable in case of "do-while" statement if it contains a "continue" statement.

21.5 Specifying time constraints of array access

21.5.1 Related attributes

- Following attributes are used for specifying the time constraints of array access

Attribute	Description	Settings target
<code>/* Cyber interval_array = array_interval */</code>	Set the time constraint specified in <i>array_interval</i> in the array	Array
<code>/* Cyber group_interval_array = array_interval */</code>	Set the time constraint specified in <i>array_interval</i> in the array	Array

- "/" (slash) which specifies condition of multiple time constraints simultaneously can be specified for interval of interval_array attribute.

Example of abbreviation	Description
<code>R/W->R:range</code>	Abbreviation of "R->R:range & W->R:range"

- Time constraint condition can be specified in the array_interval of "interval_array" attribute by using the following interval specification and multiple conditions can be specified by joining them with "&" (AND).

Interval specification	Description
<code>R->W :range</code>	Specification of the cycle count from reference initiation to assignment initiation within the scope of <i>range</i>
<code>R->R: range</code>	Specification of the cycle count from reference initiation to next reference initiation within the scope of <i>range</i> .
<code>W->W: range</code>	Specification of the cycle count from assignment initiation to next assignment initiation within the scope of <i>range</i>
<code>W->R: range</code>	Specification of the cycle count from assignment initiation to reference initiation within the scope of <i>range</i>

Range specification	Description
nT	n cycle (fixed constraint) (many times the constraint is not fulfilled)
$nT-$	at least n cycle (minimum constraint)
$-nT$	within n cycle (maximum constraint)
$nT-mT$	more than n cycle and within m cycle (interval constraint)

- Following are allowed as value of group_interval_array attribute

Interval specification	Description
$R<->R:range$	Limit the number of cycles in all references to the specified value- "range".
$R<->W:range$	Limit the number of cycles in all reference and assignment operation to the specified value- "range"
$W<->W:range$	Limit the number of cycles in all assignments to the specified value- "range".

Range specification	Description
$0T$	Simultaneous specification
$-nT$	within n cycle (maximum constraint)

Only bidirectional specification ($<->$) is allowed for direction. For timing, one can only specify "0T" or maximum value specification using "-nT".

21.5.2 Specifying time constraints for consecutive array accesses (interval_array)

The "interval_array" is used to specify the timing of reference and assignment of array.

Example:

```
var(0:8) ary1[256] /* Cyber interval_array = R->R/W:1T- & W->W/R:2T-
*/;
    // 1 cycle or more from assignment to next (reference or
    // assignment)
    // 2 cycles or more from assignment to next (reference or
    // assignment)
mem(0:16) m1[512] /* Cyber interval_array = R/W->R/W:2T- */ ;
    // from (reference or assignment)
    // 2 cycles or more till next (reference or assignment)
```

21.5.3 Time constraints for all array accesses (group_interval_array)

In automatic pipeline scheduling, multiple array accesses mentioned in the conditional exclusion are scheduled in a single stage and attribute group_interval_array is provided to avoid hazards.

The range of occurrence of array access can be specified using the "attribute group_interval_array". For example, the attribute value "R<->R:0T" specifies that all reads are going to happen within a single state. Whereas, the attribute value "R<->R:-2T" specifies that all reads should happen within the 3 states.

21.5.4 Difference between group_interval_array and interval_array

The group_interval_array attribute differs from interval_array attribute in the following ways.
While interval_array is the time constraint between two sequential accesses, whereas the group_interval_array is the time constraint which is applicable to all occurring accesses.

```
mem ary[128] /* Cyber interval_array = R<->R:-2T */
or /* Cyber group_interval_array = R<->R:-2T */;
process main(){
    out0 = ary[adr0]; /* READ0 */
    out1 = ary[adr1]; /* READ1 */
    out2 = ary[adr2]; /* READ2 */
}
```

In the above specified description, the time constraint is set using interval_array attribute which is applicable for 'READ0 & READ1' and 'READ1 & READ2'. The group_interval_array is applicable for all intervals across "READ0", "READ1", and "READ2". Subsequently, if time constraint value of "R<->R:-2T" is specified for interval_array, this specifies an interval of up to three states for "READ0 & READ1" as well as "READ1 & READ2". This specification is possible without defining any interval requirement between "READ0 & READ2". On the contrary, if the same value is specified for

group_interval_array, it requires the accesses "READ0", "READ1", and "READ2" to be executed within an interval of three states.

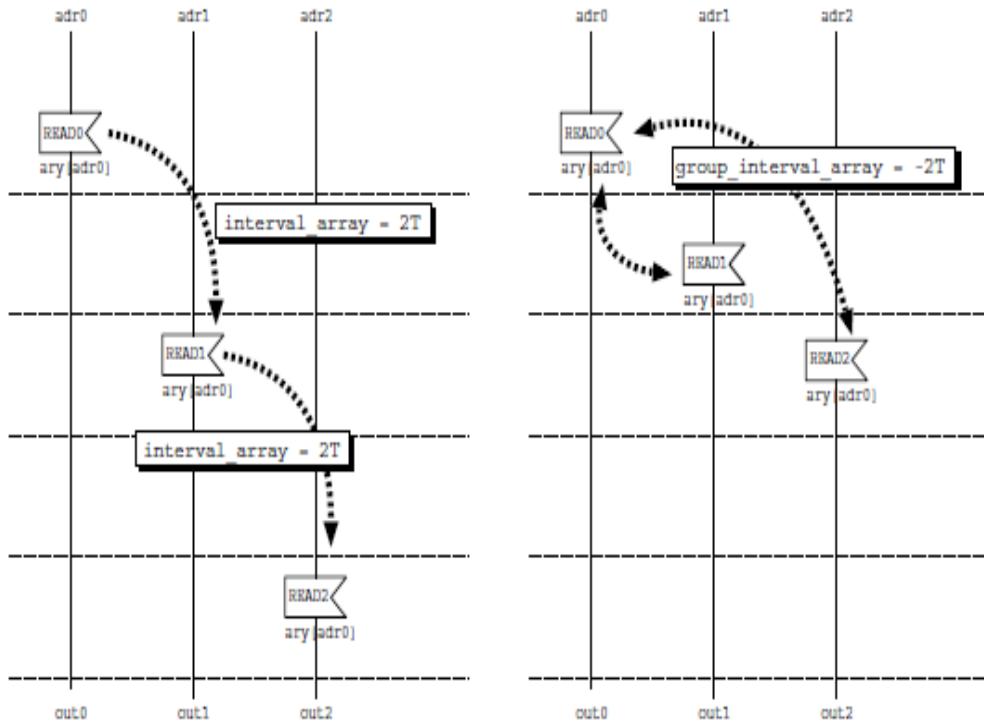


Figure 84: interval_array and group_interval_array

21.6 Specifying time constraint using id

21.6.1 Related attributes

- Following attributes are used for specifying the time constraint scheduling

Attribute	Description	Settings target
<code>/* Cyber id = <i>id</i> */</code>	Put identifier ID in Operation, Statement for which time constraint is to be specified.	Statement, Element, Assignment destination
<code>/* Cyber interval_id = <i>interval</i> */</code> <code>/* Cyber interval_id# = <i>interval</i> */</code>	Specify the step time constraint between IDs in <i>interval</i>	Process function, Function definition statement
<code>/* Cyber path_interval_id = <i>interval</i> */</code> <code>/* Cyber path_interval_id#</code>	Specify the path time constraint between IDs in <i>interval</i>	Process function, Function definition statement

= interval */		
---------------	--	--

- Following interval specification is used in *interval* of *interval_id*, *path_interval_id* attribute and the condition of time constraint is specified.

Interval specification	Description
<i>id1->id2:range</i>	Specification to maintain interval between, the initiation of process with identifier id1 to the initiation of process with identifier id2 within number of cycles specified by range.
<i>id1->id2:range id2->id1:range</i>	Specification to either maintain the interval between the beginning of the process with identifier id1 to the process with identifier id2 within number of cycles specified by value range1 or to maintain the interval between the beginning of process id2 and process id1 within the value range2. Note that interval specification is restricted to the time constraints between only 2 identifiers. " " (OR) cannot be specified for time constraints of more than three identifiers like <i>id1->id2: range id2->i3: range</i> .

- As for interval of "interval_id, path_interval_id " attribute, the abbreviated notation "/" (slash) can be specified when user wants to specify the time constraint condition among multiple identifiers id simultaneously. Similarly, to specify the same time constraint to multiple identifiers id, the abbreviated notation "<->" can be specified.

Example of abbreviated form	Description
<i>id1/id2->id3:range1</i>	Abbreviated form of " <i>id1->id3:range1&id2->id3:range1</i> "
<i>id1<->id2:range1</i>	Abbreviated form of " <i>id1->id2:range1 id2->id1:range1</i> "

However, "/" and "<->" cannot be specified simultaneously, and time constraint using "/" cannot be combined with "|" (OR).

NG1: *id1<->id2/id3:-2T*
 NG2: *id1->id2/id3:-2T | id1<-id2/id3:-2T*

- In *path_interval_id* attribute the time constraints (including <-> of abbreviated format) combined with "|"(OR) can only be specified in case it turns out to be similar block at the time of scheduling and there is a limitation that there will be an error if it turns out to be separate block.

- In the above range, the following scope specifications can be specified:

Range specification	Description
<i>nT</i>	<i>n</i> cycle (fixed constraint)
<i>nT-</i>	at least <i>n</i> cycle (minimum constraint)
<i>-nT</i>	within <i>n</i> cycle (maximum constraint)
<i>nT-mT</i>	more than <i>n</i> cycle and within <i>m</i> cycle (interval constraint)

- 2 methods of synthesizing in a way that all the conditions of multiple time constraints can be fulfilled.
 - Intervals are connected with "&"(AND) in the specification of time constraints of interval_id and path_interval_id. In the below example the execution cycle of id4 is 2 cycles till the execution step of id1 and the execution cycle of d3 is after 3 cycles from id1.

```
/* Cyber interval_id = id1->id2:-2T & id1->id3:3T- */
```

- Multiple interval_id and path_interval_id are specified by putting numeric character after them similar to interval_id1, interval_id2. In the below example, the execution cycle of id1 and id2 is after more than 3 cycles, the execution cycle of id2 and id3 is after more than 2 cycles, and id3 is executed within 6 cycles after 2 cycles of id1.

```
/* Cyber interval_id1 = id1<->id2 : 3T- & id2<->id3 : 2T- ,
interval_id2 = id1->id3 : 2T-6T */
```

In case many same numeric characters are specified after interval_id and path_interval_id then the one which was specified earlier will be ignored. However, in case character string of a part of numeric character is different as in interval_id1 and interval_id001 then it will be considered that a different numeric character has been specified and both will become valid. In the below example time constraint where OK is written is valid and the time constraint where NG is written will be ignored.

```
NG: interval_id1 = id1->id2 : 1T,
OK: interval_id1 = id1->id2 : 2T-,
OK: interval_id001 = id1->id2 : -3T,
OK: path_interval_id1 = id1->id2 : -4T
```

- Value of id attribute is specified in id of the interval_id attribute. However, for the id attribute in the loop to be unrolled, ":" can be added after in order to specify the time constraint for id with specific count of loop iteration. Iteration id specified with ". numeric character" can be specified using ".next". Next iteration id can be specified

using ".next (numeric character)" . And hence, id of next iteration of the circuit can be specified. First iteration is treated as 1.

```
/* Cyber unroll_times = all */
for(i=0;i<10;i++){
    /* Cyber id = A */
    v = a;
}
/* Cyber interval_id = A.1 -> A.2 :1T */
    Specify time constraint of 1T against A when first time i=0 and
second time i=2.
/* Cyber interval_id = A -> A.next : 1T */
    Specify time constraint of 1T against A when i=n number of times
and i=(n+1)number of times
/* Cyber interval_id = A -> A.next(2) : 1T */
    Specify time constraint of 1T against A when i=n number of times
and i=(n+2) number of times.
```

In case of nested loop, specification can be done by nesting with additional "...". Specification is done in sequence starting from internal loop. Omission is also possible, and is handled as same iteration within the loop.

21.6.2 id attribute usage method

For the specification of time constrained scheduling, identifier id can be set/fixed in element, assignment destination (left side of assignment statement) using the "id" attribute.

21.6.2.1 id attribute of statement

It is possible to specify the "id" attribute only in the assignment statement. When this attribute is put in the assignment statement, it specifies the timings as given below.

- If the assignment destination is the output variable or in-out (bi-directional) variable, then the timings of the output of assignment destination output can be specified.

```
out reg(0:8) out0;
...
/* Cyber id = ID1 */
out0 = v0;
```

- If the assignment destination is not the output variable or in-out variable, the execution timing of operation evaluated last before the assignment can be specified.

```
var(0:8) v0, v1, v2, v3;
...
/* Cyber id = ID2 */
v0 = v1 + v2 * v3;
// v0 = v1 + v2 * /* the same as Cyber id = ID2 */ v3;
/* Cyber id = ID3 */
v1 = v2; /*attribute is set to assignment itself
           if operation does not exist*/
```

21.6.2.2 id attribute of element

The elements which can attach "id" attribute are reference of input variables and operators.

- When added to operator, the timing in which that operation is executed can be specified.
- When added to reference of input variable, the timings in which the value is input from that input variable can be specified.

```
in ter(0:8) in0;
var(0:8) v0, v1, v2, v3;
...
v0 = v1 /* Cyber id = OP1 */ v2;
v3 = in0 /* Cyber id = IN1 */ ;
..
```

21.6.2.3 id attribute of assignment destination

In case id attribute is added in assignment destination the behavior will be similar to the time when id attribute is added in statement.

- If assignment destination is output variable or in-out variable then timing of this output can be specified.

```
out reg(0:8) out0;
...
out0 /* Cyber id = ID1 */ = v0;
```

- If the assignment destination is not the output variable or in-out variable, the execution timing of operation evaluated last before the assignment can be specified.

```
var(0:8) v0, v1, v2, v3;
...
v0 /* Cyber id = ID2 */ = v1 + v2 * v3;
// the same as v0 = v1 + v2 /* Cyber id = ID2 */ v3;
v1 /* Cyber id = ID3 */ = v2;
//attribute is set to assignment itself
//if operation does not exist
```

21.6.3 Specifying step time constrains between ids (interval_id)

In case user wants to specify input/output timings or timing for operation execution, "interval_id" attribute can be used.

1. In process, statement, assignment, operation, and input reference targeted as time constraints, id can be specified using the "id" attribute.
2. Time constraints between ids can be specified in the "interval_id" attribute.

Attribute interval_id is specified in the function definition statement. Attribute id can be specified for not only process function but also for functions other than process. However, blank space in the name of id value is global and since same id is specified multiple times, it cannot be used even if function is different.

Examples:

```

/* Cyber interval_id = OP1->OP2:5T- & OP1->OP3:10T- */
void func1() {
    ...
    v3 = v1 + /* Cyber id = OP1 */ v2;
    v6 = v4 + /* Cyber id = OP2 */ v5; /* Separate 5 or more cycles
from OP1 operation*/
    v9 = v7 + /* Cyber id = OP3 */ v8; /* Separate 10 or more cycles
from OP1 operation*/
    ...
}
/* Cyber interval_id = OP4->OP5:1T */
void func2() {
    ...
    v = v1 + /* Cyber id = OP4 */ v2;
    v = v + /* Cyber id = OP5 */ v3;
    ...
}

```

Since the interval_id is a thing (step time constraint) which specifies the distance between steps (states) at the time of scheduling so it does not guarantee that the execution timing between ids specified actually will be according to that. In the below example though the states between ID1 and ID2 is at the gap of 3 states but at the time of execution o1 will be output 2 cycles after the output of o0.

Example:

```

in ter(0:1) c0;
in ter(0:8) i0, i1, i2, i3;
out ter(0:8) o0, o1;
// Cyber interval_id = ID1->ID2:3T
process main() {
    var(0:8) t;
    // Cyber id=ID1
    o0 = i0;
    if(c0) {
        t = i1 + i2 + i3;
    } else {
        wait(1);
        t = i1;
    }
    // Cyber id=ID2
    o1 = t;
}

```

The timing to be scheduled for time constraint between elements of then/else node of conditional branching can be adjusted with step time constraint. In the below example, since the time constraint of OT has been set between 2 calculations therefore the calculation at then side and calculation at else side will be executed in same cycle.

Examples:

```

in ter(0:1) c0;
in ter(0:8) i0, i1, i2;
out ter(0:8) o0;
// Cyber interval_id = ID1->ID2:0T
process main() {
    var(0:8) t;
    while(1) {
        if(c0) {
            t = (i0 + i1) /* Cyber id=ID1 */ i2;
        } else {
            t = (i0 /* Cyber id=ID2 */ i1) + i2;
        }
        o0 = t;
    }
}

```

21.6.4 Specifying path time constraint between ids (path_interval_id)

path_interval_id can be used in case the user wants to specify the timing of input, output, timing when the operation is executed.

path_interval_id will be specified in a manner similar to interval_id.

path_interval_id specifies the timing when the circuit is executed actually, so it is not relevant to authenticity etc. of conditional branching and guarantees the time constraint (path time constraint) specified in all the paths. In the below example ID1 and ID2 are separated by 3 cycles therefore irrelevant to the value of c0, o0 will be output 3 cycles after the output of o0.

Example:

```

in ter(0:1) c0;
in ter(0:8) i0, i1, i2, i3;
out ter(0:8) o0, o1;
// Cyber path_interval_id = ID1->ID2:3T
process main() {
    var(0:8) t;
    // Cyber id=ID1
    o0 = i0;
    if(c0) {
        t = i1 + i2 + i3;
    } else {
        wait(1);
        t = i1;
    }
    // Cyber id=ID2
    o1 = t;
}

```

In case path time constraint is set in between the elements of then/else node of conditional branch then since it becomes the time setting of circuit execution therefore the loop surrounding to that then/else node is rotated and the timing to be executed next is controlled. In the example

below loop is rotated after the calculation (starting point) of then node is executed and the calculation (end point) of then else is executed after 3 cycles.

Example:

```

in ter(0:1) c0;
in ter(0:8) i0, i1, i2;
out ter(0:8) o0;
// Cyber path_interval_id = ID1->ID2:3T
process main() {
    var(0:8) t;
    while(1) {
        if(c0) {
            t = (i0 + i1) /* Cyber id=ID1 */ i2;
        } else {
            t = (i0 /* Cyber id=ID2 */ i1) + i2;
        }
        o0 = t;
    }
}

```

In case an element in which number of cycles during execution is not known is between starting point and end point the path time constraint does not fulfill the time constraint with a decided upper limit as like nT. In the below example since there is a wait loop with unknown number of cycles during execution so path time constraint is not fulfilled and error occurs.

Example:

```

in ter(0:1) c0;
in ter(0:8) i0, i1;
out ter(0:8) o0, o1;
// Cyber path_interval_id = ID1->ID2:3T
process main() {
    // Cyber id=ID1
    o0 = i0;
    wait(c0); // The number of loop iteration is indefinite
    // Cyber id=ID2
    o1 = i1;
}

```

Path time constraint can also specify the cycle interval between starting point and the starting point after 1 loop is rotated with .next. In that case .next can only be set in end point side of time constraint id and it is considered as time constraint specifying the timing of execution of end point after the rotation of 1 cycle of most inner side loop statements (while, for, do-while statement) to which that starting point and the end point belong. In the below example, since the path time constraint is set using the .next therefore after o0 the number of cycles till o1 is output after rotation of 1 loop is 3 cycles.

Example:

```
in ter(0:8) i0, i1;
out ter(0:8) o0, o1;
// Cyber path_interval_id = ID1->ID2.next:3T
process main() {
    while(1) {
        // Cyber id=ID1
        o0 = i0;
        // Cyber id=ID2
        o1 = i1;
    }
}
```

The path time constraint is a function that can control the timing between 2 elements. However, in case of the below mentioned there is a limitation that controlling cannot be done, and the specified timing is not reflected.

- When stall occurs due to signal specified with stall_control
- When stall occurs due to universal memory interface function
- When stall for avoiding hazard occurs with loop folding
- When transition changed due to goto statement in allstates
- Watch statement is inside the path between starting point and end point and loop is rotated

21.7 Time constraint specification (TLMT) file

21.7.1 Related options

Option	Description
-It <i>filename</i> [.TLMT]	Specifies the time constraint file

21.7.2 Description

The time constraint of interval_id attribute and interval_array attribute can be specified with time constraint specification (TLMT) file also without using the attributes. Syntax of time constraint specification (TLMT) file is as per below.

Time constraint specification	Description
/* comment */	From "/*" to "*/" is ignored as comment. Nesting of comment is not allowed
// comment	From "://" till the end of row is ignored as comment
interval_id = <i>interval</i>	Specifies the time constraint between identifiers
<i>name.interval_array</i> = <i>interval</i>	Specifies the reference of array name, time constraint related to assignment

```

interval_id =
// comment
ABC->DEF:4T &
/* comment*/
ABC/DEF->XYZ:3T;
data.interval_array = // Specify the signal name
R->W:4T &
R->R:4T;
```

21.8 Time constraint violation

21.8.1 Description

It is not always true that if various types of time constraints have been set then the execution timing in the synthesized circuit will always fulfill the time constraints.

Since the time constraint violation message is output in case time constraint is not fulfilled so the resource constraint and description needs to be adjusted/prepared in such a way that the time constraints are fulfilled.

21.8.2 Message output at the time of time constraint violation

Time constraint scheduling algorithm implemented currently has many limitations such as the specified time constraint does not fulfill the result of scheduling etc. Error is generated in case the result of scheduling and execution timing does not fulfill the time constraint completely. Warning is displayed in case of others. For example F_BT6847 is output in case time constraint is not fulfilled in scheduling block.

F_BT6847: Things between \$ inside the scheduling_block could not be kept in 1 state
[Countermeasure] Confirm the constraint, number of cycles etc.

[Source row]

9(test.bdl): $x = a + b + c;$

Case where time constraint specified in latency_loop attribute and interval_id attribute is not fulfilled is also same and Error is output in case the result of scheduling and execution timing does not fulfill the time constraint completely.

Further, there will be a message in case of time constraint violation when the designer used time constraints set implicitly rather than the time constraints specified explicitly. W_BT6854 warning or F_BT6854 error will be output in case time constraints used for avoiding hazards is not fulfilled in loop folding.

Performance will go down due to occurrence to stall at the time of pipeline execution in case the

W_BT6854: Time constraints for avoiding hazard in pipelining of loop was not fulfilled. Following interval should be less than 0T however it is 2T in the synthesized circuit.

above mentioned message is generated in loop folding. The time constraint in loop folding has been mentioned in **Section 17.2.7** in detail.

21.8.3 Support at the time of time constraint violation

In case time constraint is violated it can be fulfilled by adjusting the resource constraint. In the example below when only one adder is there then the time constraint of scheduling block cannot

be fulfilled but if there are 2 adders then there is a possibility that the time constraint can be fulfilled.

```
in ter char a, b, c;
out ter char x;
process entry(void) {
    /* cyber scheduling_block */
    $ 
    x = a + b + c;
    $
}
}
```

In case it is obvious that scheduling will fail irrespective of how the resource constraint has been specified then F_BT6361 error or W_BT6361 warning will be displayed.

F_BT6361: The following time constraint cannot be fulfilled due to long delay.
[Countermeasure] Lengthen the block cycle, or ease up/mitigate the time constraint

For instance in the example of description using the scheduling block, since reference is there after writing to register therefore it cannot be kept in 1 cycle irrespective of the kind of resource constraint.

```
in ter char a, b, c;
out ter char x;
process entry(void) {
    /* cyber scheduling_block */
    $
    x = a + b + c;
    $
}
}
```

In case F_BT6361 error or W_BT6361 warning is output, since time constraint cannot be fulfilled even after adjusting the resource constraint so there is a need to change the description. In the example mentioned above it is required to insert \$ between write and read of register.

However, there is a limitation that F_BT3631 or W_BT6361 message are not generated even if it is obvious that scheduling cannot be done. Therefore it is always not true that time constraint can be fulfilled only by adjusting the resource constraint even if these messages do not appear.

22 Automatic Pipeline Scheduling

22.1 Overview

This section covers the following:

- Pipelined and non-pipelined circuit synthesis.
- Restrictions in the automatic pipelined synthesis.
- Forwarding register in pipelined synthesis.
- Methods of suppressing the forwarding register.
- Structural hazards of functional units.
- Data hazards of memory ports.
- "Pipeline stall" functionality.

Automatic pipeline scheduling refers to the automatic generation of pipelined circuits from behavioral description. This section explains the functionality of automatic pipelining.

Circuit operations are same as the operations applicable in describing automatic scheduling mode (**section 17.2**) of loop folding in **chapter 17**, however there are following differences:

- This is only applicable when all stages work in synchronization. There is an error when there is a structural hazard between the stages and there is a possibility of stalling when loop is included in the description.

However, even if a data hazard which is different from loop folding is detected for array, a warning is generated and the stall circuit is not generated. Due to this there is a possibility that there is a circuit which has different operations from the input operation description when warning is generated. (An option that gives an error when data hazard of array is detected is prepared).

- All stages start working after Reset, and PROLOG operation is not conducted.

Therefore, when compared to loop folding the applicable scope is limited but it becomes useful as generated circuit.

22.1.1 Related Options

Option	Description
-ZZpipeline [=#]	Performs Automatic synthesis of pipelined circuits with DII # of cycles. In case not specified, a value of 1 is assumed for DII.
-Zarray_data_hazard=AVOID -Zarray_data_hazard=CHECK -Zarray_data_hazard=NO	Error for data hazard for arrays Warning for data hazard for arrays (default) Does not perform data hazard check for arrays

22.1.2 Related Attributes

Attributes	Description	Settings Target
/* Cyber pipeline_forwarding=NO */	Does not perform synthesis of the signal as forwarding register	Signal
/* Cyber group_interval_array = val */	Sets time constraints specified in "val" to array	Array
/* Cyber Array_data_hazard=AVOID */	Data hazard for array is taken as error	Array
/* Cyber Array_data_hazard= CHECK */	Warns for data hazard for array	Array
/* Cyber Array_data_hazard= NO */	Does not check data hazard for array	Array
/* Cyber pipeline_interval_check = NO */ *	Does not perform structural hazard check	Signal

* Attribute pipeline_stall_control is integrated with attribute stall_control and becomes obsolete. (Refer to **section 23**).

22.2 Pipeline circuit

In non-pipelined synthesis, circuits are synthesized by executing the process function again from the beginning after execution of the process function ends.

For example (refer to **Figure**), in a case where the execution of the process function requires four clocks, then state ST1_01 to state ST1_04 are implemented in such a way that they get executed in order. Once the execution of the first invocation (indicated as 'A' in the **Figure**) is completed then the next process function (indicated as 'B' in the **Figure**) is executed.

However, in the case of a pipelined synthesis, circuits are synthesized based on the assumption that the successive process function can be executed without waiting for the completion of prior invocation of process function. For example (refer to **Figure**), in the execution of above mentioned process function, if circuit requiring four clock cycles for execution is synthesized with automatic pipeline scheduling with DII as 1, then a circuit is generated in which these four pipeline stages, 1st to 4th stage have already been executed. Even if the first process function invocation (indicated as 'A' in the **Figure**) has not been completed, the next process function invocation (indicated as 'B' in the **Figure**) is executed.

Hereafter, the execution of the part which can be repeated once is called as "iteration".

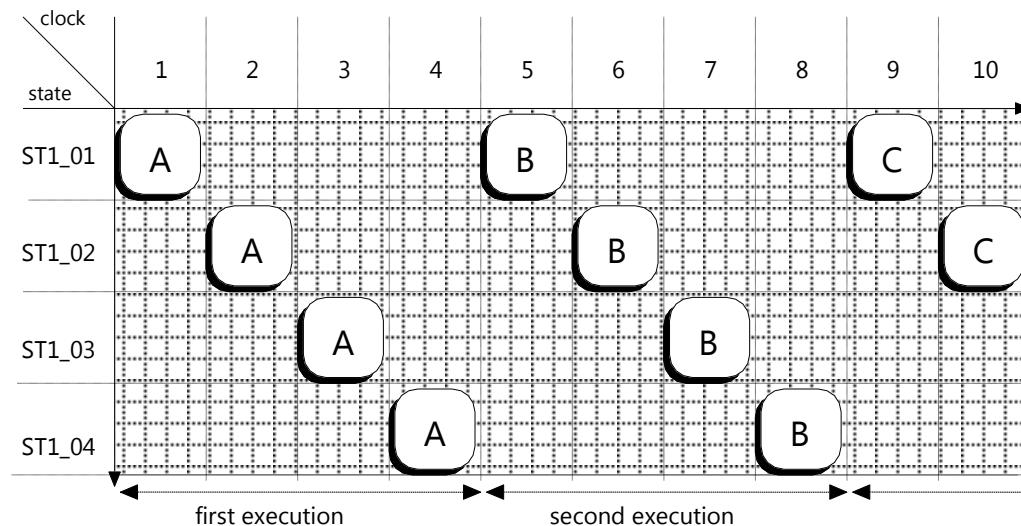


Figure 85: Operation of non-pipelined circuit

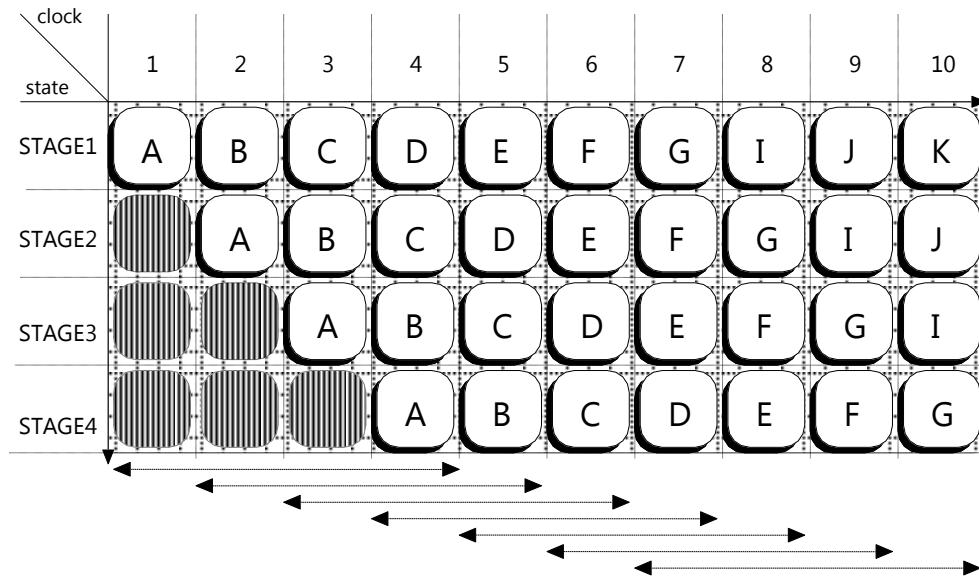


Figure 86: Operation of pipelined circuit

22.3 Restrictions of Automatic pipeline scheduling

There are certain restrictions regarding automatic pipelined synthesis.

22.3.1 Restrictions Regarding Control Structure

- Automatic pipelined synthesis is not possible for "while", "do- while" and "for ()" statements that don't allow loop unrolling. Only "for" statements that allow loop unrolling can be described. Infinite loop enclosing the entire process function can be described as well.
- Automatic pipelined synthesis is not possible when infinite loop enclosing the entire process is described with "break" and "continue" used within the loop.
- Automatic pipelined synthesis is not possible if "goto" statement is present in the loop.
- "wait" statement is permitted only when its argument is constant other than 0 and it is not in conditional branch. Synthesis cannot be done in auto-pipelined scheduling, if argument of wait statement remains as variable, or if argument of wait statement is constant '0', or if wait statement is present inside the conditional branch.

In case above mentioned restriction is violated, the following error messages will be generated.

F_BT4648: There is another loop inside loop for folding
[Action] Rewrite foldable loop

F_BT4647: wait() is not in the top level inside loop
[Action] Move wait() to top level inside loop

F_BT4621: Loop folding cannot be done for specified loop statement
[Action] Confirm the description and check whether loop folding is possible in the loop statement

F_BT1441: wait/watch statement is not constant in auto pipeline process.
[Action] Set the wait/watch statement to 1.

In case wait() statement is used in conditional branch, following error message will be generated.

F_BT7987: The data of multiple loops/blocks by the automatic pipeline conversion is not supported.
[Action] Review loop or GOTO statement of description.

22.3.2 Restrictions Regarding Pipeline Structure Hazards

- Input and output cannot have more than DII number of ports assigned to each signal since an error will be generated if specific port is accessed in multiple stages where the condition of exclusions is taken into consideration to avoid such errors.

For details, refer to **section 22.5.2**.

- Number of memory accesses for each array shall be restricted to DII times the number of ports for that array. However, the condition of exclusions is taken into consideration.

For example, when synthesis is done by specifying 1 in DII access to a single port memory shall be restricted to one for every array.

For details, refer to **section 22.5.5**.

- Number of functional units shall be restricted to the DII times the number of the functional unit specified in the constraint file. However the condition of exclusions can be specified to avoid such errors.

For details refer to **section 22.5.1**.

22.3.3 Synthesis Specific Restrictions of Automatic Pipelined circuits

- Synthesis of a function as inline expansion or functional unit is supported. Synthesis of function as "goto" conversion is not supported.

If the above mentioned restriction is violated, then the following error message gets generated:

```
F_BT7987: The data of multiple loops/blocks by the automatic
pipeline conversion is not supported.
[Action] Review loop or GOTO statement of description.
```

- Multi-cycle functional unit that requires number of cycles greater than specified DII value cannot be used. Even if that multi-cycle function is specified in the functional unit constraint, Cyber does the synthesis without using those multi-cycle functional units.

```
W_BT7981: It does not support in case cycle count (=3) of
multicycle functional unit add08_08u is bigger than DII (=1) in
auto pipeline.
[Action] adjust the delay value or clock cycle of functional
unit constraint file.
```

Even when warning message(s) are generated, synthesis will continue without using this multi-cycle functional unit. Therefore, following error will occur in case there is an operation in which there is no functional unit which can be used other than the relevant functional unit.

```
F_BT6745: Operator (+) [Bit width 8,8]
Functional unit corresponding to ((unsigned ter(0:8) + vb(0:8)) is not
in the functional unit constraint file.
[Action] Describe the corresponding functional unit in the functional
unit constraint file
```

- Stage numbers can be suffixed to the name of the register regardless of the specification of share_name attribute.

In the example given below, when variable "v" has been specified to be assigned to register name "REG", the variable will actually be allocated to multiple registers with names REG_## (## signifies the stage number).

```
var (0:8) v /* Cyber share_name = REG */;
```

Specific messages will not be displayed in this case.

22.4 Forwarding Register

As previously mentioned in **section 22.2**, in a pipelined synthesis, the succeeding process invocation is initiated prior to the completion of current process invocation.

However, in case there is a transfer of values between iterations, if the assignation of previous iteration is not completed then the value cannot be referred in the succeeding iterations. This

circuit is called a "forwarding" circuit and registers which are used for transfer of values to successive invocations are called as "forwarding registers".

Auto-pipelined synthesis of the above description is shown in **Figure**.

At the first stage, value is read from "v0" (inside dotted lines on the left side of the **Figure**). The value contained in "v0" was actually read from "in2" during previous iteration. Concurrently, for the next iteration, value from "in2" is being stored in "v0" (dotted line on the right side of **Figure**). In this scenario, register "v0" is serving as a forwarding register. Operation of forwarding is shown in **Figure**.

```

in ter(0:8) in0, in1, in2 ;
out ter(0:8) out0 ;
process foo(){
    var(0:8) t0,t1,i1,v0 = 0 ;

    while(1){
        t0 = v0; /* Refer value of previous iteration*/
        t1 = t0 + in0;
        i1 = in1;
        out0 = t1 + i1;
        v0 = in2 ; /* Assign value for succeeding iteration */
    }
}

```

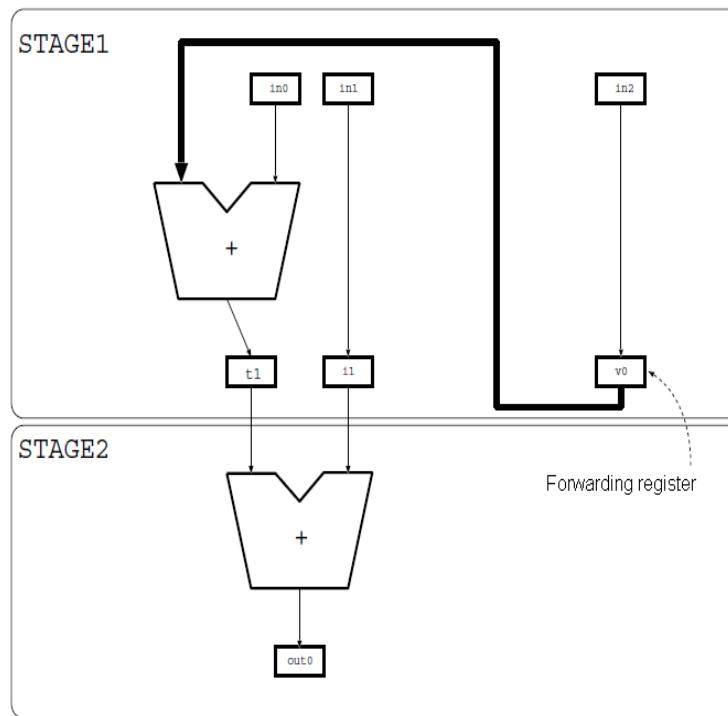


Figure 87: Forwarding circuit

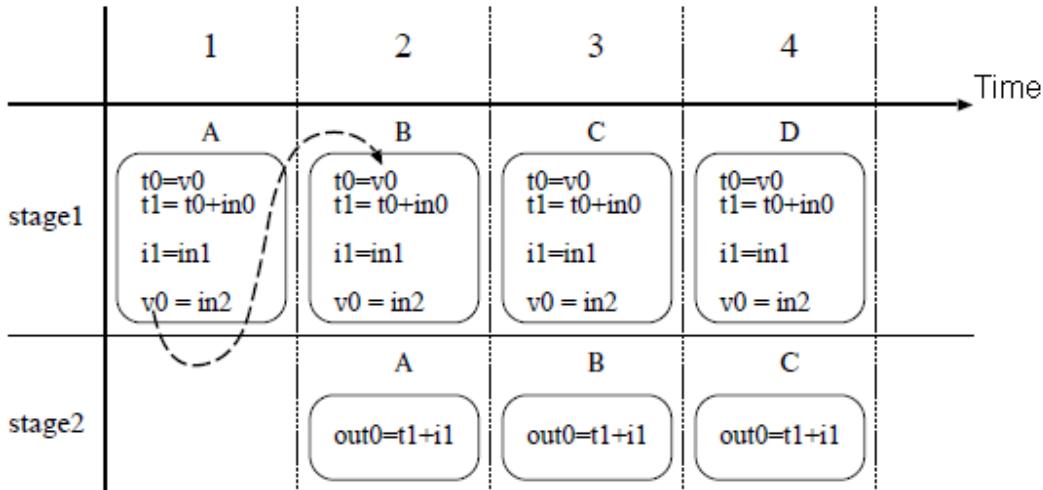


Figure 88: Forwarding operation

The values contained in the forwarding register should be set prior to referencing. In a pipelined circuit as each iteration works in succession, the following conditions should be fulfilled for forwarding register.

- Last assignment should be done from the state of initial referencing of forwarding register till the state which is before the state after DII.

Unless the above specified conditions are satisfied, the automatic pipelining function displays the following error message:

```
F_BT5651: Since there is data dependency between the iterations
of variable (RG_v0_03) so the processing of folded loop is
stalled. State1 of stage 1 of succeeding iteration cannot be
started till state 1 of stage 3 is completed.
[Action] change the description so that the related processes
are executed at the same stage (or within DII).
```

```
[Source line]
17(foo.c) :           v0 = x ;
[Source line]
15(foo.c) :           x = ((v0 + in0) + in1) + in3;
```

The above mentioned error indicates that the state of reference of the forwarding register in line 17 of foo.c and the state of the forwarding register in line 15 do not fulfill the above mentioned conditions. The error message has same expression as for the one in the folding loop of **section 17.2.4**.

If synthesis is stopped due to the occurrence of these errors specified above, the description and the synthesis constraints need to be reviewed. This is done for finding the possibilities to comply with the above specified conditions. Following are the cases where the above specified conditions cannot be fulfilled:

- Delay caused by the following operations are too significant to comply with requirements:
 - Computation of stored value of forwarding register
 - Operations using value stored in forwarding register

For example, in the description specified below when synthesis is done with DII=1, if the overall delay in the two additions that carry out the update of the "static" variable "s0" is greater than one clock cycle, the above specified conditions cannot be fulfilled and there is an error.

```
in ter(0:8) in1,in2;
out ter(0:8) out0;
var(0:8) s0 = 0;
process foo() {
    s0 = (s0 + in1) + in2 ;
    out0 = s0 ;
}
```

- For detailed overview on structural hazard which can exist in relation to inputs, outputs, and memory ports, see **Section 22.5.2, 22.5.5** below.
- In case "Cyber" makes an incorrect decision regarding the use of global variables and local variables of processes as forwarding registers, then these global and process local variables need not be synthesized as forwarding registers, refer to **section 22.4.1** below.

22.4.1 Suppression of the Forwarding Registers

Depending upon the description contents, there may be cases where Cyber synthesizes a register as forwarding registers but designer does not want that register to be forwarding register.

These cases occur when Cyber is incapable of correctly analyzing the requirement of forwarding registers during the analysis of data flow. The existence of unintended forwarding registers may have adverse influence on the circuit area or may result in synthesis errors.

In such cases, the synthesis of forwarding registers may be suppressed by doing one of the following:

- Do not refer the value clearly assigned by the previous iteration and assign constants to variables used for first iteration.
In case the infinite loop which encloses entire process functions is not described then it implies that the value of previous iteration is not referred by changing the variable to non static local variables of functions.
- Specify the "pipeline_forwarding = NO" attribute in the declaration statement of variable.

This attribute is used to forcibly stop the analysis for synthesizing specified variable as forwarding register. One needs to be careful after the specification of this attribute as the circuit generated will not have an operation as per BDL input.

```

in ter(0:8) in1,in2;
out ter(0:8) out0;
var(0:8) s0 /* Cyber pipeline_forwarding = NO */ = 0;
process foo(){
    s0 = s0 + in1 + in2 ;
    out0 = s0 ;
}

```

22.5 Structural Hazards

22.5.1 Structural Hazards of Functional units

In automatic pipeline scheduling and automatic loop folding function, there can be cases where sufficient functional unit counts are not specified in functional unit constraint file for pipelining. Such cases result in following error in order to prevent structural hazards occurrence.

```

F_BT6311: functional units supporting operation (+) [bit width 8,8] are
insufficient, hence pipelining by DII (1) can not be done.
[Action] Increase DII value or supporting functional unit count.

```

In this case, synthesis can be done for pipelined circuit by specifying sufficient functional unit constraint counts. If it is desired not to change constraint count as circuit area may have more priority, DII can be set to a greater value, as synthesis can be achieved by decreasing the processing speed.

If the operation is used multiple times under usage of exclusive conditions, functional unit is shared depending upon the area of functional unit. If the area of the functional unit is not set in functional unit library file or a smaller value is set then following error may be generated.

```

F_BT7457: Operation + in ST1_02 was not allocated in functional unit
add08_08u(1)
[Action] Increase functional units.

```

In this case, although it is indicated that it is better to increase the functional unit constraint count but when the value for area is not set, synthesis can be carried out by increasing the functional unit constraint count to appropriate value.

22.5.2 Structural Hazards of Input and Output Ports

Automatic pipelined scheduling and loop folding modes have capabilities to identify structural hazards regarding input and output ports. In case of structural hazard of input and output ports, a specific port is accessible to multiple stages simultaneously. This can lead to failure of accessing of intended data or failure of multiple assignments taking place. In the automatic pipeline scheduling of "Cyber", if a structural hazard is detected regarding Input and Output ports, the following error message gets generated:

```
F_BT6314: Access count of I/O port (in0) is high, pipelining can not
be done by specified DII(1)
[Action] Increase DII value or decrease number of accesses of Input
and output. Do not share ports.
[Source line]
7(test.c):    out1 = in0 ;
```

In such cases, the error file (.err) will contain following kind of warning:

```
W_BT4906: In folding loop, the number (2) of input / output
terminals for in0 exceeds DII(1).
[Action] Increase DII count or reduce descriptions of input/output
terminals.
```

For example, in the description specified below, warning is displayed due to the reason that multiple accesses for input signal "in0" are there.

```
In ter(0:8) in0 ;
out ter(0:8) out0 ;
out ter(0:8) out1 ;
process foo ( )
{
    out0 = in0 ;
    out1 = in0 ;
}
```

22.5.3 Avoiding Structural Hazards for Input and Output Ports

In case the structural hazards are detected for input and output ports, the following modifications in description are necessary:

- **If the same data content within an input stream is being used,** access to input port can be consolidated by routing the accessing through an intermediate terminal.

```
in ter(0:8) in0 ;
out ter(0:8) out0 ;
out ter(0:8) out1 ;
process foo( )
{
    var (0:8) in0_t /* intermediate variable */
    in0_t = in0 ;
    out0 = in0_t ;
    out1 = in0_t ;
}
```

- **If different data from an input stream is being used,** description should be modified to provide multiple input terminals for accessing different data streams contained within the input streams.

```

in ter(0:8) in0_0 ;
in ter(0:8) in0_1 ;
out ter(0:8) out0 ;
out ter(0:8) out1 ;
process foo()
{
    out0 = in0_0 ;
    out1 = in0_1 ;
}

```

Structural hazards will not occur when conditional exclusions are used as shown in description below even though multiple accesses of input / output signal are specified. Moreover, warning W_BT7971 also takes conditional exclusion into consideration in the calculation of the number of occurrence.

```

in ter(0:8) in0 ;
in ter c ;
out ter(0:8) out0 ;
out ter(0:8) out1 ;
process foo()
{
    if ( c ){
        out0 = in0 ;
    }else {
        out1 = in0 ;
    }
}

```

Even if the conditional exclusion is specified for accesses in input BDL, the error F_BT6314 may get generated. This condition occurs when Cyber fails to assess the conditional exclusions. In the example description specified below, although the executable conditions of the two "if" statements are mutually exclusive, Cyber cannot discriminate the exclusiveness.

```

in ter(0:8) in0 ;
in ter c;
out ter(0:8) out0,out1 ;
process foo(){
    var(0:1) c_t;
    c_t = c;
    if ( c_t ){
        out0 = in0 ;
    }
    if ( !c_t ) {
        out1 = in0 ;
    }
}

```

In such situation, synthesis can be made possible by redefining the statement using "if-else" statement as specified below.

```

in ter(0:8) in0 ;
in ter c ;
out ter(0:8) out0 ;
out ter(0:8) out1 ;
process foo()
{
    if ( c ){
        out0 = in0 ;
    } else {
        out1 = in0 ;
    }
}

```

22.5.4 Suppressing Detection of Structural Hazards of Input/Output Port

The detection of structural hazards of input/output ports can be suppressed by specifying the "pipeline_interval_check=NO" attribute. In case of following description, if a value which is being referred from the input port 'param' doesn't change during pipelining, we can use common input port at all stages. In this case, by specifying the "pipeline_interval_check=NO" attribute in declaration part, structural hazard detection of input/output ports can be suppressed and synthesis can be done.

```

in ter(0:8) in0;
in ter(0:8) param/* Cyber pipeline_interval_check=NO
*/;
out ter(0:8) out0 ;

process main()
{
    var(0:8) tmp0, tmp1 ;

    tmp0 = in0 * param;

    tmp1 = tmp0 * param;

    out0 = tmp1 * param;
}

```

22.5.5 Structural Hazards of Memory Port

Automatic pipelined scheduling and loop folding have capabilities to detect structural hazards of memory ports. Structural hazards in these ports signify the simultaneous access of a single memory port by multiple stages and in this case multiple assignments may occur.

In Automatic pipeline scheduling, synthesis is done in a way so that structural hazards don't occur. However, if the description is such that Cyber is unable to avoid the hazards with the specified port count, then the following error message is displayed.

F_BT6312: Memory's mem08A (ary) ports are not sufficient and pipelining is not possible with DII(1).
[Action] increase DII value or, increase port number of memory.

The message signifies that port 1 of memory "mem08A" allocated to array "arr" is being accessed by multiple stages. This results in structural hazard.

22.5.6 Avoiding Structural Hazards of Memory Ports

Similar to inputs / outputs structural hazards, array access will also contain the following message in the error file (.err) in case of structural hazards:

I_BT7972: The access number of array/memory signal(ary) is 2(read)
and 0(write)

The number of accesses indicated in the message takes conditional exclusion into consideration. Structural hazard of memory ports happens when the number of memory ports provided is less than the memory access count. In this situation, synthesis can be done by using multi-port memories or by reducing memory access count through algorithm review.

Additionally, false error arises due to misinterpretation of Cyber on conditional exclusions and can be resolved in a similar way as it was resolved in case of structural hazards of I/O ports.

22.6 Data Hazards of Memory Port

22.6.1 Overview

If the reading and writing of an array is not at the same stage in Automatic pipeline scheduling, there is a possibility of occurrence of 3 data hazards i.e., RAW (read after write), WAR (write after read) and WAW (write after write).

Unlike loop folding in automatic pipeline scheduling, a circuit in which data hazard may occur is generated by giving the following warning by default. That is why, when the following warning is given, there is a possibility that a circuit will be generated which will have operations that differ from the described input operations.

W_BT5659: Since there is data dependency between iterations of array, process of folded loop is stalled. State 1 of stage 2 of succeeding iteration cannot be started till state 1 of stage 5 is completed.

[Action] Modify so that related processes are executed at the same stage (or within DII).

RAW hazard is a problem that occurs when the value written in memory in current iteration in description is read from memory in the successive iteration. The value intended for read operation cannot be read as the write operation to memory is not completed.

WAR hazard is a problem that occurs when the value from memory is read in current iteration in description and value in memory is written by next iteration. The value intended for read operation cannot be read as the write operation to memory takes place prior to completion of read operation.

WAW hazard is a problem that occurs when value written in memory in current iteration in description is further written in memory by the next iteration. The next write operation is conducted even if read operation for first memory is not completed and if there was same address the intended value cannot be written.

22.6.2 Options and Attributes to counter data hazards

In automatic pipeline scheduling, synthesis is done such that data hazard is prevented as much as possible, and there is a functionality that detects the possibility of data hazard occurrence.

Option	Description
-Zarray_data_hazard=AVOID	Error for data hazard of array
-Zarray_data_hazard=CHECK	Check the data hazard of array and generate the error (default)
-Zarray_data_hazard=NO	Does not check data hazard of array

By specifying “-Zarray_data_hazard=AVOID” option, detection and avoidance of data hazards in all memories are performed. There is a possibility that Cyber may not be able to avoid the data hazard, then an error is generated.

Attribute	Description	Settings Target
/* Cyber array_data_hazard= AVOID */	Error for data hazard of array	Array
/* Cyber Array_data_hazard= CHECK*/	Check the data hazard of array and generate the error	Array
/* Cyber Array_data_hazard=NO*/	Does not check data hazard of array	Array

Further, data hazard of arrays synthesized as memory is detected and avoided by specifying the "array_data_hazard=AVOID" attribute. There is a possibility that Cyber may not be able to avoid the data hazard and in that case an error is generated.

22.7 Summary

This section covered the following:

- In non-pipelined synthesis, circuit synthesis is done based on the assumption that the next invocation of a process function is initiated after the execution of the current invocation has been completed.
- In the case of a pipelined synthesis, circuits are synthesized based on the assumption that the next invocations of the process function can be done in respective clock cycles without waiting for the completion of prior invocation of process function.
- Automatic pipelined synthesis is not possible for "while ()", "do- while ()" and "for ()" statements that don't allow loop unrolling.
- In automatic pipeline synthesis, multi-cycle functional unit cannot be used.
- The values contained in the forwarding register should be set prior to referencing. The pipelined circuit is designed to invoke process function in every clock cycle without waiting for completion of the current process function invocation.
- The synthesis of forwarding registers needs to be suppressed where the existence of unintended forwarding registers may have adverse influence on the circuit area.
- In automatic pipeline synthesis and automatic loop folding function, error is displayed to prevent the triggering of the structural hazard. However, synthesis can be done for pipelining by specifying sufficient functional unit constraint count.
- Automatic pipelined synthesis mode has capabilities to identify structural hazards regarding input and output ports.
- **WAR** hazard refers to the problem when a memory Write operation by next function invocation (scheduled) After Read operation by current invocation actually happens prior to this Read operation, which results in a case where intended data can't be read as it has been destroyed.
- **RAW** hazard is a problem that happens when a memory Read attempt After memory Write operation (that occurred in earlier process function invocation) leads to incorrect data being read due to the reason that write operation has not been completed.

23 Stall Functionality

23.1 Overview

This section describes the behavior of stall functionality. The stall functionality stalls the operation temporarily whenever external stall input is enabled. This functionality is also called as hold functionality.

23.1.1 Related option

Option	Description
-Zope_stall=create	Input the stall port in pipelined built in functional unit
-Zope_stall=ignore	Generate compensating circuit in pipelined built in functional unit

23.1.2 Related attributes

Attribute	Description	Settings Target
/* Cyber stall_control = high */	Use input signal as control signal of stall functionality (Active High)	Input signal
/* Cyber stall_control = low */	Use input signal as control signal of stall functionality (Active Low)	Input Signal

23.2 BDL Description for Stall Functionality

In order to use stall functionality, the following specified description can be used.

Attribute stall_control is added to 1 bit input signal that forms stall input signal. Polarity is specified in attribute value. If value of the attribute is high, circuit will stall when stall input signal is 1. If value of the attribute is low, circuit will stall when input signal is 0.

```

in ter          STALL /* Cyber stall_control = high *;
in ter (0:8) in0 ;
...
process main () {
    ...
    (process)
    ...
}

```

23.3 Stall - Overview of the Operation

When stall functionality is used, circuit is synthesized so as to carry out the following operation at the time of stall.

- During the stall operation, the present value of internal registers is retained and is not updated.⁵
- At the time of stall operation, the chip select (cs) signal is not asserted in order to disable the memory operation. Details are specified later.
- The input/output operations are not valid during stall operation. Therefore, validating signals for input/ output operations are not asserted during stall operations. Details to be specified later.
- Stall signal is applied to pipelined functional units of built-in operations (operations other than functions which is converted into functional unit) specified in functional unit constraint file (LMT or FLIB/FCNT). Details to be specified later.
- A stall signal is applied to function which is converted into functional unit. Details to be specified later.
- Write Enable (we) signal is not asserted for shared registers (register declared as 'shared reg') to halt the updating of values during the stall operation.
- Write Enable (we) signal is not asserted for shared register arrays (arrays declared as 'shared reg') to halt the updating of values during the stall operation.

23.4 Memory Operations during Stall - Synchronous Pipeline Memory

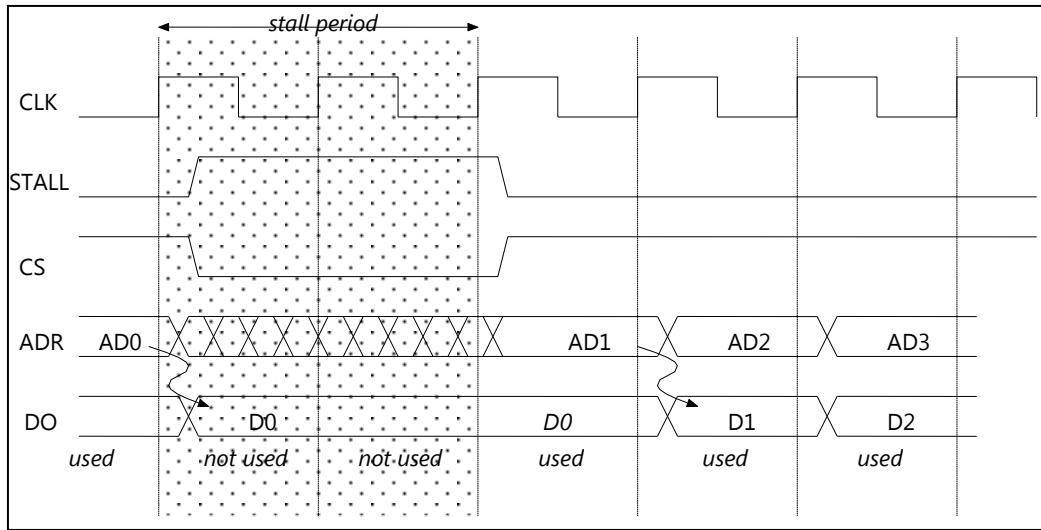
In case of memory operations during stall for synchronous pipeline memory, the memory which has chip select port can also be stalled. This type of port is known as CS port.⁶

Memory Library file (MLIB/MCNT) file will contain CS port specification, in case CS ports exists for allocated memory (specified with keyword "cs", "rs", "ws"). Also memory constraint file (MLMT) will contain CS port if option –Zmem_cs is specified or if attribute mem_cs is specified for an array.

If CS port exists in a synchronous pipelined memory, (memory which contains pipeline specification like x2, x3 etc in MLIB, MCNT or delay specification (DELAY) in MLMT) the circuit is configured such that during the stall operation, the CS port is deasserted. Additionally, during the stall operation, read enable (re) and write enable (we) ports are deasserted as well.

⁵ Excluding few registers. For example, the register of the stall compensated circuit (described later) are updated even during stall operations.

⁶ It is supposed that deasserting the CS port of memory provided with CS port will completely halt the operation in Cyber.



If the CS port is not present in a synchronous pipelined memory, unless specified otherwise, a pipelined compensated circuit is generated (refer to **section 23.9** for details). Further, during stall operation read enable (re) port, write enable (we) ports are not asserted.

23.5 Memory Operations during stall - Synchronized Non-Pipelined Memory

If the CS port is present in a non-pipelined memory (memory for which absolute delay specification or cycle specification (1T, 2T etc) is specified in MLIB/MCNT file or delay specification (DELAY) of MLMT file), circuit is configured to deassert the CS port during stall operation. Also, read enable port (re) and write enable port (we) are also deasserted. If the CS port is not present in this kind of memory, read enable (re) port and write enable (we) port are deasserted during the stall operation. It is different from synchronized pipeline memory and compensated pipelined circuit is not generated in this case.

23.6 Input Output Operations during stall

As I/O operations are not valid during the stall operation, valid signals (refer to **Section 7.10**) that generates I/O timings are deasserted. Behavior of valid signal during pipeline stall has been indicated in each of the following cases:

- "in ter" (**Figure 89: Behavior of the valid signal for "in ter" during the stall operation**)
- "in reg" (**Figure 90: Behavior of valid signal for "in reg" during the stall operation**)
- "out ter" (**Figure 91**)
- "out reg" (**Figure 92**)

In case of "in reg", the input specified prior to stall operation is valid after the stall operation, and it needs to be carefully provided as input to the circuit. In order to prevent complexities, it is advised to not to change the value of external input port during the stall operation.

23.7 Operation of Built-in Operator during Stall

A stall signal is applied to pipelined functional units for built-in operators. Circuit synthesized will differ based upon existence or nonexistence of stall input to concerned functional unit.

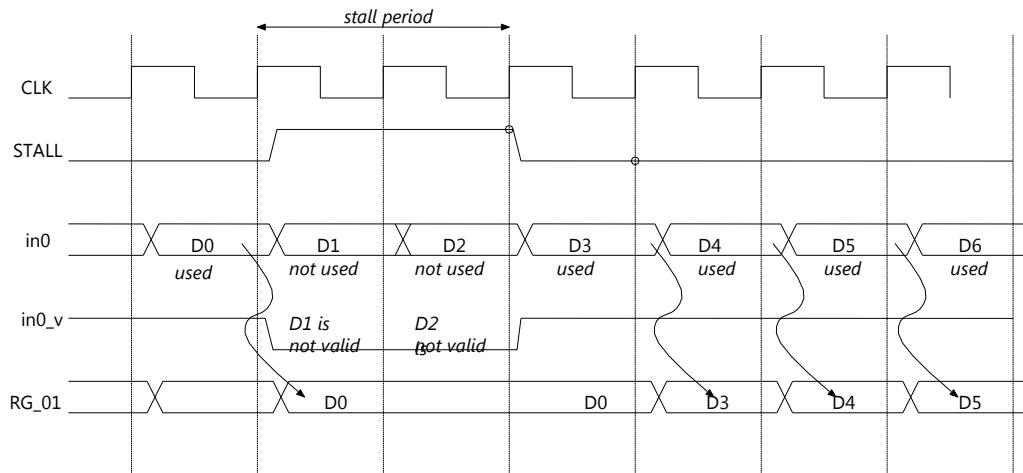


Figure 89: Behavior of the valid signal for "in ter" during the stall operation

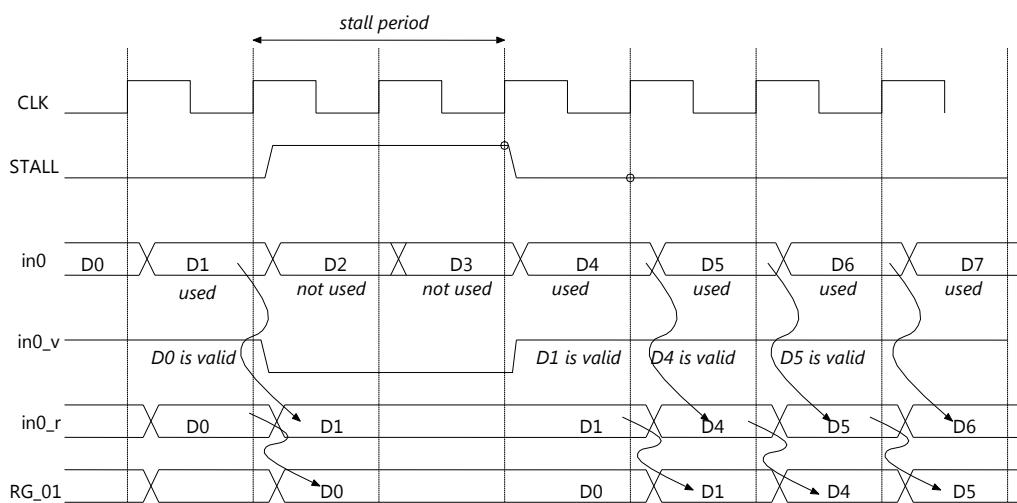


Figure 90: Behavior of valid signal for "in reg" during the stall operation

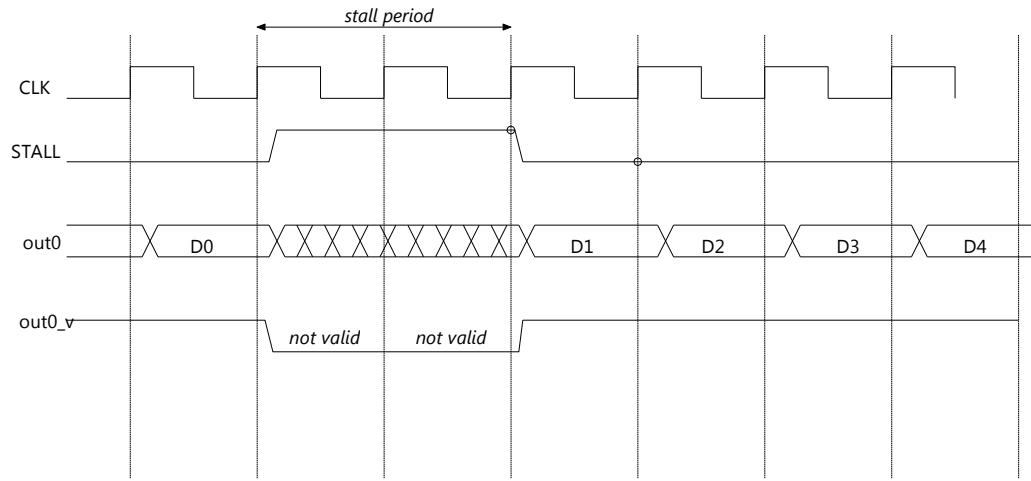


Figure 91: Behavior of valid signal for "out ter" during the stall operation

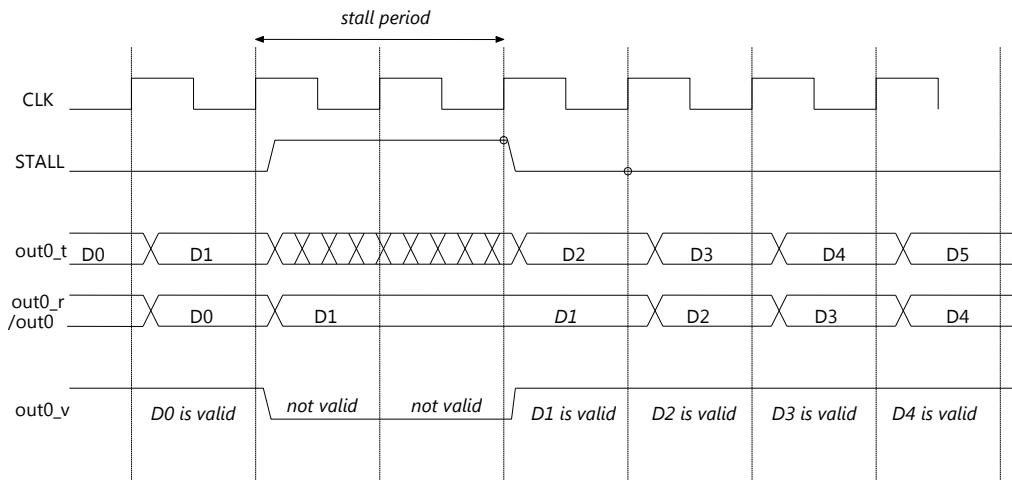


Figure 92: Behavior of valid signal for "out reg" during the pipeline stall operation

- **Pipeline functional unit with a stall input.**

In this case, stall port of circuit is connected to stall input port of pipelined functional unit. Stall input port of parent process (synthesis target process) is directly connected as it is, to pipelined functional unit.

- **Pipeline functional unit, without stall input.**

In this case, a pipeline stall compensated circuit is generated. (Described later in **section 23.9**)

Presence or absence of the stall input to pipeline stall input is specified as follows:

- In the case of the functional unit constraint file (LMT) specification, , it is supposed that stall input port is present in all the pipelined functional units by default. Existence of stall input port can be specified using “-Zope_stall” option.
- In the case of the functional unit library file (FLIB) specification, existence of stall input port can be specified by PIPELINE_STALL or STALL_COMPENSATION field of each functional unit. In case value of field STALL_COMPENSATION is CREATE, pipelined stall compensated circuit is generated without using stall input port. In case value of field STALL _COMPENSATION is IGNORE, stall input port is used without generating pipelined stall compensated circuit. In case value of field PIPELINE_STALL is YES, stall input port is generated and used, and in case value is NO, stall input port is not generated. Further, generation of stall input port can be specified using “-Zope_stall” option. And in case both are specified, FLIB specification is given priority. Note that if NO is specified for PIPELINE_STALL, and IGNORE is specified for STALL_COMPENSATION, both compensated circuit and stall input port will not be generated, therefore, normal operation will not be carried out.

```
@FLIB{
    NAME          add08cmp
    KIND          +
    BITWIDTH     8,8
    DELAY         10x2
    SIGN          UNSIGNED
    PIPELINE_STALL NO
    STALL_COMPENSATION CREATE
}
```

23.8 Operation for Converting Functional Unit for User-Defined Functions during stall

Stall signal can be supplied to functional unit for user-defined functions. However, synthesized circuit depends on the existence of stall input of functional unit.

- **Functional unit with a stall input port.**

In this case, stall port of parent process (synthesis target process) is connected to stall input port of functional unit for user-defined functions. Stall input port of parent process (synthesis target process) is directly connected as it is, to stall input port of functional unit.

- **Functional unit without a stall input.**

In case functional unit for user-defined functions is pipelined functional unit; a pipeline stall compensated circuit is generated. (Described later in **section 23.9**)

In case functional unit for user defined functions is other than pipelined functional units, neither stall input signal nor is assurance circuit generation carried out. Therefore, note that in such cases normal operation will not be carried out except combinatorial circuit.

Whether to have pipelined stall input or not can be specified using keyword "h", which specifies the stall input requirement in the bit width (BITW) field of either functional unit library file (FLIB/FCNT) or functional unit constraint file (LMT).

```
#@CLK 1000
#@UNIT 1/100ns
#IDX NAME LIMIT KIND      BITW      DELAY AREA POWER
1     fnc_c  1    @fnc_c   (8,8,c,r),8  x4    #no stall input
2     fnc_n  1    @fnc_n   (8,8,c,r,h),8 x4    #stall input is there
```

In case stall port is specified for parent process, note that stall port will be generated in functional unit of user-defined functions generated by -Ztop and stall port is added to functional unit count file (FCNT), where information of module definition of function to be synthesized as functional unit is provided.

23.9 Pipeline Stall Compensated Circuit

In case a stall input port is not present in a synchronous pipelined memory or pipelined functional unit, a pipeline compensation circuit (refer to **Figure 93**) is generated because this circuit temporarily holds the memory or output value of the functional unit during the stall operation. (Refer to **section 23.4, 23.7 and 23.8** for the conditions to generate a pipeline stall compensation circuit.)

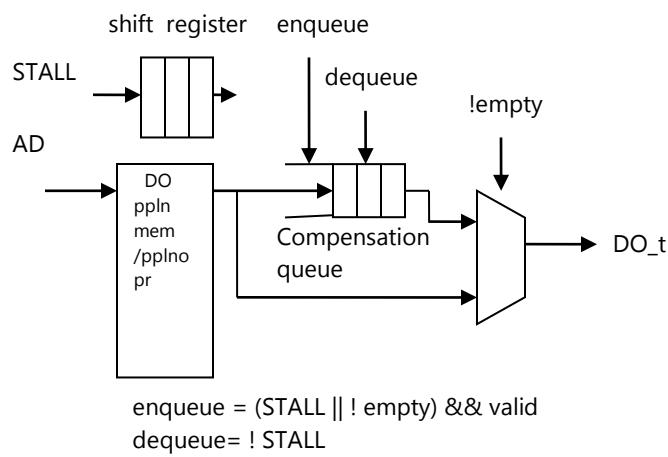


Figure 93: Pipeline stall compensated circuit

Pipeline compensation circuit operates in such a manner that it stores valid output values (output values for *data inputted during non-stall operation*) of memory / functional unit during stall

operation into a FIFO (queue) and when stall period is over, it queues these value out during appropriate timing (refer Figure 94)

There are restrictions in pipeline compensation circuit; (1) up to two stages in case of synchronous pipelined memory and (2) up to four stages in case of pipelined functional unit.

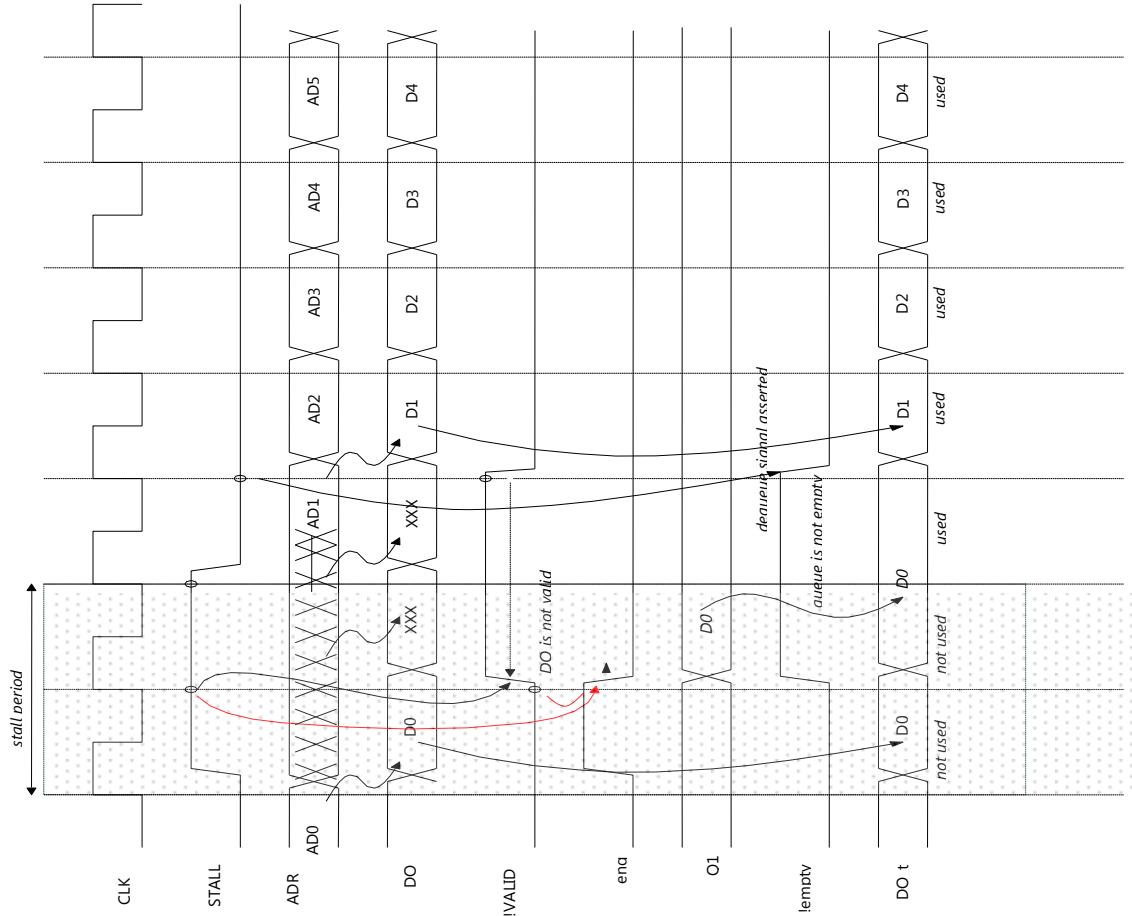


Figure 94: Waveform of pipeline stall compensated circuit

23.10 Summary

This section covered the following:

- In the case of memory operations during stall for synchronous pipeline memory, the chip select port that is present in memory macro operation is also stalled. This type of port is termed as chip select port or the CS port.
- Since I/O operations are not valid during the stall operation, valid signals that generate I/O timings are deasserted. Behavior of valid signal during pipeline stall depends whether it is reg type or ter type.
- In the case of the functional unit constraint file (LMT) specification, by default, it is supposed that stall input port is present in all the pipelined functional units. Existence of stall input port can be specified using “-Zope_stall” option.

- In the case of the functional unit library file (FLIB) specification, existence of stall input port can be specified by PIPELINE_STALL or STALL_COMPENSATION field of each functional unit.
- Stall signal can be supplied to functional unit for user-defined functions. Synthesized circuit depends on the existence of stall input of functional unit.
- If a stall input port doesn't exist in a synchronous pipelined memory or pipelined functional unit, a pipeline compensation circuit is generated to hold the output value of memory or the functional unit temporarily during the stall operation.

24 Hierarchical Setting

24.1 Overview

This section provides description for Cyber synthesis flow in hierarchical settings.

24.2 Related options

Option	Description
-Zlower_module-	Synthesize for lower module (default)
Ztop_module	Synthesize for top module

24.3 Recommended Synthesis Flow

While considering the Cyber synthesis flow in hierarchical setting, bottom up synthesis is recommended.

Functionality partition of setting target is performed and in case setting is done in cross multiple hierarchical format, then synthetic sequence of each process is considered to be bottom up. At the time of synthesizing each process, synthesis is recommended after specifying the LMSPEC file generated during sub hierarchical synthesis of that process.

For example, when 2 sub hierarchy "foo" and "bar" exists under the parent hierarchy "top", then the recommended synthetic process will change to {foo, bar} → top (However, synthetic process of process foo and process bar is not considered).

Below, **Figure 95** shows the synthesis flow of each process foo, bar, top. First, synthesis of process foo and process bar is performed and LMSPEC file of process foo and process bar is generated.

Secondly, at the time of process top synthesis in these parent hierarchies, foo.LMSPEC, bar.LMSPEC files are specified and synthesized.

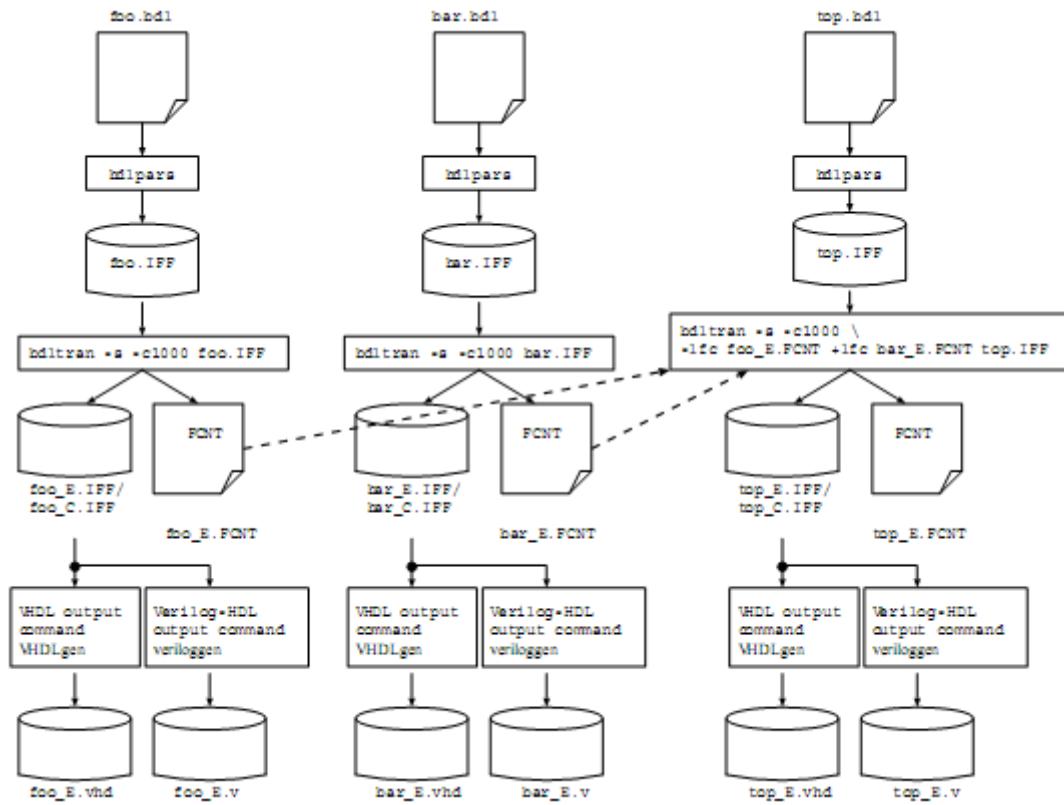


Figure 95: Recommended Synthesis Flow

24.4 Merits of Bottom-Up Synthesis

Merits of the flow of bottom up synthesis are as follows.

1. Early detection of port declaration miss and reduction in the port name adjustment efforts of parent hierarchy by the function of port consistency check.
 - Port declaration miss at the time of setting can be detected early by port consistency check between the hierarchy and adjustment.
 - At the time of parental hierarchical synthesis, as the port of hierarchy is generated in accordance with the port name unrolling rule of child hierarchy, it is possible to reduce the efforts occurred in port name consistency check between the hierarchy.
2. Reduction Effect in Synthesis Area

As interface generation of high order hierarchy is performed, which is based on the port information of sub hierarchy, the need of generating MUX ends, depending on the circumstances and have synthesis area reduction effect.

24.4.1 Port Consistency Check and Adjustment Functionality

User can check the consistency before the RTL simulation by port consistency check performed with the sub hierarchy during the parent hierarchy synthesis. For example, port x is declared in sub hierarchy foo as shown below. In case port x is not declared in the description of defmod of parent hierarchy "top", then the following error message is displayed by specifying the LMSPEC file of sub hierarchy during the synthesis of parent hierarchy "top" and error terminates.

F_BT5079: x port is described in DEFMOD of LMSPEC file (foo.LMSPEC), but does not exist in member of defmod foo

[Countermeasure]Revise the description of defmod. Or, modify the description of hierarchy foo and synthesize again.

foo.bdl(Sub hierarchy description):

```
in ter (0:8)x;
in ter (0:8)y;
out ter (0:8)z;
process foo() {
    z=x;
    $ 
    z=y;
}
```

top.bdl(Parent hierarchy description):

```
defmod foo {
    in unsigned ter(0:8) y;
    out unsigned ter(0:8) z;
} INST_foo;
in unsigned ter(0:8) y;
out unsigned ter(0:8) z;
process top()
{
    INST_foo.y ::= y;
    z ::= INST_foo.z;
    return;
}
```

In case these ports are not described in defmod part during parent hierarchy description related to clock-reset signal/ logic bist signal/ external functional unit signal generated automatically during sub hierarchical synthesis, then there is a functionality of automatic insertion/automatic connection of these signals and port consistency of hierachal setting is adjusted automatically.

For example, after sub hierarchy description is synthesized, it is assumed that clock/resets signal which is automatic generated result, port configuration changes as shown below.

foo.LMSPEC:

```

#@VERSION{1.00}
#@CNT{foo}
#@KIND{BASIC_OPERATOR}
#@CLK 1000
#@UNIT 1/100ns
@PROCESS{
    NAME foo
    ...
    DEFMOD {
        in ter(0:8) x /* glo, x */;
        out ter(0:8) y /* glo, y */;
        clock CLOCK/* c */;
        reset RESET/* r */;
    }
}
#@END{foo}

```

If this foo.LMSPEC file is specified during the synthesis of parent hierarchy description, clock/reset port is automatically inserted in defmod as shown below. Moreover, it is automatically connected with clock/reset port.

Parent hierarchy description

(before clock/reset auto insertion /auto connection):

```

defmod foo {
    in ter(0:8) x;
    out ter(0:8) y;
} INST_foo;
in ter(0:8) x;
out ter(0:8) y;
process Top(){
    INST_foo.x ::= x;
    y ::= INST_foo.y;
    return;
}

```

Parent hierarchy description

(after clock/reset auto insertion/ auto connection):

```

defmod foo {
    in ter(0:8) x;
    out ter(0:8) y;
    clock CLOCK;
    reset RESET;
} INST_foo;
in ter(0:8) x;
out ter(0:8) y;
clock CLOCK;
reset RESET;
process Top(){
    INST_foo.x ::= x;
    y ::= INST_foo.y;
    INST_foo.CLOCK ::= CLOCK;
    INST_foo.RESET ::= RESET;
    return;
}

```

Further as shown below, array port x declared in sub hierarchy foo is unrolled with x_c00 ~ x_c03 by array_index attribute value, but array port x existing in defmod of top hierarchy is unrolled automatically with x_c00 ~ x_c03 by importing LMSPEC file of sub hierarchy foo during the synthesis of parent hierarchy. Therefore, as port of parent hierarchy is generated in accordance with port name unrolling rule of sub hierarchy during synthesis of parent hierarchy, user can reduce the efforts occurring while checking the port name consistency between hierarchies.

foo.bdl(sub hierarchy description):

```
in ter (0:8)x[4]
/*Cyber
array_index_suffix=_c*/;
in ter (0:8)y;
out ter (0:8)z;
process foo(){
    z = x[0];
    $ ...
}
```

top.bdl(parent hierarchy description):

```
defmod foo {
    in unsigned ter(0:8) x[4];
    // -> expanded as "x_c00"~"x_c03"
    in unsigned ter(0:8) y;
    out unsigned ter(0:8) z;
} INST_foo;
in unsigned ter(0:8) x[4];
//-> expanded as "x_c00"~"x_c03"
in unsigned ter(0:8) y;
out unsigned ter(0:8) z;
process top() {
    INST_foo.x[0] ::= x[0];
    //->INST_foo.x_c00 ::= x_c00;
    INST_foo.x[1] ::= x[1];
    //->INST_foo.x_c01 ::= x_c01;
    ...
}
```

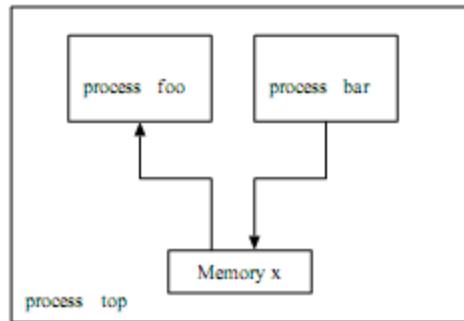
24.4.2 Synthesis Area Reduction effect

Synthesis area reduction effect is explained in concrete terms.

Below, memory existing in parent hierarchy considers the respective reference/assignment case in sub hierarchy foo/bar.

At this time, in the sub hierarchy "foo", port for shared memory generates only the read based port (RE, RA, RD) after synthesis because only reference of memory x is carried out. Further, in the sub hierarchy "bar", port for shared memory generates only the write enable port (WE, WA, WD) after synthesis because only assignment of memory x is carried out.

During the synthesis of parent hierarchy "top", output circuit of parent hierarchy "top" changes as shown in Figure 97 by specifying the LMSPEC file of sub hierarchy foo/bar. On the other hand, in case parent hierarchy "top" is synthesized without specifying the LMSPEC file of sub hierarchy foo/bar, then it changes as shown in Figure 98 and redundant MUX is generated.

**Figure 96:** Memory Access Description

top.bdl(parent hierarchy description):

foo.bdl(sub hierarchy description):

```

in ter (0:8)foo_adr;
out ter (0:8)y;
shared mem (0:8)x[256];
process foo(){
    y = x[foo_adr];
}

```

bar.bdl(sub hierarchy description):

```

in ter (0:8)z;
in ter (0:8)bar_adr;
shared mem (0:8)x[256];
process bar(){
    x[bar_adr] = z;
}

```

```

defmod foo {
    in ter(0:8) foo_adr;
    out ter(0:8) y;
    shared mem(0:8) x[256];
} INST_A;
defmod bar {
    in ter(0:8) z;
    in ter(0:8) bar_adr;
    shared mem(0:8) x[256];
} INST_B;
in     ter(0:8) foo_adr;
out    ter(0:8) y;
in     ter(0:8) z;
in     ter(0:8) bar_adr;
mem(0:8) x[256];
process top(){
    allstates {
        assign(x, INST_A.x);
        assign(x, INST_B.x);
    }
    INST_A.foo_adr::=foo_adr;
    y ::= INST_A.y;
    INST_B.z ::= z;
    INST_B.bar_adr::=bar_adr;
    return;
}

```

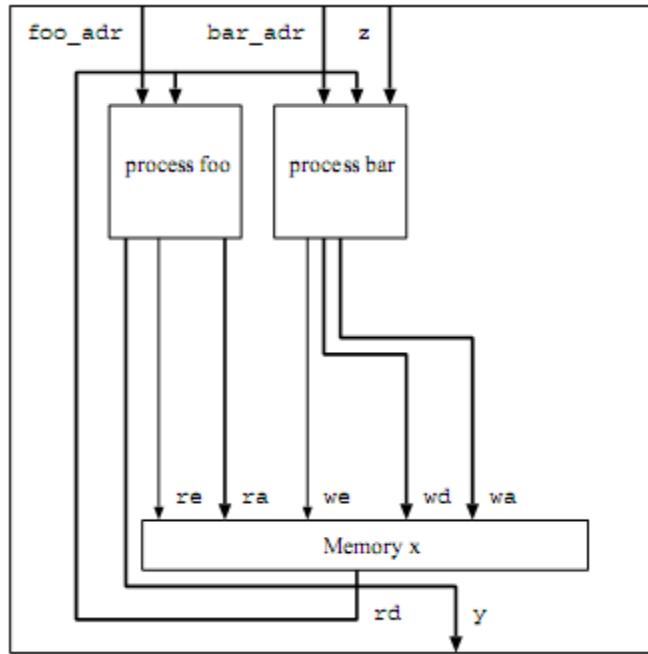


Figure 97: Parent Hierarchy Data Path (When LMSPEC File is specified)

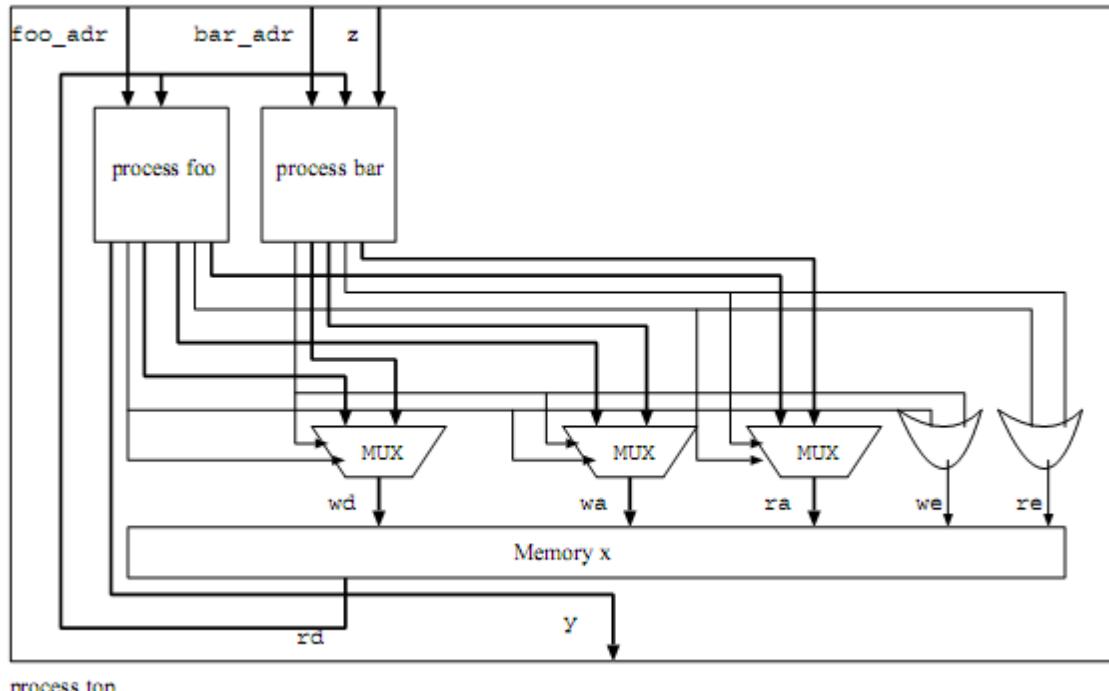


Figure 98: Parent Hierarchy Data path (When LMSPEC file is not specified)

24.5 Synthesis functionality for lower module

In this section, description related to synthesis functionality for lower module is given.

24.5.1 Related option

Option	Description
-Zlower_module	Synthesize for lower module (default)
-Ztop_module	Synthesize for top module

24.5.2 Description

By default or when the “-Zlower_module” option has been specified, behavioral synthesis is carried out considering lower module (not the top level module) as the target module. When the “-Ztop_module” option has been specified, top module is considered for synthesis. It is not recommended to specify the “-Ztop_module” option if the target module for synthesis is the lower module.

25 Constraint File Generation

25.1 Overview

This section describes constraint file generation for functional units and memories required during behavioral synthesis.

Following are the methods to generate constraint file:

- Generation of constraint file and synthesis is done by using "Functional unit constraint: none-use mode" (refer to section **25.2.1**) and "Memory constraint: none-use mode (refer to section **25.2.2**)". (Recommended)
- Generation of constraint file only. (Not recommended)

25.1.1 Related Options

Options for "functional unit constraint: none-use/use mode" and "memory constraint: none-use/use-mode":

Option	Description
-Zresource_fcnt= GENERATE	Synthesizes after required functional unit constraint gets automatically generated.
-Zresource_fcnt=USE	Synthesizes only with functional unit available in functional unit count file (FCNT) specified with '-lfc' option.
-Zresource_fcnt=AUTO	Specifies 'USE' when '-lfc' option is specified and 'GENERATE (default)' when '-lfc' is not specified.
-Zresource_mcnt=GENERATE	Synthesizes after required memory constraint gets automatically generated.
-Zresource_mcnt=USE	Synthesizes only with memory available in memory count file (MCNT) specified with '-lmc' option.
-Zresource_mcnt=AUTO	Specifies 'USE' when '-lmc' option is specified and 'GENERATE (default)' when '-lmc' is not specified.
-Zresource_cnt=GENERATE	Same as options -Zresource_fcnt=GENERATE and -Zresource_mcnt=GENERATE.
-Zresource_cnt=USE	Same as options -Zresource_fcnt=USE and -Zresource_mcnt=USE.

Option	Description
-Zresource_cnt=AUTO	Same as options -Zresource_fcnt=AUTO and -Zresource_mcnt=AUTO.

- Options for FLIB and FCNT file generation

Option	Description
-Zflib_fcnt_out	Generates FLIB, FCNT file for basic functional unit File name (circuit name -auto.FLIB, circuit name-auto.FCNT)
-Zflib_fcnt_out= <i>filename</i> [.FLIB]	Generates FLIB, FCNT file with the file name specified in <i>filename</i> [.FLIB], <i>filename</i> .FCNT
-Zflib_fcnt_arith_macro_out	Generates FLIB, FCNT file for arithmetic macro functional unit File name (circuit name -amacro-auto.FLIB, FCNT)
-Zflib_fcnt_arith_macro_out= <i>filename</i> [.FLIB]	Generates FLIB, FCNT file specified with file name in <i>filename</i> [.FLIB], <i>filename</i> .FCNT
-Zfcnt_out	Generates FCNT file for basic functional unit File name (circuit name=auto.FCNT)
-Zfcnt_out= <i>filename</i> [.FCNT]	Generates the FCNT file with the file name specified in the <i>filename</i> [.FCNT]
-Zfcnt_arith_macro_out	Generates FCNT file for arithmetic macro functional unit File name (circuit name -amacro-auto.FCNT)
-Zfcnt_arith_macro_out= <i>filename</i> [.FCNT]	Generates = <i>filename</i> [.FCNT] FCNT file with file name specified in <i>filename</i> [.FCNT]
-Zflib_out	Generates FLIB file for the basic functional unit File name(circuit name=auto.FLIB)
-Zflib_out= <i>filename</i> [.FLIB]	Generates the FLIB file with the file name specified in the <i>filename</i> [.FLIB]
-Zflib_out_create=APPEND	Generates FILB file that adds functional unit entry, which is not included in input FLIB
-Zflib_out_create=DIFF	Generates FILB file, which includes only those functional unit entry that are not included in input FLIB (default)

Option	Description
-Zflib_out_create=FIT	Ignores the input FLIB and generates FLIB file
-Zflib_out_estimate=YES -Zflib_out_estimate=NO	Estimates the circuit information of a functional unit entry Does not estimate the circuit information of functional unit entry (default)
-Zflib_out_sign=MIX -Zflib_out_sign=EACH -Zflib_out_sign=NONE	Uses mixed sign functional unit (ASIC: default) Do not use mixed sign functional unit, Uses signed functional unit, unsigned functional unit (FPGA: default) Does not specify the sign to functional unit
-Zflib_out_multi=YES -Zflib_out_multi=NO	Uses adder-subtracter, complex comparator (ASIC: default) Does not use adder-subtracter, complex comparator (FPGA: default)
-Zflib_out_limit=[L#][:M#][:S#]	Specifies the maximum value of constraint count L#: maximum value of constraint count of functional unit of largest area M#: maximum value of constraint count of functional unit of medium area S# : maximum value of constraint count of functional unit of small area

- Options for MLIB and MCNT file generation

Options	Description
-Zmcnt_out	Generates MCNT file File name (circuit name-auto.MCNT)
-Zmcnt_out =filename[.MCNT]	Generates MCNT file with the file name specified in filename[.MCNT]

-Zmlib_mcnt_out	Generates MLIB and MCNT file File name (circuit name-auto.MLIB and auto.MCNT)
-Zmlib_mcnt_out =filename[.MLIB]	Generates MLIB and MCNT files with the file name specified in filename[.MLIB] and filename.MCNT
-Zmem_kind=R#	Allocates memory with read only port of specified count
-Zmem_kind=W#	Allocates memory with write only port of specified count
-Zmem_kind=RW#	Allocates memory with read and write port of specified count
-Zmem_kind=R#W#	Allocates memory with read only and write only port of specified count (default:R1W1)
-Zmem_bitw=#	Specifies data width of memory to be allocated
-Zmem_re	Outputs 're' port to DEFMOD
-Zmem_cs	Outputs 'rs', 'ws' and 'cs' port to DEFMOD
-Zmem_be	Outputs 'be' port to DEFMOD Outputs ENABLEPIN
-Zmem_clocking=enable	Outputs 'rclk', 'wclk'and 'clk' port to DEFMOD
-Zmem_clocking=always	Same as above
-Zrw1=1port	Outputs 'wd' and 'rd' port to DEFMOD
-Zrw1=2port	Ouputs 'dt' port to DEFMOD

25.2 Functional unit constraint: None-use/Use mode and Memory constraint: None-use/use mode

- Options

Option	Description
-Zresource_fcnt=GENERATE	Synthesizes after required functional unit constraint gets automatically generated.

Option	Description
-Zresource_fcnt=USE -Zresource_fcnt=AUTO	Synthesizes only with functional unit available in functional unit count file (FCNT) specified with '-lfc' option. Specifies 'USE' when '-lfc' option is specified and 'GENERATE (default)' when '-lfc' is not specified.
-Zresource_mcnt=GENERATE -Zresource_mcnt=USE -Zresource_mcnt=AUTO	Synthesizes after required memory constraint gets automatically generated. Synthesizes only with memory available in memory count file (MCNT) specified with '-lmc' option. Specifies 'USE' when '-lmc' option is specified and 'GENERATE (default)' when '-lmc' is not specified.
-Zresource_cnt=GENERATE -Zresource_cnt=USE -Zresource_cnt=AUTO	Same as options -Zresource_fcnt=GENERATE and -Zresource_mcnt=GENERATE. Same as options -Zresource_fcnt=USE and -Zresource_mcnt=USE. Same as options -Zresource_fcnt=AUTO and -Zresource_mcnt=AUTO.

- Recommended synthesis flow
 - 1) Prepare standard functional unit library (standard.FLIB) file and basic library file (BLIB) (refer to section **I.4**).
Temporary name of file is standard.FLIB and standard.BLIB.
 - 2) Synthesize after specifying standard FLIB and BLIB file.

```
bdltran ... -lfl standard.FLIB -lb standard.BLIB foo.IFF
```

Since '-lfc' and '-lmc' options are not specified, user synthesizes by using None-use mode of functional unit constraint (-Zresource_fcnt=GENERATE) and None-use mode of Memory constraint (-Zresource_mcnt=GENERATE). Required functional unit constraint files (FLIB and FCNT) and memory constraint files (MLIB and MCNT) are automatically generated in synthesis.

- 3) Resynthesize after changing the constraint count of FCNT and MCNT files. If you want to synthesize after specifying functional unit constraint and memory resource constraint, '-lfc' and '-lmc' options are specified, therefore, user synthesizes by using use-mode in functional unit constraint (-Zresource_fcnt=USE) and memory constraint (-Zresource_mcnt=USE) respectively.

```
bdltran ... -lfl standard.FLIB -lb standard.BLIB foo.IFF
          +lfl foo-auto.FLIB +lfc foo-auto.FCNT
          +lml foo-auto.MLIB +lmc foo-auto.MCNT
```

25.2.1 Functional unit constraint- None-use/use-mode

- Functional unit constraint: None-use mode

- o Overview

In none-use mode, user synthesizes without specifying resource constraint of functional unit. It is not necessary to specify resource constraint as it is appropriately calculated in Cyber Behavioral synthesis system.

Calculated resource constraint of functional unit is generated in "Functional unit library file (FLIB)" and "Functional unit count file (FCNT)".

It is necessary to specify "Basic library file (BLIB)" and "Standard functional unit library file (Standard FLIB)" in order to synthesize using None-use mode.

- o Generated files

Usually, FLIB and FCNT files of functional unit and arithmetic macro are generated. (Same as those files that are generated using `-Zplib_fcnt_out` and `-Zplib_fcnt_arith_macro_out` option.)

```
*-auto.FLIB  
*-auto.FCNT  
*-amacro-auto.FLIB  
*-amacro-auto.FCNT
```

Functional unit (functional unit with large bit width, pipeline functional unit etc.) that is not available in standard FLIB is output in generated FLIB file.

Functional unit constraint count required for synthesis is generated in FCNT file (generated).

- o If FLIB file (other than standard FLIB file) is specified

The content of specified FLIB file is inherited to generated FLIB file if specified FLIB file is similar to generated FLIB file.

- Functional unit constraint: Use mode

- o Overview

In use mode, user synthesizes after specifying FLIB and FCNT file. It is used when user wants to assign constraint in behavioral synthesis system.

Synthesis using use-mode is done, if resource constraint is required to be specified manually on the basis of circuit or device specification.

- Conditions for None-use and use mode of functional unit constraint

Synthesis mode (none-use/use mode) of functional unit constraint is determined based on the specification of "`-Zresource_fcnt`" option and availability of FCNT files.

- o Functional unit constraint: None-use mode

In any of the following cases, user synthesizes by using none-use mode of functional unit constraint. (Specification of BLIB file and standard FLIB is necessary.)

- * If FCNT file is not specified with option –Zresource_fcnt=AUTO (default).
- * If –Zresource_fcnt=GENERATE is specified.

Synthesis is carried out by using none-use mode irrespective of whether FCNT specification is available.

FLIB and FCNT files that are synthesized using none-use mode, are automatically generated during synthesis (first time synthesis), with the state in which FLIB and FCNT files are not available. If you want to change the mode for synthesis from use to none-use, you can synthesize by keeping the specified FLIB and FCNT files as it is and specifying –Zresource_fcnt=GENERATE.

- o Functional unit constraint: Use-mode

In any of the following cases, user synthesizes by using use-mode of functional unit constraint.

- * If FCNT file is specified with option –Zresource_fcnt=AUTO (default).
- * If –Zresource_fcnt=USE is specified.

- Options for functional unit constraint generation

- o Constraint files are generated only if the following options, for functional unit constraint file, are specified. (Ignore even if –Zresource_fcnt=GENERATE option is specified.)

`-Zflib_fcnt_out, -Zflib_out, -Zfcnt_out,
-Zflib_fcnt_arith_macro_out, -Zfcnt_arith_macro_out`

- o The following options are valid for none-use mode of functional unit constraint also.

`-Zflib_out_sign, -Zflib_out_multi, -Zflib_out_limit,
-Zflib_out_create, -Zflib_out_estimate`

- o In functional unit: none-use mode, -Zflib_out_estimate=YES is default option.

25.2.2 Memory Constraint: None-use/Use-mode

- Memory constraint: None-use mode
- o Overview

In none-use mode, user synthesizes without specifying resource constraint of memory. It is not necessary to specify resource constraint as it is appropriately calculated in Cyber behavioral synthesis system.

Calculated resource of memory resource constraint is generated in "Memory library file (MLIB)" and "Memory count file (MCNT)".

It is necessary to specify "Basic library file (BLIB)" for synthesizing using None-use mode.

- o Generated files

MLIB and MCNT files are generated. (Same as those files that are generated using –Zmlib_mcmt_out option.)

*-auto.MLIB
*-auto.MCNT

- o If MLIB file is specified

The content of specified MLIB file is inherited to generated MLIB file if specified MLIB file is similar to generated MLIB file.

If memory is available for synthesis in specified MLIB file, synthesize by using that memory.

- Memory constraint: Use mode

- o Overview

In use-mode, user synthesizes after specifying MLIB and MCNT files. It is used when constraint is assigned in behavioral synthesis system.

Synthesis using use-mode is done, if resource constraint is required to be specified manually on the basis of circuit or device specification.

- Conditions for none-use and use-mode of memory constraint

Synthesis mode (none-use/use mode) of memory constraint is determined based on the specification of “–Zresource_mcmt” option and availability of MCNT files.

- o Memory constraint: none-use mode

In any of the following cases, user synthesizes by using none-use mode of memory constraint. (Specification of BLIB file is necessary.)

- * If MCNT file is specified with option –Zresource_mcmt=AUTO (default).
- * If –Zresource_fcnt=GENERATE is specified.

Synthesis is carried out by using none-use mode irrespective of whether MCNT specification is available.

MLIB and MCNT files that are synthesized using none-use mode, are automatically generated during synthesis (first time synthesis), with the state in which MLIB and MCNT files are not available.

If you want to change the mode for synthesis from use to none-use, you can synthesize by keeping the specified MLIB and MCNT files as it is and specifying –Zresource_mcmt=GENERATE.

- o Memory constraint: use mode

In any of the following cases, user synthesizes by using use-mode in memory constraint.

- * If MCNT file is specified with option –Zresource_mcmt=AUTO (default).
- * If –Zresource_mcmt=USE is specified.

- Options for memory constraint generation

- o Completes the generation of constraint file only if the following options for memory constraint file are specified. (ignore even if-Zresource_mcnt=GENERATE option is specified.)

`-Zmllib_mcnt_out, -Zmcnt_out`

- o The following options are valid in none-use mode of memory constraint also.

`-Zmcnt_format, -Zmem_kind, -Zmem_bitw`

25.3 Generation of Functional Unit Library (FLIB) file and Functional Unit Count (FCNT) files

25.3.1 Overview

It is recommended to use “none-use mode” of functional unit constraint (see section **25.2.1**) to generate FLIB and FCNT files. (As per the method explained in that section, it is possible to generate FLIB and FCNT files. However it is not recommended.)

This section describes two methods for FLIB file (refer to **section 39.7**) and FCNT file (refer to **section 39.8**) generation.

- FLIB and FCNT file generation (-Zflib_fcnt_out)
- FCNT file generation (-Zfnct_out)

25.3.2 Related options

Option	Description
<code>-Zflib_fcnt_out</code>	File name generating the FLIB and FCNT file for basic functional unit (circuit name=auto.FLIB, circuit name=auto.FCNT)
<code>-Zflib_fcnt_out =filename[.FLIB]</code>	Specifies file name in <code>filename[.FLIB]</code> , <code>filename.FCNT</code> and generates FLIB, FCNT file
<code>-Zflib_arith_macro_out</code>	Generates FLIB, FCNT files for arithmetic macro functional unit File name(circuit name-amacro-auto.FLIB, FCNT)
<code>-Zflib_arith_macro_out =filename[.FLIB]</code>	Generates FLIB, FCNT files with file name specified in <code>filename[.FLIB]</code> , <code>filename.FCNT</code>
<code>-Zflib_out</code>	File name generating the FLIB file for basic functional unit (circuit name=auto.FLIB)

Option	Description
-Zplib_out=filename[.FLIB]	Specifies file name in filename[.FLIB] and generates FLIB file
-Zplib_out_create=APPEND	Generates FILB file that adds functional unit entry, which is not included in input FLIB
-Zplib_out_create=DIFF	Generates FILB file that includes only functional unit entry, which is not included in input FLIB (default)
-Zplib_out_create=FIT	Ignores input FLIB, and generates FILB file
-Zplib_out_estimate=YES	Estimates the circuit information of functional unit entry
-Zplib_out_estimate=NO	Does not estimate circuit information of functional unit entry (default)
-Zplib_out_sign=MIX	Uses mixed sign functional unit (ASIC: default)
-Zplib_out_sign=EACH	Does not use mixed sign functional unit and uses signed functional unit, unsigned functional unit (FPGA: default)
-Zplib_out_sign=NONE	Does not specify sign in functional unit
-Zplib_out_multi=YES	Uses adder-subtractor and complex comparator (ASIC: default)
-Zplib_out_multi=NO	Does not uses adder-subtractor and complex comparator (FPGA: default)
-Zplib_out_limit=[L#]:[M#]:[S#]	<p>Specifies maximum value of constraint count</p> <p>L#: maximum value of constraint count of functional unit of largest area</p> <p>M#: maximum value of constraint count of functional unit of medium area</p> <p>S# : maximum value of constraint count of functional unit of small area</p>

(a) The operation flow is same as below when standard functional unit library (standard FLIB) is used .

(1) FLIB having standard functional unit library (#@LIBTYPE{STANDARD} specification. Below, standard FLIB) file is prepared (refer to **section I.4**).

(2) FLIB file, FCNT file are generated for behavioral synthesis. Functional unit, having pipeline functional unit specification (FU_IP) in functional unit with large bit width or ope_synth attribute (refer to **section 20.10.1**), not included in standard FLIB is generated as additional FLIB file by default. In case, functional unit included in standard FLIB is adequate, then an empty FLIB file not including the functional unit is generated.

```
bdltran ... -lfl standard.FLIB -Zflib_fcnt_out foo.IFF
```

(3) Synthesis is done with generated FLIB file, FCNT file specified in synthesis option. In case, change in FLIB file or FCNT file by source modification or option change etc. is required, then additional FLIB file and FCNT file can be updated by specifying -Zflib_fcnt_out option at the same time with the following option. Further, FLIB file and FCNT file for arithmetic macro functional unit can be generated by specifying the -Zflib_fcnt_arith_macro_out option.

```
bdltran ... -lfl standard.FLIB +lfl foo-auto.FLIB -lfc foo-auto.FCNT ...
```

(b) Operational flow in case of standard FLIB not used is same as below.

(1) FLIB file, FCNT file are generated for behavioral synthesis.

Functional unit required in synthesis is output to the generated FLIB file, FCNT file. #@LIBTYPE{GENERATED} is specified in the generated FLIB file. This #@LIBTYPE specification shows that it is not standard FLIB.

```
bdltran ... -Zflib_fcnt_out foo.IFF
```

The following message is output in case functional unit is added in generated FLIB file. When this message is output, then execution of below(2) FLIBgen is required.

I_BT4387: FLIB, in which delay and area are not set, is generated. Delay, area is set with executing the functional unit library automatic setting tool (FLIBgen)

(2) Delay or area is set in FLIBgen for generated FLIB file. (refer to **section I.3**)

```
FLIBgen foo-auto.FLIB
```

(3) Synthesis is done with specifying the generated FLIB file, FCNT file in synthesis option.

```
bdltran ... -lfl foo-auto.FLIB -lfc foo-auto.FCNT ...
```

In case change in FLIB file or FCNT file is required by source modification or option change, then FLIB file and FCNT file can be updated by specifying `Zplib_fcnt_out` option with generated FLIB file, FCNT file.

```
bdltran ... -lfl foo-auto.FLIB -lfc foo-auto.FCNT -Zplib_fcnt_out ...
```

In the generated FLIB file, functional unit (functional unit with delay or area set) of FLIB file specified by option is inherited.

When FLIB file with no standard FLIB specification is used, in the generated FLIB file, synthesis is possible by specifying only generated FLIB file and FCNT file because the functional unit required in synthesis is included.

```
bdltran ... -lfl foo-auto.FLIB -lfc foo-auto.FCNT ...
```

Further, FLIB file and FCNT file for arithmetic macro functional unit can be generated by specifying the `-Zplib_fcnt_arith_macro_out` option.

The option of 5 categories related to `-Zplib_out` option generating only option `-Zplib_fcnt_out` and FLIB for basic functional unit is described below.

- `-Zplib_out_create`
- `-Zplib_out_estimate`
- `-Zplib_out_mult`
- `-Zplib_out_sign`
- `-Zplib_out_limit`

The option `-Zplib_out_create` is set to APPEND by default. This option appends a functional unit entry which is not present in the input FLIB file. In case `-Zplib_out_create` is set to DIFF, it generates an FLIB file which includes only those functional unit entries, which are not included in input FLIB file. In case of `-Zplib_out_create` is set to FIT, the input FLIB is ignored while generating the FLIB file. If the input FLIB file is not specified while using the `-Zplib_out_create` option, the `-Zplib_out_create` option would be ignored. Thus, it would behave as if `-Zplib_out_create` was not specified at all.

The FLIB and FCNT file are explained by using the following module2.bdl. For this example, it is assumed that main.FLIB is specified in behavioral synthesis option.

[Description] module2.bdl**[Input FLIB] main.FLIB**

```
in ter(0:6) a1, a2, a3;          #@VERSION{1.00}
in ter(0:8) b1, b2;            #@LIB{main}
in ter(0:3) d1;                #@UNIT 1/100ns
@FLIB {
    NAME      add8u
    KIND      +
    BITWIDTH 8
    DELAY    100
    AREA     400
    SIGN     UNSIGNED
}
@FLIB {
    NAME      add8
    KIND      +
    BITWIDTH 8
    DELAY    110
    AREA     410
    SIGN     SIGNED,UNSIGNED
}
#@END{main}
```

In case -Zflib_out_create=APPEND is specified (which is also set by default), the following FLIB and FCNT files are generated.

[Output FLIB] module2-auto.FLIB

```

#@VERSION{1.00}
#@LIB{main}
#@UNIT 1/100ns
@FLIB {
    NAME      add06_07u
    KIND      +
    BITWIDTH  6,7
    DELAY     1t
    SIGN      UNSIGNED
}
@FLIB {
    NAME      add06_07
    KIND      +
    BITWIDTH  6,7
    DELAY     1t
    SIGN      SIGNED,UNSIGNED
}
@FLIB {
    NAME      add8u
    KIND      +
    BITWIDTH  8
    DELAY     100
    AREA      400
    SIGN      UNSIGNED
}
#@END{main}

```

[Output FCNT] module2-auto.FCNT

```

#@VERSION{1.00}
#@CNT{main}
#@UNIT 1/100ns
@FCNT {
    NAME      add06_07u
    LIMIT    1
}
@FLIB {
    NAME      add8u
    LIMIT    2
}
@OPERATOR{
    NAME      bar
    KIND      @bar
    LIMIT    1
    BITWIDTH (3),8
    DELAY    1t
}
#@END{main}

```

In case -Zplib_out_create=DIFF is specified, the following FLIB and FCNT files are generated.

[Output FLIB] module2-auto.FLIB

```

#@VERSION{1.00}
#@LIB{main}
#@UNIT 1/100ns
@FLIB {
    NAME      add06_07u
    KIND      +
    BITWIDTH  6,7
    DELAY     1t
    SIGN      UNSIGNED
}
@FLIB {
    NAME      add06_07
    KIND      +
    BITWIDTH  6,7
    DELAY     1t
    SIGN      SIGNED,UNSIGNED
}
#@END{main}

```

[Output FCNT] module2-auto.FCNT

```

#@VERSION{1.00}
#@CNT{main}
#@UNIT 1/100ns
@FCNT {
    NAME      add06_07u
    LIMIT    1
}
@FLIB {
    NAME      add8u
    LIMIT    2
}
@OPERATOR{
    NAME      bar
    KIND      @bar
    LIMIT    1
    BITWIDTH (3),8
    DELAY    1t
}
#@END{main}

```

In case -Zplib_out_create=FIT is specified, the following FLIB and FCNT files are generated.

[Output FLIB] module2-auto.FLIB

[Output FCNT] module2-auto.FCNT

```

{@VERSION{1.00}
{@LIB{main}
{@CLK 1000
{@UNIT 1/100ns
@FLIB{
  NAME      add06_07u
  KIND      +
  BITWIDTH 6,7
  DELAY    1t
  SIGN     UNSIGNED
}
@FLIB{
  NAME      add06_07
  KIND      +
  BITWIDTH 6,7
  DELAY    1t
  SIGN     SIGNED,UNSIGNED
}
@FLIB{
  NAME      add08_09u
  KIND      +
  BITWIDTH 8,9
  DELAY    1t
  SIGN     UNSIGNED
}
@FLIB{
  NAME      add08_09
  KIND      +
  BITWIDTH 8,9
  DELAY    1t
  SIGN     SIGNED,UNSIGNED
}
#@END{main}
}

{@CNT{main}
{@CLK 1000
{@UNIT 1/100ns
@FCNT{
  NAME      add06_07u
  LIMIT    1
}
@FCNT{
  NAME      add08_09u
  LIMIT    2
}
@OPERATOR{
  NAME      bar
  KIND      @bar
  LIMIT    1
  BITWIDTH (3),8
  DELAY    1t
}
#@END{main}
}

```

In case of `-Zplib_out_estimate=NO` and default, the template format of FLIB file is generated. In the template format of FLIB file, circuit information is not set therefore, along with using constraint file of behavioral synthesis, a functional unit library delay and area setting tool (Refer **section I**) is also used and it is required to set circuit information for each functional unit inside FLIB file. "1T" indicates 1 cycle delay of each functional unit, which is set in the FLIB file of template format, besides this area etc. is not set, therefore caution will be required. In case `-Zplib_out_estimate=YES` option is specified, the circuit information of a functional unit, which is automatically generated from each functional unit inside input FLIB, is estimated. However, in case functional unit entry which cannot be estimated from each functional unit inside input FLIB, a functional unit of template format is generated. In this case, above mentioned functional unit library, delay area setting tool is used and it is necessary to set the circuit information of functional unit in a template format. Through this functionality, the design efficiency will increase as there is no need to frequently characterize the FLIB file. Further, the designer who cannot use logical synthesis tool can create FLIB file that includes circuit information of functional unit. If -

Zplib_out_estimate=YES option is specified along with the input FLIB then the behavior is same as when option -Zplib_out_estimate is not specified.

Functionality of option -Zplib_out_multi, -Zplib_out_sign, and -Zplib_out_limit is provided in **section 25.3.3**.

Further, in case it is possible to implement array in combinatorial circuit, functional unit for combinatorial circuit implementation of array is generated at the time of FCNT file generation. Details are provided in **section 25.3.3**.

[Description] module3.bdl**[Input FLIB] main.FLIB**

```
in ter(0:8) a1, a2;                      #@VERSION{1.00}
in ter(0:10) b1, b2;                      #@LIB{main}
out reg(0:16) z;                          #@UNIT 1/100ns
process main(){
    z = a1 + a2;
    $
    z = b1 + b2;
    $
}
@FLIB {
    NAME      add8u
    KIND      +
    BITWIDTH 8
    DELAY    100
    AREA     400
    SIGN     UNSIGNED
}
@FLIB {
    NAME      add16u
    KIND      +
    BITWIDTH 16
    DELAY    200
    AREA     600
    SIGN     UNSIGNED
}
#@END{main}
```

FLIB generated during specification of
 -Zflib_out_estimate=NO
[Output FLIB] module2-auto.FLIB

```

#@VERSION{1.00}
#@LIB{main}
#@UNIT 1/100ns
@FLIB {
  NAME      add8u
  KIND      +
  BITWIDTH  8
  DELAY    100
  AREA     400
  SIGN     UNSIGNED
}
@FLIB {
  NAME      add10_11u
  KIND      +
  BITWIDTH 10,11
  DELAY    1t
  SIGN     UNSIGNED
# AREA
}
@FLIB {
  NAME      add16u
  KIND      +
  BITWIDTH 16
  DELAY    160
  AREA     600
  SIGN     UNSIGNED
}
#@END{main}

```

FLIB generation during specification of
 -Zflib_out_estimate=YES
[Output FCNT] module2-auto.FCNT

```

#@VERSION{1.00}
#@LIB{main}
#@UNIT 1/100ns
@FLIB {
  NAME      add8u
  KIND      +
  BITWIDTH  8
  DELAY    100
  AREA     400
  SIGN     UNSIGNED
}
@FLIB {
  NAME      add10_11u
  KIND      +
  BITWIDTH 10,11
  DELAY    125
  AREA     450
  SIGN     UNSIGNED
}
@FLIB {
  NAME      add16u
  KIND      +
  BITWIDTH 16
  DELAY    160
  AREA     600
  SIGN     UNSIGNED
}
#@END{main}

```

25.3.3 Generation of FCNT file

Option	Description
-Zfcnt_out	Generation of FCNT file for basic functional unit File name (circuit name-auto.FCNT)
-Zfcnt_out= <i>filename</i> [.FCNT]	Specifies file name in <i>filename</i> [.FCNT] and generates FCNT file
-Zfcnt_arith_macro_out	Generation of FCNT file for arithmetic macro functional unit File name (circuit name-amacro-auto.FCNT)
-Zfcnt_arith_macro_out = <i>filename</i> [.FCNT]	Generates FCNT file with file name specified in <i>filename</i> [.FCNT]

-Zflib_out_sign=MIX	Uses mixed sign functional unit (default)
-Zflib_out_sign=EACH	Uses signed functional unit, unsigned functional unit
-Zflib_out_sign=NONE	Does not specify sign of functional unit
-Zflib_out_multi=YES	Uses adder-subtractor and complex comparator (default)
-Zflib_out_multi=NO	Does not uses adder-subtractor and complex comparator
-Zflib_out_limit =[L#][:M#][:S#]	<p>Specifies maximum value of constraint count</p> <p>L#: maximum value of constraint count of functional unit of largest area</p> <p>M#: maximum value of constraint count of functional unit of medium area</p> <p>S# : maximum value of constraint count of functional unit of small area</p>

If -Zfcnt_out option is specified, the Cyber behavioral synthesis system will generate a FCNT file. A FCNT file contains functional units that are required for synthesizing the behavioral description. The type of functional unit will correspond to the type of the functional unit inside the input FLIB file. Therefore, the user must specify both -Zfcnt_out option and the input FLIB file simultaneously. The input FLIB file is specified using the -lfl option.

The following options can be used to control a FCNT generation regarding -Zfcnt_out option for basic functional unit.

Only -Zflib_out_limit is valid with option -Zfcnt_arith_macro_out for arithmetic macro functional unit-Zflib_out_limit

- -Zflib_out_mult
- -Zflib_out_sign
- -Zflib_out_limit

The -Zflib_out_multi option is set to YES by default. This option gives a higher priority to, complex functional units as compared to simpler ones and includes them in the FCNT file. For example, an adder-subtractor would be given a higher priority over an adder or a subtracter, and complex comparator would be given a higher priority than a simple functionality comparator.

On the contrary, if -Zflib_out_multi is set to NO, simpler function units are given a higher priority as compared to complex ones. For example, adder or subtracter would be given a higher priority over an adder-subtractor and simple comparator would be given a higher priority over complex comparator.

The `-Zflib_out_sign` is set to `MIX` by default. This option gives a higher priority to mixed sign functional units and includes them in the FCNT file. In other words, in case mixed sign functional unit and unsigned (or, signed) functional unit is included in the input FLIB file; the mixed sign functional units would be given a higher priority. On the contrary, in case of `-Zflib_out_sign` is set to `EACH`, the unsigned (or signed) functional unit is given priority over mixed sign functional units. Further, specifying `-Zflib_out_sign=NONE` and `-Zfcnt_out` simultaneously is not supported.

Constraint count in the FCNT file specifies the number of functional operators that would be used to synthesize the behavioral description. However, this count can be controlled by the user by the option `-Zflib_out_limit`. This option specifies the maximum value of functional unit constraint count.

Classification	Type of operation
Functional unit of large area	<code>*, /, %, <<, >></code> , function functional unit, array of logic implementation
Functional unit of medium area	<code>+,-,<,>,<=,>=</code> (bigger than 8 bit)
Functional unit of small area	<code>+,-,<,>,<=,>=</code> (8 bit or less)

Scheduling Mode	Functional unit of large area	Functional unit of medium area	Functional unit small area
Manual scheduling	1	1	1
Automatic scheduling	5	10	20
Automatic pipeline scheduling	No limit	No limit	No limit

If an array has to be implemented as a combinatorial circuit, the information of functional unit count is specified by the array name inside `@OPERATOR` in FCNT file. Cyber Behavioral Synthesis System generates functional unit constraint count in FCNT file, if Behavioral Synthesis System can judge that array inside behavioral description is synthesized as combinatorial. Array can be synthesized as combinatorial circuit by using the attribute or ter type of array is initialized at the time of declaration.

[Description] module1.bdl

```

in ter(0:6) a1, a2, a3 ;
in ter(0:8) b1, b2;
in ter(0:10) c1, c2, c3;
in ter(0:3) d1;

out reg(0:6) x;
out reg(0:8) y;
out reg(0:10) z;

process main(){
    ter(0:8) bar[8]
        = {1,2,3,4,5,6,7,8};

    x = a1 + a2 + a3 +
    bar[d1];
    y = b1 + b2;
    z = c1 + c2 + c3;
}

```

[Input FLIB] main.FLIB

```

#@VERSION{1.00}
#@LIB{main}
#@UNIT 1/100ns
@FLIB {
    NAME add8u
    KIND +
    BITWIDTH 8
    DELAY 100
    AREA 400
}
@FLIB {
    NAME add12u
    KIND +
    BITWIDTH 12
    DELAY 160
    AREA 600
}
#@END{main}
# COMMENT
}
#@END{main}

```

[Output FCNT] module1-auto.fcnt

```

#@VERSION{1.00}
#@CNT{main}
#@UNIT 1/100ns
@FCNT {
    NAME add8u
    LIMIT 4
    # COMMENT
}
@FLIB {
    NAME add12u
    LIMIT 2
    # COMMENT
}
@OPERATOR{
    NAME bar
    KIND @bar
    LIMIT 1
    BITWIDTH (3),8
    DELAY 1t
    # AREA
    # COMMENT
}
#@END{main}
# COMMENT
}
#@END{main}

```

25.4 Generation of Memory Library (MLIB) file, Memory Count (MCNT) file

25.4.1 Overview

It is recommended to use “none-use mode” of memory constraint (see section **25.2.2**) to generate MLIB and MCNT files. (As per the method explained in this section, it is possible to generate MLIB and MCNT files. However it is not recommended.)

In this section, the two methods for generating MLIB (refer to **section 39.10**) file, MCNT (refer to **section 39.11**) file are explained below.

- Generation of MCNT file (-Zmcnt_out)
- Generation of MLIB, MCNT file (-Zmlib_mcmt_out)

In case shared memory is used in hierarchical setting etc., multiple processes, then MLIB is prepared beforehand and MLIB is commonly used in all process. Only MCNT is generated in every process and operational procedure is recommended. In this case, MLIB prepared earlier can be generated by executing -Zmlib_mcmt_out in the top level process.

25.4.2 Related Option

Option	Description
-Zmcnt_out	Generates MCNT file File name (circuit name –auto.MCNT)
-Zmcnt_out = <i>filename</i> [.MCNT]	Generates MCNT file specified with file name specified in <i>filename</i> .MCNT
-Zmlib_mcmt_out	Generates MLIB, MCNT file File name (circuit name –auto.MLIB, circuit name–auto.MCNT)
-Zmlib_mcmt_out = <i>filename</i> [.MLIB]	Generates MLIB, MCNT file with file name specified in <i>filename</i> [.MLIB], <i>filename</i> .MCNT
-Zmem_kind=R# -Zmem_kind=W# -Zmem_kind=RW# -Zmem_kind=R#W#	Specifies KIND of MLIB generated in - Zmlib_mcmt_out Specifies read only port Specifies write only port Specifies read and write dual use port Specifies read only port and write only port (R1W1 is default)
-Zmem_bitw=#	Specifies the data bit width of memory to be allocated
-Zmem_re	Outputs ‘re’ port to DEFMOD

-Zmem_cs	Outputs 'rs', 'ws', and 'cs' port to DEFMOD
-Zmem_be	Outputs 'be' port to DEFMOD Outputs ENABLEPTN
-Zmem_clocking=enable	Outputs 'rclk', 'wclk', and 'clk' port in DEFMOD
-Zmem_clocking=always	Same as above
-Zrw1=1port	Outputs 'wd', and 'rd' port to DEFMOD
-Zrw1=2port	Outputs 'dt' port to DEFMOD

25.4.3 Related Attributes

Attributes	Description	Setting Target
/* Cyber mem_re = create */	Outputs re port to DEFMOD	Signal
/* Cyber mem_cs = create */	Outputs rs, ws, cs port to DEFMOD	Signal
/* Cyber mem_be = create */	Outputs be port to DEFMOD Outputs ENABLEPTN	Signal
/* Cyber mem_clocking =enable */	Outputs rclk, wclk, clk port to DEFMOD	Signal
/* Cyber mem_clocking = always */	Same as above	Signal
/* Cyber univ_mem_if =create */	Outputs rreq, rack, wreq, wack, req, ack port to DEFMOD	Signal

25.4.4 MCNT File Generation

If -Zmcnt_out option is specified, then MCNT file required for behavioral synthesis is generated. Memory registered in MCNT file is selected from the memory types in the input MLIB file. For this purpose, at the time of specifying the -Zmcnt_out option, it is necessary to specify MLIB file in -lml option at the same time.

Memory selected in -Zmcnt_out option is decided by input description and MLIB file and behavioral synthesis option. Information of bit width or data elements or attributes of the arrays declared in input description, furthermore, appropriate memory from behavioral synthesis option are selected from MLIB file.

Options and attributes related to -Zmcnt_out option is shown below.

- Related option
- -Zmem_re
- -Zmem_cs
- -Zmem_be
- -Zmem_clocking
- -Zrw1

- Related attributes
 - mem_re
 - mem_cs
 - mem_be
 - mem_clocking
 - univ_mem_if

By specifying these options and attributes, it is possible to specify selected memory types. For example, options or attributes specifying the configuration of memory port of -Zmem_re option etc. are prepared.

Following is the example of the MCNT generated through -Zmcnt_out option. In the input description, 2 types of array, in which bit width and data elements and attributes are different is declared. In MLIB, 3 types of memory, in which the configuration of port is different is registered. From the set value, memory MEMB16W256_CS attached port of port selection for array M1, memory MEMB16W256 for array M2 is selected respectively. (module4-auto.FCNT).

[Behavioral description] module4.bdl

```
in ter(0:8) a1;
in ter(0:16) a2;
in ter(0:6) b1;
in ter(0:8) b2;
out reg(0:8) x1;
out reg(0:16) x2;
mem(0:8) M1[64] /* Cyber mem_cs = create */;
mem(0:16) M2[256];
process entry()
{
    M1[b1] = a1;
    M2[b2] = a2;
    x1 = M1[b1];
    x2 = M2[b2];
}
```

Input MLIB] main.MLIB

```
#@VERSION { 1.00 }
#@LIB { entry }
@MLIB {
    NAME MEMB16W256
    KIND R1,W1
    BITWIDTH 16
    DELAY 1T
    WORD 256
    DEFMOD {
        in ter (0:8) RA1 /* ra:1 */;
        out ter (0:16) RD1 /* rd:1 */;
        in ter (0:1) RE1 /* re:1 */;
        in ter (0:8) WA2 /* wa:2 */;
        in ter (0:16) WD2 /* wd:2 */;
        in ter (0:1) WE2 /* we:2 */;
    }
}
@MLIB {
    NAME MEMB16W256_CS
    KIND R1,W1
    BITWIDTH 16
    DELAY 1T
    WORD 256
    DEFMOD {
        in ter (0:8) RA1 /* ra:1 */;
        out ter (0:16) RD1 /* rd:1 */;
        in ter (0:1) RE1 /* re:1 */;
        in ter (0:1) RS1 /* rs:1 */;
        in ter (0:8) WA2 /* wa:2 */;
        in ter (0:16) WD2 /* wd:2 */;
        in ter (0:1) WE2 /* we:2 */;
        in ter (0:1) WS2 /* ws:2 */;
    }
}
@MLIB {
    NAME MEMB16W256_BE
    KIND R1,W1
    BITWIDTH 16
    DELAY 1T
    WORD 256
    ENABLEPTN 8,8
    DEFMOD {
        in ter (0:8) RA1 /* ra:1 */;
        out ter (0:16) RD1 /* rd:1 */;
        in ter (0:1) RE1 /* re:1 */;
        in ter (0:1) RS1 /* rs:1 */;
        in ter (0:8) WA2 /* wa:2 */;
        in ter (0:16) WD2 /* wd:2 */;
        in ter (0:1) WE2 /* we:2 */;
        in ter (0:1) WS2 /* ws:2 */;
        in ter (0:2) BE2 /* be:2 */;
    }
}
#@END { entry }
```

[Output MCNT] module4-auto.FCNT

```
#@VERSION { 1.00 }
#@CNT { entry }
@MCNT {
    NAME    MEMB16W256_CS
            # for M1
    LIMIT   1
}
@MCNT {
    NAME    MEMB16W256
            # for M2
    LIMIT   1
}
#@END { entry }
```

25.4.5 Generation of MLIB, MCNT File

MLIB file and MCNT file are generated when -Zmlib_mcnc_out option is specified.

MLIB (MCNT) entry of these files is decided on the basis of information of bit width or data elements or attributes of array allocated to memory in the arrays declared in behavioral description, furthermore on the basis of usage method etc. if corresponding array.

- Memory Name (NAME)

Decided by bit width and data elements of arrays. Memory name can be specified by specifying the attribute sig2mu in array.

- Type (KIND)

Decided by -Zmem_kind option and assignment of array and existence or nonexistence of reference and attribute set in array. Read port and write port, in case of assignment to corresponding array with array reference, write port, in case of only assignment, read port, only in case of reference is set. Read port and write port, in case attribute array = RAM specification is there in array, read port, in case attribute array = ROM specification is specified. Moreover, by specifying the -Zmem_kind option, port count can be specified.

- Bit Width of Data (BITWIDTH)

Bit width of declared array is basically equivalent to the bit width of data. However, it is possible to generate memory with lesser data bit width than the bit width of declared array if "-Zmem_bitw" option is specified. For example, declare the array of 40 bit width and synthesis will become possible by using the memory of bit width of data 16 in three usable ways, if -Zmem_bitw=16 option is specified.

- Memory Size (WORD)

Decided by array data element.

- Access Delay (DELAY)

Generally 1T is set. In case of pipeline memory x2 is set.

- Port Information (DEFMOD)

Set on the basis of related options and attributes. Related options and attributes are similar to -Zmcnt_out (**section 25.4.4**). Based on these specifications, port information required in memory is specified in MLIB file.

- Byte Enable Pattern Specification (ENABLEPTN)

Set when specification of byte enable exists.

- Number of Constraints (LIMIT)

Memory count required in synthesis is set.

In the below example, if -Zmlib_mcnt_out is executed, then MLIB, MCNT files are generated for array foo and bar allocated to memory. Regarding array foo, memory MEM8W256 is generated. This array is not only assigned in reference, therefore KIND becomes R1 and as attribute mem_re, mem_cs specification are there, re port, rs port are specified in DEFMOD specification. Regarding array bar, memory MEMB16W128 is generated. As this array has assignment with reference, KIND becomes RW1, and attribute mem_be specification is there, therefore ENABLEPTN is specified. Further, be port is also specified in DEFMOD.

[Behavioral description] module5.bdl

```
in ter(0:8) a;
in ter(0:7) b;
in ter(0:16) c;
out reg(0:8) x;
out reg(0:16) y;
process main()
  mem(0:8) foo[256]/* Cyber mem_re=create, mem_cs=create */;
  mem(0:16) bar[128]/* Cyber mem_be=create */;
  while (1) {
    bar[b] = c;
    $;
    x = foo[a];
    y = bar[b];
    $;
  }
}
```

[Output MLIB] module5-auto.MLIB

```
#@VERSION { 1.00 }
#@LIB { main }
@MLIB {
    NAME MEMB8W256
    KIND R1
    BITWIDTH 8
    DELAY 1T
    WORD 256
    DEFMOD {
        in ter (0:8) RA1 /* ra:1 */;
        out ter (0:8) RD1 /* rd:1 */;
        in ter (0:1) RE1 /* re:1 */;
        in ter (0:1) RS1 /* rs:1 */;
    }
}
@MLIB {
    NAME MEMB16W128
    KIND RW1
    BITWIDTH 16
    DELAY 1T
    WORD 128
    DEFMOD {
        in ter (0:7) AD1 /* ad:1 */;
        out ter (0:16) RD1 /* rd:1 */;
        in ter (0:16) WD1 /* wd:1 */;
        in ter (0:1) WE1 /* we:1 */;
        in ter (0:2) BE1 /* be:1 */;
    }
}
ENABLEPTN 8+
}
#@END { main }
```

[Output MCNT] module5-auto.MCNT

```
#@VERSION { 1.00 }
#@CNT { main }
@MCNT {
    NAME MEMB8W256
    LIMIT 1
}
@MCNT {
    NAME MEMB16W128
    LIMIT 1
}
#@END { main }
```

25.4.6 MCNT File Generation in case of Hierarchical Description

In case shared type variable is specified in hierarchical description (for details refer to "User manual Hierarchical setting column editing Connection method of multiple module") or shared type variable is allocated to memory, then MCNT file is generated in INSTANCE-type format shown below.

- Instance name (INSTANCE)

Array name is specified. In case array declaration is shared, then "shared" is added in prefix.

- Synthesis method specification (RESOURCE)

Decided by signal declaration. MEM, in case of mem declaration, REG, in case of reg declaration, REGARY, in case of array in reg declaration is specified.

- Priority order specification (PORTPRIORITY)

Not output by default. Hence, after MCNT output, it is necessary to specify the priority order based on the requirement.

The below example is the top level description. When -Zmlib_mcnt_out is executed against this description, then MLIB file and MCNT file are generated. Regarding the array M allocated in the memory declared in shared, memory MEMB8W16 is generated in MLIB, MCNT file. As this array is declared in shared, the extension format for defmod interface is specified. shared.M is specified in INSTANCE of MCNT and MEM is specified in the RESOURCE. Further, only MCNT is generated for the array ary achieved in the register array declared in shared and shared.ary is specified in INSTANCE, REGARY is specified in RESOURCE.

[Behavioral description] module6.bdl

```

out reg(0:8) x;
out reg(0:4) y;
shared mem(0:8) M[16];
shared reg(0:4) ary[8];
defmod foo {
    out ter(0:8) o1;
    out ter(0:4) o2;
    shared mem(0:8) M[16];
    shared reg(0:4) ary[8];
} inst1;
process main() {
    allstates {
        assign(M, inst1.M);
        assign(ary, inst1.ary);
    }
    x ::= inst1.o1;
    y ::= inst1.o2;
}

```

[Output MLIB] module6-auto.MLIB

```

#@VERSION { 1.00 }
#@LIB { main }
@MLIB {
    NAME MEMB8W16
    KIND RW1
    BITWIDTH 8
    DELAY 1T
    WORD 16
    DEFMOD {
        in ter (0:4) AD1 /* ad:1 */;
        out ter (0:8) RD1 /* rd:1 */;
        in ter (0:8) WD1 /* wd:1 */;
        in ter (0:1) WE1 /* we:1 */;
    }
}
#@END { main }

```

[Output MCNT] module6-auto.MCNT

```

#@VERSION { 1.00 }
#@CNT { main }
@MCNT {
    INSTANCE shared.ary
    RESOURCE REGARY
}
@MCNT {
    NAME MEMB8W16
    INSTANCE shared.M
    RESOURCE MEM
}
#@END { main }

```

25.4.7 Precautions

In case of sharing the memory in hierarchical description, it is recommended that common MLIB file should be specified when the synthesis of top level, sub hierarchy. Hence, you have only to generate MLIB file by -Zmlib_mcmt_out command when the synthesis of top level module. Therefore by using the generated MLIB file, setting flow like the generation of only MCNT, through -Zmcnt_out is preferred in sub hierarchy. In case of preparing for MLIB file beforehand, it is recommended to execute -Zmcnt_out command in all process.

25.5 Summary

This section covered the following:

- When the “-Zflib_fcnt_out” option is specified, Cyber generates a template for the FLIB and the FCNT file automatically from the statement.
- When “-Zflib_out” option is specified, it only generates the template of the FLIB file automatically from statement.

- When the “-Zmlmt_out” option is specified, it automatically generates the template of functional unit constraint file from statement.
- When the “-Zmlmt_out” and “-Zmlmt_out = *filename*[.MLMT]” options are specified, the template of memory constraint file is automatically generated from the statement.

26 MultiCycle functional unit/ MultiCycle Memory

26.1 Overview

This section explains about the pipeline functional unit, multicycle functional unit, multi cycle memory and arithmetic macro functional unit.

26.1.1 Related Options

Options	Description
-Zmulti_cycle_ope=reg -Zmulti_cycle_ope=latch Zmulti_cycle_ope=negFF	Inputs from same register (default) Inputs multiple signal after switch over by switching signal using the through latch Inputs multiple signal after switch over by switching signal using the negative phase register
-Zmulti_cycle_mem=reg -Zmulti_cycle_mem=latch - Zmulti_cycle_mem=negFF	Inputs from same register (default) Inputs multiple signal after switchover by switching signals using the through latch Inputs multiple signal after switchover by switching signals using the negative phase register
-Zfsm_dec=slow -Zfsm_dec=fast	Circuit which considers a long delay of select signals (default) Circuit which assumes the delay of the select signal to be small

26.1.2 Related Attributes

Attribute	Description	Settings Target
/*Cyber multi_cycle_input=true*/	Input variable is input directly to the multicycle operation/memory assuming that it is stored externally	Input variable
/* Cyber arith_macro */ /* Cyber arith_macro=name */	Specify the statement assumed to be arithmetic macro operation Specify the statement assumed to be arithmetic macro operation after specifying	Statement Statement

	functional unit name	
--	----------------------	--

26.2 Pipeline Functional Unit

Pipeline functional unit have the register within the functional unit and for input data at each cycle, output at each cycle is possible. Designware connection of Synopsys for ASIC and creation of automatic reasoning description for logical synthesis tool for FPGA, CORE Generator IP connection for Xilinx and for Altera company, Megafunction is possible. Further, it is also possible to use the design assets of pipeline functional unit for RTL or Gate.

Below are the pipeline functional units supporting the logical synthesis description.

- Pipeline multiplier
- Pipeline divider
- Pipeline surplus calculator

Regarding these functional units, description using automatic recommendation description or CORE Generator/IP Catalog (IP catalog supports pipeline multiplier only) or Megafunction for DesignWare or FPGA can be used. Further, to the extent simulation model of pipeline functional unit is supported, regarding VHDL description and Verilog-HDL description, it is mentioned in **section C.10** and **section D.11** respectively.

In case pipeline functional unit is used, then at first it is necessary to prepare a functional unit library (FLIB) for pipeline functional unit for the purpose of behavioral synthesis. Specify attribute ope_synth = FU_IP (Refer to **section 20.10.1**) and by using the FLIB, FCNT file generation functionality (refer to **section 25.3.1**), FLIB for pipeline functional unit can be generated. Further, it can be used after editing the template file (template-asic_pipe.FLIB, template-fpga_pipe.FLIB) for pipeline functional unit included in the installation set. In this case, validity of reset port (RESET_PORT) or enable port (PIPELINE_STALL) etc. can be specified by FLIB file. Further, it is possible to perform the specification (STAGE_SEARCH) etc. of automatically searching the number of stages of pipeline stages. Further, in FPGA logical synthesis tool, it is possible to specify (DSP_MACRO) automatic reasoning description or IP Generator description (refer to **section 39.7**). Afterwards, delay or area can be set by the setting tool of delay area for functional unit library (FLIBgen) execution. At this time, it is necessary to specify the frequency. In the generated FLIB, target device information is also set. At present, description regarding operation classification or FLIB setting supporting the estimation in LIBgen is described in **section I.9**.

```
% FLIBgen -c 1000 -syn_tool ise -fpga_family spartan6  
-fpga_device ... fpga_pipeline.FLIB
```

As the flow using the standard functional unit library (FLIB) (Refer to **section I.4**) is recommended, FLIB and standard FLIB created above are specified in synthesis operation of bdltran and behavioral synthesis are required.

```
% bdltran -s -c1000 -lfl standard.FLIB
+lfcc fpga_pipeline.FLIB ...
```

In the intermediate file generated after bdltran execution, (_E.IFF), functional unit information and target device information set in above process is included. Therefore, when FLIB created for Design Compiler is used, then in the generated RTL description, pipeline functional unit of DesignWare is included. In case FLIB created for FPGA is used, then the description according to the specification for DSP_MACRO specification etc. pipeline functional unit of FLIB is generated in the generated RTL description. However, regarding FLIBgen not supporting the operation classification, the RTL of black box is generated as a separate file. It is necessary to add the instance description when needed.

Below, example of Verilog-HDL description is shown and it is also same for VHDL description also.

```
% veriloggen foo_E.IFF
```

Example) Example of pipeline multiplier unit output of DesignWare

```
module main_mull2_24u_pipe ( i1 ,i2 ,CLOCK ,o1 );
parameter I1_WIDTH = 12;
parameter I2_WIDTH = 12;
parameter O1_WIDTH = 24;
input  [I1_WIDTH-1:0] i1 ;
input  [I2_WIDTH-1:0] i2 ;
input      CLOCK ;
output [O1_WIDTH-1:0] o1 ;
wire      inst_TC ;
assign inst_TC = 1'h0 ;
DW02_mult_3_stage #(I1_WIDTH, I2_WIDTH) U1 ( .A(i1) ,.B(i2) ,
.TC(inst_TC) ,.CLK(CLOCK) ,.PRODUCT(o1) );
Endmodule
```

Example) Example of pipeline multiplier unit output of automatic reasoning description for FPGA

```
module main_mull2_24u_pipe ( i1 ,i2 ,CLOCK ,o1 );
  input [11:0] i1 ;
  input [11:0] i2 ;
  input      CLOCK ;
  output [23:0] o1 ;
  reg    [11:0] i1_r1 ;
  reg    [11:0] i2_r1 ;
  reg    [23:0] o1_r1 ;
  assign o1 = o1_r1 ;
  always @ ( posedge CLOCK )
    begin
      i1_r1 <= i1 ;
    end
  always @ ( posedge CLOCK )
    begin
      i2_r1 <= i2 ;
    end
  always @ ( posedge CLOCK )
    begin
      o1_r1 <= ( i1_r1 * i2_r1 ) ;
    end
endmodule
```

Example) Example of pipeline multiplier unit of CORE Generator

```
module main_mull2_24u_pipe ( i1 ,i2 ,CLOCK ,RESET ,
  o1 ,mul12_24u_pipe1_en );
  input [11:0] i1 ;
  input [11:0] i2 ;
  input      CLOCK ;
  input      RESET ;
  output [23:0] o1 ;
  input      mul12_24u_pipe1_en ;
  mul12_24u_pipe_ip
  INST_mull2_24u_pipe_ip ( .a(i1) ,.b(i2) ,.clk(CLOCK) ,
    .sclr(RESET) ,.ce(mul12_24u_pipe1_en) ,.p(o1) );
endmodule
```

Script for logical synthesis can be generated using logical synthesis script generation tool (Lsscrge) and logical synthesis can be carried out easily. In case of ISE, it is also possible to generate script for CORE generator at the same time. Generation of CORE generator IP is possible by execution of script.

Example) Example of pipeline multiplier output of Megafunction

```

module main_mul12_24u_pipe ( i1 ,i2 ,CLOCK ,o1
                            ,mul12_24u_pipe1_en );
  input [11:0] i1 ;
  input [11:0] i2 ;
  input          CLOCK ;
  output [23:0] o1 ;
  input          mul12_24u_pipe1_en ;
  mul12_24u_pipe_ip INST_mul12_24u_pipe_ip (.aclr(1'b0) ,
                                                .clken(mul12_24u_pipe1_en) ,.clock(CLOCK) ,
                                                .dataaa(i1) ,.dataab(i2) ,.result(o1) );
endmodule

// synopsys translate_off
'timescale 1 ps / 1 ps
// synopsys translate_on
module mul12_24u_pipe_ip ( aclr ,clken ,clock ,dataaa
                           ,dataab ,result );
  input      aclr ;
  input      clken ;
  input      clock ;
  input [11:0] dataaa ;
  input [11:0] datab ;
  output [23:0] result ;
  wire [23:0] sub_wire0;
  wire [23:0] result = sub_wire0[23:0];
  lpm_mult lpm_mult_component (
    .aclr (aclr),
    .clock (clock),
    .dataab (dataab),
    .clken (clken),
    .dataaa (dataaa),
    .result (sub_wire0),
    .sum (1'b0));
  defparam
    lpm_mult_component.lpm_hint = "MAXIMIZE_SPEED=5",
    lpm_mult_component.lpm_pipeline = 2,
    lpm_mult_component.lpm_representation = "UNSIGNED",
    lpm_mult_component.lpm_type = "LPM_MULT",
    lpm_mult_component.lpm_widtha = 12,
    lpm_mult_component.lpm_widthb = 12,
    lpm_mult_component.lpm_widthp = 24;
endmodule

```

Script for logical synthesis can be generated using the logical synthesis script generation tool (Lsscrge). Logical synthesis can also be performed easily. In case of ISE, it is also possible to generate the script for COREGenerator at the same time and CORE Generator IP can be generated by execution of script. Refer to **section A.4** regarding execution method of logical synthesis script.

Further, verification of RTL description including IP is possible by use of the RTL test bench generation tool (tbgen). At this time, it is necessary to specify simulation model of each IP vendor. Below is the example in case of using the ModelSim simulator of Mentor Graphics Company in Verilog-HDL description.

Example) Verification example including DesignWare (Simulator: ModelSim)

```
% tbgen -scr=modelsim -scr_vlog_option=
'`-y $SYNOPSYS/dw/sim_ver +libext+.v' ...
```

Example) Verification example including CORE generator IP (Simulator: ModelSim)

```
% tbgen -scr=modelsim -scr_vlog_option=
'`-y $XILINX/verilog/src/unisims +libext+.v' ...
```

In case setting design of pipeline functional unit exists in RTL or gate, then it is necessary to change the NETLIST specification to EXIST in FLIB or FCNT. Corresponding functional unit is generated as a black box in a different file by performing the above specification. Connection is enabled by initiating the module of setting design in that file.

Example) Example of FCNT files specifying the NETLIST

```
#{@VERSION{2.00}
#{@CNT{PIPELINE}
#{@KIND{BASIC_OPERATOR}
#{@CLK 1000
#{@UNIT 1/100ns
@FCNT{
  NAME      mul12_24u_pipe
  LIMIT    1
  NETLIST  EXIST
}
#{@END{PIPELINE}}
```

26.3 Multicycle Functional Unit/Multicycle Memory

This section explains about the multicycle functional unit/memory which requires multiple cycles in execution.

Multicycle functional unit is a functional unit in which clock cycle performs the delay operation over a number of cycles in small circuit. Input data to multicycle functional unit must be retained and stored in register during execution cycle.

Similarly in multicycle memory, it takes long time even from clock cycle in memory access and it is the memory which performs memory read and memory write over number of cycles. Address data, enable signal and write data to memory must be retained and stored in the register during read operation or write operation.

In this section, following two points are explained.

- Specification method of multi cycle functional unit/memory
- Current methodology of specifying multi cycle functional unit/memory in automatic scheduling

26.3.1 Specification Method in Manual Scheduling

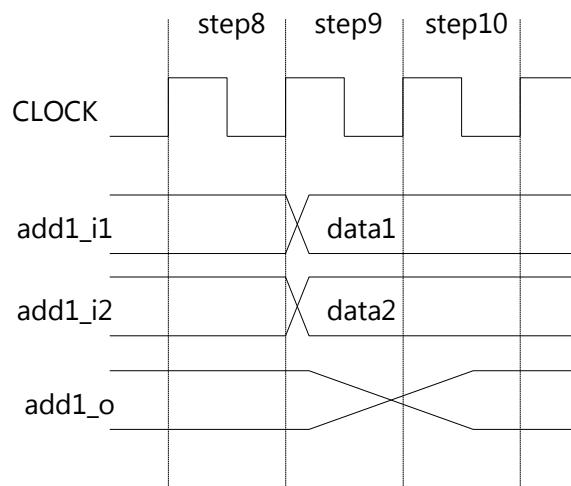
In manual scheduling, multicycle functional unit/memory can be directly specified using continuous assignment statement (:=).

Multicycle functional unit can be specified using continuous assignment as illustrated in the example below. Input variables to multicycle functional unit must hold value for a few cycles using the "reg" variables.

Example of adder which requires two cycles:

```

reg(0:8) add1_i1,add1_i2;
ter(0:8) add1_o;
add1_o ::= add1_i1 + add1_i2;
:
$ /* step 8 */
add1_i1 = data1;
add1_i2 = data2;
$ /* step 9 */
$ /* step 10 */
outdata = add1_o;
$
```



In the example above, value is assigned to register variables in step 8, register output gets updated in step 9, and addition is executed in step 9 & 10.

When a multicycle memory is implemented internally, the memory is specified by converting a function into a functional unit (refer to **section 15.4.3**). Write data, enable signal and addresses are passed to this function as arguments. The read data of memory is specified as the return value of function and can be assigned using the concatenation (::=) to a variable that represents read-data.

Memory read and memory write must be specified using an assignment and reference of data, address and enable signal directly. In the case of external memory, I/O port should be directly specified without using a function conversion into a functional unit.

Example of internal memory:

```
extern var(0:8)
mem_module( var(0:8) write_data,
            var(0:1) write_enable,
            var(0:1) read_enable,
            var(0:6) address );
process example5(){

    reg(0:1) we,re;
    reg(0:6) ad;
    reg(0:8) wdata;
    reg(0:8) rdata;
    rdata ::= mem_module(wdata,we,re,ad);
    :
    :
}
```

Functional unit constraint file:

#IDX	NAME	LIMIT	KIND	BITW	DELAY	AREA
1	mem_module	1	@mem_module	(8,1,1,6),8	8000	400

Example of external memory:

```
in ter(0:8) mem1_ReadData;
out reg(0:8) mem1_WriteData;
out reg(0:1) mem1_WriteEnable;
out reg(0:1) mem1_ReadEnable;
out reg(0:6) mem1_Address;
process example6(){
    reg(0:8) data;
    mem1_Address = 50;
    mem1_WriteEnable = 1;
    $
    $
    data = mem1_ReadData;
    mem1_ReadEnalbe = 0;
    $
    $
    mem1_WriteEnable = 0;
}
```

26.3.2 Specification Method in Automatic Scheduling

- Multicycle functional unit

A functional unit can be specified using the functional unit constraint file. A value greater than a clock cycle can be set as delay of respective functional unit in the LMT file. Otherwise, delay of more than or equal to 2 can be specified in cycle specification. Then a relevant functional unit can be generated as multicycle functional unit.

(For functional unit constraint file, refer to **section 39.2**, for functional unit library file, refer to **section 39.7**)

Functional unit constraint file

```

#@LIB { example1 }
@FLIB{
    NAME      multi_add32
    KIND      +
    BITWIDTH 32
    DELAY    2T
    AREA     400
}
@FLIB{
    NAME      multi_mul32
    KIND      *
    BITWIDTH 32
    DELAY    4T
    AREA     800
}
#@END { example1 }

```

Functional unit count file

```

#@CNT { example1 }
@FCNT {
    NAME multi_add32
    LIMIT 1
}
@FCNT {
    NAME multi_mul32
    LIMIT 1
}
#@END { example1 }

```

- Multicycle memory

Multicycle memory is specified by using the memory constraint (MLMT) file in automatic scheduling mode. By setting a value greater than one clock cycle as the delay of corresponding memory in the MLMT file or if setup delay of more than 2 cycles is specified, then the memory unit is generated as a multicycle memory. (For memory constraint file, refer to [section 39.3](#), for memory library file refer to [section 39.10](#))

Functional Unit Library File

```

#@LIB { example1 }
@FLIB{
    NAME      multi_add32
    KIND      +
    BITWIDTH 32
    DELAY    2T
    AREA     400
}
@FLIB{
    NAME      multi_mul32
    KIND      *
    BITWIDTH 32
    DELAY    4T
    AREA     800
}
#@END { example1 }

```

Functional Unit Count File

```

#@CNT { example1 }
@FCNT {
    NAME multi_add32
    LIMIT 1
}
@FCNT {
    NAME multi_mul32
    LIMIT 1
}
#@END { example1 }

```

26.3.3 Implementation Method of Multicycle Functional Unit/Memory in Automatic Scheduling

26.3.3.1 Multiplexer Select Signal

Option	Description
Specification of the implementation method of multicycle functional unit	
-Zmulti_cycle_ope=reg	Inputs from same register (default)
-Zmulti_cycle_ope=latch	Inputs multiple signals after switch over by switching signal using the through latches.
- Zmulti_cycle_ope=negFF	Inputs multiple signals after switch over by switching signal using the negative registers.
Specification of the implementation method of multicycle memory	
-Zmulti_cycle_mem=reg	Inputs from same registers (default) (not supported)
-Zmulti_cycle_mem=latch	Inputs multiple signals after switch over by switching signal using the through latches.
- Zmulti_cycle_mem=negFF	Inputs multiple signals after switch over by switching signal using the negative registers.
Selection of circuit that generates the select signal of input multiplexer in multicycle operation/memory	
-Zfsm_dec=slow	A secure circuit to take care of long delays in the select signals(default).
-Zfsm_dec=fast	A circuit assuming delay in the select signals to be small.

In automatic scheduling mode, circuits will be synthesized to retain input data for multicycle operations / memory in the same register throughout the execution cycles.

When some conditions have been applied for operation execution or functional unit is shared by multiple operations, a multiplexer may be generated at the input of functional units. If a multiplexer is generated, the selection signal of multiplexer must be retained throughout execution cycle. Therefore, a circuit is generated to retain selection signals throughout execution cycles.

By default, input of multicycle functional unit is synthesized such that same register is used. Thus, the circuit to retain the select signal is not generated. However, the cycle for register transfer is required. Current version of Cyber doesn't support synthesis of a circuit which uses the same register as an input of multicycle memory. Therefore, if input requires multiple registers or if it is constant, it will be treated as an error.

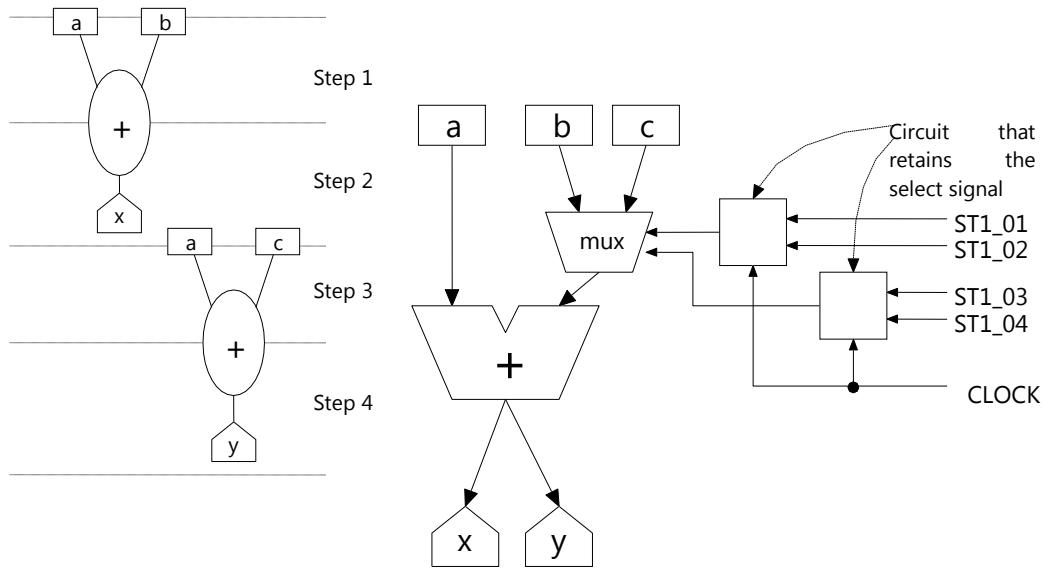


Figure 99: Select signal of multiplexer of multicycle functional unit

A circuit can retain select signals using the following two methods:

- Using a latch (options “-Zmulti_cycle_ope=latch” and “-Zmulti_cycle_mem=latch”)
- Using a negative flip-flop (options “-Zmulti_cycle_ope=negFF” and “-Zmulti_cycle_mem=negFF”)

While using a negative flip-flop, it is necessary that all select signals must stabilize by half period of clock cycle.

Following are the two ways by which a circuit can retain select signals (**Figure 100**):

- By default or when the “-Zfsm_dec=slow” option has been specified, circuit is generated by using only select signals that are in a state to start the operation. In this case, though the die area used will be large, circuit which is safe in delay is generated.
- When the “-Zfsm_dec=fast” option has been specified, a circuit with smaller area is generated using select signals of all states to execute the operation. However, it is necessary in this circuit that select signals must become stable by half period of the clock cycle. However, this is not a necessity for the select signal of starting state.

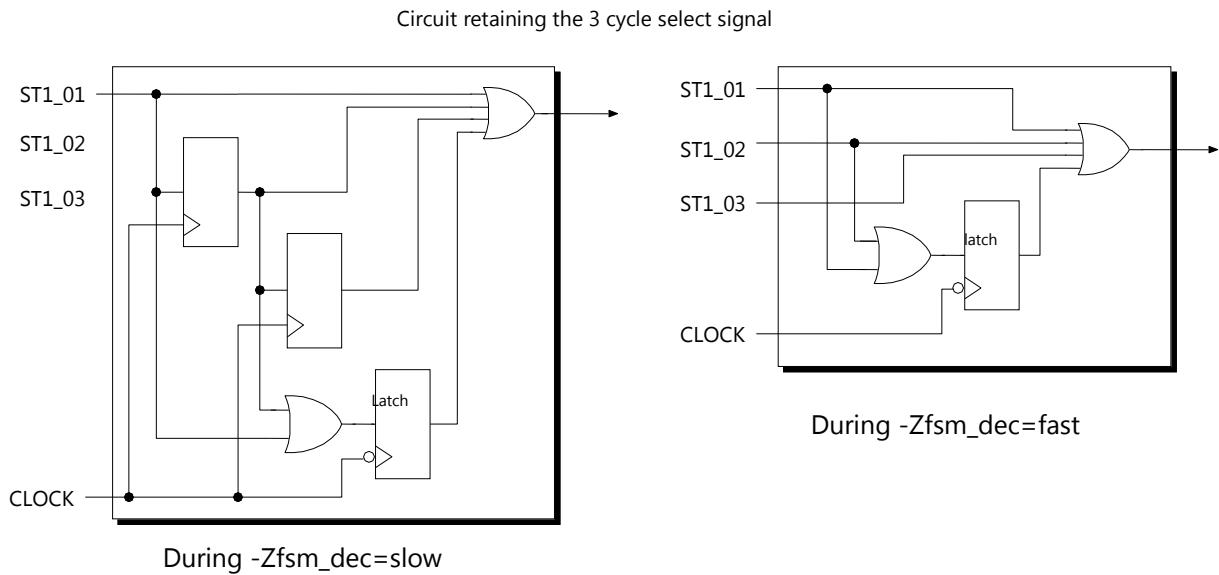


Figure 100: Circuit to retain select signal of multiplexer for 3 cycles

Restrictions:

- When the “-Zmulti_cycle_ope=reg” option is specified, there are cases where the specification of options “-SR” or specification of attribute “reg_bind” for input signals of multicycle functional unit is not valid.

26.3.3.2 Input Variable Directly Used in MultiCycle Functional Unit /Memory

Attribute	Description	Setting Target
/*Cyber multi_cycle_input=true*/	Input variable is input directly to multicycle operation /memory assuming that it is stored outside.	Input variable

An input to a multicycle functional unit / memory must be held throughout execution cycles.

In case an input variable is used in multicycle functional unit / memory, input data is referenced after storing it in register. Hence, it will cause one clock cycle delay in input.

In case it is ensured that input data is retained throughout execution cycle externally, the input data need not be retained in a register.

If the /* cyber multi_cycle_input=true */ attribute is specified for an input variable as shown below, it will be assumed that input data will be retained externally throughout execution cycles of multicycle functional unit/ memory. In addition, input data will be applied directly to multicycle functional unit/ memory without allocating it to a register first.

However, this attribute will not be valid in default case of “-zmulti_cycle_ope=reg”. Moreover, it will be first stored to a register and referenced afterwards in the generated circuit.

Description example:

```
in ter(0:8) /* Cyber multi_cycle_input = true */;
out reg(0:8) y;
process multi_ope() {
    y = x + 1;
}
```

Functional Unit Library File

```
#@LIB { multi_ope }
@FLIB{
    NAME add8
    KIND +
    BITWIDTH      8
    DELAY 3T
    AREA 400
}
#@END { multi_ope }
```

Functional unit count file

```
#@CNT { multi_ope }
@FCNT {
    NAME      add8
    LIMIT    1
}
#@END { multi_ope }
```

26.4 Arithmetic Macro Functional Unit

This section explains about the arithmetic macro functional unit.

26.4.1 Related attributes

Attribute	Description	Setting Target
/*Cyber arith_macro */	Specifies statement assumed to be arithmetic macro operation	Statement
/*Cyber arith_macro=name */	Specifies statement assumed to be arithmetic macro operation after specifying functional unit name	Statement

26.4.2 Overview

Arithmetic macro operation is the functionality in which the operational expression including the targeted multiple operations are synthesized as functional unit. High speed functional unit of multi input functional unit or compound function unit, etc. can be realized by using arithmetic macro functional unit. Particularly when Design Compiler Ultra of Synopsys Company is used, high efficiency functional unit can be mounted in circuit and the quality of circuit is expected to improve.

In the below example, operational expression specifying the attribute arith_macro is synthesized as one functional unit. By realizing multiple operations into one functional unit, the optimization effect of operation process is feasible for circuit which can be easily applied in logical synthesis without adjusting the resource allocation of behavioral synthesis process, further where arranging the hierarchy of generated circuit is not required.

```

in ter(0:8) a, b, c, d;
out reg(0:8) x;
process entry(){
    /* Cyber arith_macro */
    x = a + b + c + d;
}

```

26.4.3 Synthesis Method of Arithmetic Macro Functional Unit

The following specifications are required for performing synthesis by using arithmetic macro functional unit.

- arith_macro attribute specification for target operational expression
- Functional unit library (FLIB) file, Functional unit count (FCNT) file specification for Arithmetic macro functional unit

26.4.3.1 Attribute Specification of target Operational expression

In the target operational expression of arithmetic macro functional unit, it is necessary to specify the attribute arith_macro specifying the synthesis as a functional unit. This attribute is specified only for operational expression and it is not possible to specify in control statement, etc. of if statement etc.

The target operation type of arithmetic macro functional unit is the below operation and it is possible to target only in case of highest order of operational expression regarding comparison operation and shift operation.

- Addition
- Subtraction
- Multiplication
- Comparison operation
- Shift operation

Further, Synthesized functional unit can be specified by specifying the functional unit name by attribute arith_macro=name. Resource allocation can be controlled as per requirement through

this functionality. If functional unit library is auto generated (refer to **section 25**) after specifying functional unit name, then functional unit library file is generated with specified functional unit name.

26.4.3.2 FLIB file, FCNT file specification for Arithmetic Macro Functional Unit

In the synthesis done by using arithmetic macro functional unit, it is necessary to specify the functional unit library (FLIB) file and functional unit count (FCNT) file for arithmetic macro functional unit. While specifying these FLIB, FCNT files, option -lfl (+lfl), -lfc (+lfc) similar to specification of functional unit library file, functional unit count file for basic functional unit are used.

In the FLIB file for arithmetic macro functional unit, adequate information of operational expression can be specified in the item (KIND) specifying the functional unit type. Below, example of FLIB file for arithmetic macro functional unit is shown.

$u8=((u8+u8)+u8)+u8$ is specified in KIND and the addition result of unsigned 8 bit variable of 4 unit shows the functional unit generated by unsigned 8 bit. In the process of resource allocation, as structure of expression (priority order of operation) along with bit width or sign is also recognized with functional unit type, therefore attention is required while preparing FLIB.

Further, refer to **section 25** for the functionality of automatically generating FLIB, FCNT file for arithmetic macro functional unit.

```
#@VERSION{1.00}
#@LIB{entry}
#@KIND{ARITH_MACRO_OPERATOR}
#@CLK 1000
#@UNIT 1/100ns
@FLIB{
    NAME AMU8AAAAU8U8U8U8
    KIND u8=((u8+u8)+u8)+u8
    DELAY 100
    AREA 150
}
#@END{entry}
```

26.4.4 Points of Concern

- This function assumes the optimization of Design Compiler Ultra of Synopsys Company. For this purpose, it is recommended to specify YES in -dc_ultra option at the time of functional unit library proposal.
- In case output language is Verilog-HDL, then it is possible to specify verilog 1995 format, 2001 format through option -signed_fu. When this option is specified, it is necessary to specify the same option even at the time of functional unit library proposal.
- The following message is output when this function is used.

I_BT4091: Below expression is assumed to be arithmetic macro functional unit of AMU8AAAAU8U8U8U8(u8=((u8+u8)+u8)+u8)
[Source line]
8(foo.bdl): x = a + b + c + d;

26.5 Summary

This topic covered the following:

- A multicycle functional unit is a functional unit that takes more than one clock cycle to perform an operation.
- In manual scheduling, multicycle functional unit/memory can be directly specified using continuous assignment statement (::=).
- In automatic scheduling, a multicycle functional unit can be specified using the functional unit constraint (LMT/FLIB) file.
- Multicycle memory is specified using the memory constraint (MLMT) file in automatic scheduling mode.
- If a multiplexer is generated, the selection signal of multiplexer must be retained throughout execution cycle

27 Language Level Conversion Function

27.1 Overview

This section covers the following:

- This section explains the following functionalities regarding language level conversion
 - Invariant expressions in loops
 - Common sub-expressions elimination
 - Functionality to reduce the stages in operation tree

27.2 Invariant Expression in Loops

This sub-section explains about functionality that is responsible for transfer of invariant expressions from inside a loop to outside the loop. However, this functionality is available only for automatic scheduling mode.

27.2.1 Related Options

Option	Description
-oL	Transfer invariant expressions outside the loop (available only in automatic scheduling mode)
-o-L	Do not transfer invariant expressions outside the loop (default)
-Zopt_loop_lvl=#	Maximum number of loop hierarchies while transferring invariant expression outside the loop, is set to #.
-Zopt_loop_lvl=A	Do not set any maximum limit on number of loop hierarchies while transferring invariant expression outside the loop, such that invariant expressions are transferred outside the loops to the extent possible.

27.2.2 Description

Variables that are not assigned inside a loop are not going to be modified in the loop. Therefore, any operation involving only these unassigned variables will always return the same result during loop repetitions. Hence, loop can be optimized by transferring these operations outside the loop, as it would lead to lesser number of operations inside the loop. This conversion can be enabled by specifying the option “-oL”.

Input description	Transferring invariant expression inside the loop
--------------------------	--

```
in ter(0:1) flag;
in ter(0:8) a,b;
out reg(0:8) c,d;
process example1() {
    var(0:8) x,y;
    x = a;
    y = b;
    while( flag ){
        c = x + y;
        d = x - y;
    }
}
```

```
in ter(0:1) flag;
in ter(0:8) a,b;
out reg(0:8) c,d;
process example1() {
    var(0:8) x,y;
    var(0:8) OL_01,OL_02;
    x = a;
    y = b;
    OL_01 = x + y;
    OL_02 = x - y;
    while( flag ){
        c = OL_01;
        d = OL_02;
    }
}
```

In the case of specifying the “-oL” option, maximum loop count transferred will be converted as 1 loop. This implies that even if there is a nested hierarchy of two or more loop levels, invariant expression is only transferred outside of only one loop.

Input description	When maximum no. of loop steps are 1
--------------------------	---

```
in ter(0:1) flag1;
in ter(0:1) flag2;
in ter(0:8) a,b;
out reg(0:8) c,d;
process example2() {
    var(0:8) x,y;
    x = a;
    y = b;
    while( flag1 ){
        while( flag2 ){
            c = x + y;
        }
        d = x - y;
    }
}
```

```
in ter(0:1)
flag1,flag2;
in ter(0:8) a,b;
out reg(0:8) c,d;
process example2() {
    var(0:8) x,y;
    var(0:8)
OL_01,OL_02;
    x = a;
    y = b;
    OL_02 = x - y;
    while( flag1 ){
        OL_01 = x + y;
        while( flag2 ){
            c = OL_01;
        }
        d = OL_02;
    }
}
```

In maximum loop count conversion during this transfer, numbers of stages are specified with numbers in -Zopt_loop_lvl option. Or specify “A” in place of numbers when possible number of stages to be transferred is to be increased.

When "-Zopt_loop_lvl=2" or the "-Zopt_loop_lvl=A" is specified:

```

in ter(0:1) flag1,flag2;
in ter(0:8) a,b;
out reg(0:8) c,d;

process example2() {

    var(0:8) x,y;
    var(0:8) OL_01,OL_02;

    x = a;
    y = b;
    OL_02 = x - y;
    OL_01 = x + y;
    while( flag1 ){
        while( flag2 ){
            c = OL_01;
        }
        d = OL_02;
    }
}

```

27.3 Common Sub-Expressions Elimination

This section explains functionality to eliminate redundant common sub-expressions at language level.

27.3.1 Related Options

Option	Description
-oK	Eliminates redundant operations and array references at language level (default)
-o-K	Do not eliminate redundant operations and array references at language level
-Zopt_redundant_ope=ope_only	Restricts elimination of redundant operation and array reference at language level to elimination of operations only
-Zopt_redundant_ope=array_only	Restricts elimination of redundant operation and array reference at language level to elimination of array references only

Option	Description
-Zcse_limit=#	Increase restriction limit of number of redundant operations / array references for elimination by # times

27.3.2 Related Attributes

Attribute	Description	Settings Target
/*Cyber array_refer=asitis*/	Do not optimize redundant array references	signal array declaration
/*Cyber array_refer=optimize*/	Optimize redundant array references	signal array declaration
/*Cyber redundant_ope=asitis*/	Do not optimize redundant array references and operations	Expression element
/*Cyber redundant_ope=optimize*/	Optimize redundant array references and operations	Expression element

27.3.3 Description

If an operation with the same inputs is performed multiple times, the results of the operation do not change. Similarly, the result of "read" operation does not change if array values are read from the same address. In such cases, result of the first operations is parameterized intermediately. These operations can be reduced by replacing the subsequent operations in intermediate variable reference.

Input description

```

in ter(0:8) a,b;
out ter(0:8) c;
process example1() {
    var(0:8) x,y;
    mem(0:8) ary[256];
    x = a;
    y = b;
    if (ary[x] > x + y) {
        c = ary[x];
    } else {
        c = x + y;
    }
}

```

Elimination of common sub-expression

```

in ter(0:8) a,b;
out ter(0:8) c;
process example1() {
    var(0:8) x,y,z;
    mem(0:8) ary[256];
    var(0:8) M_01, M_02;
    x = a;
    y = b;
    M_01 = ary[x];
    M_02 = x + y;
    if (M_01 > M_02) {
        c = M_01;
    } else {
        c = M_02;
    }
}

```

When “-Zopt_redundant_ope=ope_only” option is specified, only redundant operations are considered for elimination. Similarly, when “-Zopt_redundant_ope=array_only” option is specified, only redundant array references are considered for elimination.

**-Zopt_redundant_ope=ope_only
specification**

```

in ter(0:8) a,b;
out ter(0:8) c;
process example1() {
    var(0:8) x,y,z;
    mem(0:8) ary[256];
    var(0:8) M_01;
    x = a;
    y = b;
    M_01 = x + y;
    if (ary[x] > M_01) {
        c = ary[x];
    } else {
        c = M_01;
    }
}

```

**-Zopt_redundant_ope=ary_only
specification**

```

in ter(0:8) a,b;
out ter(0:8) c;
process example1() {
    var(0:8) x,y,z;
    mem(0:8) ary[256];
    var(0:8) M_01;
    x = a;
    y = b;
    M_01 = ary[x];
    if (M_01 > x + y) {
        c = M_01;
    } else {
        c = x + y;
    }
}

```

Moreover, the following optimization targets can be controlled individually by using attributes:

- Do not optimize even when redundant reference for that array signal exists when `array_refer=asitis` attribute is specified.
- Optimize if redundant reference for that array signal exists when `array_refer=optimize` attribute is specified.
- Do not optimize even if that operation are redundant when `redundant_ope=asitis` attribute is specified.
- Optimize if that operation is redundant when `redundant_ope=optimize` attribute is specified.

Attribute specification is given preference over option specification. When contradicting attributes are specified for a signal array declaration and a reference element expression of that array, the attribute specified for reference element expression is given preference.

Relationship between the “`array_refer`” attribute, “`redundant_ope`” attribute, and options are summarized in the following table:

	<code>array_refer=asitis</code>	<code>array_refer=optimize</code>	Not specified during declaration
<code>redundant_ope=asitis</code>	As it is	As it is	As it is
<code>redundant_ope=optimize</code>	optimization	optimization	optimization
not specified during reference	As it is	optimization	depends on all the options

```

in ter(0:8) x;
out ter(0:8) y;
out ter(0:8) z;
out ter(0:8) w;
process main()
{
    mem(0:8) a[255] /* Cyber array_refer=asitis */;
    var(0:8) i,j;
    i = x;
    j = x;
    y = a[i];
    y = i + j;
    z = a[i] /* Cyber redundant_ope=optimize */;
    z = i + /* Cyber redundant_ope=asitis */ j;
    w = a[i];
}

```

If the description specified above is synthesized using the “-oK” option, then it is synthesized based upon relation between the attribute and the option. As in the case of test circuits, an attribute can be specified to read values only from array in spite of redundancy.

```

in ter(0:8) x;
out ter(0:8) y;
out ter(0:8) z;
out ter(0:8) w;

process main()
{
    mem(0:8) a[255];
    var(0:8) i,j;
    var(0:8) M_01;
    i = x;
    j = x;
    M_01 = a[i];
    y = M_01;
    y = i + j;
    z = M_01;
    z = i + j;
    w = a[i];
}

```

As elimination of common sub-expressions is time-consuming, there are restrictions on the number of expressions optimized.

The number of redundant expressions may exceed the designated limit in certain cases during synthesis, like when “for” loop is being unrolled or a function is synthesized using inline expansion.

If it is required to eliminate common sub-expressions, the restriction limit can be increased using the “–Zcse_limit” option. In this case, limit value is increased by the number of times of the value specified after '=' in the option.

```
%bdltran -Zcse_limit=2 -s -c1000 -oK abc.bdl
(Maximum limit of number of common sub-expressions for
elimination is increased by twice its original value)
```

27.4 Functionality to reduce the stages of operation tree

This section explains functionality that reduces stages of operation tree. This functionality is valid only in automatic scheduling mode.

27.4.1 Related option

Option	Description
-oI	Reduces the stage of operation tree
-o-I	Do not reduce the stage of operation tree (Default)

27.4.2 Related Attributes

Option	Description	Settings Target
/* Cyber thr_block=YES */	Reduces the stage of operation tree	Block, control statement, function definition
/* Cyber thr_block=NO */	Do not reduce the stage of operation tree	Block, control statement, function definition

27.4.3 Description

This functionality uses feature (association law and commutative law) of arithmetic operation, and converts the operation into a tree structure executable in parallel. At this time, optimum operation tree is generated by considering the delay in input of arithmetic operation or delay of functional unit. Reduction in the latency or delay of circuit is possible by using this function.

Input description

```

in ter(0:8) a, b, c, d;
out reg(0:8) z;

process example1(){
    z = a + b + c + d;
}

```

At the time of -oI specification (default)

```

/*operation having 3 stages*/
z = ((a + b) + c) + d;

```

At the time of -oI specification

```

/* operation having 2 stages
 */
z = (a + b) + (c + d);

```

By using attribute, optimization target can be individually controlled.

```

in ter(0:8) a, b, c, d;
out reg(0:8) z;

process example2(){
    var(0:8) t0, t1;

    /* Cyber thr_block = YES */
    {
        t0 = a + b;
        t1 = t0 + c;
        out0 = t1 + d;
    }
}

```

However, there is a limitation that attribute is not valid inside the function which is included in the block specifying the attribute.

27.4.4 Points to be noted

- Targeted operations are subtraction, multiplication, logical operations.
- The functionality which is included in operation tree will execute in parallel.
Therefore,
 - Latency or delay is reduced while circuit area tends to increase.

- If the operation listed below cannot be executed in parallel, sometimes latency is not improved.
 - * When operand of target operation is array reference, and that array is synthesized as register array or memory
 - * When operand of target operation includes operation, and there are few functional unit constraints for that operation, say, 1.
 - * When operand of target operation is input variable
- Operation tree, which ignores "(" specified in input description, is generated
- When this functionality is applicable, following message is displayed

```
I_BT8371: Stages of operation tree are reduced
[Source row]
6(foo.bdl): x = a + b + c + d;
```

- When this functionality is executed, the message given below gets generated in hint file (.tips), when target operation is included in description

```
H_BT9017: 4 or more variables are added
[Hint] If attribute -oI or thr_block is specified, operation
will get executed in parallel, and there is a possibility
that latency will disappear.
[Source row]
6(foo.c): z = a + b + c + d;
```

27.5 Bit Width Reduction Function for Variables

This section explains about the function for reducing bit width of the variables as per description.

27.5.1 Related Option

Option	Description
-Zbitw_opt=ALL	All variables are targeted for bit width reduction (default)
-Zbitw_opt=LOGIC_TYPE	Variables other than variables already specifying bit width, are targeted for bit width reduction.
-Zbitw_opt=INTERNAL	Variables other than already declared variables, are targeted for bit width

-Zbitw_opt=NO	All variables are not targeted for bit width reduction
-Zmem_bitw_opt	Array allocated in memory are targeted for bit width reduction (FPGA: default)
-Zmem_bitw_opt=NO	Array allocated in memory is not targeted for bit width reduction (ASIC: default)

27.5.2 Related Attributes

Attribute	Description	Settings Target
/* Cyber bitw_opt=YES */	Variables are targeted for bit width reduction	Variable
/* Cyber bitw_opt=NO */	Variables are not targeted for bit width reduction	Variable
/* Cyber value_range=#1 - #2 */	Specify the no input from input variable only for values from #1 to #2	Input variable, Bidirectional variable

27.5.3 Overview

This function statically analysis the valid bit width of variables described. It is a function to reduce the bit width. Specifically, as variable v is assigned only for constant number 0 and constant number 1 as shown below figure, int (signed 32 bit) at the time of declaration is not required and sufficient with unsigned 1 bit. Therefore, it a function to reduce variable v in unsigned 1 bit.

```
in ter(0:1) flag;
in ter(0:8) in1;
out reg(0:8) o;
process ex1(){
    int v;
    v = 0;
    while( c ){
        if( flag ){
            v = 1;
        }
    }
    if( v == 1 ){
        o = in1;
    }
}
```

27.5.4 Specification of Variable for Bit Width Reduction

This function can specify target, non target in variables separately by using option -Zbitw_opt, -Zmem_bitw_opt and attribute bitw_opt. However, following variables are excluded from bit width reduction without any relation with option, attribute.

- Input variable, output variable, bidirectional variable
- shared variable
- Variable referred or assigned in allstates
- Variable referred or assigned in all combined statement
- Array allocated in byte enabled memory

Regarding arrays, either of the -Zbitw_opt and -Zmem_bitw_opt is excluded when exemption of bit width reduction is specified. By default, variables other than array allocated in memory are target for bit width reduction.

When option -Zbitw_opt=LOGIC_TYPE is specified, then variables declared in ter, reg, var, mem are assumed to be bit width reduction target, and variables generated internally with variables declared in char, short, int, long, long long are assumed to be bit width reduction target. By using both ter, reg, var, mem and char, short, int, long, long long, variables declared becomes bit width target.

```

        /* During -Zbitw_opt=LOGIC_TYPE
specification*/
ter(0..0)      vt;      /* Excluded from bit width reduction*/
reg(0:4)       vr;      /*          "          */
mem(0:32)     vm[256]; /*          "          */
ter           vtl1;   /*          "          */
int            i;       /* Bit width reduction target*/
char           c;       /*          "          */
int reg       ir;      /*          "          */

```

Regarding SystemC, when option `-Zbitw_opt=LOGIC_TYPE` is specified, variables declared in `sc_int`, `sc_uint`, `sc_bigint`, `sc_bignum` are excluded from bit width reduction.

Variables generated internally are variables generated when multiple operations are performed in 1 statement as shown below.

```

var(0:8) a,b,c;
signed var(0:10) y;
y = a - b - c;

var(0:8)      a,b,c;
signed var(0:10) y;
signed var(0:9) tmp;
tmp = a - b;
y = tmp - c;

```

In case option `-Zbitw_opt=INTERNAL` is specified, then variables declared as per description reduces only for variables generated internally as exclusion. When option `-Zbitw_opt=NO` is specified, bit width reduction of variables also including internally generated variables is not performed.

27.5.5 Input Range Specification of Input Variable

In input variable, bidirectional variable, value_range attribute specifying the scope of input value exists.

If this attribute is specified, then that variable inputs only the value of that scope and bit width reduction is performed. In the below example, variable `i1` of 4 bit indicates that only the value, from 1 to 12 within 15 from 0 is represented by 4 bit is entered.

```
in ter(0:4) i1 /* Cyber value_range = 1 - 12 */;
```

27.5.6 Display of variables reduced

Reduced variables are output in the below format in Register allocation information file (.REG). Register allocation information file is not output by default. If –OR option or –OX option are added then it is output.

```
----- list of shrink variable -----
original: (s32) a_1 /* foo.c:3 */
    --> (s8) a_11
    --> (u8) a_12
-----
Sign before
reduced and
bit width      Variable name      Related line number
↓              ↓                      ↓
original: (s32) a_1                  /* foo.c:3 */
    --> (s8) a_11
    --> (u8) a_12
↑              ↑
Sign after
reduced and
bit width      Variable name
```

In above example, variable a_1 of signed 32 bit declared in 3rd row of file foo.c shows separation and reduction in both variable a_11 of signed 8 bit and variable a_12 of unsigned 8 bit.

However, the variable name displayed here is the variable name after internal conversion and structured variable etc. are displayed after unrolling. Hence, it is necessary to specify the original variable based on line number.

27.5.7 Restrictions

- As analysis is done statically, there are always cases of no reduction in expected bit width. At that time if bit width is specified in a portion of variable, then bit width of other variable, referring to that variable is reduced.
 - Specifically in the below example, for loop rotates only 10 times, but in the bid width reduction function for variable, there is a limitation of bit width of variable j not reduced as it assumed that it rotates for infinite times because information of "rotation only for 10 times" is not used.

(However, reduction is done for variable i recognized as counter variable of for loop.)

```
in ter(0:8) in1,in2;
out reg(0:32) out1,out2;
process ex2(){
    int i,j,v;
    j = 0;
    /* Cyber unroll_times = 0 */
    for(i = 0; i < 10; i++) {
        j += in1 ;
        v = j + 1;
        out2 = v * in2;
    }
    out1 = i + j;
}
```

If variable j is explicitly declared in 11 bit, then variable v referring to variable j is automatically reduces to 12 bit.

- Even if target, non target for bit width optimization is specified separately in variable using option or attribute, there is an invalid restriction when specified variables cannot be used in constant propagation and variable propagation.
- With regards to parameter of function and return value of function, those functions are considered to be the target during inline expansion of function, goto conversion of function, but not reduced at the time of user-defined functional unit conversion.

27.6 Summary

This course covered the following:

- Any operation involving unassigned variables will return the same result during loop repetitions.
- The invariant expressions are operations that don't change their result during loop execution.
- An invariant expression would only be transferred outside a single level of loop when the “-oL” option is specified.
- The “-Zopt_loop_lvl” option can be specified to change the maximum levels of loop stages for transfer to a higher number.
- If an operation with same inputs is performed multiple times or if array values are read from same location, results of operation / “read” do not change, and hence there is some redundancy in specification.
- When “-Zopt_redundant _ope=ope_only” option is specified, only redundant operations are considered for elimination.
- Similarly, in case the “-Zopt_redundant_ope=array_only” option is specified, only redundant array references are considered for elimination.
- If the option “-oI” or the attribute “thr_block=YES” is specified, the stages in operation tree gets reduced.

28 Logical level conversion functionalities

28.1 Overview

This section covers the following:

- Merge the same conditions of a multiplexer
- Simplify the condition logic of a multiplexer
- Distribute the "ter" variable during manual scheduling
- Delete unreferenced registers
- Convert a variable whose assignment appears only in the initialized declaration to a constant.
- Generate the rewrite circuit of register value
- Optimize the redundant logical functions
- Flatten the multistage multiplexer

28.1.1 Related Options

Option	Description
-Zshare_nmux_cond	Merge select conditions of multiplexers
-Zsimplify_nmux_cond[=YES]	Simplify the select control logic of multiplexer (default)
-Zsimplify_nmux_cond=NO	Do not simplify the select control logic of multiplexer
-Zopt_ter_rename	In manual scheduling, distribute <i>ter</i> variable (Default)
-Zopt_ter_rename=NO	In manual scheduling, do not distribute 'ter' variable
-Zreg_asitis	Do not delete unreferenced registers
-Zreg_optimize	Delete unreferenced registers (Default).
-Zinit_reg=asitis	Do not convert the register variable, which has been assigned only in the initialized declaration to a constant
-Zinit_reg=optimize	Convert the register variable, which has been assigned only in initialized declaration, to a constant (Default)
-Zopt_reg_feedback[=YES]	Do not generate when rewrite circuit of register value is not required (default)

-Zopt_reg_feedback=NO	Generate as value is preserved even if rewrite circuit of register value is not required
-Zopt_redundant_logic[=YES]	Optimize redundant logical operations (Default)
-Zopt_redundant_logic=NO	Do not optimize redundant logical functions
-Zflatten_nmux=DP	Flatten the multistage multiplexer in data-path (default)
-Zflatten_nmux=FSM	Flatten the multistage multiplexer in FSM
-Zflatten_nmux=ALL	Flatten all multistage multiplexers
-Zflatten_nmux=NO	Do not flatten multistage multiplexer

28.1.2 Related Attributes

Attribute	Description	Settings Target
/* Cyber init_reg = asitis */	Do not convert the register variable, which has been assigned only in the initialized declaration to a constant	Register variable
/* Cyber init_reg = optimize */	Convert the register variable, which has been assigned only in initialized declaration to a constant	Register variable

28.2 Merging Similar Condition of Multiplexers

Option	Description
-Zshare_nmux_cond	Merge condition logic of multiplexers

Sometimes, different multiplexers use similar condition in a generated circuit. Hence, logical circuit is possible for generating similar condition for each multiplexer.

It is a function of merging logical gate generating condition into one when condition of each multiplexer is of entirely same in this function. (**Figure 101**)

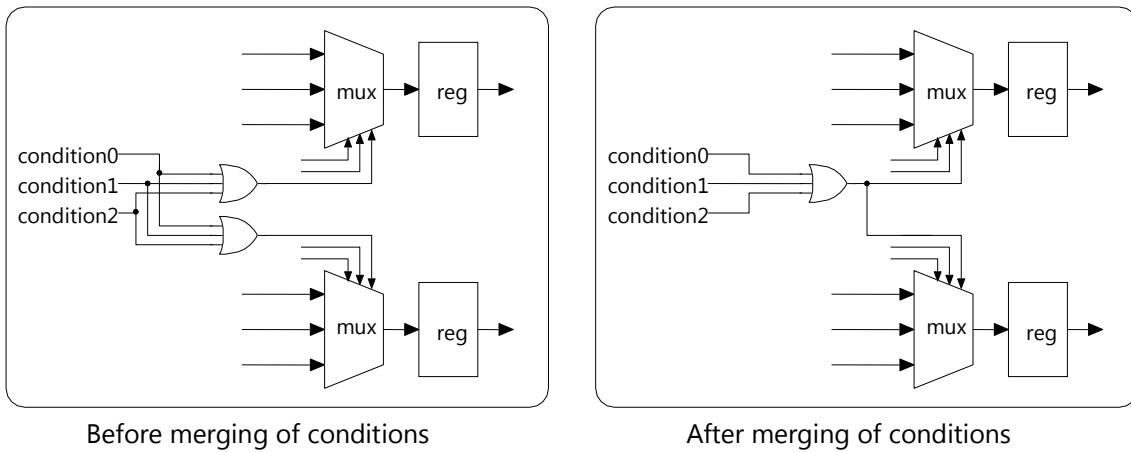


Figure 101: Merging same conditions of multiplexer

28.3 Simplification of Logic Condition of Multiplexers

Option	Description
-Zsimplify_nmux_cond	Simplify the selection control logic of multiplexer. (default)
-Zsimplify_nmux_cond=NO	Do not simplify the selection control logic of multiplexer

In some of the designs, multiplexers are placed immediately before functional units and have select logic based on AND of the same set of signals. In such cases, sometimes, circuit function is not altered even after the deletion of AND gates. Moreover, at the same time the logic stages are reduced for control of section multiplexer.

In this function, there is AND with similar signal in all conditional clause of single multiplexer and at the time of redundancy, simplification to reduce AND is achieved. Using this option, redundant AND gates can be eliminated as shown below (**Figure 102**).

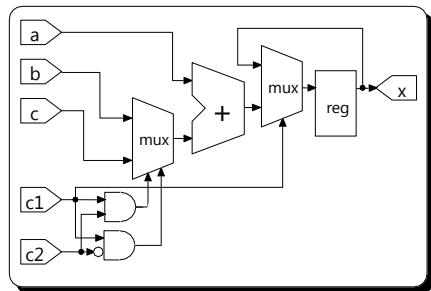
However, in some cases, area is increased due to the duplication of control.

Input Description

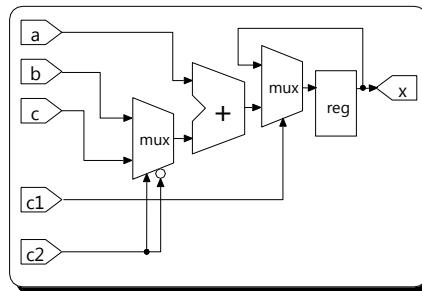
```

in ter  (0:8) a, b, c , ;
in ter  (0:1) c1, c2;
out reg (0:8) x;
process main(){
    if ( c1){
        if ( c2)  x = a + b;
        else      x = a + c;
    }
}

```



Synthesis Circuit



Synthesis Circuit after logical simplification

Figure 102: Conditional logic simplification of multiplexer.**28.4 Distribution of “ter variable” in Manual Scheduling**

Option	Description
-Zopt_ter_rename	Distribute ‘ter variable’ in manual scheduling (default)
-Zopt_ter_rename=NO	Do not distribute ‘ter variable’ in manual scheduling

In manual scheduling, when there are multiple assignments to a “ter” variable, each operation is synthesized as a circuit with selection through a multiplexer. As shown below, the “ter” variable “t1” has been assigned to three different values in different statements i.e. “a”, “b” and “c+d”. Therefore, it will be synthesized in three inputs multiplexer. In addition, the “ter” variable “t1” is referenced in two different statements as “t1-1” and “x = t1”. Therefore, multiplexer’s output will be passed on to a subtracter and another variable “x” (**Figure 103**).

The "ter" variables do not retain their values across the different states. So even if a "ter" variable is distributed to multiple "ter" variables across different states and conditions, its meaning is not lost. In the example shown below, there will be no logical changes even if "t1" is distributed to variables "t1_1" and "t1_2" in different states.

This distribution of "ter" variables can lead to the reduction in the number of multiplexer stages. Moreover, false loops and false paths interleaved with "ter" variables can be avoided. However there are cases when area increases due to replication of condition logic.

Input Description	Distribution of 'ter' variable
<pre>in ter(0:8) a,b,c,d; in ter(0:1) c1; out reg(0:8) x; process example5() { ter(0:8) t1; if(c1){ t1 = a; }else{ t1 = b; } x = t1 - 1; \$</pre>	<pre>in ter(0:8) a,b,c,d; in ter(0:1) c1; out reg(0:8) x; process example5() { ter(0:8) t1_1,t1_2; if(c1){ t1_1 = a; }else{ t1_1 = b; } x = t1_1 - 1; \$</pre>

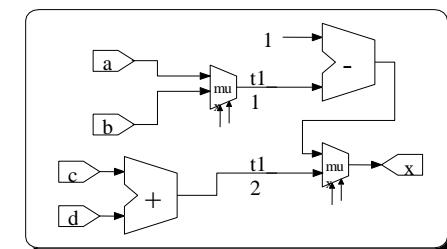
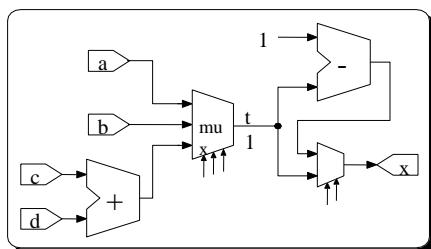


Figure 103: Distribution of 'ter' variable during manual scheduling

28.5 Deletion of Unreferenced Registers

Options	Description
-Zreg_optimize	Delete unreferenced registers (default).
-Zreg_asitis	Do not delete unreferenced registers.

The behavior of a circuit does not get altered even if the unreferenced registers are deleted. Therefore, it reduces the circuit area by deleting unreferenced registers.

However in testing, it may be desirable to retain unreferenced registers to help in observation, etc. In these cases, by specifying the “-Zreg_asitis” option, unreferenced registers are generated by retained circuit.

```

in ter(0:1) f;
in ter(0:8) a,b;
out reg(0:8) x;
process reg_optimize(){
    reg(0:8) r = 0;
    reg(0:8) r_tmp; /* unreferenced register*/
    if( f ){
        r = a + r;
    }else{
        r = b;
    }
    $x
    = r;
    r_tmp = r;
    $
}

```

However, there are following limitations.

- Unreferenced register array which are not developed are not deleted.
- Automatic scheduling deletes the register variable which are not supported and not referred.
- Even in manual scheduling, there are cases when register connected to input are not deleted.

28.6 Converting a register variable, having an assignment only in the initialized declaration, to a constant

Options	Description
-Zinit_reg=asitis	Do not convert the register variable, which has been assigned only in the initialized declaration to a register.
-Zinit_reg=optimize	Convert the register variable, which has been assigned only in the initialized declaration, to a constant (Default)

Attribute	Description	Settings Target
/* Cyber init_reg = asitis */	Convert the register variable, which has not been assigned only in the initialized declaration to a register.	Register variable
/* Cyber init_reg = optimize */	Convert the register variable, which has been assigned only in the initialized declaration to a constant	Register variable

In case the register that has been assigned only in the initialized declaration, will be converted to a constant which will have constant value or array by default, i.e. register will not be generated.

If option –Zinit_reg=asitis is specified, register variable is synthesized as register only. Further, in case it needs to be separately specified, specified variable is synthesized as register only by adding the attribute " /* Cyber init_reg=asitis */ ", to the variable.

In the following example, register variable 'r' has been assigned only in the initialized declaration, therefore, the reference part can be replaced with a constant. Further, register array M is an array having assignment only in the initialized declaration, and its subscript at the time of reference is not a constant, therefore, it is synthesized by the array logic implementation. When these variables are required to be synthesized as register, the same can be done by adding the attribute "/* Cyber init_reg=asitis */" or specifying option " –Zinit_reg=asitis " for these variables.

```

in ter(0:1) f;
in ter(0:8) a;
in ter(0:3) b;
out reg(0:8) x;

reg(0:8) r = 1;

reg(0:8) M[8] ={1,2,3,4,5,6,7,8};

process reg_optimize(){

var(0:8) v;

if( f ){
    v = a + r;
}else{
    v = M[b];
}
$ 
x = v;
$ 
}

```

However, at the time of auto scheduling mode, there is a limitation that there will be no effect on variables that are not array.

28.7 Generation of Rewrite Circuit of Register Value

Option	Description
-Zopt_reg_feedback[=YES]	Do not generate when rewrite circuit of register value is not required (default)
-Zopt_reg_feedback=NO	Generates for preserving the value even if rewrite circuit of register value is not required

In case the register value is referred only in the next assigned state, then rewrite circuit is not generated and circuit area is reduced because the rewrite circuit (feedback loop) preserving the register value is not required.

In case the following example is synthesized in manual scheduling, then rewrite circuit preserving the register value is not generated because register v1 can refer the assigned value only in next state. On the other hand, rewrite register is generated, as register v2 must retain the assigned value for 2 or more cycles.

```

in ter(0:1) f;
in ter(0:8) a1,a2;
out reg(0:8) b;
in ter(0:8) c1,c2;
out reg(0:8) d;
    reg(0:8) v1;
    reg(0:8) v2;
process entry()
{
    if( f ){
        v1 = a1;
        v2 = c1;
    }
    else {
        v1 = a2;
        v2 = c2;
    }
    $
    b = v1 + v2;
    $
    d = v2 + 10;
    $
}

```

In order to observe the value after stimulation, by specifying option -Zopt_reg_feedback=NO, a rewrite circuit is generated even when it is considered unnecessary for preserving those value which may not be needed.

28.8 Optimization of Redundant Logical Functions

Options	Description
-Zopt_redundant_logic	Optimize the redundant logical functions. (Default).
-Zopt_redundant_logic=NO	Do not optimize the redundant logical functions

A similar I/O logic operation used in multiple locations can be compiled at a single location. This type of optimization is generally performed by logical synthesis tools. However, it can be explicitly achieved during behavioral synthesis. Moreover, the estimation of area, net number, and pin pair number can be brought close to the estimation result of logical synthesis optimization.

Where there is redundant logical operation, logical function are optimized and compiled in 1. (Figure 104)

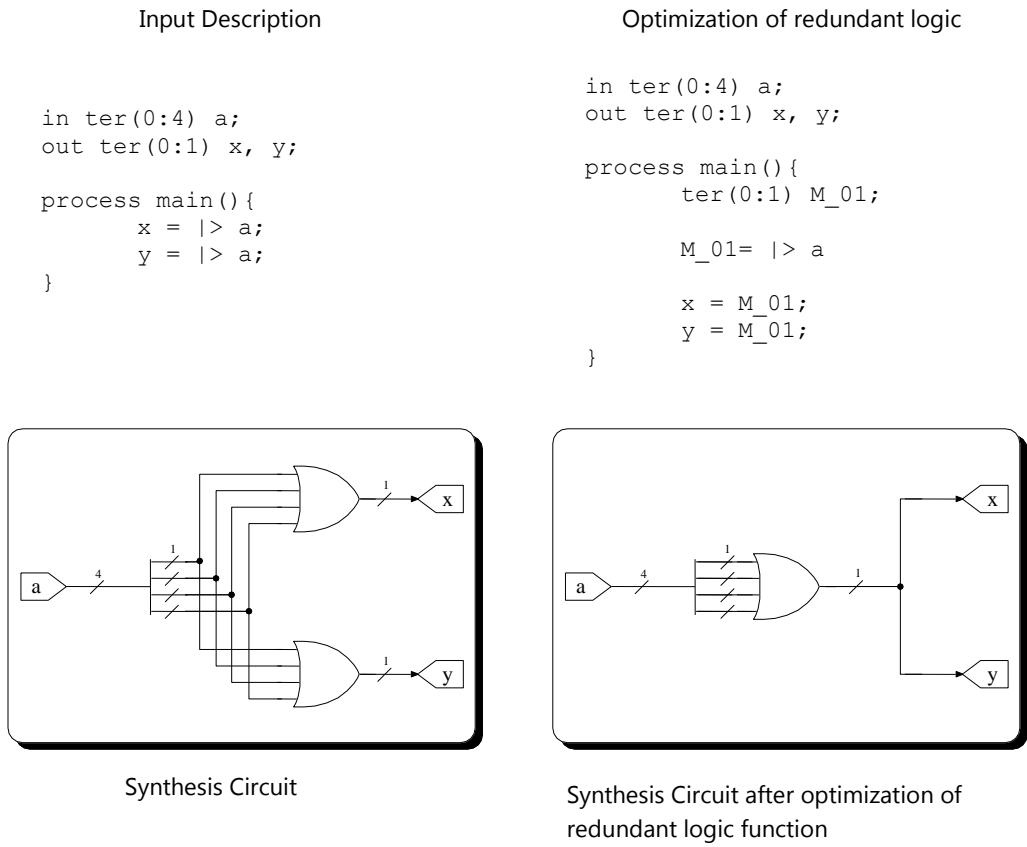


Figure 104: Optimization of redundant logic function

28.9 Flattening of multistage multiplexer

Options	Description
<code>-Zflatten_nmux=DP</code>	Flatten the multistage multiplexer in data path (default)
<code>-Zflatten_nmux=FSM</code>	Flatten the multistage multiplexer in FSM
<code>-Zflatten_nmux=ALL</code>	Flatten the all multistage multiplexers
<code>-Zflatten_nmux=NO</code>	Do not flatten multistage multiplexer

Sometimes the multiplexer generated by behavioral synthesis has multistage structure. In other words, synthesis is carried out in such a way that output of a certain multiplexer gets connected to data input of other multiplexer, and it is not used at any other place. In such cases, multiplexer can be restructured by merging 2 multiplexers into 1.

This functionality flattens the multistage multiplexer into a single stage multiplexer, when the output from the previous stage is used only for data input of the subsequent stage (**Figure 105**). By default, this functionality is applied only to data path portion because the application of this functionality to FSM portion can increase the area after logic synthesis. In case - Zflatten_nmux=ALL is specified, it will be applied to both data path portion and FSM portion.

Input Description

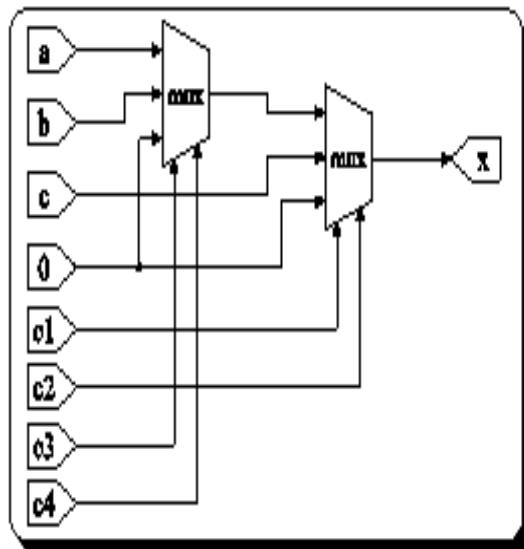
```
in ter (0:1) c1, c2, c3,c4;
in ter (0:4) a, b, c;
out ter (0:1) x;

process main () {
    x : := nmux {
        when (c1) :
            nmux {
                when (c3):a;
                when (c4):b;
                default   :0;
            };
        when (c2): c;
        default   : 0;
    };
}
```

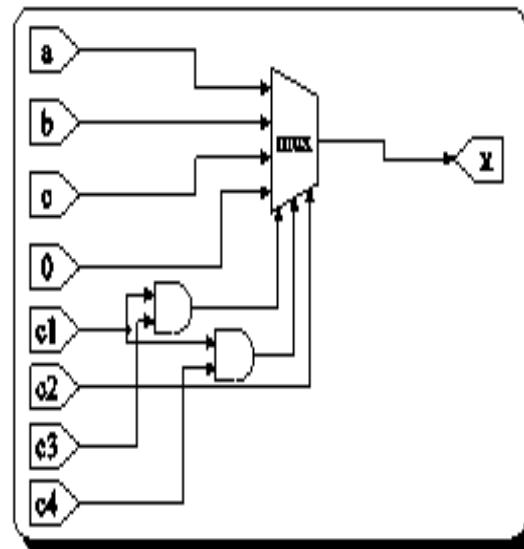
Optimization of redundant logic

```
in ter(0:1) c1,c2,c3,c4;
in ter(0:4) a, b, c;
out ter(0:1) x;

process main() {
    x : := nmux {
        when (c1&c3):a;
        when (c1&c4):b;
        when (c2)   :c;
        default      :0;
    };
}
```



Synthesis Circuit



Synthesis circuit after multiplexer flattening

Figure 105: Flattening of Multistage Multiplexer**28.10 Summary**

This section covered the following:

- A logical level conversion can generate common select condition for each multiplexer using the "-Zshare_nmux_cond" option.
- The redundant AND gates can be eliminated using the "-Zsimplify_nmux_cond" option.
- The "-Zreg_optimize" option can be used for the deletion of unreferenced registers.
- The "-Zreg_asitis" option can be used to avoid deletion of unreferenced registers.
- Deletion of unreferenced registers reduces the circuit area by deleting such registers.
- Using the "-Zopt_redundant_logic" option, redundant functions are optimized and compiled into a single function.

29 Combinatorial Loop

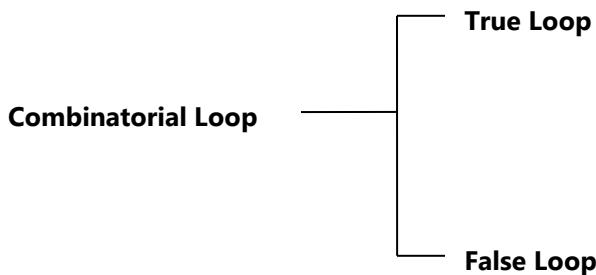
29.1 Overview

This section covers the following:

- Types of combinatorial loops
- Function to detect the "True" loops
- Combinatorial loop detection function
- Avoiding the "False" loop in resource allocation

This section describes the combinatorial loop included in synthesized circuit. A "combinatorial loop" is a loop which does not include any register in its structure.

Combinatorial loop is of two types. One is structurally combinatorial loop but no meaningful signal is propagated logically along the loop. This is known as the "False" loop. On the other hand, the "True" loop has a possibility of being logically activated.



A circuit may oscillate abnormally if it has a "True" loop in its structure.

The "False" loops do not cause any problem as far as the circuit behavior is concerned. However, logic synthesis tools and static delay analyzers may face problems due to their presence in the circuit.

In behavioral synthesis, the parameters to generate these combinatorial loops differ depending on the scheduling modes.

Manual scheduling: Cyber may generate both "True" and "False" loops based on the description.

Automatic scheduling: Cyber will not generate "True" loops in automatic scheduling mode. There is a possibility of "False" loops being generated.

In case a True loop is detected, it is required to modify the description and avoid the loop.

In case a False loop is detected, it can be avoided by specifying the scheduling option or resource allocation option. In case it is avoided by scheduling option, resources are saved and processing

steps are extended. In case it is avoided by specifying resource allocation option, resources to be used are increased without changing the processing steps. In case both are specified, avoidance by scheduling option is given a higher priority.

This section explains about the following combinatorial loop detection and avoiding functionalities.

- "True" loop detection in manual scheduling
- "False" loop, Combinatorial loop detection functionality
- "False" loop avoidance scheduling in automatic scheduling
- Loop avoidance resource allocation
- Command to detect combinatorial loop, false loop of synthesized circuit (refer **Appendix E**)

29.1.1 Related Options

Option	Description
-Wfalse_loop	Shows a warning on detecting a false loop, combinatorial loop (default)
-Wfalse_loop=NO	Does not detect any false loop, combinatorial loop
-Wfalse_loop_max=#	Set the maximum value of false loop, combinatorial loop detection to # units (default 10 units)
-Wexit	Exit if there exists any warning by following options: -Wattr, -Wfalse_loop, -Wall_loops
-SF	Scheduling which doesn't create false loop (only automatic scheduling mode)
-Af	Avoid false loop in resource allocation
-A-f	Does not avoid false loop in resource allocation (default) (Applicable only for allocation algorithm 2)
-At	Prioritize the delay reduction by allocation (FPGA: default) Does not prioritize delay reduction by

Option	Description
-A-t	allocation (ASIC: default) (Applicable only for allocation algorithm2)

29.2 “True” Loop Detection in Manual scheduling

In manual scheduling, if there is a True loop, the following error will be generated. In that case, it is required to modify the relevant part in description so as to prevent it from becoming a loop.

F_BT8915: There exists a combinatorial loop

However, this loop check has a limitation that True loops that pass through continuous assignment will not give an error. If loop avoiding resource allocation functionality of **section 29.5** is used for a True loop that passes through continuous assignment, it will be able to carry out the check.

If a loop that cannot be avoided by resource allocation is detected in functionality of loop avoidance resource allocation, the following error will be generated. In that case, it is necessary to modify the description of relevant part to be output by warning of W_BT7455 in error file (.err).

```

W_BT7455:Unavoidable loop is detected in allocation.
[Countermeasure] Change the description in a way that it
does not become a loop
    loop exist among the following
    -----
    [Source row]
    ....
    -----
F_BT7456: Unavoidable loop exists in allocation.
[Countermeasure] Refer to error file (.err) and change the
description in way that it does not become a loop

```

29.3 False loop, Combinatorial Loop Detection Function

Option	Description
-Wall_loops	Shows a warning on detecting false loop, combinatorial loop (default)
-Wall_loops=NO	Does not detect false loop, combinatorial loop
-Wfalse_loop_max=#	Change the maximum value of false loop, combinatorial loop detection count to # units (default 10 units)
-Wexit	Exit if there exists any warning by following options: -Wattr, -Wfalse_loop, -Wall_loops

False loop, combinatorial loop (true loop) is detected without any classification and warning is output.

```
W_BT5901: False loop/combinatorial loop exists
.
.
.

W_BT5902: False loop/combinatorial loop of 1 unit exists
[Action]Specify any of the option -SF, SSF or -Zassign=2 -Af when false loop
is to be cancelled
```

False loop, combinatorial loop detection is not performed when -Wfalse_loop=NO option is specified.

When -Wfalse_loop_max=# option is specified, maximum value of detected loop count can be changed. In case option is not specified, then maximum value of loop detection count is set at 10 unit. When maximum value is exceeded, then W_BT5903 warning is issued.

```
W_BT5903: Loop detection is aborted as maximum value of false
loop/combinatorial loop detection count exceeds 1 unit.
[Action]-Wfalse_loop_max is specified while changing the maximum value of
false loop/ combinatorial loop detection count
```

As false loop, combinatorial loop detection function are not supported in low order module declared in the defmod and user-defined functional unit, simulated error is generated because

these modules are detected after assuming that there is an existence of path which does not traverse through register for all input to all output. In that case, warning of W_BT5905 is generated in place of W_BT5901 and apart from W_BT5902, warning of W_BT5904 is generated.

W_BT5905: There is a possibility that false loop/combinatorial loop including low order hierarchy exists.

•
•
•

W_BT5904: There is a possibility that false loop/combinatorial loop including low order hierarchy of 1 unit exists.

Exists when false loop, combinatorial loop when option –Wexit is specified.

29.4 False Loop Avoidance Scheduling

Option	Description
-SF	Scheduling which doesn't create "False" loop (Applicable only for automatic scheduling)

Synthesizes the circuit by avoiding "False" loops in scheduling during automatic scheduling when this option is specified.

For example, when the description illustrated below in **Figure 106** is synthesized, usually the circuit is generated including the "False" loop by default.

The false loop is detected for the reason that for the condition when flag is true, the path generated is from (+ → -) whereas when flag is false, the path generated is from (- → +), but for both these conditions an adder and a subtractor are being used, thus seems to forming a loop.

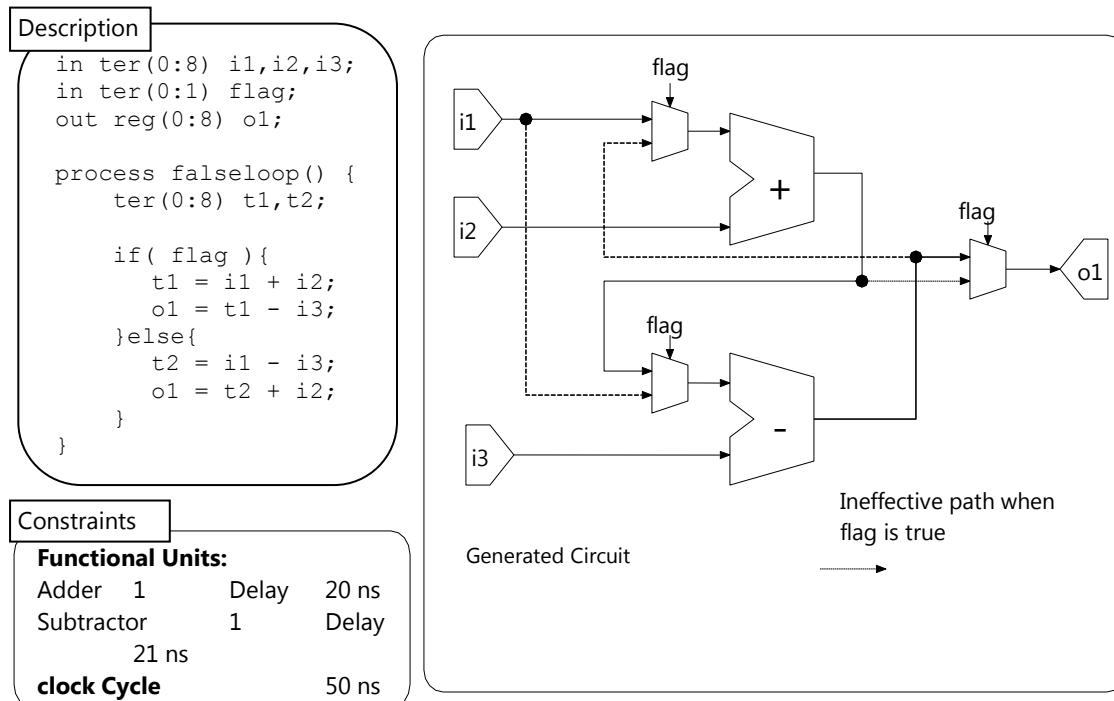


Figure 106: Circuit including false loop

Here, when scheduling is carried out to avoid the false loop, the scheduling is carried out such that the path towards $- \rightarrow +$ is not created in one condition provided the path $+ \rightarrow -$ already exists. For that, the operation of $o1 = t2 + i2$ gets scheduled in the next state and cancels the "False" loop. Two cycles are necessary in the operation.

It is not possible to carry out operation in one cycle without including the "False" loop with one adder. In this case, if there are two adders in the functional unit constraint file, then it is possible to carry out operation in one cycle without including a "False" loop.

29.5 Loop Avoidance in Resource Allocation

Option	Description
-Af -A-f	Avoid loops in resource allocation Does not avoid loops in resource allocation (default) (Applicable only for allocation algorithm 2)
-At -A-t	Prioritize the delay reduction by allocation Does not prioritize delay reduction by allocation (default) (Applicable only for allocation algorithm2)

When the “-Af” or “-At” option is specified, it avoids loops in resource allocation and synthesizes the circuit.

For example, when description illustrated in **Figure 106** is synthesized, the default circuit including loops is generated. When the “-Af” option is specified, the memory port allocation and functional unit allocation is rejected for generating loops in resource allocation. In case functional unit becomes insufficient, an error message is displayed as shown below.

The “operation” part of the error signifies unallocated operations, and the “function unit” specifies functional unit to which allocation of operation was attempted.

```
F_BT7458: Could not avoid loop by allocation
[countermeasure] Either change allocation or increase functional
unit/ memory port or remove-se, -MS5
operation:
15(loop.c): o1_1(0:8) = t1(0:8) - input(i3(0:8));
function unit: sub08(2)
```

In this situation, the subtractor sub08 specified in the functional unit constraint file should be increased (from one unit) to two units as instructed in order to avoid the false loop formation and enable the synthesis of the circuit.

Even on specifying –At option, functional unit allocation or memory port allocation that generates false loops in resource allocation is excluded. However, unlike –Af option, in case functional unit and decoder enabled register array are not sufficient, avoidance is carried out by increasing the resources equal to or more than constraint count.

Restrictions

- As far as the “-Af” and “-At” options are concerned, these are effective only in allocation algorithm 2. However, manual scheduling uses allocation algorithm 1 by default. Therefore, the “-Zassign=2” option should be used to specify the allocation algorithm 2. (Refer **section 20**)
- There is a need to specify basic library (BLIB) file for -At option. In case basic library (BLIB) file is not specified, -At option will become invalid.
- In case read data port and write data port of external memory are synthesized in bidirectional port, it is not possible to avoid loops using the “-Af” and “-At” option.

29.6 Summary

This section covered the following:

- Combinatorial loop is of two types: “False” loop and “True” loop.
- In manual scheduling, Cyber may generate both “True” and “False” loops based on the behavioral description.
- Cyber will not generate “True” loops in automatic scheduling mode.
- The “-Wall_loop” option is used to detect combinatorial loops.
- The “-Wfalse_loop” option detects “False” loop during automatic scheduling.
- When the “-Af” option is specified, it avoids “False” loop in resource allocation and synthesizes circuit.

30 Gated clock circuit

30.1 Overview

Gated clock circuit is one of the techniques of the low power consumption design. In a gated clock circuit, the clock of a register is disabled when the register is not updated. The section explains the functionality by which the gated clock circuit is added to a register.

- Gated clock circuit of latch type.
- Gated clock circuit of logic type (AND type, OR type).
- Gated clock circuit of estimated type in logical synthesis tool

30.1.1 Related options

Options at the time of the VHDL/Verilog-HDL output (type specification)

Option	Description
-Zgated_clock	Synthesizes register as gated clock
-Zgated_clock=YES	Synthesizes register as gated clock
-Zgated_clock=NO	Do not synthesize register as gated clock (Default)
-Zgated_clock=#	Synthesize register of # bits or more as gated clock

Following options exist in VHDL output command vhdlgen, Verilog-HDL output command verilogen.

Option	Description
-reg_feedback	Register value kept is represented by feedback format

30.1.2 Related Attributes

Attribute	Description	Settings Target
/* Cyber gated_clock=YES */	Specified variable allocated in register is synthesized as gated clock.	Variable
/* Cyber gated_clock=NO */	Specified variable allocated in register is not synthesized as gated clock.	Variable

30.2 Circuit for Gated Clock

By this functionality, power consumption can be decreased effectively when gated clock is applied in register by not overwriting the value of register.

In behavioral synthesis, for circuit area reduction, the value kept in register at the time of dont care is optimized such that the value of register is re-written without retaining rh value. (**Section 28.7**).

If the intention of using the gated clock which uses the option -Zgated_clock or attribute gated_clock during behavioral synthesis is indicated, the circuit is added in such a way the value of register is retained at the time of dont care. Hence, there are cases when frequency update of register decreases and changes to low power consumption; however on the other hand, there are cases when it leads to the increase of circuit area or delay.

This functionality is meant for adding the circuit for increasing the conditions which retains the value of register during behavioral synthesis and in order to implement gated clock circuit in reality, it is necessary to specify by logical synthesis tool, etc.

Gated clock functionality can be specified by variable unit by adding attribute gated_clock in variable. If it is specified like option -Zgated_clock=8, gated clock functionality is specified variables of 8 bit or more. In case option -Zgated_clock or -Zgated_clock=YES is specified, gated clock functionality is specified in variable of 4 bit or more which is similar with -Zgated_clock=4.

30.3 Output format of Gated Clock Circuit and Register

In RTL output, it is assumed as the format of maintaining the value of register despite of gated clock specification and logical synthesis tool is output by "clock enable format" which can be easily recognized as gated clock.

Circuit image and example of VHDL output and Verilog-HDL output is shown below.

Example: VHDL output (case format is multiplexer)

```

data_en <= ((ST1_05d or ST1_03d) or ST1_01d);
VHDLgen_Label1005 : process(ST1_05d,ST1_03d,ST1_01d,
                           data_t_c1,in3,in2,in1)
begin
    data_t_c1 <= ST1_05d & ST1_03d & ST1_01d;
    case data_t_c1 is
        when "100" =>
            data_t <= in3;
        when "010" =>
            data_t <= in2;
        when "001" =>
            data_t <= in1;
        when others =>
            data_t <= (others => 'X');
    end case;
end process;
VHDLgen_Label1006 :process(CLOCK, RESET)
begin
    if ( RESET = '1' ) then
        data <= "00000000";
    elsif ( CLOCK'event and CLOCK = '1' ) then
        if ( ( data_en = '1' ) ) then
            data <= data_t;
        end if;
    end if;
end process;

```

Example: Verilog-HDL output (case format is multiplexer)

```

always @ ( ST1_01d or in1 or ST1_03d or in2 or ST1_05d or in3 )
    case ( { ST1_05d , ST1_03d , ST1_01d } )
        3'b100 :
            data_t = in3 ;
        3'b010 :
            data_t = in2 ;
        3'b001 :
            data_t = in1 ;
        default :
            data_t = 8'hx ;
    endcase
assign data_en = ( ST1_05d | ST1_03d | ST1_01d ) ;
always @ ( posedge CLOCK or posedge RESET )
    if ( RESET )
        data <= 8'h00 ;
    else if ( data_en )
        data <= data_t ;

```

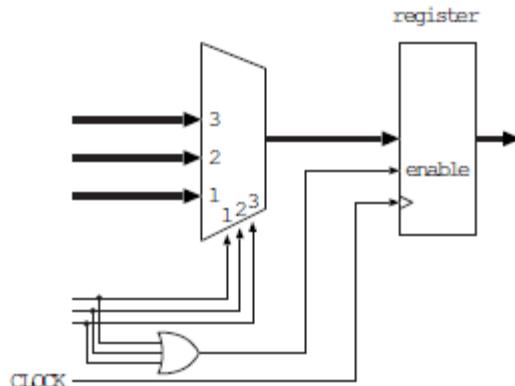


Figure 107: Register of clock enable format

As a format of maintaining the value of register, feedback format is provided as an option (option -reg_feedback). This format has been provided for compatibility with old version. Hence, below register which is output in clock enable format in old version even when this option is specified, is output in clock enable format.

- Register which is specified as gated clock at the time of behavioral synthesis
- Register stalled in stall_control attribute (Refer **section 23**)

Example: VHDL output (case format is multiplexer)

```
VHDLgen_Label1005 : process(ST1_05d,ST1_03d,ST1_01d,
                           data_t_c1,in3,in2,in1,data)
begin
  data_t_c1 <= ST1_05d & ST1_03d & ST1_01d;
  case data_t_c1 is
    when "100" =>
      data_t <= in3;
    when "010" =>
      data_t <= in2;
    when "001" =>
      data_t <= in1;
    when "000" =>
      data_t <= data;
    when others =>
      data_t <= (others => 'X');
  end case;
end process;
VHDLgen_Label1006 :process(CLOCK, RESET)
begin
  if ( RESET = '1' ) then
    data <= "00000000";
  elsif ( CLOCK'event and CLOCK = '1' ) then
    data <= data_t;
  end if;
end process;
```

Example: Verilog-HDL output (case format is multi plexer)

```

always @ ( data or ST1_01d or in1 or ST1_03d or in2 or ST1_05d
or in3 )
  case ( { ST1_05d , ST1_03d , ST1_01d } )
    3'b100 :
      data_t = in3 ;
    3'b010 :
      data_t = in2 ;
    3'b001 :
      data_t = in1 ;
    3'b000 :
      data_t = data ;
    default :
      data_t = 8'hx ;
  endcase
always @ ( posedge CLOCK or posedge RESET )
  if ( RESET )
    data <= 8'h00 ;
  else
    data <= data_t ;

```

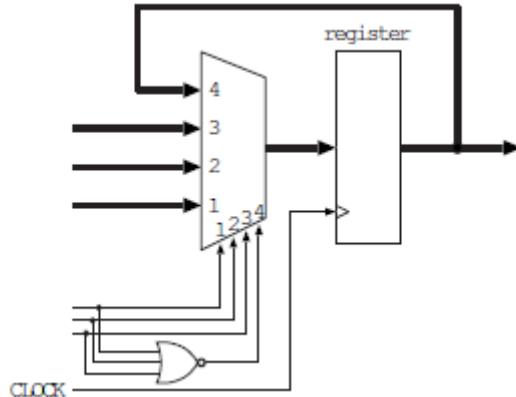


Figure 108: Register of feedback format

30.4 Summary

This section covered the following:

- Latch type gated clock is generated through a "AND" which takes clock and output of latch as an input. Latch which is used here takes control signal as an input and set signal to latch is clock.
- There are two types of gated clock of logic type:
 - One is the circuit in which AND gate is used. In this type of gated clock, the value of clock to register is maintained as "LOW".
 - The other one is the circuit in which OR gate is used. In this type of gated clock the value of clock to register is maintained as "HIGH".
- There are two specification methods.
 - One method is to add the "gated_clock" attribute.
 - The other method is specifying the option.

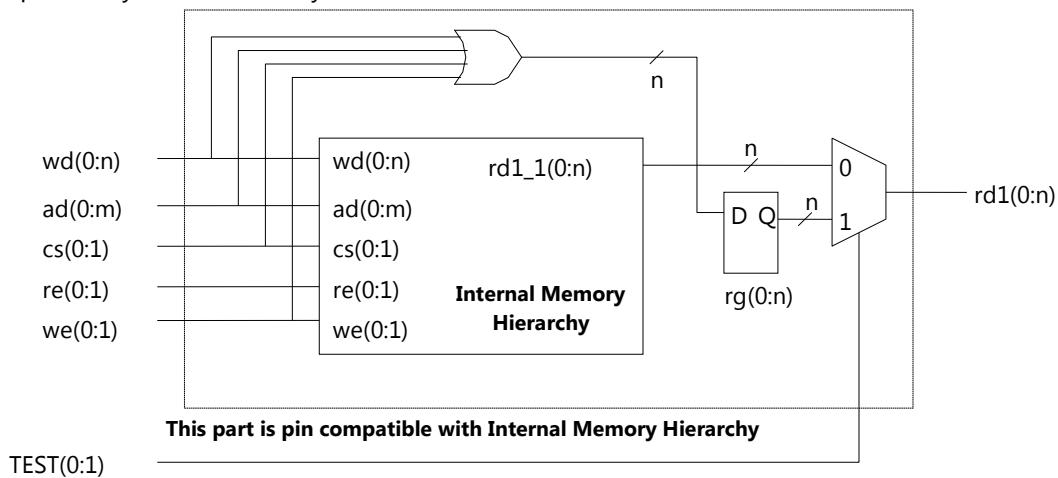
31 Generation of Wrapper for Bypassing Internal Memory

31.1 Functional Overview

This section covers the following:

- The options for generating wrappers
- The names prohibited for use as test port name

A wrapper is generated to bypass a black box (currently only for internal memory) inside a synthesized circuit hierarchy. It is used for inserting Built In Self Test (BIST). The bypass circuit is activated only when test port is asserted. At each port of internal memory, a bypass circuit is generated as shown below. When a bypass circuit is not generated, the part shown in dotted line in figure is pin compatible with internal memory hierarchy. However, this part is not made specifically into a hierarchy.



31.2 Related options

Option	Description
-ZZlogicbist	Generate a wrapper meant for bypassing internal memory with test port name as "TEST".
-ZZlogicbist= <i>name</i>	Generate a wrapper meant for bypassing internal memory with test port name as " <i>name</i> ".

When specifying the option above, Cyber generates wrapper for bypassing internal memory used for logic BIST. In case test port name is not specified, then "TEST" is used for the same.

31.3 Remarks

1. The following names are prohibited for use as test port name.
 - CLOCK and RESET
 - Input ports or output ports that appears in description or those automatically generated by Cyber.
2. For each port, when the total bit number of all input ports is less than the total bit number of all output ports, it results in an error. Therefore, one should be careful while using ROM.

31.4 Summary

This section covered the following:

- A wrapper is generated to bypass a black box to be used for inserting BIST.
- The "-ZZlogicbist" option generates a wrapper meant for bypassing internal memory used for logic BIST.
- CLOCK and RESET are prohibited as test port name.
- Input ports or output ports that appears in description or those automatically generated by Cyber are also prohibited.

32 Interrupt (“allstates” Statement)

32.1 Overview

This section covers the following about the “allstates” statement.

- Interrupt using allstates statement.
- Operations to execute in each cycle in automatic scheduling.
- Restrictions of allstates statement.

32.2 Interrupt Using the “allstates” Statement

Following conditions can be specified using the “allstates” statement:

- An operation that is to be forcibly triggered by a signal as an interrupt.
- When a soft reset needs to be specified.

Example 1 below shows the generation of a circuit where the control is transferred to the reset state when input variable “ad” is not 0. In Example 2, a circuit is generated where control is transferred to label “L1” when input variable “flag” is not 0.

Example 1

```
in ter(0:8) a, b, ad;
out ter(0:8) c;

allstates {
    if (ad != 0) {
        exit(1);
    }
}
process main() {
    reg x, y;
    :
    :
}
```

Example 2

```
in ter(0:1) flag;
in ter(0:8) a, b, ad;
out ter(0:8) c;
process main() {
    reg x, y;
    allstates {
        if (flag != 0) {
            goto L1;
        }
        :
        L1:
    }
}
```

32.3 Operations Executed at every Step in Automatic Scheduling

The continuous assignment, which indicates the step-wise execution operation, is not supported and generates an error in auto scheduling. Hence allstates statement is used in place of continuous assignment.

```
in ter(0:8) a;
out ter(0:8) b;
process main() {

    allstates {
        b = a;
    }
    :
    :
}
```

32.4 Restrictions of “allstates” Statement

- Only following conditional statements can be specified in “allstates” statement.
 - “if” statement
 - * “else” clause cannot be described
 - * Nesting of two or more hierarchies cannot be described
 - * Cannot be described in *for* loop
 - * When the conditions of multiple “if” statements are overlapped, behavior of circuit is unpredictable when all of them are “True” as according to the specification, all of them are evaluated at same time. (There is possibility of changes in future)

In following example, operation output is unpredictable when both ‘a’ and ‘b’ are “True”.

```
allstates {
    if( a ){
        goto L1;
    }
    if( b ){
        goto L2;
    }
}
```

Hence, it is necessary to clearly specify the priority like the following:

```
allstates {
    if( a ){
        goto L1;
    }
    if( !a && b ){
        goto L2;
    }
}
```

- "for" statement
 - * Only *for* statement which can be unrolled can be described
 - * Cannot be described in then/else section of *if* statement
 - * Loop control variable (loop counter) cannot be used outside *allstates* statement

```
in ter(0:8) a[4];
out ter(0:8) b[4];
process main() {
    int i;
    allstates {
        for (i = 0; i < 4; i++) {
            b[i] = a[i];
        }
    }
    :
    // i cannot be used outside
    allstates statement
    :
}
```

- "goto" statement and "exit" statement
 - * It can only be described in "then" construct of "if" statement
- "nmux", "dmux" and assignment statements
 - * Cannot be specified inside an "if" construct
 - * Concatenation (:) cannot be used for assignment destination.
- Function invocation
 - * Only function (void type function) having no return value is invoked
 - * Function invoked from *allstates* statement are inline expanded
 - * Contents of function invoked from *allstates* statement is treated as *allstates* statement
 - * Statement which can be described in function is similar to the item described in *allstates* statement
 - * Function invoked from *allstates* statement cannot be invoked outside *allstates* statement

```

struct foo {
    in ter(0:8) a, b;
    out ter(0:8) c;
} x, y;
in ter(0:8) i;
out ter(0:8) s, t;
process main() {
    allstates {
        func(&x, &s);
        func(&y, &t);
    }
    :
    // func() invoking is not possible outside
    allstates statement
    :
}
void func(struct foo *p, out ter(0:8) *o) {
    *o = p->a | p->b;
    p->c = i;
}

```

- Only the following syntax elements can be used in all the statements of "allstates" statement
 - Input output variables
 - Constants
 - ==, !=, and logical operations (&, |, ^, ~, !, &&, ||, ::, |>, &>, ..)
 - * In case ==, != is specified in functional unit constraint file, it is included in arithmetic operation
 - Arithmetic operations (+, -, *, /, %, >, ..)
 - * It cannot be used for functional call
 - * It cannot be used in condition part of "if" statement or "dmux" statement.
 - Pointer variable / Pointer reference
 - * Pointer variable and pointer indirect referencing can be described
 - * Pointer used inside *allstates* statement cannot be used outside *allstates* (It should be completed within *allstates* statement)
 - Variables
 - * See Note 1 below
 - * It cannot be used in condition part of "if" statement or "dmux" statement, and in condition part of "when" clause of "nmux" statement
- "allstates" statement cannot be specified in a function other than process function.
- In case a "goto" statement has been specified in "allstates" statement, label specified for jump destination should exist within the process function.
- Even when description contents are different, "allstates" statement cannot be specified at two locations.
- There is no guarantee that variable value after transition will be retained.

Note

Currently, in automatic scheduling, when "ter" variables and "var" variables are referenced and assigned only within "allstates" statement, they are treated as 'ter' type. When they are referenced and assigned both inside & outside "allstates" statement, they are treated as "reg" type.

As there is a possibility that above specification may undergo changes, it is recommended to use allstates statement in the following way:

- Use the "ter" type for variables referenced and assigned only inside "allstates" statement.
- Use "reg" type for variables referenced and assigned both inside and outside "allstates" statement.
- Do not use "var" type variables inside "allstates" statement.

32.5 Summary

This section covered the following:

- The "allstates" statement is used when an operation that is to be executed is forcibly triggered by a signal as an interrupt.
- The "allstates" statement is also used when a soft reset needs to be specified.
- In auto scheduling, the "allstates" statement is used instead of "continuous assignment" operator to specify an operation to be executed in every cycle.

33 FSM generation

33.1 Overview

This section explains the generation of FSM (Finite State Machine). It covers the following:

- Handling of invalid state.
- Partitioning of FSM.

33.1.1 Related Options

Option	Description
-Zstreg=add_dflt	Ensures that the reset state is the next state after invalid states (default)
-Zstreg=no_dflt	Next state after invalid state is not defined
-Zfsm_st	Generates FSM using a single register for all the states(ASIC: default)
-Zfsm_st=#	Generates FSM using multiple state registers, such that each register has a maximum of '#' states.
-Zfsm_st=1	Generates FSM by allocating 1-bit register for each state (one hot encoded).
-Zfsm_st=NO	Generates FSM with explicit state transition format. (FPGA: default)

33.2 Handling of Invalid States

Option	Description
-Zstreg=add_dflt	Ensures default next state of invalid states is reset state (default)
-Zstreg=no_dflt	Next state after invalid state is not defined

FSM is synthesized by binary encoding. In case of a binary encoding, when the state count is not a power of 2 (2, 4, 8...), state register will have certain values to which a state has not been allocated. These unallocated values are called invalid states. For example, an FSM with six states and a reset

state has a total of seven states. Therefore, the state register will be of 3-bit. Suppose value '0' is allocated to the reset state and values '1' to '6' are allocated to remaining six states. As a result, the value "7" is not allocated to any state and thereby this value becomes an invalid state. In an ideal circuit operation, transfer to an invalid state does not happen. However, considering the external disturbances like noise etc, FSM may be transferred to an invalid state. To take care of this, FSM is generated in such a way that it is transferred from an invalid state to a reset state for safety reasons. (**Figure 109**)

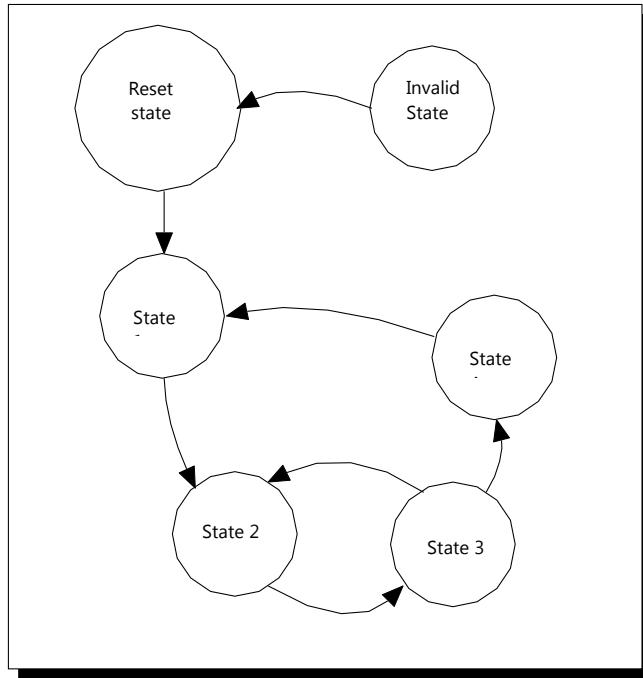


Figure 109: Transfer destination of invalid state of FSM

In above figure, a circuit is designed to have transfer of invalid states to reset state. However, to reduce the circuit area, the transfer of invalid state to reset state is possible without the circuit. When “-Zstreg=no_dflt” option is specified, an FSM is generated that cannot be transferred from invalid states to reset state. In this case, after state is transferred to invalid state then operation of FSM is not predictable.

33.3 Partition of State Register

Option	Description
-Zfsm_st	Generates FSM using a single register for all the states(ASIC: default)
-Zfsm_st="#	Generates FSM using multiple state registers, such that each register has a maximum of '#' states.

Option	Description
-Zfsm_st=1	Generates FSM by allocating 1-bit register for each state (one hot encoding).
-Zfsm_st=NO	Generates FSM with explicit state transition format. (FPGA: default)

The following three methods can be used for generating FSM using registers:

- FSM is generated using a single register for all the states by default. As the area required for a single register is small, thus circuit with smaller area is generated. However this causes increase in the decoder size and delay.
- FSM is generated using multiple registers using the “-Zfsm_st=#” option. In this method, the decoder size will be relatively smaller leading to lesser decode delay. However multiple state registers will require more register area.
- When FSM is generated using 1-bit register for each state, it becomes one hot encoded state machine where one register represents one state. In such a case, largest register area will be required. However, there will be no decode delay because there will be no requirement of decoder.

Figure 110 demonstrates all of above three methods. In the first case, FSM of 6 states is generated using a single 3-bit register. Second case is when FSM is partitioned into two 2-bit registers, each representing 3 states (-Zfsm_st=3). Final case is the one of hot encoding, where each state is represented by a 1-bit register (-Zfsm_st=1).

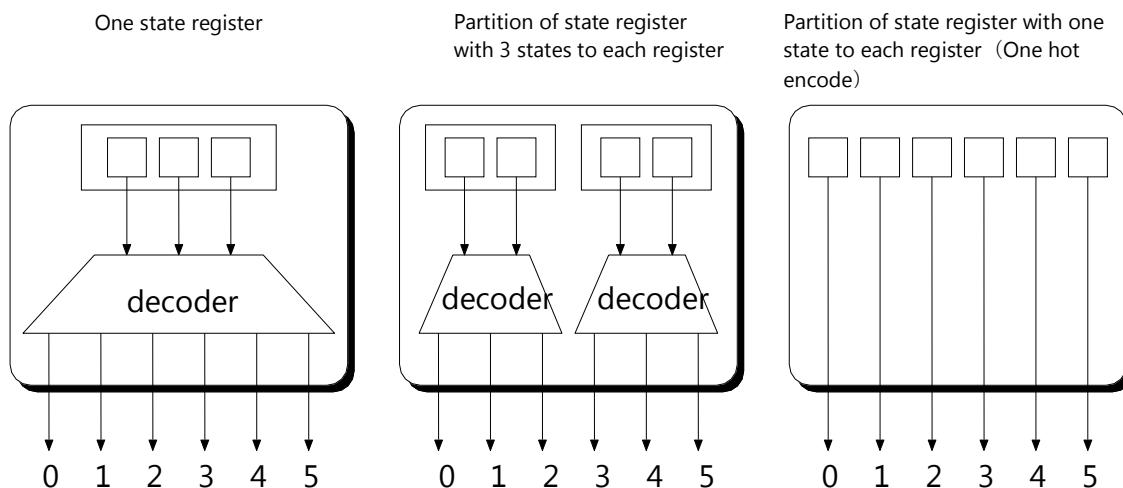


Figure 110: partitioning of state register

The “-Zfsm_st=NO” option outputs FSM with explicit state transition format. However, when this option is specified, the partitioning of state register is not done.

33.4 Summary

This section explained the following:

- FSM is synthesized by binary encoding
- When the state count is not a power of 2, the unallocated values are called invalid states.
- By default FSM is generated in such a way that control is transferred from an invalid state to a reset state for safety reason.
- When “-Zstreg=no_dflt” option is specified, an FSM is generated that cannot be transferred from invalid states to reset state.
- Following three methods may be used for generating the FSM using registers:
 - using a single register for all the states (default).
 - using multiple registers using the “-Zfsm_st=#” option.
 - using 1-bit register for each state.

34 Macro Option

34.1 Overview

This section describes the usage of the macro options.

34.2 Related Options

Option	Description
-MS#	Specifies the synthesis execution speed priority level, $1 \leq \# \leq 5$
-MA#	Specifies area priority level $1 \leq \# \leq 5$
-MPL#	Specifies cycle count priority level $1 \leq \# \leq 3$
-MPD#	Specifies delay priority level $1 \leq \# \leq 3$

34.3 Description

A macro option is a combination of various options.

Each macro option is a collection of options that will have a particular effect on the results. Higher the priority level specified, options that have appropriate effect on the result are included.

1. Macro option for prioritizing the synthesis execution speed

It is a macro option that reduces the execution time for synthesis. The execution time for synthesis decreases as the priority level increases. In this option, option of resource allocation "-AF" or false loop check option "-Wfasel_loop" etc. are included. However, the area and execution cycle counts may increase because the "macro" option restricts the scope of optimization.

2. Macro option for Area priority

This macro option decreases the circuit area. The circuit area decreases as the specification level increases. In this option, scheduling option "SS" etc. are included. In this case, execution cycle count may increase.

3. Macro option for Cycle count priority

This macro option decreases the execution cycle count (number of cycles).

Currently it is not supported.

4. Macro option for Delay priority

This macro option reduces the maximum delay. Currently it is also not supported.

NOTE: If the same types of macro options with different priority levels are specified, then only the last one will be preferred and considered as valid macro option. Hence, one needs to be careful while specifying macro options, as in case options that are part of macro option are specified after macro option, such options will be invalid.

34.4 Summary

This topic covered the following:

A macro option is a combination of various options.

- Macro option for synthesis execution speed priority decreases the synthesis time.
- Macro option for area priority decreases the circuit area.
- Macro option for Cycle count priority decrease the execution cycle count.
- Macro option for Delay priority reduces the maximum delay.

35 Dump Output

35.1 Overview

This section explains the “dump” attributes that is specified in bdl code to display a variable’s value during simulation.

35.1.1 Related Attributes

Attributes	Description	Target Objects
/*Cyber dump = {"format", arg} */	Display the value of specified variable in the simulation.	Expression element, assignment point

35.2 Input Description

The “dump” attribute is used to display the value of a variable at an assignment or a reference during simulation. The attribute can be specified at an assignment or a reference of a variable that is to be displayed and a “C” like format string (similar to argument of ‘printf’ function).

Ex:

```
x = a /* Cyber dump = {"%d", #} */;
y /* Cyber dump = {"comment1 %d comment2", #} */ = b;
```

Following are the features of the “dump” attribute:

- ‘#’ *arg* indicates the attributes adding the variable of a reference or an assignment. Currently, only one ‘#’ can be specified in *arg*, and operations like “#+1” in “*arg*” is not possible.
- Output conversion specifier which can be written in format is currently specified only %d, %x, and %o. Others such as %s, %c etc are not supported.
However, depending on the output language, attention is required as %d, %x, and %o are not supported.
- Within the format string, '\n', space, and '\t' are supported. However, depending on the output language, the format string might not be supported, so attention should be paid on this.
- ‘%%’ can’t be specified in format string.
- At present, the attribute does not support a case where expression (reference or assignment for which attribute is specified) is removed due to synthesis.

35.3 Handling of “dump” attribute in Verilog-HDL output

35.3.1 Related Options

Following are the options related to the “*dump*” attribute in the Verilog-HDL output command:

Options	Description
-dump_on	Enable the dump attribute (Default).
-dump_off	Ignore the dump attribute.
-dump_ref= <i>name</i>	Specification of dump description reference destination layer name.

35.3.2 Functional Description

When the “*dump*” attribute is specified in the input description, a separate module is generated in verilog that outputs the dump. This module is named as “*ProcessName_dump*” and is stored in file “*filename_E_DUMP.v*”.

When the “-dump_off” option is specified, the “*dump*” attribute will be ignored, and the module that outputs the dump is not generated.

To access signal of synthesized circuit in module that outputs the dump; appropriate hierarchy name is appended to signals. By default, it outputs with the instance name as “testbench.INST_0”. User can override this default instance name by using the “-dump_ref” option.

Input file example: foo.c

```
process process_name() {
    reg(0:8) r;
    .
    .
    if( f ){
        r/* Cyber dump = {"1:%d",#} */ = a;
    }
    .
    .
}
```

Output file example: foo_E_DUMP.v

```

module process_name_dump ( CLOCK ,delta_CLOCK );
    input          CLOCK ;
    input          delta_CLOCK ;
    always @ ( posedge CLOCK )
        begin
            $display("----CLOCK----");
            end
    always @ ( posedge delta_CLOCK )
        begin
            if ( testbench.INST_0.INST_2.U_01 )
                begin
                    #0 $display("1:%d(RHS)",
                                testbench.INST_0.INST_2.r);
                    @(posedge delta_CLOCK);
                    #0 $display("1:%d(LHS)",
                                testbench.INST_0.INST_2.r);
                end
            end
        endmodule

```

35.3.3 Example of dump output

The module which outputs the dump must input the “delta_CLOCK” signal in addition to clock signal. This signal takes the delta delay (after clock edge) and it is required to set value assigned to a register. The output dump will display both the values before and after the register is set on clock edge. The value before the clock edge trails “(RHS)” while the value after the clock edge trails “(LHS)”.

Example of dump output

```

#1: 0 (RHS)
# ----CLOCK----
# 1: 1 (LHS)
# ----CLOCK----
# ----CLOCK----
# 1: 1 (RHS)
# ----CLOCK----
# 1: 0 (LHS)

```

35.4 Handling of “dump” attribute in VHDL output

35.4.1 Option

- Enable/Disable options

When the "dump" attribute is specified in BDL, the following options can be used to enable/disable it during VHDL output:

- dump_on: enable the dump attribute (default).
- dump_off: ignore the dump attribute.
- Specifies the way to generate the standard output

The value of the generated VHDL output is dumped as the standard output value. However the way to output a value to standard output may differ from simulator to simulator. The VHDL output supports one of the following options:

- 1) /dev/stdout
 - 2) STD_OUTPUT
- dump_stdout{1|2}: Method to specify standard output during dump [1]
- 1: "/dev/stdout"
 - 2: "STD_OUTPUT"

35.4.2 Example of "dump" Output

```
---- CLOCK ----
:ST=00:
---- CLOCK ----
:ST=01:
---- CLOCK ----
:ST=10:
X = 0000FFFF
---- CLOCK ----
:ST=11:
```

35.4.3 Constraints

- "%d" can be used only in signal of unsigned values of 31-bit or less and signed values of 32-bit or less. This is due to the limitation of CONV_INTEGER () used to output 10-digit decimal numbers.
- Escape sequences like '\n', '\t' etc are ignored.
- Currently, it does not support clock edge specification. Values are always dumped on positive edge.
- Currently, the "dump" attribute attached with "dmux" is ignored.

35.5 Summary

This section covered the following:

- When the "dump" attribute is specified in the input description (BDL code), a separate module is generated in verilog/VHDL, that outputs the dump.
- The "dump" attribute is used to display a variable reference or assignment during simulation.

36 Error / Warning

36.1 Overview

This section explains about options to control the messages displayed in case of wrong description and settings as specified below. Besides, various types of functional options that check the description and generated circuits will also be discussed. This section covers the following:

- Message types and control option
- Functionality to check multiple assignments in manual scheduling
- Functionality to check 'nmux statement' assigned using continuous assignment statement
- Functionality to check bit width of operands of 'XOR function'
- Functionality to check bit width of allocated functional unit
- Functionality to check the error of attributes
- Functionality to check the part that behaves differently from C Language in manual scheduling
- Functionality to check paths violating delay specifications in manual scheduling
- Functionality to detect false loop, combinational loops
 - Refer to **section 29.3**
- Functionality to give warning about description influenced by specification changes in BDL version 3.0.

36.1.1 Related Options

Options	Explanations
-E0	Do not display anything including errors and warnings
-E1	Display errors (Default)
-E2	Display errors and strong warnings
-E3	Display errors and warnings
-E4	Display errors, warnings and important information messages
-E5	Display errors, warnings and all information messages

Options	Explanations
-EF0	Do not output errors in the file
-EF1	Output errors in the file
-EF2	Output errors and strong warning in the file
-EF3	Output errors and warnings in the file
-EF4	Output errors, warnings and important information messages in the file
-EF5	Output errors, warnings and all information messages in the file(Default)
-Wsuppress_warning=#[:#...]	Suppress display of specified number warning / information
-EW0	Relax standard for errors and make some errors as warnings
-EW1	Default error level(Default)
-EW2	Tighten standard for errors, and make some warnings as errors
-EJ	Output message in Japanese
-EE	Output message in English (By default, exist in environment variable)
-Wmulti_asgn=ignore	Do not check for multiple assignments
-Wmulti_asgn=careful	Treat suspicious multiple assignments as an error
-Wxor_bitw	When bit widths of 2 inputs of 'XOR operation' differ, treat it as an error and terminate synthesis.
-Wope_asgn_bitw	Warn, when allocated functional unit's bit width is bigger than the bit width of original operation.
-Wfalse_loop	Warn when a false loop, combinatorial loop is detected (Default)
-Wfalse_loop=NO	Do not check for false loop, combinatorial loop

Options	Explanations
-Wfalse_loop_max=#	Set the maximum value of detection count of false loop, combinatorial loop as # units (Default 10 units)
-Wattr -Wattr=error -Wattr=NO	Warn when attribute is erroneous (Default) Error when attribute is erroneous Do not check whether attribute is erroneous or not
-Was_C_lang	Check the part which behaves differently from C language in manual scheduling. Output only errors in the file and in case of duplicate errors, output only once.
-Was_C_lang=0	Same as -Was_C_lang.
-Was_C_lang=1	Output errors and warnings in a file. In case of duplicate errors/warnings, output them only once.
-Was_C_lang=2	Output errors and warnings in a file. In case of duplicate errors/warnings, output them all.
-Wcritical_path[=#]	Output paths where delay is more than #, to a file in manual scheduling
-Wexit	Terminate as error when there are 2 types of warnings (viz. -Wattr, -Wfalse_loop)

Options for BDL Input Command

Option	Explanation
-WBDL3.0=no_warning	Gives warning about the descriptions influenced by specification changes in BDL version 3.0 Does not give warning (default)
-WBDL3.0=warning	Gives warning
-WBDL3.0=error	Terminates after giving an error whenever a warning is encountered

36.1.2 Related Attributes

Attributes	Explanations	Setting Target
/* Cyber nmux_multi_asgn = check */	In case when destination variable of 'nmux' statement assigned using continuous assignment, is also assigned elsewhere, it will be treated as an error.	nmux statement
/* Cyber nmux_multi_asgn = ignore */	In case when destination variable of 'nmux' statement assigned using continuous assignment, is also assigned elsewhere, it will not be treated as an error.	nmux statement

36.2 Types of Error and Warning

As shown in the figure below, messages generated consist of four parts: [Type and Number], [Description], [Countermeasure for resolving problem] and [Related information]. Depending upon message, sometimes [Countermeasure for resolving problem] and [Related information] might not be included, and hence not present. Also, the format of [Related information] in messages may differ depending upon requirement.

As shown in **Table 3** below, there are four types of messages, and the format of [Type and Number] is different for these. Output to the standard error output device and error file can be

controlled using options for each message type. Currently there are two kind of warnings, [Strong Warning] and [Warning]. Also, some information will be displayed regardless of the [Information].

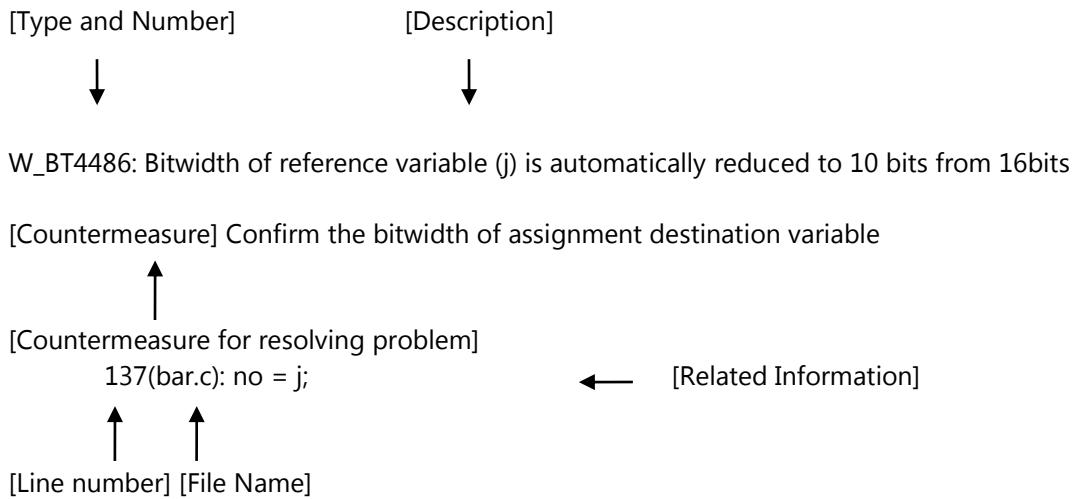


Table 3: Types of Messages

Type	Format	Explanation
Information	I_BT????	Display information about synthesis (including what and how)
Warning	W_BT????	Display points that may lead to problem in synthesis
Error	F_BT????	Display points that may lead to problem in synthesis and terminate synthesis
Program Error	X_BT????	Display occurrence of point besides specification

Option	Explanation
-Wsuppress_warning=# [:#...]	Suppress the display of warning/information of specified number

When too many warnings or information are being displayed, in order to suppress particular number warning or information, it can be done by specifying warning or information number that need to be suppressed with the '-Wsuppress_warning' option.

Example:

```
%bdltran foo.IFF -c2500 -E5 -Wsuppress_warning=4169:4703
```

36.3 Changing the message language

Message language can be changed by -EE, -EJ options. However, English and Japanese language can be automatically changed by setting the environment variables etc, without even specifying the option. The settings for each OS are summarized as follows:

- UNIX based OS

Message language is decided by the following environment variable, as per the locale structure in UNIX.

- LC_ALL
- LC_MESSAGES
- LANG

The sequence of priority of each environment variable is

LC_ALL > LC_MESSAGES > LANG

Value is specified in locale name in the below format.

```
language[_territory][.code-set][@modifier]
```

Display language is decided by the language part of locale name. For example, the classification will be as follows:

language part	Language
ja, Japanese	Japanese
en	English
Except those mentioned above	English

However, the set locale name differs by platform, therefore, caution is required.

In RedHat based Linux,

locale name can be confirmed by following command.

```
% locale -a
```

In case there is no specification for environment variable, it will be in English.

- Windows based OS

The language of displayed message can be changed by changing the language setting options.
(In case of Windows XP, [Control Panel] → [Regional and Language options])

In case Japanese language is set through user language settings, Japanese language notations will be displayed. If the language other than Japanese is specified, then English notations will be displayed.

36.4 Error Level Control

Options	Explanation
-EW0	Relax standard for errors and make some errors as warnings
-EW1	Default error level(Default)
-EW2	Tighten standard for errors, and make some warnings as errors

Some designers prefer to treat some warnings as errors to interrupt synthesis if these warnings are detected. In contrast, sometimes certain designers may not want to interrupt synthesis and let it continue. They prefer these errors to be treated as warnings.

To take care of these scenarios, above specified options may be used to treat certain warnings as errors or certain errors as warnings as per situation.

As of now, the level change can be done only for few errors and warnings using these options. There is a plan to include them gradually in future.

36.5 Functionality to Check Multiple Assignment in Manual Scheduling

Options	Explanations
-Wmulti_asgn=ignore	Do not check for suspicious multiple assignments
-Wmulti_asgn=careful	Treat suspicious multiple assignments as an error

In manual scheduling, parallel operations in one state can be specified as per language specification. Therefore, if multiple assignments are made for a variable in one state, the value of variable cannot be decided. This scenario is known as multiple assignments.

Example of Multi Assignments

```

in ter(0:1) f;
in ter(0:8) i1;
in ter(0:8) i2;
out reg(0:8) o;

process main() {
    reg(0:8) a;
    a = i1;
    if(f){
        a = i2;
    }$ 
    o = a;
    $ 
}

```

F_BT1509 : Signal 'a' has been assigned multiple times in step ST1_01
[Countermeasure] Rewrite so that multiple assignments don't occur.
11(test.bd1) : a(0:8) <- i2 (0:8);
9 (test.bd1) : a(0:8) <- i1 (0:8);

As shown in above case, when it is clear that multiple assignments have occurred, it will be treated as an error and synthesis will be terminated.

Example When Multiple Assignments are Suspected

```

in ter(0:1) f1, f2;
in ter(0:8) i1, i2;
out reg(0:8) o;

process main() {
    reg(0:8) a;
    a = i1;
    $ 
    if(f1){
        o = a;
    }
    if(f2){
        o = i2;
    }
}

```

W_BT1505: There is possibility that signal '0_1' has been multi assigned by step ST1_02.
[Countermeasure] Check whether it has been actually multi assigned. If it is a real multi assign, then it will rewrite.
15(test.bd1) : 0_1(0:8) <- i2 (0:8);
12(test.bd1) : 0_1(0:8) <- a (0:8);

The description that becomes multiple assignments by above mentioned condition cannot be judged statically, therefore, synthesis is carried out after giving a warning message. In case one wants to treat this situation as an error and terminate the synthesis, the '-Wmulti_asgn=careful' option can be specified.

F_BT1505: There is possibility that signal '0_1' has been multi assigned in step ST1_02.

[Countermeasure] Check whether it has been actually multi assigned. If it is a real multi assign, then rewrite the description.

```
15(test.bd1) : 0_1(0:8) <- i2 (0:8);
12(test.bd1) : 0_1 (0:8) <- a (0:8);
```

Further, if one wants the warning for multiple assignment, when it cannot be cancelled statically not to be output, it can be done by specifying the '-Wmulti_asgn=ignore' option.

Restrictions

- In case of assignments to array elements, when subscripts are same, it will be treated as multiple assignments scenario. However, in case subscripts are different, it will not be treated as multiple assignments. Currently, arrays are out of scope of multiple assignments check. (However, when array subscripts are constant numbers and array variables are unrolled, multiple assignment checks will be run.)
- In case of the 'nmux' statement specification, if multiple states of 'when' construct are set, it will be a multiple assignment. However, it will be out of scope of multiple assignment check.
- In case the assignment of a variable is done in multiple bit parts in the same state, there may be a possibility of multiple assignments even if bit positions are not overlapping. Especially, attention is required in case of array, as no warnings will be generated.

36.6 Functionality to check 'nmux' statement of continuous assignment

Attributes	Explanations	Setting Target
/* Cyber nmux_multi_asgn = check */	In case when destination variable of 'nmux' statement assigned using continuous assignment, is also assigned elsewhere, it will be treated as an error.	nmux statement
/* Cyber nmux_multi_asgn = ignore */	In case when destination variable of 'nmux' statement assigned using continuous assignment, is also assigned elsewhere, it will not be treated as an error.	nmux statement

In manual scheduling, 'nmux' and 'dmux' statements of continuous assignment (::=) has presently become specification as shown below:

When the destination variable of 'nmux' statement is present at assignment destination in another assignment statement which is not a "continuous assignment" statement, then these assignments are executed if none of conditional clauses of 'nmux' and 'dmux' statement is true. In addition, in case no conditional clause is true and no assignment statement other than the "continuous assignment" assignment exists, the 'default' clause is used for assignment.

As shown in example below, during step '1' state if the conditions 'c1' or 'c2' are not true, the value of '1' is assigned to the variable 'r'. In step '1' state, when either condition 'c1' or 'c2' is true then the following can occur:

- In case c1 is true, value 'd1' is assigned to variable 'r'.
- In case c2 is true, value 'd2' is assigned to variable 'r'.

At step '2' state, when 'c1' and 'c2' are not true, the value 'd3' is assigned to variable 'r' based on the "default" clause.

```

in ter (0:1) c1,c2;
in ter(0:8) d1,d2,d3;

out reg(0:8) o;

process main() {
    reg(0:8) r;
    r ::= nmux{
        when( c1 ) : d1;
        when( c2 ) : d2;
        default      : d3;
    };
    while(1){
        /* step 1*/
        r = 1;
        $
        /* step 2*/
        o = r;
        $
    }
}

```

If the above specified description is synthesized, the following warning will be generated. But if this is the intended behavior, then there is actually no problem.

```

W_BT1511: Signal 'r' is multi assigned with 'nmux(dmux)' of continuous
assignment at step ST1_00.
[Countermeasure] If it is intended, then there is no problem.
    18(test.bd1) : r(0:8) <- 1(0:8);
    10( test.bd1): r(0:8) ::= nmux

```

However, if other assignment to assignment destination variable of continuous statement nmux statement as shown above exists when not intended, then attribute /*Cyber nmux_multi_asgn =

check */ generating error is prepared. If this is added to nmux statement, the following error is generated when other assignment exists.

```

:
reg(0:8) r;
:
/* Cyber nmux_multi_asgn = check */
r ::= nmux {
    when( c1 ) : d1;
    when( c2 ) : d2;
    default     : d3;
};
:
F_BT1512: Signal 'r' with 'nmux (dmux)' using continuous assignment has
been assigned again at step ST1_00.
[Countermeasure] Correct the specification where signal 'r' in step st1_00
is assigned.
    19(test.bd1) : r(0:8) <- 1(0:8);
    11( test.bd1): r(0:8) ::= nmux

```

On other hand, if the attribute specification */* Cyber nmux_multi_asgn = ignore */* is used, no warning or error will be generated. If this is annexed along with the 'nmux' statement, there won't be any warning or error displayed even if any other assignment exist.

```

:
reg(0:8) r;
:
/* cyber nmux_multi_asgn = ignore */
r ::= nmux {
    when( c1 ) : d1;
    when( c2 ) : d2;
    default     : d3;
};
:

```

36.7 Bit Width Check Function of XOR Operation

Options	Explanations
-Wxor_bitw	When bit widths of 2 operand of 'XOR operation' differ, treat it as an error and terminate synthesis.

As shown below, when bit widths of 2 operands of XOR operation are different, *by default*, the operand with smaller bit width is expanded to match with the larger bit width of other operand. In case the above mentioned option is specified, different bit width of operands of XOR operation will be treated as an error.

```

var(0:6) a;
var(0:8) b;
var(0:8) c;
c = b ^ a;

```

F_BT1317: Input bit widths of XOR operation are different (8, 6, b (0:8) ^ a (0:6))
 [Workaround] Rewrite so that the bit widths of XOR operation inputs are equal.
 10(test.bd1): c(0:8) = b(0:8) ^ a(0:6);

36.8 Function for Checking the Bit Width of Allocated Functional unit

Option	Explanation
-Wope_asgn_bitw	Warn when the bit width of allocated functional unit is larger than the bit width of allocated functional unit.

As shown below, sometimes operation requiring a smaller bit width is allocated to larger bit width functional unit. This option displays a warning message for such cases. If this option is specified, then functional unit destination type, name and corresponding description will be displayed in a message as shown below.

Input Description

```

in ter(0:8) a;
in ter(0:8) b;
out reg(0:8)c;

process main() {
    c = a + b;
}

```

Functional Unit Library File

```

#@LIB { example1 }
@FLIB{
    NAME      add10
    KIND      +
    BITWIDTH 8
    DELAY    500
    AREA     250
}
#@END { example1 }

```

Functional Unit Count File

```

#@CNT { example1 }
@FCNT {
    NAME add10
    LIMIT 1
}
#@END { example1 }

```

W_BT5407: Bigger functional unit (ope=add10, bitw=10) than the required bit width is allocated to the operation.

[Countermeasure] Specify functional unit having appropriate bit width in the functional unit constraint file.

```
11(test.bdl): c_1(0:8) = a(0:8) + b(0:8);
```

36.9 Function to Check Attributes

Option	Explanation
-Wattr	Warns that the attribute is wrong (Default).
-Wattr=error	Shows error when attribute is wrong
-Wattr=NO	Does not execute check to find out whether the attribute is wrong or not.
-Wexit	Process terminates if an error is detected in attribute specification.

BDL input command 'bdlpars' may provide the following option:

Option	Explanation
-OB	Output the BDL file after parsing.

Sometimes, the desired results as per designer's intention are not obtained while using attributes. This is due to the specification of attribute at the wrong place. Some common mistakes are as follows:

- The special keyword "Cyber" or "cyber" is specified at the beginning in a comment to imply the attribute. However, if there is a mistake in the keywords, then it will be treated as a simple comment.
 - [Confirmation Method]
 It is possible to confirm that the attribute has been recognized by using the "-OB" option in the BDL input command 'bdlpars' and generating the BDL file after parsing.
- Incorrect attribute name

In the following example, the attribute is mentioned as 'unroll_time'. However, the correct name is 'unroll_times'.

Example /* Cyber unroll_time = all */
 for (i =0; i < 10; i++) {

- [Confirmation Method]

Following message is displayed as a warning in case attribute name which does not exist is specified:

W_BT1404: Wrong attribute name 'unroll_time' has been described.

Or the specified place is wrong.

[Countermeasure] Check the attribute specification of input description.

If "-Wattr=NO" is specified, then attribute check will not be done. In case "-Wexit" is specified or when "-Wattr=error" is specified, then attribution check will not result in a warning but in an error and thus synthesis abortion.

- Incorrect location specified

In the example mentioned below, the 'folding' attribute has been appended to the function definition part. Ideally, folding attribute must be appended to the loop statement.

Example

```
/* Cyber folding = 1 */
var(0:8) func( var(0:8) i) {
    :
}
```

- [Confirmation Method]

If the attribute is appended at incorrect location, then the following message will be displayed as warning:

W_BT1404: Wrong attribute name 'folding' has been described.

Or the specified place is wrong.

[Countermeasure] Check the attribute specification of input description.

If "-Wattr=NO" is specified, then attribute check will not be done. In case "-Wexit" or "-Wattr=error" is specified, then attribute check will not result in a warning but in an error and thus synthesis abortion.

LIMITATIONS

Depending on the attributes, there are cases when there is no effect on specified variable and statement, etc. but there is no assurity if the message is generated.

36.10 Functionality to Check the Portion of Behavior Deviating from C Language in Manual Scheduling

Options	Explanations
-Was_C_lang	While using manual scheduling this option checks the constructs that behave differently from C' language. In case of error duplications, error message is displayed only once.
-Was_C_lang=0	Same as '-Was_C_lang'.
-Was_C_lang=1	Outputs errors and warnings to a file. In case of duplicate errors, error message is displayed only once.
-Was_C_lang=2	Outputs errors and warnings to a file. In case of duplicate errors, error message is displayed every time.

The BDL check function has been developed for finding differences of behavioral execution of operations between 'C' language and BDL. These behavioral differences cause problem in the design flow at behavior level verification using 'C' simulation during behavioral synthesis in manual scheduling mode.

By using this function, the issues present in the description can be found out immediately and prevent unintentional synthesis of the circuit that is not intended by designer.

This section explains the usage and points to be taken care during the functional overview of BDL check function and usage.

36.10.1 Usage

The BDL check function has been incorporated in main program of Cyber, 'bdltran', and the BDL check function is enabled by appending the '-Was_C_lang' option⁷ with the 'bdltran' command line options.

By specifying this option, the BDL check is executed concurrently with synthesis and the check result is saved in a file with .Clog extension.

The specification method of this option is as follows:

⁷ This option is only valid during manual scheduling mode. Please note that this option may be missed in case of automatic scheduling

Format:

-Was_C_lang[=level]

Level: Check level

Explanation:

-Was_C_lang

-Was_C_lang=0 : Output only fatal errors. Duplicate messages will not be output.

-Was_C_lang=1 : Output fatal errors and warnings. Duplicate messages will not be output.

-Was_C_lang=2 : It will output fatal errors and warnings and will output duplicate messages also.

Example:

36.10.2 Function Overview

Following important points are checked in the BDL check function:

- When the 'ter' type variables are referred before assignment (F_BT1496)

References of the 'ter' variables refer to values assigned in the same cycle. In case no assignment occurs to a 'ter' variable, then the (default) initial value will be used. Hence, if a reference occurs in a cycle without assignment or when reference occurs prior to an assignment, behavior will be different from the 'C' language.

Example:

```
ter (0:8) terminal;
terminal = x;
$ 
y = terminal ; /* no assignment has occurred in the same step */
```

Description:

Values for the 'ter' type variables are initialized during start of every cycle. Therefore, in case the values are referenced before execution of the assignment, it will result in the generation of circuits that has behaviors different from 'C' language.

- In case the 'reg' type variables are referenced after assignment during the same cycle (F_BT1493)

Reference of the 'reg' variable implies the use of values assigned during the previous cycles If a reference of a reg type variable follows an assignment for the same reg type variable within a cycle, the behavior of the generated circuit will be different from 'C' language. This is because the in this case previous value will be used for referencing (register behavior in a

hardware circuit).

Example:

```
reg (0:8) register;
register = x;
y = register; /* Reference the value before update */
```

Description:

The "reg" type signal 'register' is not updated immediately after an assignment. So, if 'register' is referenced before completion of a cycle, a circuit will be generated having behavior different from 'C' language.

- In case the 'reg' type arrays are referenced during the same cycle of assignment but after completion of assignment. (W_BT1488)

The problem described in earlier point regarding the 'reg' variables will also occur for the 'reg' type arrays. In case of assignment/referencing of the same register array element, handling of the 'reg' type array is identical to the 'reg' type variable, which results in value update timing issues.

However, because array subscript is generally specified with a variable, it cannot be determined statically whether the same or different array elements are accessed. Cyber will generate a warning specifying that there is a chance of behavior being different from that of 'C' language.

Example:

```
reg(0:8) register[4];
register[i] = x;
y = register[j];
```

Description: When variable 'i' and 'j' are the same, register[i] and register[j] refer to the same register array location and as the value will not be updated immediately after assignment, so the behavior will be different from 'C' language.

36.10.3 BDL Rules for Using This Function

Conventions to be followed while using this BDL feature have been enumerated below. This feature works effectively by describing it as per the following conventions:

- Rewrite in a form, which does not use partial bits.

Example:

✗	x(0:4) = a; x(4:4) = b;
✓	x = a::b;

Description: If any partial bits are used, synthesized circuit will behave differently than it's in 'C' language, but presently this aspect is not checked.

36.10.4 Appendix

36.10.4.1 Collection of BDL Description Techniques

BDL description methods for better circuit synthesis in manual scheduling mode have been specified below.

- Conditional branching without full-case, should be restricted to minimum requirement.
Describe 'else' in case of 'if block', 'default case' in case of 'switch block' as much as possible. In all other cases, it is considered to be without full-case and in order to make full-case; circuits for other conditions are appended automatically.
Particularly, when the assignment for the 'ter variable' is not full-case, it is strongly recommended to make it to full-case because in most cases unintended circuit will be generated.

36.10.4.2 Collection of error messages generated in Clog file

The messages generated by BDL check feature are as follows:

- W_BT1471
This warning is generated when a false condition exists.
- W_BT1472
This warning is generated when an 'exit statement' exists.
Since the 'exit statement' implies return to the reset state in BDL, so when the process is not the main () function, the operation will be clearly different.
It will give warning when an 'exit statement' is detected.
- W_BT1473
This warning is generated when an undefined function has been used. Undefined functional part can be considered as an IP core. It is not known, how many cycles are required for the IP core to operate because the functional part is not defined. This function performs the check process with the assumption that it is a combinational circuit which is executed within 1 cycle.
- F_BT1481
This error is generated if code which cannot be executed exists after the 'break, continue, and return statement'.
- W_BT1483
This warning is generated when the 'return statement' is used at a place other than at the end of the function.

Example:

```
✗  if( x == 0) return a;
✓   if( x != 0) {
    ...
}
return a;
```

Description: According to the BDL language specification, the 'return statement' cannot be specified except at the end of the function definition.

A warning will be generated when a 'return statement' is detected except at the end of user defined function.

- **W_BT1484**

This warning is generated in case of multiple number of assignments are done to the same array location.

Example:

```
register[i] = x;
register[j] = y;
```

Description: In case the value of variables 'i' and 'j' are the same, then register[i] and register[j] will be the same register and multiple assignments are not allowed to the same register in BDL.

When multiple assignments are done to the same name reg type array, there is a possibility of multiple assignments. Therefore, a warning is generated in case of more than two assignments in a single cycle.

- **F_BT1486**

This error is generated in case ::= transfer type is used.

- **W_BT1487**

This warning is generated when the 'case block' does not end by a 'break statement'.

- **F_BT1488**

This error is generated when the same location of "reg type" array is referred in the same cycle after assignment.

- **F_BT1490**

This error is generated when the subscript of the 'ter type array' is not constant. According to the specification of the BDL language, usage of variable as a subscript of the 'ter type array' is not allowed.

- **F_BT1492**

This error is generated when multiple assignments are done to the same signal.

In BDL, multiple assignments to the same signal are not allowed.

In case multiple assignments are done in the same cycle, it will result in a fatal error.

- **F_BT1493**

This error is generated when the same 'reg variable' is referred in the same cycle after assignment.

- **W_BT1494**

This warning is generated when either the '&&' operator or the '||' operator is used.

In C language, the evaluation is executed in order from the left side and stopped when the conditions are confirmed. But in BDL, parallel evaluation is done. Since there is possibility of difference of behavior depending upon the description contents, warning output is given.

- **F_BT1496**

This error is generated when the 'ter variable' is referred before assignment.

- F_BT1497
This error is generated when a syntax that cannot be used in C language exists.
- W_BT1498
This warning is generated when '\$' is not specified immediately after 'goto statement'.
- W_BT1499
This warning is generated when the 'return statement' is used in a process function.
The 'return statement' in a process function implies the return to first state in BDL. Therefore, if the process is not main () or if the reset state exists, then the behavior will definitely differs.
A warning is generated when the 'return statement' is detected in the process function.

36.11 Functionality to check Delay Violation Path in Manual

Scheduling

Option	Explanation
-Wcritical_path[=#]	In manual scheduling, paths where delay exceeds the value of # are output to a file.

36.11.1 Overview

In contrast, to automatic scheduling, in manual scheduling, the circuit is generated as per the scheduling description specified by the user. The delays between the registers are not adjusted automatically to be within a clock cycle. Therefore, sometimes it may generate paths having delays more than the clock cycle. There is a high possibility that the problems of these paths may not be detected till the post processing of simulation results including delay and STA (Static timing analysis). This can result in turnaround time (TAT) getting significantly longer for correcting the problems.

This functionality is provided with the objective of identifying paths having larger delays (with a potential of violating delay specifications) during early stages of behavioral synthesis thereby preventing problems before they occur.

36.11.2 Usage

If '`-Wcritical_path=#`' is specified, then the paths having delays exceeding '#' are output to the file named *filename.CPI*. If only '`-Wcritical`' is specified and '`=#`' is omitted, then the paths having delay exceeding the clock cycle are output to the file named *filename.CPI*. In addition, it will display the following information:

```
I_BT8934:1critical path got detected.
[Countermeasure] For details, refer to filename.CPI.
```

Following information is generated for each path in the *filename.CPI*:

```
I_BT8931: 1st critical path is following:
Required time: 1000    arrival time: 1200    slack: -200 (VIOLATED)
```

Inc	Total	Expression
300	300	12(test.c): if ((unsigned ter(0:1)) (a(0:8) > 10(0:8)))
300	600	16(test.c): t(0:8) = (unsigned ter(0:8)) (b(0:8) + d(0:8));
600	1200	18(test.c): x_r(0:16) = (unsigned ter(0:16)) (t(0:8) * e(0:8));

required time	Overall delay of threshold path specified by '-Wcritical_path' option.
arrival time	Overall delay of path
slack	Value after deducting the overall delay of path from specified threshold.
Expression	Description corresponding to operation execution path
Inc	Delay of the operation
Total	Overall delay till that operation

36.11.3 Restrictions

- Option ‘-Wcritical_path’ is available only in manual scheduling mode.
- It does not support continuous assignment statement (Ex: a::= b+c). In the current version, path of the “continuous assignment” statement is ignored.
- Will give error when a loop is detected.
- Sometimes error may be generated when the number of registers is large.
- Sometimes, error may be generated when a redundant assignment is detected.

36.12 Functionality to generate Warning about Description influenced by Specification Changes in BDL version 3.0

Option for BDL input command

Option	Explanation
-WBDL3.0=no_warning	Gives warning about description influenced by specification changes in BDL version 3.0 Does not give warning (default)
-WBDL3.0=warning	Gives warning
-WBDL3.0=error	Terminate the program by giving an error when warning is detected

36.12.1 Overview

Given below are the specification changes done in BDL version 3.0. Due to these changes there is a possibility that the behavior may differ from existing description. This functionality aims to highlight the points, where behavior has changed in description in specification of BDL version 3.0 and to generate warnings.

If -WBDL3.0=warning is specified during BDL input, the following items are checked at the time of BDL input as well as behavioral synthesis. If -WBDL3.0=error is specified, error will be generated followed by termination of program even if a single warning is detected during BDL input or behavioral synthesis.

- Points to be checked during BDL input
 - Specification change when physical type is not specified
 - Specification change of assignment to bool type
- Points to be checked during behavioral synthesis
 - Specification change for evaluation sequence of && and ||
 - Specification change for “continue statement” when \$ is present at the end of the loop
 - Change in the execution sequence for \$ present at the last in the loop and resetting part of “for statement”
 - Specification change in return statement when \$ is present at the end of the function
 - Specification change for loop unrolling of “for statement” in manual scheduling mode
 - Specification change related to retention of variable value after process function completion

Regarding the specification changes given below in BDL version 3.0, an error or warning is always generated, regardless of specifying this option.

- As these items are discontinued, an error is generated during BDL input
 - Discontinuation of par statement

- Discontinuation of port type, ctlvar type, bus type
- Discontinuation of other labels
- Discontinuation of register transfer, port transfer, tristate transfer, latch transfer
- Discontinuation of formatting of const(0:4) i = 1
- As these items are planned to be discontinued, a warning is shown during BDL input
 - Specification change for parameters of process function
 - Specification change of output function
 - Discontinuation of timing descriptor ^
 - Unification of reset, clock formatting
- Due to specification changes, the description has no meaning for synthesis, therefore an error is generated during behavioral synthesis.
 - Specification change of "ter" variable when it is initialized at the declaration time in manual scheduling.

36.12.2 Warning List related to specification change in BDL3.0

- Warning during BDL input
 - Specification change when physical type is not specified

```
"foo.c", line 3: warning: physical type is not declared from BDL
3.0
          in/out/inout variable changed from ter type to var
          type
[Countermeasure] Explicitly declare by ter type <W2005>
```

- Specification change in assignment to bool type

```
"foo.c", line 13: Warning: The assignment to bool is converted
into True/False value from lowest bit of assignment variable from
BDL3.0 onwards <W2009>
```

- Specification change in output function

```
"foo.c", line 11: Warning: Output, where assignment destination
is output variable and not argument, is not supported <W2006>
"foo.c", line 13: Warning: Output, where output variable of
argument and assignment variable is different, are not supported
<W2007>
```

- Discontinuation of timing descriptor ^

"foo.c", line 13: Warning: Please delete timing descriptor as it's obsolete. <W3001>

- Unification of reset and clock formatting

"foo.c", line 1: Warning: in cannot be specified for clock/reset signal (clk) <W2812>
"foo.c", line 1: Warning: ter cannot be specified for clock/reset signal (clk) <W1002>

- Error during BDL input

- Discontinuation of par statement

"foo.c", line 13: Please specify in "for statement" as "par statement" has been discontinued < F9202>

- Discontinuation of port type, ctlvar type, bus type

"foo.c", line 1: ctlvar type is discontinued <F9203>
"foo.c", line 2: port type is discontinued <F9203>
"foo.c", line 3: bus type cannot be used <F9015>

- Discontinuation of other label

"foo.c", line 11: other label is discontinued therefore, re-write the description using default label <F9204>

- Discontinuation of register transfer, port transfer, tristate transfer, latch transfer

"foo.c", line 7: Transfer type "<-" has been discontinued therefore, write description by = <F3054>

- Discontinuation of formatting of const(0:4) i = 1

"foo.c", line 4: From BDL 3.0 onwards, bit specification cannot be done for const that does not have physical type
[Action] Declare the physical type and specify the bits
Example) const var(0:2) <F4066>

- Warning during behavioral synthesis

- Specification change for evaluation sequence of && and ||

W_BT4010: From BDL3.0 onwards, evaluation of right operand is executed by authencity of left operand of `&&`, `||`
 [Action] In order to operate like before, it should be modified in a way that value of right operand is referred after assigned to a variable

```
(example) if( a && b++ ){ } --> tmp = b++;
if( a && tmp ){ }
```

- Specification change in "continue" statement when \$ is present at the end of the loop

W_BT4002: From BDL3.0 onwards, changes are done such that cycle cannot be divided by continue, when timing descriptor \$ is present at the last of the loop
 [Action] Add timing descriptor above continue to synchronize with the previous behavior

- Specification change in "return statement" when \$ is present at the end of function

W_BT4003: From BDL3.0 onwards, changes are done so that cycle cannot be divided by return, when timing descriptor is present at the end of function
 [Action] Add timing descriptor \$ above the return to synchronize with the previous behavior

- Change of the execution sequence for \$ at the end of loop and resetting part of "for statement".

W_BT4004: From BDL3.0 onwards, the execution of for loop resetting part has been changed to cycle after the execution of the loop body end
 [Action] Shift the timing descriptor from the end of loop body to the end of resetting part.

- Specification change in loop unrolling of "for statement" in manual scheduling

W_BT4005: From BDL 3.0 onwards, changes are done such that ter type cannot be used as variable of loop counter of unrolled for loop
 [Action] Change the loop counter to var type

W_BT4006: From BDL3.0 onwards, changes are done such that timing descriptor is required between conditional part of initial setting part and resetting part when reg type is used for loop counter of unrolled for statement
 [Action] Change the loop counter to var type that does not require timing descriptor \$

- Specification change related to retention of value at the time of process function completion

W_BT4007: From BDL3.0 onwards, changes are done such that local variable (v) does not hold register value at the time of process function completion

[Action] Enclose the entire process function in infinite loop ,or define variable as global signal, to synchronize with the previous behavior

W_BT4008: From BDL 3.0 onwards, changes are done such that global variable, static variable (`main_v`) retains the value of register

[Action] Define variable as local variable of process function or exclude the static, to synchronize with the previous behavior.

- Error during behavioral synthesis
 - Specification change of "ter" variable when initialized at the declaration time in manual scheduling

F_BT4009: From BDL3.0 onwards, initialization during declaration of ter type variable is not possible

[Action] Change the physical type of variable from ter type to var type

- Error message when specification change is detected

F_BT4001: From BDL 3.0 onwards, description having different specification is detected

[Action] Refer to error file (.err), and change the description

36.12.3 Restrictions

- In case WBDL3.0=error is specified, there is a limitation that even if relevant description is detected, F_BT4001 error message is not generated, in case synthesis terminates due to some other error.

36.13 Functionality to warn any adverse effect of &&, ||

Option	Description
-Wandor_side_effect	Gives warning in case there are any adverse effect in right operand of &&,
-Wandor_side_effect=NO	Gives no warning in case there are any adverse effect in right operand of &&, (default)

Execution sequence of &&, || from BDL version 3.0 changes the right operand to non evaluated specification when the value is decided by left operand along with C language.

Hence, in case the operations, etc. having possibility of adverse effect in right operand is described, then the logic for controlling the execution of right operand can be created by left operand value.

If option -Wandor_side_effect is specified, then right operand of &&, || is checked and possibility, etc. for existence of adverse effect is also checked. In case there is a possibility of any adverse effect, then the following warning is displayed. By specifying this, it is possible to check whether possibility for adverse effect in right operand of &&, || without aiming is specified or not.

```
W_BT4250: Adverse effect in right operand of &&, ||
[Source line]
9(foo.c): if( a && v++ ) {
```

- Target behavioral description
 - Operation having adverse effect
 - ++(previous value/after value), --(previous value/after value)
 - =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
 - Function call
 - input/output function
 - Input variable (in ter/in reg), output variable (out ter), bidirectional variable (inout)

(Adverse effects are determined only when reference and input function of input variable, bidirectional variable generates valid signal in valid attribute etc.)
 - Array reference (as reference besides elements is not produced)
 - Division, Surplus calculation (as division, surplus through 0 is not performed)
 - Shift operation (as shift through false is not performed)

36.14 Summary

This topic contents can be summarized as follows:

- Error/Warning messages generated consists of four parts: [Type and Number], [Description], [Action for resolving problem] and [Related information].
- There are four types of messages in which the format of [Type and Number] is different.
- Some designers prefer to convert some warnings as errors so as to interrupt synthesis as soon as that warning is detected.
- In contrast, sometimes designers may not want to interrupt synthesis and let it continue even after some trivial errors are detected.
- If user wants to treat multiple assignment situation as an error and wants to terminate the synthesis, it can be done by specifying the '-Wmulti_asgn=careful' option.
- If one wants to disable multiple assignment check functionality, it can be done by specifying the '-Wmulti_asgn=ignore' option.
- While considering variable's bit width, if it is judged that the loop condition will remain true and result in an infinite loop, it will be treated as an error.

- The BDL check function has been developed for finding differences of behavioral execution of operations between 'C' language and BDL.
- The BDL check function has been incorporated in main program of Cyber, 'bdltran' and by appending the '-Was_C_lang' option to the 'bdltran' command line, the BDL check function can be enabled.
- In manual scheduling, the delays between the registers are not adjusted automatically to be within a clock cycle. Therefore, sometimes it may generate paths having delays more than the clock cycle.
- If '`-Wcritical_path=#`' is specified, then the paths having delays exceeding '#' are output to the file named *filename.CPI*.
- If only '`-Wcritical`' is specified and '`=#`' is omitted, then the paths having delay exceeding the clock cycle are output to the file named *filename.CPI*.
- In the file named *filename.CPI*, BDL code lines that are involved in critical path are displayed. Here one line is displayed for every BDL code line that shows timing delays and expression at that point.

37 Information File

37.1 Overview

This section describes various information files generated and their contents. This section covers the following:

1. List types of output information files
2. The features of Summary File (.SUMM)
3. The features of Error File (.err)
4. The features of Log File (.LOG)
5. The features of Synthesis information files (.CSV format)
6. The features of BDL File for analysis (.BDL)
7. The features of information file (.INF)
8. Memory and register array usage with usage tables
9. Circuit quality report
10. Text version circuit quality report (.QOR)
11. Options concerning the circuit quality report
12. State transition diagram file (.dty) in Graphviz format

37.1.1 Related Options

Options	Description
-OB	Output BDL file for each level of analysis.
-O-B	Do not output BDL file for every level of analysis (Default).
-OI	Output information file.
-O-I	Do not output information file (Default).
-OW	Output important warning file.
-O-W	Do not output important warning file (Default).
-OA	Output file for verification.
-O-A	Do not output file for verification (Default).
-OR	Output register allocation information file.
-O-R	Do not output register allocation information file (Default).
-OS	Output step information file.
-O-S	Do not output step information file (Default).

Options	Description
-OF -O-F	Output functional unit usage information file. Do not output functional unit usage information file (Default).
-OP -O-P	Output I/O port usage information file. Do not output I/O port usage information file (Default).
-OM -O-M	Output memory usage information file. Do not output memory usage information file (Default).
-OU -O-U	Output resource usage information file. Do not output resource usage information file (Default).
-OX -O-X	Output above mentioned information files. Do not output above mentioned information files (Default).
-OT -O-T	Output hint file (Default). Do not output hint file.
-OC -O-C	Output synthesis information file of CSV format (Default). Do not output synthesis information file of CSV format.
-Zcsv_lst -Zcsv_lst= <i>filename</i>	Add the synthesis information of CSV format in "Input filename_list.csv" file. Add the synthesis information of CSV format in <i>filename</i> file.
-Zdotty_out	Output the state transition graph in the format corresponding to Graphviz.
-wd= <i>directory</i>	Set specified directory as path for all information file outputs.

Options related to Output BDL Files for Analysis	
Options	Description
-OI	Display only 1 line number at the top of the line of the corresponding source.
-O-I	Do not display the line number of corresponding source (Default).
-Oo	Display all the corresponding source line numbers at the end of the line in the form of comments (Default).
-O-o	Do not display all the corresponding source line numbers at the end of the line in the form of comments.
-Ot	Display the attributes (Default).
-O-t	Do not display the attributes.
-Oy	Display the cast (Default).
-O-y	Do not display the cast.
-Oe	Display the 'bitw' attribute (Default).
-O-e	Do not display the 'bitw' attribute.
-Ow	Display the bit information of signal, constant number (Default).
-O-w	Do not display the bit information of signal, constant number.
-O2	Display the constant numbers in binary format.
-O-2	Display the constant numbers in decimal format (Default).
-Zundisp_fname	Omit the display of file name when only one input file is given.

37.2 List of Output Information File

There are three types of output information files. The first type consists of those files that are always generated and the second type consists of information files that are not generated when certain synthesis options have been specified, and third consists of information file that are generated only when certain synthesis options are specified. Following indicates the list of information files to be output.

Further, depending on the synthesis mode, there is also an information file which is not output.

Table 4: Default Output File

File Name	Extension	Format
Summary file	.SUMM	Text
Error File	.err	Text
Log File	.LOG.gz	gzip compressed sh script
Circuit quality report (text format)	.QOR	Text
Circuit quality format (HTML format)	.QOR.HTML	HTML format
Internal format file of state transition description (before resource sharing)	_C.IFF	Binary
Internal format file of state transition description (after resource sharing)	_9.IFF	Binary
Internal format file of configuration description	_E.IFF	Binary
Synthesis information integrated file	.VSG	Binary
Lower module specification		Text
Power evaluation file for FPGA (for Altera)	LMSPEC	csv file
Power evaluation file for FPGA (for Xilinx)	_pwr.csv _pwr.xpe	xpe file

Table 5: File that can control output by option

File Name	Extension	Format	Option
Synthesis information file of CSV format	.CSV	csv format	-O-C
Hint file	.tips	Text	-O-T

Table 6: Output file of option

File Name	Extensions	Options
Resource Allocation Information File	.INF	-OI
Important Warning file(warn)file	.warn	-OW
BDL file of state transition description(before resource sharing)	_C.BDL	-OB
BDL file of state transition description(after resource sharing)	_9.BDL	"
BDL file of configuration description	_E.BDL	"
File for verification	.AID	-OA
Register allocation information file	.REG	-OR
Step information file	.STP	-OS
State transition graph file of Graph viz format.	_5.dty	-Zdotty_out
"	_C.dty	-Zdotty_out
Following are at the time of automatic scheduling.		
I/O port usage information file	.PUV	-OP
Functional unit usage level information file	.FUV	-OF
Memory usage level information file	.MUV	-OM
Resource usage level information file	.USE	-OU

37.3 Summary File (.SUMM)

This file generates the summary of synthesis information displayed during the execution of synthesis. Below, output result during execution of sample data "sort.c" in manual scheduling is shown. Each section of output result is annotated with * sign followed by a number which is described in

Table 7.

```
Copyright (C) 1988-2008 NEC Corporation. All rights reserved.  
bdltran version : 5.02.00 Fri Feb 1 17:33:24 JST 2008  
    (BIF version : 3.02h)  
    make at Fri Feb 1 17:45:57 JST 2008  
Option -c2000 -sN -lb sample.BLIB -lfl sort-auto.FLIB -lfc sort-  
auto.FCNT sort.IFF  
bdlpars sort.c  
##### CyberI started (manual scheduling mode) #####  
[Clock period] 2000 (1/100ns) [define]  
[Datapath delay] 2000 (1/100ns)  
[Control delay] 0 (1/100ns)  
***** Initial # of operations in the input behavior description  
*****  
==== Operators ====  
!= : 4 (1bit:1, 4bit:3)  
++ : 1 (4bit:1)  
> : 1 (32bit:1)  
== : 3 (4bit:3)  
++ : 1 (4bit:1)  
- : 1 (4bit:1)  
Data transfer : 51  
==== array / memory ====  
read : 10  
write : 10  
***** After optimizing & expanding the behavior description  
*****  
==== Operators ====  
!= : 3 (4bit:4)  
> : 1 (32bit:1)  
== : 3 (4bit:3)  
- : 1 (4bit:1)  
- : 2 (4bit:2)  
+ : 1 (1bit:1)  
Data transfer : 60  
==== array / memory ====  
read : 10  
write : 13
```

```
#####
[Performance index max/min] 6 / 4
MEM :                                         ※10
                                         ※11

REG :
  Used      declared      used      bit * count      ※12
  Bit       bit       count
-----+
  1          1          1
-----+
  3          3          1
-----+
  4          4         32
-----+
  32         32         11
-----+
Total                                484

MUX :
  1 bit   : (1way: 1)                                         ※13
  2 bit   : 2way: 2
  3 bit   : (1way: 1), 9way: 1
  4 bit   : (1way: 2), 2way:34 , 3way: 1 , 10way: 3
  32 bit  : (1way: 1), 2way:10 , 3way: 1 , 10way: 1
  Total   : 1511 (# of fanins)
DECODER:
  4to10: 4                                         ※14

FU :
  Fu name      area      delay      pipeline      count      ※15
                           stage
-----+
  addsub04_04u    153     1.99      -           1
  gop32_01u       330     9.13      -           1

Unused FUs:
-----+
None

Total states : 6 (+Reset)                                         ※16
Total net count      : 1501                                         ※17
Total pin pair count : 5333                                         ※18
Const fanout        : 119                                         ※19

Critical path delay   : 16.57ns                                         ※20
False path           : Unchecked                                         ※21
Multi-cycle path     : Unchecked                                         ※22
Combinational loop   : Unchecked                                         ※23

Net count:
  bit      net      bitXnet      ※24
-----+
  1        110      110
  2         5       10
  3        14       42
  4        77      308
  10        4       40
  32        26      832
-----+
Total                1342
```

Pin pair:		count	sub total
Fanout	bit		
<hr/>			
45	1	2	90
15	1	1	15
14	4	1	56
1	2	5	10
1	1	77	77

5

```

Total                               2659
Fanout for consts:
value          fanout
0              92
1              27
-----
Total           119

```

26

Basic Library Name: sample (BLIB Version 2.00)
TOTAL AREA 14768 (FU: 421 + REG: 8712 + MUX: 5018 + DEC: 192 + MISC: 425)
+ pin pair 5333(net 1501) + 1FSM of 6(+Reset)states
CyberI Normal finish (manual scheduling mode) [sort]

⊗ 27

⊗ 28

⊗ 29

(For detailed information, please refer to **section 37.11** Circuit Quality Report (QOR/.QOR.HTML))

Table 7: List of Summary File Information

* 1	CWB Version information.
* 2	Command-line options specified for Synthesis
* 3	Option at the time of BDL input
* 4	Mode display
* 5	Clock cycle given as constraint.
* 6	Data path delay constraint during constraint clock cycle.
* 7	Assumed delay of FSM and control logic during constraint clock cycle.
* 8	Number of assignments and references of operation, number and array used during file input.
* 9	Number of assignments and references of operation, number and array after the conversion
* 10	Maximum and minimum number of states required for transition from Initial state to each state.
* 11	Type, number and estimated area of used memory.
* 12	Total number of used Flip Flop and bit number, declared bit number, quantity used for each register.
* 13	Bit width of multi processor and input way number wise quantity.
* 14	Type and quantity of each decoder.
* 15	Type and quantity of functional units used.
* 16	Number of states
* 17	Total number of wiring.
* 18	Total number of Pin Pairs. (Total number of fan-out)
* 19	Constant number of Fan-outs.
* 20	Estimated maximum delay time.
* 21	Number of false paths.
* 22	Number of multi-cycle paths.
* 23	Number of combinatorial loops.
* 24	Breakdown of the number of wiring between register, multiprocessor and functional unit.
* 25	Breakdown of the number of Pin Pairs between register, multiplexer and functional unit.
* 26	Breakdown of constant number of Fan-outs.
* 27	Library name and version used in estimated area.
* 28	Total estimated area and wiring number, total Pin pair and state number.
* 29	Display mode and completion state.

37.4 Error File (.err)

"Information", "warning" and "error" are generated in error file. For details on errors and warnings, please refer to **section 36**.

When the "-OW" option is specified, important warning file (.warn) is generated. This is provided as a separate option because important warnings may sometimes be buried along with other error and information messages in the error file. Currently, warning file contains messages related to multiple assignments as mentioned in **sections 36.5, 36.6**. However, in the future releases of Cyber, more messages will be included in the list.

37.5 Log File (.LOG.gz)

In gzip compressed file, all the files used in synthesis, like "source file", "header file", and "various constraint files" etc, as well as "options" are consolidated into one file. When LOG file is executed as a shell script, the respective files are regenerated and the BDL input option is generated in the BDL input parameter file "bdlpars.bpprm", whereas synthesis option is generated in parameter file "btltran.prm". However, options related to the RTL output are not generated.

37.6 Lower module specification File (.LMSPEC)

It is a file consolidating the circuit information of synthesized process and name, area information and port information of synthesized process are generated.

When high level hierarchy appropriate process is synthesized, then the following merits can be achieved by reading this lower module specification defined information file after specifying -ll

- Easy port consistency check between hierarchies
- Reduction effect of synthesis area of high level hierarchy

For details, please refer to **chapter 24**.

37.7 Hint file (.tips)

Option	Explanation
-OT	Output hint file (default)
-O-T	Do not output hint file

Hint file outputs hints for circuit in order to obtain a better circuit. The hint may not necessarily be always effective; however, generally, it suggests changes in constraints for synthesis or modification in description so that better results can be obtained.

37.8 Synthesis Information Files (.CSV Format)

Option	Explanation
-OC	Generate synthesis information file (CSV) of CSV format. (Default).
-O-C	Do not generate synthesis information file (CSV) of CSV format.
-Zcsv_lst	Add synthesis information of CSV format in file "inputfilename_list.csv".
-Zcsv_lst= <i>filename</i>	Add synthesis information of CSV format in file ' <i>filename</i> '.

Following information is output in CSV format.

Further, option "-Zcsv_1st" adding the synthesis information to the file for creating executed synthesis results list is prepared.

Table 8: Information of CSV file

Total Estimated area	AREA
Number of states	state
Estimated area of functional unit	FU
Estimated area of register	REG
Estimated area of multiplexer	MUX
Estimated area of logical function	MISC
Estimated area of memory	MEM
Total Pin Pair number	Pin-pair
Number of wires	net
Maximum number of state transitions from primary state	max
Minimum number of state transitions from primary state	min
Average number of state transitions from primary state	ave

Example

AREA, state, FU, REG, MUX, DEC, pin-pair, net, max, min, ave, MISC, MEM
 14694, 6, 388, 8694, 5026, 192, 5255, 1478, 4, 4, 0.00, 394, 0

37.9 BDL File for Analysis (.BDL)

When “-OB” option is specified, the BDL file (-C.BDL,-9.BDL) of state transition description and BDL file (-E.BDL) of configuration description are generated for the analysis of behavioral synthesis process. However, there is no guarantee that this BDL file can be used as behavioral synthesis input for analysis.

Following are the options to control the format to be output to the above mentioned BDL format file:

Options related to generated BDL file for analysis	
Options	Description
-OI	Display only 1 line number at the top of the line of the corresponding source.
-O-I	Do not display the line number of corresponding source (Default).
-Oo	Display all the corresponding source line numbers at the end of the line in the form of comments (Default).
-O-o	Do not display all the corresponding source line numbers at the end of the line in the form of comments.
-Ot	Display the attributes (Default).
-O-t	Do not display the attributes.
-Oy	Display the cast (Default).
-O-y	Do not display the cast.
-Oe	Display the ‘bitw’ attribute (Default).
-O-e	Do not display the ‘bitw’ attribute.

-Ow	Display the bit information of signal, constant number (Default).
-O-w	Do not display the bit information of signal, constant number.
-O2	Display the constant numbers in binary format.
-O-2	Display the constant numbers in decimal format (Default).
-Zundisp_fname	Omit the display of file name when only one input file is given

- -OI Displays line number of corresponding source in only 1 line head.
Displays line number at the beginning of the corresponding source line as shown below. In case of multiple lines, only line number is displayed with one and not others.

```
110 : right(0:4) = node_right_node[p(0:4)](0:4);
111 : left(0:4) = node_left_node[p(0:4)](0:4);
113 : if (left(0:4) != 0(0:4)) {
```

- -Oo Display all the corresponding line numbers at the end of the source line in the form of comments.

As shown below, line number corresponding to the source of each line are displayed in the form of comment /* line# File name: Line number*/.

```
p(0:4) ::= nmux {
    when (U_16(0:1)) : stack_rd00(0:4);          /* line# sort.c:128 */
    when (U_13(0:1)) : right(0:4);                /* line# sort.c:121 */
    when (U_11(0:1)) : left(0:4);                 /* line# sort.c:117 */
    when (ST1_04d(0:1) | U_01(0:1)) : 0(0:4);    /* line# sort.c:77,101 */
    when (U_08(0:1)) : right_node_rd00(0:4);     /* line# sort.c:92 */
    when (U_06(0:1)) : left_node_rd00(0:4);      /* line# sort.c:85 */
    default : p(0:4) ;
};
```

- -Ot Display the attributes
Display the attribute in the output BDL file. In case listing file is difficult to visualize, then specify option '-O-t' which will hide the attributes.
- -Oy Display the cast (type conversion).
Display the cast in the output BDL file. When it is difficult to see because of lots of casts, then specify the "-O-y" option, which will not display the cast.
- -Oe Display the 'bitw' attribute
Display the bit width attribute expressing the bit width of generated result of function. If it is invisible due to attributes, then by specifying"-O-e" option, the display will be stopped. Sometimes the bit width attribute get converted to cast during the synthesis and are displayed as casts.

- -Ow Display the bit width information of signals and constant number

As shown below, Bit-width information is displayed during reference and assignment of signals and constant numbers. Specify the option "-O-w", for stopping the display of bit information.

Displays Bit-width

```
if (RG_01(0:4) != 10(0:4)) {  
    data_ter(0:32) = i_data(0:32);  
    nod_value[RG_01(0:4)](0:32) = data_ter(0:32);  
    data(0:32) = data_ter(0:32);  
    p(0:4) = 0(0:4);  
    goto ST1_03;  
}
```

Do not display Bit-width

```
if (RG_01 != 10) {  
    data_ter = i_data;  
    node_value[RG_01] = data_ter;  
    data = data_ter;  
    p = 0;  
    goto ST1_03;  
}
```

- -O2 Display the constant numbers in binary.

Normally constant numbers are shown in decimal format. When the "-O2" option is specified, numbers will be displayed in binary. However, indefinite constant number for bit width is displayed like decimal.

Display in decimal

```
if (RG_01(0:4) != 10(0:4)) {  
    :  
    p(0:4) = 0(0:4);  
    :  
}
```

Display in binary

```
if (RG_01(0:4) != 0b1010) {  
    :  
    p(0:4) = 0b0000;  
    :  
}
```

- -Zundisp_fname omits the display of file name when only one input file is given.
Generally, when line information is output, file name and line number are shown as a set.
However, when only one input file is listed, then it can be omitted.

37.10 Resource Allocation Information File (.INF)

In the INF file, the resource allocation information (transfer information or input output information etc) of synthesized circuit is classified and outputted into the following 7 types:

1. Usage of functional units (Operator Usage Table)
2. Memory, register array usage (Memory Usage Table)
3. Module Usage (Module Usage Table)
4. Register Transfer information (Register Transfer Table)
5. Terminal Transfer information (Ports Transfer Table)
6. Condition information (Condition Table)
7. I/O information (IO Usage Table)

Given below is an example where description of 1 is synthesized and explained after INF file is generated.

Example 1: Input description (gcd1.bdl)

```

in ter(0:8) xi,yi;
out ter(0:8) GCDout;
process GCD()
{
    /* Synthesis by internal memory*/
    var(0:8) x[2]/* Cyber array=RAM */;
    x[0] = xi ; x[1] = yi;
    while(x[0] != x[1]) {
        if(x[0] < x[1] )
            x[1] = x[1]-x[0];
        else
            x[0] = x[0]-x[1];
    }
    GCDout = x[0];
}

```

Functional unit library file (gcd1.FLIB)

```

#@LIB { GCD }
@FLIB{
    NAME      lop08
    KIND      <
    BITWIDTH 8
    DELAY    442
    AREA     52
}
@FLIB{
    NAME      sub08
    KIND      -
    BITWIDTH 8
    DELAY    927
    AREA     74
}
#@END { GCD }

```

Functional unit count file (gcd1.FCNT)

```

#@CNT { GCD }
@FCNT {
    NAME lop08
    LIMIT 1
}
@FCNT {
    NAME sub08
    LIMIT 1
}
#@END { GCD }

```

Memory library file (gcd1.MLIB)

```

#@LIB { GCD }
@MLIB {
    NAME mem08A
    KIND R1,W1
    BITWIDTH 8
    DELAY 300
    ...
}
#@END { GCD }

```

Memory library file (gcd1.MCNT)

```

#@CNT { GCD }
@MCNT {
    NAME mem08A
    LIMIT 1
}
#@END { GCD }

```

37.10.1 Functional Unit Usage (Operator Usage Table)

This section shows the usage of functional unit. The targeted functional units are the one that have been described in the functional unit constraint file. Below, the description of each item generated is shown by using an example:

Symbol	Meaning
no	Number
name	Name of the functional unit
in_1	1 st operand of operation
in_2	2 nd Operand of operation
wid	Bit width
state	State used
Condition	Condition used

```
[Operator Usage Table]
no. name      ope      in_1          in_2          wid     state       condition
1. lop081ot   <        TR_01(0:8)    RG_00(0:8)    8        ST1_06     T03
2. sub081ot   -        RG_01(0:8)    RG_00(0:8)    8        ST1_07     T04 && T05'
sub081ot     -        RG_00(0:8)    RG_01(0:8)    8        ST1_07     T04 && T05
```

Here "T03" signifies the "Condition name". In the Condition Table, "T03" is described in terms of signals or expression where it is applicable in the design. Usage of " " with "T05'" indicates inverse of given condition. In case "-" is specified, it signifies unconditional execution.

In the description above, "ST1_06" and "ST1_07" signify the "State". Normally, "ST1_00" is reset state, whereas "ST1_01" is 1st State and other states are named as "ST1_02", "ST1_03" and so on.

In the 1st line, in state "ST1_06", in case of condition "T03" is satisfied, a 8-bit comparator (<) is used to compare terminal signal "TR_01" and register "RG_00"(TR_01 < RG_00). Here, "RG_00" and "TR_01" are described in register transfer information and terminal transfer information tables, respectively.

Both 2nd and 3rd lines list the usage of 8-bit subtracter functional unit named "sub081ot", which is used in state "ST1_07". Here, in case condition "T04 && T05'" is true, "RG_00" is subtracted from "RG_01". Whereas, in case "T04 && T05" is true, "RG_01" is subtracted from RG_00".

37.10.2 Memory and Register Array Usage (Memory Usage Table)

This section shows usage of Memory and Register array. Targeted memory is the memory used from memory constraints. Register array that are variable unrolled and register array implemented as logic are out of the scope.

Below, the description of each item generated is shown by using an example

Symbol	Meaning
no	Number
name	Array/Memory Name
kind	Memory type
addr	Access Address
data	Data read/written
state	Used State
Condition	Used Condition

```
[Memory Usage Table]
no. name      kind      addr      data      state      condition
1. x(mem08A) R1,W1
               0(0:1)    ---- ST1_06   T03
               0(0:1)    ---- ST1_05   T01'
               1(0:1)    ---- ST1_05   T01
               1(0:1)    ---- ST1_04   -
               0(0:1)    ---- ST1_03   -
               1(0:1)    RG_00(0:8) ST1_08   T06 && T07
               0(0:1)    RG_00(0:8) ST1_08   T06 && T07'
               1(0:1)    yi(0:8)    ST1_02   -
               0(0:1)    xi(0:8)    ST1_01   -
```

Here, 「name」 indicates the array as specified in the input description. Whereas, the name within parenthesis indicates the name within memory constraint file. For example, as shown above, array 「x」 in 「x(mem08A)」 is allocated in memory 「mem08A」. In case of register arrays, in 「name」 field, array name is specified, whereas in parenthesis array name is appended with "_reg". For example, register array for array M is listed as 「M (M_reg)」.

In the field 「kind」, memory type as specified in memory constraint file is displayed e.g. 「R1, W1」 specify a memory with Read port 1/write port 1. In case of register arrays, available decoder count is displayed. 「RW1」 displays that one decoder for Reading/ writing is available.

The 「addr」, 「data」, 「state」 and 「cond」 shows the access information for a memory/register array read/write. The 「addr」 specifies the memory access address whereas 「Data」 means the data that is written to the register array/memory. In case listing shows 「----」, it means data is read from memory / register array.

37.10.3 Module Information (Module Usage Table)

This section shows the information of module generated. Here, module name and port name are listed after inserting 「=」 behind item name.

Sign	Meaning
name	Module name
input	List of input ports
output	List of output ports
inout	List of inout ports

```
[Module Usage Table]
name = GCD
input = xi(0:8), yi(0:8), RESET, CLOCK
output = GCDout(0:8),
```

The port information, like 「xi(0:8)」, specifies both signal name and bit-width information. Bit information is displayed in ascending order. Further, as the signal which does not have bit information omits 「(0:1)」, therefore start bit is 「0」 and bit width is 「1」.

37.10.4 Register Transfer Information (Register Transfer Table)

This section shows the information transferred to register used in circuit generated.

Symbol	Meaning
no	Number
name	Register name
source	Transfer source
state	Transferred State
Condition	Transferred Condition

```
[Register Transfer Table]
no.    name          source           state      condition
1.     FF_00(0:1)    |>(RG_00(0:8) ^ mem08A1_rdl(0:8)) ST1_04      -
2.     FF_01(0:1)    lop081ot(0:1)      ST1_06      T03
3.     RG_00(0:8)    sub081ot(0:8)      ST1_07      T04 && T05'
            RG_00(0:8)    sub081ot(0:8)      ST1_07      T04 && T05
            RG_00(0:8)    mem08A1_rdl(0:8)   ST1_05      T01
            RG_00(0:8)    mem08A1_rdl(0:8)   ST1_03      -
4.     RG_01(0:8)    TR_01(0:8)        ST1_06      T03
```

In the 2nd line, the register "FF_01" shows the assignment of the value of terminal "Lop081ot" when the state is "ST1_06" and the condition "T03" gets satisfied. However, caution must be observed here because register will not be updated until the next cycle. In other cases, the register will retain the value in the self feedback loop.

37.10.5 Terminal Transfer Information (Terminal Transfer Table)

This section shows the connection information within the circuit generated.

Symbol	Meaning
no	Signal
name	Terminal name (array name)
source	Transfer source
state	Transferred State
condition	Transferred Condition

```
[Terminal Transfer Table]
no. name      source                      state  condition
1. GCDout(0:8) mem08A1_rd1(0:8)          ST1_05 T01'
2. JF_01(0:1)  1(0:1)                      ST1_05 T01'
                JF_01(0:1)  0(0:1)              ST1_05 T01
3. TR_01(0:8)  mem08A1_rd1(0:8)          ST1_06 T03
4. JF_02(0:1)  1(0:1)                      ST1_08 T06
                JF_02(0:1)  0(0:1)              ST1_08 T06'
5. lop081ot(0:1) lop08@1(lop081i1(0:8),lop081i2(0:8)) -      -
6. lop081i1(0:8) TR_01(0:8)                  ST1_06 T03
7. lop081i2(0:8) RG_00(0:8)                  ST1_06 T03
8. sub081ot(0:8) sub08@1(sub081i1(0:8),sub081i2(0:8)) -      -
9. sub081i1(0:8) RG_00(0:8)                  ST1_07 T04 && T05
                sub081i1(0:8) RG_01(0:8)          ST1_07 T04 && T05'
10. sub081i2(0:8) RG_01(0:8)                 ST1_07 T04 && T05
                sub081i2(0:8) RG_00(0:8)          ST1_07 T04 && T05'
11. mem08A1_rd1(0:8) mem08A@1(mem08A1_wd1(0:8),
                                mem08A1_ra1(0:1),
                                mem08A1_wa1(0:1),
                                mem08A1_re1(0:1),
                                mem08A1_we1(0:1)) -      -
12. mem08A1_wd1(0:8) xi(0:8)                 ST1_01 -
                mem08A1_wd1(0:8) yi(0:8)           ST1_02 -
                mem08A1_wd1(0:8) RG_00(0:8)          ST1_08 T06 && T07'
                mem08A1_wd1(0:8) RG_00(0:8)          ST1_08 T06 && T07
.
.
```

Line no.1 shows that "mem08A1_rd1(0:8)" is assigned in terminal GCDout(0:8) when the state is ST1_05 and the condition T01' gets satisfied.

Further, no. 5 and 8 shows the functional units, while no.11 shows a memory.

A functional unit is shown as C language functional form where functional unit name specified in functional unit constraint file becomes the function name appended with "@numeral". Further, parameter represents the input port and the "name" of output destination represents the output port.

no. 5 is the functional unit named as "lop08@1" and has two inputs "lop081i1", "lop081i2" and have "lop081ot" as an output.

```
5. lop081ot(0:1) lop08@1(lop081i1(0:8),lop081i2(0:8)) - -
```

A memory is shown in functional form similar to functional unit. Here, memory name as specified in constraint file becomes the function name appended with "@numeral". Parameter shows the input port of memory, and "name of " output destination represents the output port.
no. 11 shows a memory called "mem08A@1", which have 5 input "mem08A_wd1", "mem08A_ra1", "mem08A_wa1", "mem08A_re1", "mem08A_we1" and have one "mem08A_rd1" as an output.

```
11. mem08A1_rd1(0:8)      mem08A@1(mem08A1_wd1(0:8),  
                           mem08A1_ra1(0:1),  
                           mem08A1_wa1(0:1),  
                           mem08A1_re1(0:1),  
                           mem08A1_we1(0:1)) - -
```

37.10.6 Condition Information (Condition Table)

This section shows the information related to condition symbol used in functional unit usage, memory usage, register transfer information, and terminal transfer information.

Symbol	Meaning
no	Number
name	Condition name
Condition	Corresponding signal/operational expression

```
[ConditionTable]
no.    name        condition
1.     T01         FF_00(0:1)
2.     T02         JF_01(0:1)
3.     T03         FF_00(0:1)
4.     T04         FF_00(0:1)
5.     T05         FF_01(0:1)
6.     T06         FF_00(0:1)
7.     T07         FF_01(0:1)
8.     T08         JF_02(0:1)
```

This is a condition table that relates condition name used in "condition" of each information table to signal name and expression within a synthesized circuit. As conditions which cannot be used only in FSM are included, there are also conditions which cannot be used within INF file like T02 or T08.

37.10.7 Input Output Information (IO Usage table)

This section shows the information related to the input from outside, output to exterior and in which state/condition it is performed.

Symbol	Meaning
no	Number
dest.	Transfer destination
Source	source of transfer
State	Transferred State
condition	Transferred Condition

```
[IO    Usage Table]
no.   dest.          source           state  condition
1.    mem08A1_wd1(0:8)  xi(0:8)        ST1_01  -
2.    mem08A1_wd1(0:8)  yi(0:8)        ST1_02  -
3.    GCDout(0:8)       mem08A1_rdl(0:8) ST1_05  T01'
```

In "dest", signal name of transfer destination is written and in "source", signal name of transfer source is written. In case of input port, "source" represents the input port name and "dest" represents the internal signal name. In case of output port, the "dest" represents the output port name and "source" represents the internal signal name.

In particular, no. 1 shows the state "ST1_01", when data is transferred from external input port "xi" to memory Write Data port "mem08A1_wd1".

37.11 Circuit Quality Report (.QOR/.QOR.HTML)

The circuit quality report contains information pertaining to the quality of the synthesized circuit (number of variables/registers used, routability etc), and the report is classified into the following 12 categories:

1. Summary (Summary)
2. Important Information (Important Information)
3. Delay Information (Delay Information)
4. Input/output information (Primary Port)
5. Memory information (Memory)
6. Functional unit information (Functional Unit)
7. Register information (Register)
8. Multiplexer information (Multiplexer)
9. Decoder information (Decoder)
10. Detailed delay timings (Timing)
11. Wiring information (Wire)
12. Loop Latency information (Latency)

Further, the following information is generated in wiring information.

- Wiring information
- Fan-in/ fan-out information of the register
- Routability information

Each content in circuit quality report (QOR.HTML) of HTML version and corresponding text version is explained below.

37.11.1 Summary

An output example of summary information in a circuit quality report containing information regarding the entire circuit is shown below.

The description of corresponding number is mentioned in

Table 9.

Cyber RTL Synthesis Report (module name : filter) ※a-1

Summary

Basic Library Name	SAMPLE (BLIB Version 2.00)	※a-2
Clock period	12ns	※a-3

※b AREA		TOTAL Area				Memory	
		Sequential	Combinational	-		-	
Design	count	REG	FU	MUX	DEC	MISC	MEM
filter (TOTAL)	-	12,435 (0%)	13,065 (31%)	11,727 (28%)	0 (0%)	3,604 (8%)	-
filter (w/o sub modules)	-	8,240	1,549	8,068	0	2,324	-
apply_filter	x1	3,195	11,416	3,669	0	1,280	0

Controller		WIRE				PORT			
TOTAL STATES	#FSM STATES / FSM	EST. DELAY	NET	PIN PAIR	TOTAL	IN	OUT	IN / OUT	
12 (+Reset)	1	12 (+Reset)	0.3ns	6,890	12,811	49	36	13	0

Figure 111: Summary of circuit quality report when synthesized at ASIC mode

Controller				PIPELINE		WIRE		PORT			
TOTAL STATES	#FSM	STATES / FSM	EST. DELAY	stages	DII	NET	PIN PAIR	TOTAL	IN	OUT	IN / OUT
1	0	-	-	2	1	82	86	42	34	8	0

※d-1 ※d-2

Figure 112: Summary of circuit quality report at the time of automatic pipelined scheduling

Cyber RTL Synthesis Report (module name : filter)**Summary**

Basic Library Name	CWBSTDLIB (BLIB Version 2.00)
Clock period	12ns

FPGA Family	StratixIV	※a-3
FPGA Device	EP4SOX70D	※a-4
FPGA Package	FC20	※a-5
FPGA Speed	-3	※a-6

Design	count	ALUTs ^{※L}	Registers	Block memory bits	DSPs	
filter (TOTAL)	-	1,488	343	32,768	0	※b-16
filter (w/o sub modules)	-	992	258	32,768	0	※b-17
apply filter	x 1	496	86	0	0	※b-18

※b-19

※b-20

※b-21

※b-22

※b-23

※b-24

Figure 113: Summary of circuit quality report when synthesized in FPGA mode**Table 9: Information list of summary**

※a-1	Module name
※a-2	Basic library and library version used in estimation
※a-3	Clock cycle
※a-4	Target family of FPGA
※a-5	Target device of FPGA
※a-6	Target package of FPGA
※a-7	Target speed grade of FPGA
※b	Area information
※b-1	Total estimated area
※b-2	Estimated memory area
※b-3	Estimated area of sequential circuit
※b-4	Estimated area of combinatorial circuit
※b-5	Each estimated area (% usage in terms of total area is in brackets)
※b-6	Each estimated area of only uppermost process (omitted only in case of uppermost process)
※b-7	Each estimated area of child function and sub module (module name starts from CMPS_ if compensation circuit is automatically generated during synthesis)
※b-8	Process name (Function name)
※b-9	Instantiated count
※b-10	Estimated area of the register
※b-11	Estimated area of functional unit

⌘b-12	Estimated area of multiplexer
⌘b-13	Estimated area of decoder
⌘b-14	Estimated area of logical operation
⌘b-15	Estimated area of memory
⌘ b-16	Each estimation
⌘ b-17	Each estimation only for highest order process (omit only in case of highest order process)
⌘ b-18	Each estimated area of child function and sub module (module name starts from CMPS_ if compensation circuit is automatically generated during synthesis)
⌘ b-19	Process name (function name)
⌘ b-20	Instantiated count
⌘ b-21	LUT estimation (memory LUT cannot be included)
⌘ b-22	Register estimation
⌘ b-23	Memory bit estimation
⌘ b-24	DSP estimation
⌘ y-1	Latency index
⌘c	FSM information
⌘c-1	Number of FSM
⌘c-2	Total number of states
⌘c-3	FSM-wise state count
⌘c-4	Decoder delay of FSM
⌘d	Pipeline information (* displayed only when pipeline synthesis is used).
⌘d-1	Number of stages
⌘d-2	Number of DIIIs
⌘e	Wiring information (refer section 37.11.8.4).
⌘e-1	Total number of wires
⌘e-2	Total number of pin pairs
⌘f	Port information
⌘f-1	Total number of input/output bits
⌘f-2	Number of input bits
⌘f-3	Number of output bits
⌘f-4	Number of input/output bits

37.11.1.1 Latency Index

This index shows the latency index value of synthesis result, in quality index of scheduling result. If the numeric character is smaller, the number of cycles (latency) in scheduling also reduces. Below mentioned digit sum are the latency index value.

```

Sequential circuit: Maximum latency (maximum latency in case of conditional divergence)
+
Fixed cycle loop: Iteration count * loop latency
+
Un-specified cycle loop: Loop latency (Count=1 or assumed value)

```

Since wait() waits in one cycle, latency is one.

37.11.2 Important Information

An output example of important information in a circuit quality report containing information of the entire circuit is shown below.

The description for corresponding number is mentioned in **Table 10**.

Important Information^{*g}

#Violation path ^{*7} (> 10ns)	<u>False path</u>	Multi-cycle path	Combinational loop ^{*2}	Latch (bit)
1	3	0	0	0

^{* g -1}
^{* g -2}

Figure 114: Important information of circuit quality report

Table 10: Information list of important information

※g	Important information (In case each option for the number of paths and number of loops is not valid, then each one of them is displayed as "Unchecked")
※g-1	Number of delay violation paths (Displayed only when -Wcritical_path is specified and with manual scheduling)
※g-2	Delay values that are delay violation paths (within brackets) (Value specified by -Wcritical_path. Clock cycle when omitted)

※g-3	Number of false paths (Valid only if -Zfalse_path_script is specified)
※g-4	Number of multi-cycle paths (Valid only if -Wfalse_path_script is specified)
※g-5	Number of false loops or combinatorial loops (Its display can be changed by the specification of -Wfalse_loop, -Wall_loops. False loop display during -Wfalse_loop specification. Combinatorial display during -Wall_loops specification. When both -Wfalse_loop, -Wall_loops are specified, the one having greater loop counts is displayed)
※g-6	Number of latches used

37.11.3 Delay Information

An output example of delay information in a circuit quality report is shown below. The description for corresponding number is mentioned in **Table 11**.

Delay Information^{*h}

Critical Path Delay	13.1ns
---------------------	--------

Total (ns)	IN	FU	MUX	DEC	MISC	MEM
13.10	0.00 (0%)	12.00 (91%)	0.50 (3%)	0.00 (0%)	0.60 (4%)	0.00 (0%)

*h-1 *h-2 *h-3 *h-4 *h-5 *h-6 *h-7

Figure 115: Delay information of the circuit quality report

Table 11: Information list of delay information

⌘h	Delay information
⌘h-1	Estimated maximum delay time (shown as "unchecked" when basic library is not specified) (When -Zfalse_path_script is specified, false path detected is eliminated)
⌘h-2	Total delay of input port (in terms of % of total delay) present in maximum delay path requested.
⌘h-3	Total delay of functional units (in terms of % of total delay) present in maximum delay path requested.
⌘h-4	Total delay of multiplexer (in terms of % of total delay) present in maximum delay path requested.
⌘h-5	Total delay of decoder (in terms of % of total delay) present in maximum delay path requested.
⌘h-6	Total delay of logical operational units (in terms of % of total delay) present in maximum delay path requested.
⌘h-7	Total delay of memory (and in terms of % of total delay) present in maximum delay path requested.

37.11.4 Input output/Memory Information

An output example of all the information pertaining to input output / memory in a circuit quality report is shown below.

The description of corresponding is mentioned in **Table 12**.

Primary Port $\otimes i$			Memory $\otimes j$						
name	type	bit	name	kind	bit	word	area	count	count(shared/outside)
pixel_in	in	8	MEM3	R1,W1	8	2048	-	2	0
hlen_in	in	12							
vlen_in	in	12							
mode_in	in	2							
pixel_out	out	8							
pixel_in_ack	out	1							
hlen_in_v	out	1							
vlen_in_v	out	1							
mode_in_v	out	1							
pixel_out_valid	out	1							

$\otimes i-1$ $\otimes i-2$ $\otimes i-3$

$\otimes j-1$ $\otimes j-2$ ↑ $\otimes j-4$ ↑ $\otimes j-6$ $\otimes j-7$
 $\otimes j-3$ $\otimes j-5$

Figure 116: Input output/memory information

Table 12: Information list of input output / memory

$\otimes i$	Input/output information
$\otimes i-1$	Port name
$\otimes i-2$	In/out/inout
$\otimes i-3$	Bit width
$\otimes j$	Memory information
$\otimes j-1$	Used memory name
$\otimes j-2$	Memory type
$\otimes j-3$	Bit width
$\otimes j-4$	Number of words
$\otimes j-5$	Estimated area
$\otimes j-6$	Number of usages (internal memory)
$\otimes j-7$	Number of usages (external memory)

37.11.5 Functional Unit Information

An output example of information pertaining to functional unit in a circuit quality report is shown below.

The description of corresponding number is mentioned in **Table 13**. However, area and delay value generated through functional unit information generates the value for functional unit constraint (LMT) file, functional unit library (FLIB) file, and functional unit count (FCNT) file. Meanwhile, estimated area for summary of functional unit generates area after estimated logical synthesis based on value of these libraries. Hence, it does not match with the total value of area for functional unit information. Similarly, delay value of functional unit for detailed delay information generates delay value after estimated logical synthesis based on the value of these libraries. Hence, it does not match with delay value of functional unit information.

Figure 117: Functional unit information**Table 13: Information list of functional unit**

※k	Functional unit information ※ Turns colored in case of pipeline functional unit
※k-1	Functional unit name
※k-2	Functional unit type
※k-3	sign
※k-4	Bit width (within brackets indicates input bit width, and outside bracket indicates output bit width. c: clock port, r: reset port, h: stall port, s: start port, e: eop signal)
※k-5	Area
※k-6	Delay value
※k-7	Number of usages
※k-8	List of unused functional units

37.11.6 Register/Multiplexer/Decoder Information

An output example of information pertaining to register, multiplexer and decoder in a circuit quality report is shown below.

The description for corresponding number is mentioned in **Table 14**

Register ※1

used bit	declared bit count	used bit * count
1	1	1
3	3	1
4	4	32
32	32	11
Total		484

※I-1 ※I-2 ※I-3 ※I-4

Multiplexer ※m

bit way count	bit * way * count	
1 1	1	
2 2	2	
3 1	1	
3 8	1	
4 1	2	
4 2	34	
4 3	2	
4 10	3	
32 1	1	
32 2	10	
32 3	1	
32 10	1	
Total		1,504

※m-1 ※m-2 ※m-3 ※m-4

Decoder ※n

bit way count
4 10 4

※n-1 ※n-3
※n-2

Figure 118: Register/multiplexer/decoder information**Table 14: Information list of register/multiplexer/decoder**

⌘I	Register information
⌘I-1	Number of used bit
⌘I-2	Number of declared bit
⌘I-3	Number of respective pair for used number of bit and declared number of bit
⌘I-4	Number of used bit x numbers
⌘I-5	Total number of used FlipFlop
⌘m	Multiplexer information ⌘ colored if bit width x number of input way exceeds 256.
⌘m-1	Bit width
⌘m-2	Number of input way
⌘m-3	Count
⌘m-4	Number of total bits per Multiplexer type
⌘m-5	Total number of bits for multiplexer
⌘n	Decoder information
⌘n-1	Bit width
⌘n-2	Number of 'way'
⌘n-3	Count

37.11.7 Detailed Delay Information

An output example of detailed delay information corresponding to maximum delay path in a circuit quality report is shown below. The description for corresponding number is mentioned in **Table 15**.

Timing *o-1							
Clock period : 1ns , Path delay : 0.7ns							
*o-3	*o-4	*o-5	*o-6	*o-7	*o-8	*o-9	
name	pin	signal	delay	arrival time	logic stage	type	
func_1	o1	func_1.V_func	0.50	0.50	0	-	
_NMUX_60	o1	RG_t	0.20	0.70	1	-	
RG_t	din	-	-	0.70	1	-	

TOTAL	IN	FU	MUX	DEC	MISC	MEM	
0.70	0.00 (%)	0.50 (71%)	0.20 (28%)	0.00 (%)	0.00 (%)	0.00 (%)	

* i:input, o:output, s:MUX select, din:FF's input, dout:FF's output

Since false path is not considered during critical path delay measurement, there is a restriction that sometimes imparted clock cycle is crossed.

Further, normally, in logical synthesis, synthesis is done by meeting the constraints even if the constraint clock cycle is crossed. However, there is a possibility that logical synthesis time will become elongated and circuit area will get extended.

False path information *o-16

Path #1

*o-17	*o-18	*o-19
name	pin	signal
c	-	c
sub08_08u@1	o1	sub08_08u1ot
add08_08u@1	o1	add08_08u1ot

Multi-cycle functional units *o-20

*o-21	*o-22
FU name	delay
func	1.50

Figure 119: Detailed delay information

Table 15: Information list of Timing

※ o	Detailed delay information (Output only the path with maximum delay value. When -QTn# option is specified, if there are multiple paths having maximum delay value, then the specified numbers are generated) (If a library is not specified, maximum delay time is shown as "unchecked")
※o-1	Clock cycle
※o-2	Estimated maximum delay time (If basic library is not specified, then maximum delay time is shown as "unchecked") (Detected false path is excluded when -Zfalse_path_script is specified)
※o-3	Name of register or functional units etc. ※ The multiplexer names are indicated as "_MUX_identification number"/"_DMUX_identification number" ※ The logical operation is indicated as "_logical operation name_identification number" (The identification number is automatically allocated unique number)
※o-4	Pin name (Other than "defmod", pin name is allocated internally)
※o-5	Signal name for corresponding _E.BDL
※o-6	Delay value
※o-7	Signal arrival time (Stored value)
※o-8	Number of logical states (stored value)
※o-9	Type of signal
※o-10	Total delay of input ports (in terms of % of total maximum delay) present in maximum delay path requested.
※o-11	Total delay of functional units (in terms of % of total maximum delay) present in maximum delay path requested.
※o-12	Total delay of multiplexer (in terms of % of total maximum delay) present in

	maximum delay path requested.
※o-13	Total delay of decoders (in terms of % of total maximum delay) present in maximum delay path requested.
※o-14	Total delay of logical operational units (in terms of % of total maximum delay) present in maximum delay path requested.
※o-15	Total delay of memory (in terms of % of total maximum delay) present in maximum delay path requested.
※o-16	False path detailed information (Not displayed in case -Zfalse_path_script option is not specified or false path does not exist) Detailed display count is specified by –QFn# option. When not specified, it is 10)
※o-17	Component name (functional unit name, function name, memory name, signal name, decoder etc.)
※o-18	Port name
※o-19	Signal name
※o-20	Multicycle path detailed information (Not displayed in case -Zfalse_path_script option is not specified or multicycle path does not exist)
※o-21	Functional unit name
※o-22	Delay value

37.11.8 Wiring Information

37.11.8.1 Wiring Information

An output example of wiring information in a circuit quality report is shown below. The description for corresponding number is mentioned in **Table 16**.

WIRE *p

*p-1	*p-2	*p-3
Total net count *1	Total pin pair count *2	Const fanout
103	148	27

[Register fanin/fanout>>](#)

[Routability>>](#)

Net count *q *3

*q-1 * q-2 * q-3	bi	net	bit x net
1	13	13	
3	6	18	
8	9	72	
Total		103	

* q-4

Pin pair count *4

*r-1 * r-2 * r-3* r-4*	fanout	bit	coun	Sub total
5	3		1	15
4	1		2	8
3	1		1	3
2	8		3	48
2	1		1	2
1	8		6	48
1	3		5	15
1	1		9	9
Total				148

* r-5

Fanout for constsr *s

* s-1	* s-2
value	fanout
0	17
1	10
Total	27

* s-3

Clock fanout *t

*t-1 * t-2	name	count
	CLOCK(0:1)	4

Reset fanout *u

*u-1 * u-2	name	count
	RESET(0:1)	4
	ST1_00d(0:1)	1

Figure 120: Wiring information

Table 16: Information list of WIRE

⌘p	Wiring information (Refer 37.11.8.4)
⌘p-1	Total wiring count considering functional units, multiplexer and constants
⌘p-2	Total pin-pair count considering functional units, multiplexer and constants (total fan out)
⌘p-3	Total number of fan out of the constant
⌘q	Classification of wiring count among the functional unit, multiplexer, register etc.
⌘q-1	Bit width
⌘q-2	Count
⌘q-3	Bit width x Count
⌘q-4	Total wiring count
⌘r	Classification of pin-pair among functional unit, multiplexer, register
⌘r-1	Number of fan-out
⌘r-2	Bit width
⌘r-3	Count
⌘r-4	Total Pin-pair number for each fan-out
⌘r-5	Total number of pin pair
⌘s	Classification of fan-out among constants
⌘s-1	Constant value
⌘s-2	Number of fan-out
⌘s-3	Total number of fan-out of the constant
⌘t	Clock fan-out information (⌘ shown only when clock is present)
⌘t-1	Clock signal name
⌘t-2	Number of fan-out
⌘u	Reset fan out information (⌘ shown only when reset is present)
⌘u-1	Reset signal name
⌘u-2	Number of fan-out

37.11.8.2 Register “fan-in / fan-out” Information

An output example of register fan-in / fan-out information in a circuit quality report is shown below. The description of corresponding number is mentioned in **Table 17**.

Register fanin/fanout cone size ^{*v}			
Fanin (Top 10 registers) ^{*v-1}		Fanout (Top 10 registers) ^{*v-2}	
*v-3	*v-4	*v-3	*v-5
Register name	cone size	Register name	cone size
RP_MM_02(0:8)	7	RG_01_01(0:1)	3
RP_RR_02(0:8)	7	RR_rg03(0:8)	2
RR_rg03(0:8)	4	RR_rg00(0:8)	2
RR_rg01(0:8)	4	RR_rg02(0:8)	2
RR_rg00(0:8)	4	RR_rg01(0:8)	2
RR_rg02(0:8)	4	RP_main_tra01(0:2)	2
RG_01_01(0:1)	1	RG_04_02(0:1)	2
RG_04_02(0:1)	1	RP_RR_02(0:8)	1
RP_main_tra01(0:2)	1	RP_MM_02(0:8)	1

Input port and number of register of transfer source Output port and number of registers of transfer destination

Figure 121: Register fan-in / fan-out information

Table 17: Information list of register fan-in, fan-out

※v	Fan-in/fan-out information of register
※v-1	List of register sorted with fan-in count
※v-2	List of register sorted with fan-out count
※v-3	Register name
※v-4	fan-in count
※v-5	fan-out count

- **Fan-in of register** is the number of transfer source from the input port of register or primary input port till the time it gets connected.
- **Fan-out of register** is the number of transfer destination from the output port of register or primary output port till the time it get connected.

37.11.8.3 Wiring Characteristics/Routability Information

An output example of wiring characteristics/ routability information in a circuit quality report is shown below. (Section **37.11.8.5**) The description of corresponding number is mentioned in **Table 18**.

Routability *w

Top 25 nets
sorted by "total" (total pin pair)

Net name	total ^{*w-3}	max ^{*w-4}
RG_t(0:8)	24	3 (8bit)
B01_streg(0:3)	18	6 (3bit)
RG_j(0:8)	16	2 (8bit)
RG_00(0:8)	16	2 (8bit)
RG_01(0:8)	16	2 (8bit)
sub08_08u1ot(0:8)	16	2 (8bit)
func_1.V_func(0:8)	8	1 (8bit)
a(0:8)	8	1 (8bit)
b(0:8)	8	1 (8bit)
c(0:8)	8	1 (8bit)
f(0:8)	8	1 (8bit)
add08_08u1ot(0:8)	8	1 (8bit)
sub08_08u1i2(0:8)	8	1 (8bit)
sub08_08u1i1(0:8)	8	1 (8bit)
CLOCK(0:1)	5	5 (1bit)
RESET(0:1)	5	5 (1bit)
ST1_05d(0:1)	4	4 (1bit)
ST1_04d(0:1)	4	4 (1bit)
ST1_01d(0:1)	3	3 (1bit)
ST1_03d(0:1)	2	2 (1bit)
ST1_02d(0:1)	1	1 (1bit)
ST1_00d(0:1)	1	1 (1bit)
e(0:8)	0	0 (8bit)
d(0:8)	0	0 (8bit)
_NMUX_62(0:8)	8	1 (8bit)

*w-3 *w-4 *w-5

Top 25 nets
sorted by "max" (maximum fan out)

Net name	total ^{*w-3}	max ^{*w-4}
B01_streg(0:3)	18	6 (3bit)
CLOCK(0:1)	5	5 (1bit)
RESET(0:1)	5	5 (1bit)
ST1_04d(0:1)	4	4 (1bit)
ST1_05d(0:1)	4	4 (1bit)
RG_t(0:8)	24	3 (8bit)
ST1_01d(0:1)	3	3 (1bit)
RG_01(0:8)	16	2 (8bit)
RG_00(0:8)	16	2 (8bit)
RG_j(0:8)	16	2 (8bit)
sub08_08u1ot(0:8)	16	2 (8bit)
ST1_03d(0:1)	2	2 (1bit)
func_1.V_func(0:8)	8	1 (8bit)
sub08_08u1i1(0:8)	8	1 (8bit)
sub08_08u1i2(0:8)	8	1 (8bit)
add08_08u1ot(0:8)	8	1 (8bit)
f(0:8)	8	1 (8bit)
c(0:8)	8	1 (8bit)
b(0:8)	8	1 (8bit)
a(0:8)	8	1 (8bit)
ST1_02d(0:1)	1	1 (1bit)
ST1_00d(0:1)	1	1 (1bit)
e(0:8)	0	0 (8bit)
d(0:8)	0	0 (8bit)
_NMUX_58(0:8)	8	1 (8bit)

*w-3 *w-4 *w-5

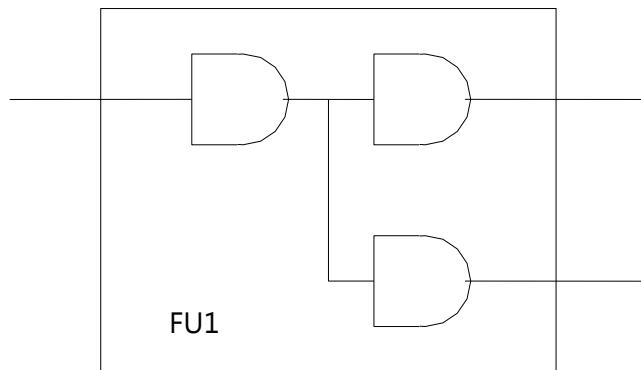
Figure 122: Wiring characteristics/Routability information

Table 18: Information list of wiring characteristics/Routability

⌘ w	Wiring characteristics/Routability information
⌘ w-1	List of signal sorted with total Pin-pair count
⌘ w-2	List of signal sorted with maximum fan-out count
⌘ w-3	Signal name ⌘In BDL, the output signal name from the multiplexer or logical operation with no name is outputted later compared to other signals. During that time multiplexer signal name is displayed as "_NMUX_identification number"/"_DMUX_identification number" The name of logical operation is displayed as "_logical operation name_identification number" (The identification number is an automatically allocated unique number) Total pin pair count for each signal.
⌘w-4	Total pin pair counted for all respective bit of every signal.
⌘w-5	Maximum value of pin pair counted for all respective bit of every signal. Numerical value in the bracket shows the bit number that has the maximum value.

37.11.8.4 Count of Wiring Information

The count of every item of wiring information ("WIRE") is explained with an example of a model using the functional unit (FU1) having an internal structure as mentioned below

**Figure 123:** Example of functional unit

Ex) Output of Summary (Text version)

```

WIRE:
net      : 4 ← Same as Total net count of wiring information
pin pair : 5 ← Same as Total pin pair count of wiring
           information
  
```

Ex) Output of wiring information (Text version)

```
Total net count      : 4 ← Number of total net and number of total pin pair
Total pin pair count : 5    which includes internal wiring and constants of
                        functional units and multiplexers etc.
Const fanout         : 0 ← Number of fan-out of constants
```

- Each value of "Total net count "/" Total pin pair count" of wiring information is a result of considering the internal wiring of functional unit or multiplexer etc. Thus, the total value of "Net count"/"Pin pair" table accumulated as a result of number of net/number of pin pair between the functional unit or multiplexer or registers etc are not same.
- If the NET/PIN_PAIR value is unspecified by the functional unit library file or basic library file that is being used, then there is no consideration to the internal wiring of the functional unit or multiplexer.
- If 1.xx is a version of basic library file that is being used, then the internal wiring of multiplexer or decoder is not considered.

37.11.8.5 Count of Wiring Characteristics Information/Routability

An example of pin-pair and routability information for _E.BDL has been described below.

Ex) Descriptive example of generated _E.BDL

```
REG01(0:4) = REG_A(0:4); ← The total bit reference are 2
REG02(0:4) = REG_A(0:4);
REG03(0:4) = REG_B(0:4);
REG04(0:2) = REG_B(0:2); ← Part reference
REG05(0:1) = REG_B(1:1); ← Part reference
```

Ex) Example of Count Result Output (Text version)

Net name	total	max
REG_A(0:4)	8	2 (4bit)
REG_B(0:4)	7	3 (1bit)

- Count of REG_A (0:4)
 6. max fanout

Since REG_A is referenced only for total bits it has, the number of fan-outs will be constants regardless of the bit-count. Hence, the maximum fan-out number is "2".
 7. Number of bit that becomes max fanout

As part reference is not done, hence ("4" bit) in the brackets because all bits changes to max fanout.
 8. total pinpair

The total is the sum of fanout number for every bit, thus $2+2+2+2 = "8"$.

- Count of REG_B (0:4)
 - max fanout
Since only REG_B(1) has the fan-out number of "3", the maximum fanout number is "3".
 - Bit number changing to max fanout
Since only REG_B(1) refers to ("1" bit) within the bracket as the maximum fan-out number is "3".
 - total pinpair
The total is the sum of fanout number of every bit, thus $2+3+1+1=7$.

37.11.9 Loop Latency Information

An output example of loop latency information in circuit quality report is shown below.

The description of corresponding number is mentioned in **Table 19**.

Latency										
Loop Type	Latency	State No.	Folding loop	Folding stages	Total folding states	Hazard	User operator	Sub loops		
L1	F	1 * N1	2	FL3	11	11	Exist	-	-	
L2	S	{3 + L1} * N2	1, 2, 3	-	-	-	-	-	L1	
total	S	Reset + L2	0, 1, 2, 3	-	-	-	-	-	L2	

※x-1	※x-2	※x-3	※x-4	※x-5	※x-6	※x-7	※x-8	※x-9	※x-10
Folding loop	Root	DII	Folding states	Hazard	User operator	Sub loops			
FL1	L1	1	1	None	-	-			
FL2	L1	1	6	Exist	-	FL1			
FL3	L1	1	4	Exist	-	FL2			

※x-11	※x-12	※x-13	※x-14	※x-15	※x-16	※x-17
-------	-------	-------	-------	-------	-------	-------

Figure 124: Loop Latency Information

Table 19: Information list of loop latency information

※ x	Loop latency information
※ x-1	Loop name (L#)
※ x-2	Loop classification ([S]equential/[F]olding)
※ x-3	Loop latency
	Reset: Reset state
	L#: Latency of loop L#
	N#: Number of repetition of L# (No limit for the number of repetition for description)
	{n1,n2,...}: Depicts the latency for every branch when branch exists in loop
	Latency index (Depicts the number of cycle until output is less for smaller value in the relative index for similar description.)
	In case repeat count (N#) is not decided, N# is calculated as 1.

※ x-4	In case repeat count is fixed, calculate with that value In case of folding loop, value requested in "(DII count × repeat count) + folding state count— DII count"
※ x-5	Number of the state configuring the loop
※ x-6	Folding loop name in the loop (FL#)
	Top loop name when there are multiple folding loop
※ x-7	Number of stages of folding loop
※ x-8	Number of state of folding loop (including inner loop)
※ x-9	Whether hazard exist in folding loop or not (Exist/None)
※ x-10	User-defined functional unit existing in loop
※ x-11	Loop existing in the inner part of loop
※ x-12	Folding loop name (FL#)
※ x-13	Loop which belongs to folding loop
※ x-14	DII of folding loop (Data Initiation Interval)
※ x-15	Number of state of folding loop (does not include inner loop)
※ x-16	Whether hazard exists in folding loop (Exist/None)
※ x-17	User-defined functional unit existing in folding loop
※ x-18	Folding loop existing in inner part of folding loop

37.11.10 Text Version Circuit Quality Report (.QOR)

Even in text version circuit quality report, information similar to HTML version is output.

An output example of circuit quality report (when synthesized in ASIC mode) in text version is shown below. The description of corresponding number is mentioned in **Table 20**.

```

Cyber RTL Synthesis report (module name : filter)      ※ A-1
Basic Library Name: SAMPLE (BLIB Version 2.00)          ※ A-2
Clock period: 12ns                                      ※ A-3

AREA:
TOTAL Area      : 40831                                ※ B
Sequential       : 12435                                ※ B-1
Combinational    : 28396                                ※ B-2
Memory          : -                                     ※ B-3
                                         : -                                     ※ B-4

filter (TOTAL):
REG   : 12435 (30%)                                 ※ B-5
FU    : 13065 (31%)                                 ※ B-6
MUX   : 11727 (28%)                                 ※ B-7
DEC   : 0 ( 0%)                                    ※ B-8
MISC  : 3604 (8%)                                   ※ B-9
MEM   : -                                         ※ B-10
count: -                                         -           ※ B-11

filter (w/o sub modules):                            ※ B-11
REG   : 9240
FU    : 1649
MUX   : 8068
DEC   : 0
MISC  : 2324
MEM   : -
count: -                                         -           ※ B-12

apply_filter:                                       ※ B-12
REG   : 3195
FU    : 11416
MUX   : 3659
DEC   : 0
MISC  : 1280
MEM   : 0
count: x 1                                         ※ B-13

Controller:
TOTAL STATES : 12 (+Reset)                         ※ C
#FSM        : 1                                     ※ C-1
STATES/FSM   : 12 (+Reset)                         ※ C-2
FSM DECODER DELAY : 0.3ns                          ※ C-3
                                         : 0.3ns          ※ C-4

Latency index : 12                                  ※ Y-1

```

PIPELINE:	※ D
stages : 6	※ D-1
DII : 1	※ D-2

WIRE:	※ E
NET : 6890	※ E-1
PIN PAIR : 12811	※ E-2

PORt:	※ F
TOTAL : 49	※ F-1
IN : 36	※ F-2
OUT : 13	※ F-3
IN/OUT : 0	※ F-4

```

Critical path delay : 12.8ns      ※ G-1
#Violation path ( > 10ns ) : 1      ※ G-2,3
False path : Unchecked            ※ G-4
Multi-cycle path : Unchecked      ※ G-5
False loop : 0                   ※ G-6

LATCH (bit):
count : 0                         ※ G-7

Latency:                                ※ X
total:
    Type: S
    Latency: Reset + L1
    Latency index: 36
    State No.: 0, 1, 2, 3, 4, 5
    Folding loop: -
    Folding stages: - , Total folding states: - , Hazard: -
    User operator: -
    Sub loops: L1
L1:                                     ※ X-1
    Type: S
    Latency: (1 + L2 + 1 + L3) * N1 [N1 >= 1]  ※ X-2
    Latency index : 35                            ※ X-3
    State No.: 1, 2, 3, 4                        ※ X-4
    Folding loop:                                ※ X-5
    Folding stages: , Total folding states: , Hazard: Exist
                                                ※ X-,7,8, 9
    User operator: -                            ※ X-10
    Sub loops: L2, L3                          ※ X-11

L2:
    Type: F
    Latency: 1 * 8
    Latency index : 9
    State No.: 2
    Folding loop: FL1
    Folding stages: 2 , Total folding states: 2 , Hazard: None
    User operator: -
    Sub loops: -
L3:
    Type: S
    Latency: L4 * 8
    Latency index : 24
    State No.: 4, 5
    Folding loop: -
    Folding stages: - , Total folding states: - , Hazard: -
    User operator: -
    Sub loops: L4
L4:
    Type: S
    Latency: {1, 2} * N4 + 1
    Latency index : 3
    State No.: 4, 5
    Folding loop: -
    Folding stages: - , Total folding states: - , Hazard: -
    User operator: -
    Sub loops: -
FL1:                                     ※ X-12
    Root: L1: , DII: 1 , Folding states: 1 , Hazard: None
                                                ※ X-13,14,15,16
    User operator: -                            ※ X-17
    Sub loops: -                             ※ X-18

```

```

PORT:                                     ※ I
name      type bitw
-----
pixel_in      in   8
hlen_in       in   12
vlen_in       in   12
mode_in       in   2
pixel_out     out  8
pixel_in_ack out  1
hlen_in_v    out  1
vlen_in_v    out  1
mode_in_v    out  1
pixel_out_valid out 1
                                         ※ I-1

REG :                                     ※ J
used      declared      used
bit        bit count    bit   * count
-----
1          1           7           7           ※ J-1
-----
2          2           2           4
-----
4          4           1           4
-----
8          8           18          144
-----
11         11          8           88
-----
Total                                247   ※ J-2
MEM :                                     ※ K
                                         (shared/
                                         outside)
name      kind   bit   word   area   count count
-----
MEM3      R1,W1  8     2,048      - 2   0       ※ K-1
MUX :
1 bit: 2way: 2 , 3way: 1 , 5way: 2 ※ L-1
2 bit: 2way: 4 , 3way: 1
3 bit: 2way: 2
4 bit: 12way: 1
8 bit: (1way: 1), 2way: 6 , 3way: 12 , 4way: 3 , 5way: 1 , 6way: 1
11 bit: 2way: 8 , 3way: 2 , 4way: 2 , 5way: 1 , 6way: 1
total: 1,118 (# of fanins)
DECODER:                                     ※ M
4to10: 4                                     ※ M-1
FU :                                         ※ N
Fu name      area delay pipeline count
          (ns) stage
-----
add11_11u    355   1.26   -     2           ※ N-1
apply_filter 19,550 12.00   -     1
decr11_12u   191    0.63   -     1
decr11_12u_11 191    0.63   -     1
incr11_11u   170    0.65   -     2
leop11_01u_1  167    0.57   -     3
Unused FUs:                                     ※ N-2
-----
None

```

```

Timing:                                     ※ O
Path: #1                                     ※ O-1
                                              arrival
                                              delay   time logic
name      / port [signal]                   ] (ns)  (ns) stage
-----
B01_streg / dout [                         ] -    0.00  0  ※ O-2
_XOR_3397  / o1  [                         ] 0.00  0.00  0
_RNOR_2993  / o1  [ST1_10d                  ] 0.20  0.20  1
ST1_10d   / o1  [                         ] 0.00  0.20  1
_AND_3404  / o1  [apply_filter_1.apply_filter_start] 0.10  0.30  2
apply_filter_1/ o1  [apply_filter_1.apply_filter_eop] 1T    12.30  2
_AND_2532  / o1  [U_33                     ] 0.10  12.40  3
_LOGIC_3074 / o1  [                         ] 0.20  12.60  4
_NMUX_1111  / o1  [RG_hcnt_vcnt           ] 0.50  13.10  8
RG_hcnt_vcnt / din [                      ] -    13.10  8
                                              sub total
class  (ns)  ratio                           ※ O-3
-----
IN     0.00  0%
FU     12.00 91%
MUX    0.50  3%
DEC    0.00  0%
MISC   0.60  4%
MEM    0.00  0%
-----
Total 13.10
False path information:                    ※ O-4
Path #1
name      pin       signal
-----
c        -         c
sub08_08u@1 o1       sub08_08ulot
add08_08u@1 o1       add08_08ulot
Multi-cycle functional units:             ※ O-5
FU name      delay
-----
func          1.50
Total net count : 6,890                  ※ P-1
Total pin pair count : 12,811            ※ P-2
Const fanout : 209                      ※ P-3
Net count: ※Q
bit      net      bitXnet               ※ Q-1
-----
1       149      149
2       13       26
3       7        21
4       21      84
8       46      368
11      31      341
12      3       36
-----
Total           1,025                  ※ Q-2

```

Pin pair:					※ R
Fanout	bit	count	sub	total	
41	1	1	41		※ R-1
37	1	1	37		
20	1	1	20		
13	4	1	52		
13	1	3	39		
<hr/>					
1	8	22	176		
1	4	20	80		
1	3	7	21		
1	2	11	22		
1	1	72	72		
<hr/>					
Total			2,360		※ R-2
Fanout for consts:					※ S
value fanout					
0	158				※ S-1
1	51				
<hr/>					
Total	209				※ S-2
Clock fanout:					※ T
name count					
<hr/>					
CLOCK(0..0)	41				※ T-1
Reset fanout:					※ U
name count					
<hr/>					
RESET(0..0)	37				※ U-1
ST1_00d(0..0)	1				
Register fanin/fanout cone size:					※ V
Fanin: (Top 10 registers)					※ V-1
Register name cone size					
<hr/>					
RG_pixel_10(7..0)	26				※ V-2
FF_vcvt(0..0)	24				
RG_pixel_12(7..0)	24				
RG_pixel_13(7..0)	24				
RG_hcnt_vcvt(10..0)	23				
RG_35(0..0)	23				
RG_pixel_17(7..0)	23				
RG_pixel_16(7..0)	23				
RG_pixel_15(7..0)	23				
RG_pixel_11(7..0)	23				
Fanout: (Top 10 registers)					※ V-3
Register name cone size					
<hr/>					
B01_streg(3..0)	43				※ V-4
RG_13(0..0)	28				
RG_hcnt_vcvt(10..0)	25				
RG_pixel_5(7..0)	25				
RG_pixel_8(7..0)	23				
RG_vcvt(10..0)	23				
RG_hcnt_hmax(10..0)	22				
RG_hmax(10..0)	22				
RG_vmax(10..0)	22				
RG_mode(1..0)	22				

Routability:			※ W
Net name	total	max	
RG_hcnt_hmax(10..0)	110	10 (11bit)	※ W-1
RG_hcnt_vcnt(10..0)	88	8 (11bit)	
RG_hmax_1(10..0)	88	8 (11bit)	
RG_vcnt(10..0)	73	13 (1bit)	
RG_vcvt_1(10..0)	53	13 (1bit)	
B01_streg(3..0)	52	13 (4bit)	
RG_pixel_5(7..0)	48	6 (8bit)	
pixel_in(7..0)	48	6 (8bit)	
RG_vmax(10..0)	44	4 (11bit)	
RG_vmax_1(10..0)	44	4 (11bit)	
CLOCK(0..0)	41	41 (1bit)	
RG_pixel_8(7..0)	40	5 (8bit)	
RG_pixel_10(7..0)	40	5 (8bit)	
RESET(0..0)	37	37 (1bit)	
RG_hmax(10..0)	33	3 (11bit)	
hlen_in(11..0)	33	3 (11bit)	
vlen_in(11..0)	33	3 (11bit)	
RG_pixel_9(7..0)	32	4 (8bit)	
RG_pixel_12(7..0)	32	4 (8bit)	
RG_pixel(7..0)	24	3 (8bit)	
RG_pixel_1(7..0)	24	3 (8bit)	
RG_pixel_2(7..0)	24	3 (8bit)	
RG_pixel_3(7..0)	24	3 (8bit)	
RG_pixel_4(7..0)	24	3 (8bit)	
RG_pixel_6(7..0)	24	3 (8bit)	
Top 25 nets sorted by "max" (maximum fan out)			
Net name	total	max	
CLOCK(0..0)	41	41 (1bit)	※ W-2
RESET(0..0)	37	37 (1bit)	
ST1_09d(0..0)	20	20 (1bit)	
RG_vcnt(10..0)	73	13 (1bit)	
RG_vcvt_1(10..0)	53	13 (1bit)	
B01_streg(3..0)	52	13 (4bit)	
RG_13(0..0)	13	13 (1bit)	
ST1_12d(0..0)	12	12 (1bit)	
FF_vcnt(0..0)	11	11 (1bit)	
U_33(0..0)	11	11 (1bit)	
U_69(0..0)	11	11 (1bit)	
RG_hcnt_hmax(10..0)	110	10 (11bit)	
ST1_01d(0..0)	10	10 (1bit)	
ST1_04d(0..0)	9	9 (1bit)	
RG_hcnt_vcnt(10..0)	88	8 (11bit)	
RG_hmax_1(10..0)	88	8 (11bit)	
U_35(0..0)	8	8 (1bit)	
U_16(0..0)	7	7 (1bit)	
RG_pixel_5(7..0)	48	6 (8bit)	
pixel_in(7..0)	48	6 (8bit)	
RG_32(0..0)	6	6 (1bit)	
ST1_10d(0..0)	6	6 (1bit)	
RG_pixel_8(7..0)	40	5 (8bit)	
RG_pixel_10(7..0)	40	5 (8bit)	
RG_35(0..0)	5	5 (1bit)	

The output example (only 1 part) of circuit quality report of text version is also displayed when synthesized with FPGA mode.

```
Cyber RTL Synthesis report (module name : filter)
Basic Library Name: CWBSTDLIB (BLIB Version 2.00)
Clock period: 12ns
FPGA Family : StratixIV                                     ※ A-4
FPGA Device : EP4SGX70D                                      ※ A-5
FPGA Package: FC29                                         ※ A-6
FPGA Speed : -3                                           ※ A-7

ALUTs           : 1,488                                       ※ B-14
Registers      : 343                                         ※ B-15
Block memory bits : 32,768                                     ※ B-16
DSPs            : 0                                           ※ B-17

filter (w/o sub modules):                                     ※ B-18
ALUTs           : 992
Registers      : 258
Block memory bits : 32,768
DSPs            : 0

count: -
apply_filter:                                                 ※ B-19
ALUTs           : 496
Registers      : 85
Block memory bits : 0
DSPs            : 0

count: x 1
Controller:
TOTAL STATES : 12 (+Reset)
#FSM          : 1
STATES/FSM    : 12 (+Reset)
```

Table 20: Information list of Text Version Circuit Quality Report

※ A-1	Module name
※ A-2	Basic library and library version used in estimation
※ A-3	Clock cycle
※ A-4	Target Family of FPGA
※ A-5	Target Device of FPGA
※ A-6	Target Package of FPGA
※ A-7	Target Speed grade of FPGA
※ B	Circuit Area information
※B-1	Total estimated area
※B-2	Estimated area of sequential circuit
※B-3	Estimated area of combinatorial circuit
※B-4	Estimated area of memory
※B-5	Estimated area of register (percentage generally present in bracket)
※B-6	Estimated area of functional unit (percentage generally present in bracket)
※B-7	Estimated area of multiplexer (percentage generally present in bracket)
※B-8	Estimated area of decoder (percentage generally present in bracket)
※B-9	Estimated area of logical operation (percentage generally present in bracket)
※B-10	Estimated area of memory
※B-11	Estimated area only for top process (Omit only in case of top process)
※B-12	Each estimated area of child function and sub module (module name starts from CMPS_ if compensation circuit is automatically generated during synthesis)
※B-13	Instanced count
※B-14	Estimation of LUT (memory LUT is not included)
※B-15	Estimation of register
※B-16	Estimation of memory bit
※B-17	Estimation of DSP
※B-18	Each estimation only for top process (omit only in case of top process)
※B-19	Each estimated area of child function and sub module (module name starts from CMPS_ if compensation circuit is automatically generated during synthesis)
※B-20	Instanced count
※ Y-1	Latency index
※C	FSM information
※C-1	All state count
※C-2	FSM count
※C-3	FSM wise state count

⌘C-4	FSM Decoder delay
⌘D ⌘D-1 ⌘D-2	Pipeline information (shown only when pipeline synthesis used) Number of pipeline stages DII count
⌘E ⌘E-1 ⌘E-2	Wiring information (Refer to section 37.11.8.4) Total wiring count Total pin pair count
⌘F ⌘F -1 ⌘F -2 ⌘F -3 ⌘F -4	Port information Total input/ output bit count Input bit count Output bit count Input/ output count
⌘G-1 ⌘G-2 ⌘G-3 ⌘G-4 ⌘G-5 ⌘G-6 ⌘G-7	Maximum delay value requested in delay calculation (In case basic library is not specified, then "Unchecked" is displayed) (Detected false path is excluded when -Zfalse_path_script is specified) Delay violation path count (Valid only when using -Wcritical_path and with manual scheduling) Threshold delay value changing to delay violation path (within brackets) (Value specified with -Wcritical_path. Clock cycle when omitted) False path count (valid only with -Zfalse_path_script is specified) Multi cycle path count (Valid only when -Zfalse_path_script is specified) False loop count or Combinational loop count (Valid only with option -Zfalse_path_script. It is enabled by default) Used latch count
⌘I ⌘I-1	Detailed port information Detailed information on ports (port name and in/ out/ inout and bit count)
⌘J ⌘J-1 ⌘J-2	Register information (Total Flip Flop count) Used Bit width, declared bit and count and count of every register All used FlipFlop count
⌘K ⌘K-1	Memory information type of mMemory name, used, bit width, estimated area, internal memory count and external memory count
⌘L ⌘L-1	Multiplexer information Bit width and count of input 'way' count
⌘M ⌘M-1	Decoder information Type and count of decoders

⌘N	Breakdown of functional unit information
⌘N-1	Functional unit name, area, delay value, pipeline stage count used and count of functional units
⌘N-2	List of unused functional units
⌘O	<p>Detailed delay information (please refer to section 37.11.7) (Outputs just only path for which the delay is maximum. In case option – QTn# is specified, specified count is output if path having maximum delay value are more than one# number of paths with maximum delays are listed) (descending order)</p> <p>(In case library is not specified in maximum delay time, " unchecked" is displayed)</p>
⌘O-1	<p>Serial number of listed path information (default :one path is listed) (Detected false path is excluded when -Zfalse_path_script is specified)</p>
⌘O-2	<p>Name and port name of register or functional unit etc, corresponding _E.DBL signal name, delay value, signal receive time (accumulated value), logical stage count (accumulated value)</p> <p> ⚡ Multiplexer name is displayed as “_NMUX_identification number” / “_DMUX_identification number” ⚡ Logical operation is displayed as “ _logical operation name_identification number” (Identification number is an automatically allocated unique number) </p>
⌘O-3	Percentage of input delay, user functional units, multiplexer, decoder, logical operation unit and memory; present inside maximum delay path found requested.
⌘O-4	<p>False path detail information (Not displayed when -Zfalse_path_script is not specified or there is no false path)</p> <p>(Specify the number of detailed display by -QFn# option. It is 10, when not specified)</p>
⌘O-5	<p>Multi cycle path details information (Not displayed when -Zfalse_path_script is not specified or there is no multi cycle path)</p>
⌘P-1	Total net count (considering internal wiring of functional units, multiplexer and constant) (refer to section 37.11.8.4)
⌘P-2	Total pin pair count (total fan out) (considering internal wiring of functional units, multiplexer and constant) (refer to section 37.11.8.4)
⌘P-3	Total fan-out count of constants

⌘Q	Detailed breakup of wiring net count (across functional unit, multiplexer, register etc.)
⌘Q-1	Bit width, and number and bit width ×x number
⌘Q-2	Total wiring count
⌘R	Detailed breakup of pin pair count (across functional unit, multiplexer, register etc.)
⌘R -1	Fan out count, bit width and counts, pin pair count
⌘R -2	Total pin pair count
⌘S	Detailed break up of fan-out of constant
⌘S -1	Constant values and respective fan-out count
⌘S -2	Total fan-out count of constant.
⌘T	Clock fan-out information (⌘ shown only when clock exists)
⌘T -1	Clock signal name and fan-out count
⌘U	Reset fan-out information (⌘ shown only when reset exists)
⌘U -1	Reset signal name and fan-out count
⌘V	Fan-in/ fan-out information of register
⌘V -1	List of register sorted by fan-in count
⌘V -2	Register name and fan-in count
⌘V -3	List of register sorted by fan-out count
⌘V -4	Register name and fan-out count
⌘W	Wiring information (refer to section 37.11.8.5)
⌘W -1	List of signals sorted by total pin pair count
⌘W -2	List of signals sorted by maximum fan-out count
⌘W -3	Signal name and total pin pair count and maximum pin pair count ⌘ The signal from multiplexer or logical operation with no name in BDL is output latter than other signal names. The signal name of multiplexer at this time is displayed as "_MUX_identification number" / "_DMUX_identification number" Name of logical operation is displayed as "_logical operation name_identification number" (Identification number is an automatically allocated unique number) Total pin pair count of bit wise counted pin pair count for every signal respectively. Maximum value in pin pair count of bit wise counted for every signal
⌘W -4	

※W -5	respectively. The numeric value displayed in the brackets is the bit count that turned/became maximum value
※X	Loop latency information
※X-1	Loop name (L#)
※X-2	Loop classification ([S]equential/[F]olding)
※X-3	Loop latency
※X-4	Reset: Reset state L#: Latency of loop L# N#: Repeat count of L# (not limited to repeat count for in description) {n1,n2,...}: Depicts the latency for every branch when branch exists in loop Latency index (Depicts the number of cycle until output is less for smaller value in the relative index for similar description.) In case repeat count is not fixed (N#), N# is calculated as 1. In case repeat count is fixed, calculate with that value In case of folding loop, value requested in_ "(DII count × repeat count) + folding state count— DII count"
※X-5	Number of state configuring the loop
※X-6	Folding loop name in loop (FL#)
※X-7	Top loop name when there are multiple folding loop
※X-8	Number of stages of folding loop
※X-9	Number of state of folding loop (inner side loop is included)
※X-10	Existence/Nonexistence of hazards in folding loop (Exist/None)
※X-11	Function functional unit existing in loop
※X-12	Loop existing in inner side of loop
※X-13	Folding loop name (FL#)
※X-14	Loop belonging to folding loop
※X-15	DII of folding loop (Data Initiation Interval)
※X-16	Number of state of folding loop (inner side loop is not included)
※X-17	Existence/Nonexistence of hazards in folding loop (Exist/None)
※X-18	Function functional unit existing in loop
	Folding loop existing in inner part of folding loop

37.11.11 Options Concerning the Circuit Quality Report

Following table shows the options related to circuit quality report.

Option	Description
-QDt#	Specifies the threshold value for total fan-out during "Sorted by total pin pair" output

-QDm#	Specifies the threshold value for max fan-out during "Sorted by maximum fanout" output
-QDn#	Specifies the number of output for "Routabilty" (Default -QDn25)
-QDn	Outputs all "Routability"
-QFn#	Specifies the detailed display count of false path (default -QFn10)
-QFn	Outputs the entire detailed display of false path
-QRi#	Outputs register having fan-in number more than the specified value
-QRo#	Outputs register having fan-out number more than the specified value
-QRn#	Outputs high order # register with fanin/fanout report ("Register fanin/fanout cone size") of register. (Default -QRn10)
-QRn	Outputs all fanin/fanout report ("Register fanin/fanout cone size") of register
-QMa	Adds memory to total area.
-QTn#	Specifies the output number of maximum delay path report ("Timing:") (Default -QTn1)

- -QDt #: Specifies the threshold value for 'total fanout' during "Sorted by total pin pair" output. As shown below, signal having total fanout bigger than the specified value is displayed in "Sorted by total pin pair" output.

Currently, this option has been given higher priority than -QDn.

"(> threshold value)" is shown alongside title to indicate that only signals with total fanout more than threshold are shown in the list.

Example). bdltran -QDt2

Sorted by total pin pair: (> 2)		
Net name	total	max
B01_streg(0:2)	8	4 (2bit)
RG_00(0:4)	8	2 (4bit)
b(0:4)	4	1 (4bit)
a(0:4)	4	1 (4bit)
ST1_02d(0:1)	3	3 (1bit)
_NMUX_34(0:4)	4	1 (4bit)
_NMUX_36(0:4)	4	1 (4bit)

- QDm#: Specifies the threshold value of the 'max fanout' during the output of "Sorted by maximum fanout". As shown below, signal having max fanout bigger than specified value is displayed in "Sorted by maximum fanout" output.

Currently, this option has been given higher priority than -QDn.

"(> 'threshold value')" is shown alongside the title to indicate that only signals with max fanout more than threshold are shown in the list.

Example) -QDm1 is specified

Sorted by maximum fanout: (> 1)		
Net name	total	max
B01_streg(0:2)	8	4 (2bit)
ST1_02d(0:1)	3	3 (1bit)
RG_00(0:4)	8	2 (4bit)
ST1_01d(0:1)	2	2 (1bit)
ST1_03d(0:1)	2	2 (1bit)

- QDn#, -QDn : Specifies the output count of "Routability"
As shown below, the signal of the number specified from each higher level is displayed. (default -QDn25). Outputs all signal in case numbers are omitted.
However, when -QDt, -QDm are specified, they are prioritize respectively.
"(Top specified value nets)" is shown alongside the title and "(All nets)" is shown during all signal output.

Example) -QDn4 is specified

Sorted by total pin pair: (Top 4 nets)		
Net name	total	max
B01_streg(0:2)	8	4 (2bit)
RG_00(0:4)	8	2 (4bit)
b(0:4)	4	1 (4bit)
a(0:4)	4	1 (4bit)
Sorted by maximum fanout: (Top 4 nets)		
Net name	total	max
B01_streg(0:2)	8	4 (2bit)
ST1_02d(0:1)	3	3 (1bit)
RG_00(0:4)	8	2 (4bit)
ST1_01d(0:1)	2	2 (1bit)

Example) -QDn is specified

Sorted by total pin pair: (All nets)		
Net name	total	max
B01_streg(0:2)	8	4 (2bit)
RG_00(0:4)	8	2 (4bit)
b(0:4)	4	1 (4bit)
a(0:4)	4	1 (4bit)
ST1_02d(0:1)	3	3 (1bit)
ST1_03d(0:1)	2	2 (1bit)
ST1_01d(0:1)	2	2 (1bit)
ST1_00d(0:1)	1	1 (1bit)
_NMUX_34(0:4)	4	1 (4bit)
_NMUX_36(0:4)	4	1 (4bit)
_NMUX_42(0:2)	2	1 (2bit)
Sorted by maximum fanout: (All nets)		
Net name	total	max
B01_streg(0:2)	8	4 (2bit)
ST1_02d(0:1)	3	3 (1bit)
RG_00(0:4)	8	2 (4bit)
ST1_01d(0:1)	2	2 (1bit)
ST1_03d(0:1)	2	2 (1bit)
b(0:4)	4	1 (4bit)
a(0:4)	4	1 (4bit)
ST1_00d(0:1)	1	1 (1bit)
_NMUX_34(0:4)	4	1 (4bit)
_NMUX_36(0:4)	4	1 (4bit)
_NMUX_42(0:2)	2	1 (2bit)

- QRi# : Specifies the threshold value for the number of fan in during "Register fanin/fanout cone size" output. As shown below, register having fan in count bigger than the specified value is displayed in "Fanin" output.

Currently this option has been given higher priority than -QRn.

"(> 'threshold value')" is shown alongside the title to indicate that only register with total fanin count more than threshold are shown in the list.

Example) -QRi2 is specified

Fanin: (> 2)	Register name	count
	RP_b_01(0:8)	6
	RP_00_01(0:8)	6
	RP_03_02(0:5)	5
	RP_main_x_02(0:8)	5
	B01_streg(0:2)	3

- -QRo# : Specifies the threshold value for the number of fan out during "Register fanin/fanout cone size" output. As shown below, register having fan out count bigger than the specified value is displayed in "Fanout" output.

Currently this option has been given higher priority than -QRn.

"(> 'threshold value')" is shown in alongside the title to indicate that only register with total fan out count more than threshold are shown in the list.

Example) -Qro2 is specified

Fanout: (> 2)	Register name	count
	B01_streg(0:2)	9
	RP_00_01(0:8)	3
	RP_b_01(0:8)	3

- -QRn#, -QRn : Specifies the output count of "Register fanin/fanout cone size" As shown below, the signal of number specified from all higher order is displayed. (default is -QRn10). Outputs all signal in case numbers are omitted. However, when -QRi, -Qro are specified, they are prioritize respectively. The "(Top 'specified value' registers)" is shown alongside the title and "(All registers)" is displayed during all register output.

Example) -QRn3 is specified

Fanin: (Top 3 registers)	
Register name	count
RP_b_01(0:8)	6
RP_00_01(0:8)	6
RP_03_02(0:5)	5

Fanout: (Top 3 registers)	
Register name	count
B01_streg(0:2)	9
RP_00_01(0:8)	3
RP_b_01(0:8)	3

Example) -QRn is specified

Fanin: (All registers)	
Register name	count
RP_b_01(0:8)	6
RP_00_01(0:8)	6
RP_03_02(0:5)	5
RP_main_x_02(0:8)	5
B01_streg(0:2)	3

Fanout: (All registers)	
Register name	count
B01_streg(0:2)	9
RP_00_01(0:8)	3
RP_b_01(0:8)	3
RP_main_x_02(0:8)	2
RP_03_02(0:5)	2

- -QTn# : Specifies the output count of "Timing".

As shown below, when there are multiple paths, path of top N units of delay time is displayed for only specified count (N) (default is -QTn1).

Serial number "Path:# serial number" of delay path from the generated higher order is displayed.

Example) -QTn3 is specified

Timing:					arrival	logic
Path:	#1	name	/ port [signal]	delay time	stage	
a		/ o1 []	-	0.00	0	
add16_16u@1		/ o1 [add16_16u@1]	4.50	4.50	0	
_NMUX_31		/ o1 [RG_r]	0.13	4.63	1	
RG_r		/ din []	-	4.63	1	
Path:	#2	name	/ port [signal]	delay time	stage	arrival logic
RG_r		/ dout []	-	0.00	0	
add16_16u@1		/ o1 [add16_16u@1]	4.50	4.50	0	
_NMUX_31		/ o1 [RG_r]	0.13	4.63	1	
RG_r		/ din []	-	4.63	1	
Path:	#3	name	/ port [signal]	delay time	stage	arrival logic
c		/ o1 []	-	0.00	0	
_NMUX_31		/ o1 [RG_r]	0.25	0.25	2	
RG_r		/ din []	-	0.25	2	

- **-QFn#, -QFn** : Specifies the detailed display count of false path
Maximum count that displays the detailed information of false path can be specified by option – QFn#. By default the maximum count that is displayed is 10. If –QFn is specified, all detected false paths are displayed.

The serial number displayed in "Path: # Serial number" is the sequence detected.

False path information:

```

Path #1
Name          pin      signal
-----
mul08_08u@1   o1       mul08_08u1ot
right08_08u@1 o1       right08_08u1ot
div08_08u@1   o1       div08_08u1ot

Path #2
Name          pin      signal
-----
sub08_08u@1   o1       sub08_08u1ot
mul08_08u@1   o1       mul08_08u1ot
right08_08u@1 o1       right08_08u1ot

Path #3
Name          pin      signal
-----
add08_08u@1   o1       add08_08u1ot
sub08_08u@1   o1       sub08_08u1ot
mul08_08u@1   o1       mul08_08u1ot

```

- Qma : Adds memory to total area

By default, memory area is not included in total area and displayed separately (**Figure 125**). In case total area needs to be displayed with memory area included, and then add –QM_A option (**Figure 126**).

AREA	TOTAL Area					Memory	
	55138						
	Sequential	Combinational					
	15015	40123					
Design	count	REG	FU	MUX	DEC	MISC	MEM
filter (TOTAL)	-	15015 (27%)	21756 (39%)	14342 (26%)	0 (0%)	4025 (7%)	1000
filter (w/o sub modules)	-	10430	2075	10711	0	2718	1000
apply_filter	x 1	4585	19681	3631	0	1307	0

Figure 125: Summary of circuit area report (without -QM_A)

AREA		TOTAL Area					
		Sequential		Combinational			Memory
		15015		40123			1000
Design	count	REG	FU	MUX	DEC	MISC	MEM
filter (TOTAL)	-	15015 (26%)	21756 (38%)	14342 (25%)	0 (0%)	4025 (7%)	1000 (1%)
filter (w/o sub modules)	-	10430	2075	10711	0	2718	1000
apply_filter	x 1	4585	19681	3631	0	1307	0

Figure 126: Summary of circuit area report (with -QMa)

QOR text version summary (without -QMa)

```

AREA:
TOTAL Area      : 55138
Sequential       : 15015
Combinational    : 40123
Memory          : 1000

```

QOR text version summary (with -QMa)

```

AREA:
TOTAL Area      : 56138
Sequential       : 15015
Combinational    : 40123
Memory          : 1000

```

However, total estimated area of standard error output and summary file (.SUMM), CSV file is not reflected in option –QMa.

37.12 State Transition Diagram file (.dty) in Graphviz Format

This file outputs the state transition of synthesis result in "Graphviz" format. This format is an open source graph drawing software developed by AT&T. Please refer to <http://www.graphviz.org> for further details on "Graphviz", and <http://www.research.att.com/sw/tools/graphviz/ref.html> for details on description format (dot language).

The _5.dty file shows the transition (label transition) before scheduling, whereas the _C.dty file shows the transition (state transition) after scheduling. State names are shown enclosed within "" while state transition is depicted using "state name 1" -> "state name 2" format, which means transition from state 1 to state 2.

In the example below, "ST_00"->"ST_01" in third line depicts transition from state ST_00 to state ST_01.

```
digraph fsm {
    size = "8, 11"
    "ST1_00" -> "ST1_01";
    "ST1_01" -> "ST1_02";
    "ST1_02" -> "ST1_03";
    "ST1_03" -> "ST1_04";
    "ST1_04" -> "ST1_05";
    "ST1_05" -> "ST1_06";
    "ST1_06" -> "ST1_03";
}
```

When this file is opened using 'dotty' command included in Graphviz, State transition diagram as shown in **Figure 127** gets displayed.

Please note that 'dotty' is graphical editor for 'dot' language. Refer to <http://www.research.att.com/sw/tools/graphviz/refs.html> for more details about its usage procedure.

```
% dotty foo_C.dty
```

Figure 127: Example of dotty execution

37.13 Power consumption evaluation file for FPGA

(_pwr.csv,_pwr.xpe)

In this section, power consumption evaluation file regarding FPAG for entering into early power estimator of Xilinx and Altera is explained.

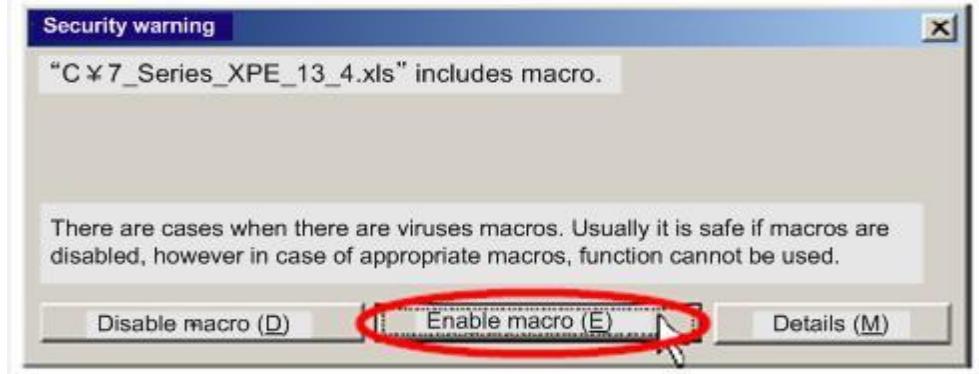
When library for FPGA is specified by BLIB or FLIB, power consumption evaluation file for FPAG is generated at the time of behavioral synthesis. In case Xilinx is selected, the end of file name is output by _pwr.xpe and when Altera is selected, the end of file name is output by _pwr.csv.

The spread sheet of early power estimator which can be downloaded for free from the home page of Xilinx and Altera is used and the method for carrying out power consumption evaluation of FPGA is explained.

37.13.1 Xilinx Power consumption Evaluation (XPower Estimator)

Spreadsheet of early power estimator of Xilinx only supports Virtex7, Kintex7, Artix7 and file input can be done from outside.

1. Download the spreadsheet for Virtex7, kintex7, Artix7 from the homepage of Xinlinx (Only the spreadsheet for Virtex7, kintex7, Artix7 can import the xpe file. The spreadsheet for other devices cannot be used)
2. Open spreadsheet by Excel. At this time, it is necessary to enable the macros.



3. Import the generated _pwr.xpe file.

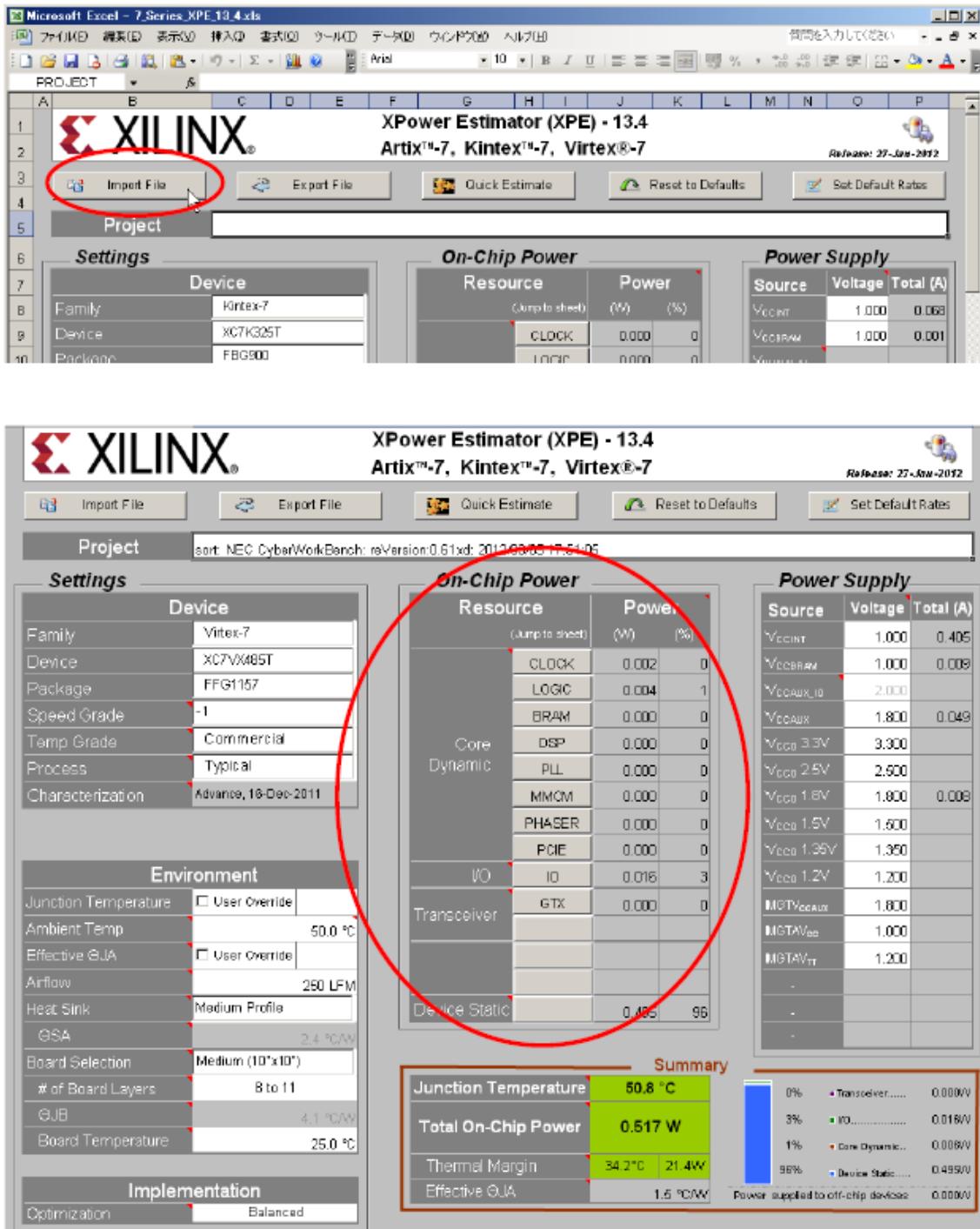
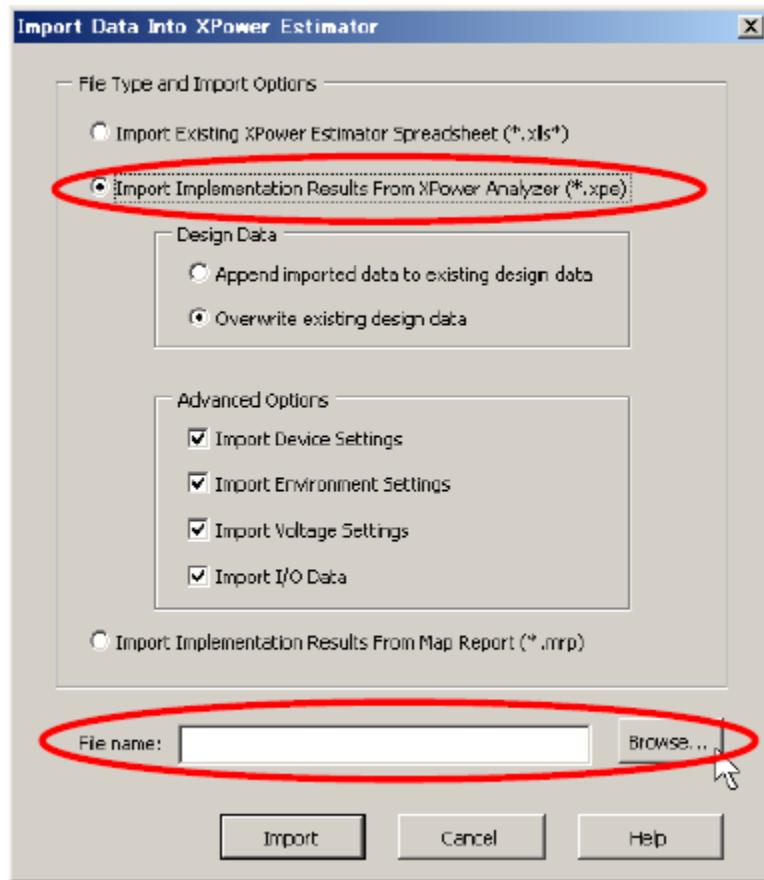


Figure 128: Result of XPower Estimator of Xilinx



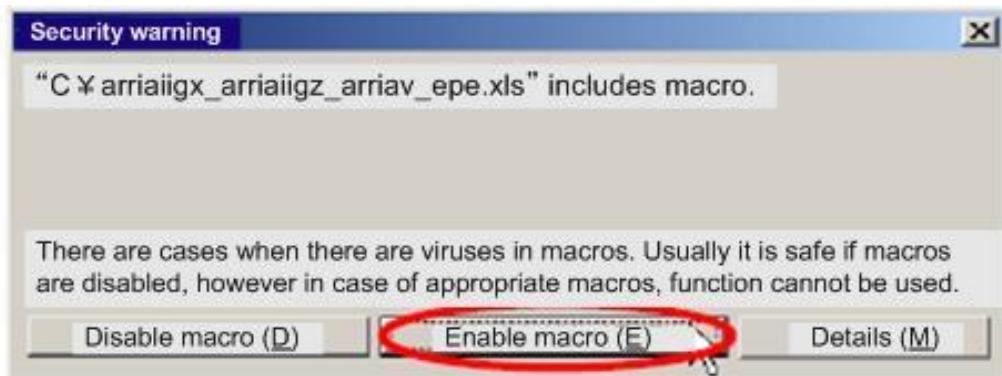
In **Figure 128**, the result of importing xpe file in spreadsheet is shown.

In the generated xpe file, items of CLOCK, LOGIC, BRAM, DSP are set by default toggle rate. Power consumption can be evaluated by changing the information of toggle rate, PLL, PCIE etc.

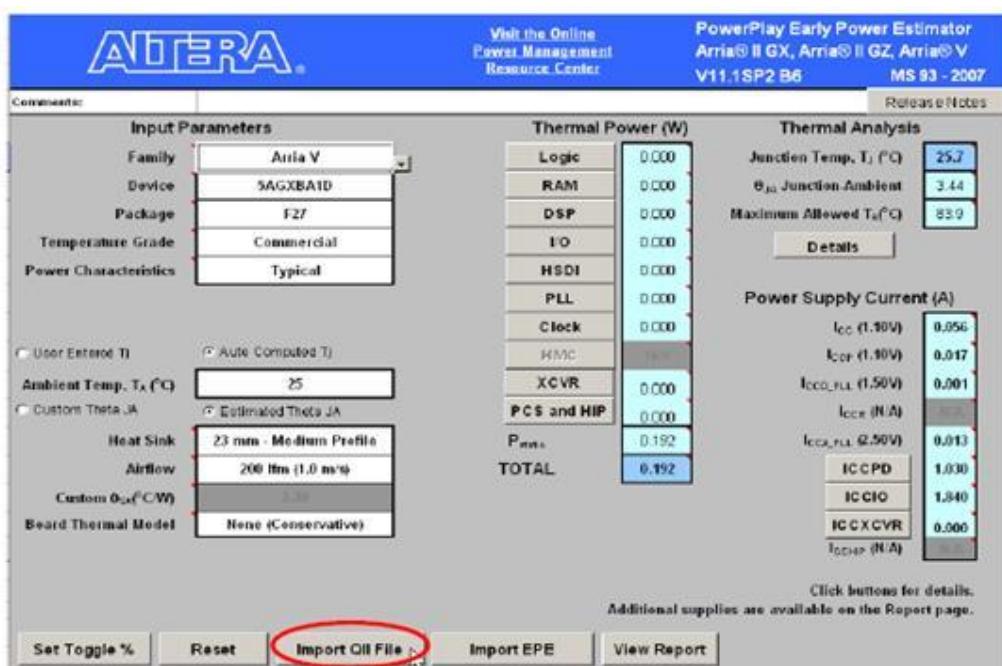
37.13.2 Altera Power consumption evaluation (PowerPlay Early Power Estimator)

Power consumption of Altera uses spreadsheet as in the case with Xilinx. _pwr.csv file is generated when Altera is selected.

1. Download the spreadsheet of target device from the home page of Altera. Caution is required as different spreadsheets are there for every device.
2. Open spreadsheet by Excel. At this time, it is necessary to enable macros.



3. Import the generated _pwr.csv file.



The result of importing _pwr.csv file in spreadsheet is shown in **Figure 129**. In the generated _pwr.csv file, items of LOGIC, RAM, DSP, I/O are set by default toggle rate. Power consumption can be evaluated by changing the information of toggle rate, PLL, HSDI etc.

37.13.3 Important Points

- Since default value is set in toggle rate, it is necessary to change for evaluating the power consumption.
- In case of hierarchical settings, the value which does not include the value within the sub module is set by the value which has been set.
- The bit width of memory used in the behavioral synthesis is set in bit width of memory port of clock RAM. However, as the bit width which can be set in block RAM through device is already decided, it is necessary to modify the big bit width than set bit width manually.

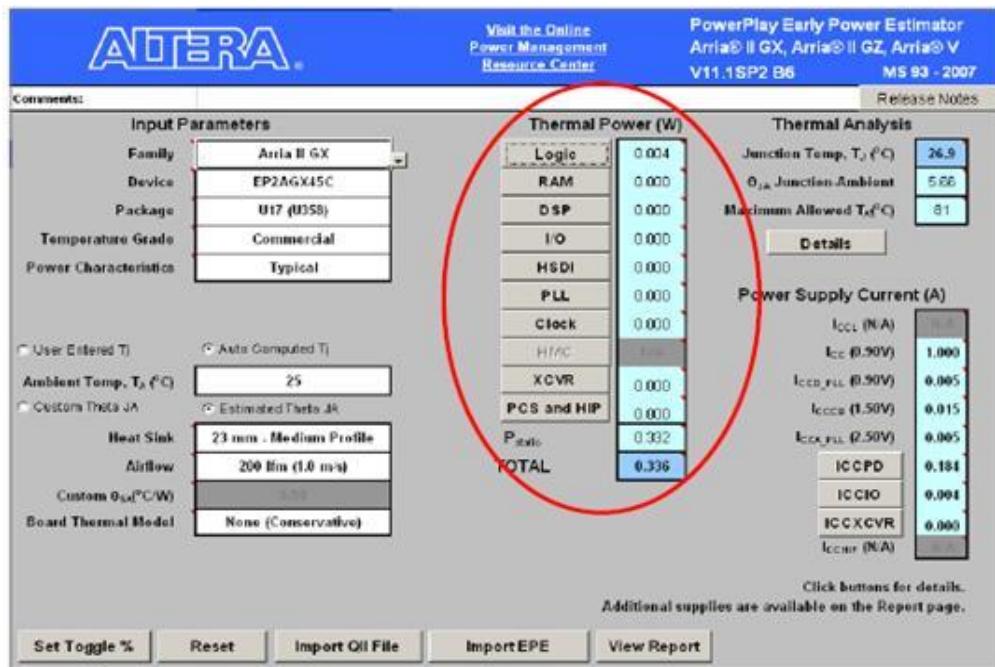


Figure 129: Result of PowerPlay Early Power Estimator of Altera

The table displays memory configuration details. The top part shows a row for MEMB16W256 with four RAMB16 instances. The bottom part shows the same row after changes, where the bit width for the first instance is explicitly set to 16-bit. Red circles highlight the 'Bit Width' column in both tables, and an arrow points from the first table's circled cell to the second table's circled cell.

Name	BRAMs	Mode	Toggle Rate	Clock (MHz)	Enable Rate	Bit Width	Write Mode	Write Rate
MEMB16W256	1	RAMB16	5.0%	100.0	100.0%	16	WRITE_FIRST	49.0%
		RAMB16	50.0%	0.0	25.0%	1	NO_CHANGE	50.0%
		RAMB16	50.0%	0.0	25.0%	1	NO_CHANGE	50.0%
		RAMB16	60.0%	0.0	25.0%	1	NO_CHANGE	50.0%

Name	BRAMs	Mode	Toggle Rate	Clock (MHz)	Enable Rate	Bit Width	Write Mode	Write Rate
MEMB16W256	1	RAMB16	5.0%	100.0	100.0%	16	WRITE_FIRST	49.0%
		RAMB16	50.0%	0.0	25.0%	1	NO_CHANGE	50.0%
		RAMB16	50.0%	0.0	25.0%	1	NO_CHANGE	50.0%
		RAMB16	50.0%	0.0	25.0%	1	NO_CHANGE	50.0%

37.14 File Output Destination Directory Specification

Option	Description
-wd= <i>directory</i>	Make file output destination directory to <i>directory</i> .

Output file is generated in the same directory where input file *filename.IFF* is present. Output file name is derived from the input file name *filename*. Please note that in case the file already exists, it will be overwritten.

While executing multiple runs of synthesis using different constraints, it may be desirable to specify the file output destination directory to directory different from input file. In that case by using the -wd option, destination directory name can be specified.

As shown in the example below, file is output to directory 'result1' which exists underneath the current directory.

```
>bdltran src/foo.IFF -c1000 -wd=result1
Output file name
    result1/foo.CSV
    result1/foo.SUMM
    result1/foo.err
    .
    .
    result1/foo_E.IFF
```

37.15 Summary

This section covered the following:

- The first type consists of output information files are those that are always output.
- The second type output information files consist of information files that are output only when certain synthesis options have been specified.
- The summary file (.SUMM) contains the summary of information generated during the synthesis.
- All "information", "warning" and "error" messages generated will be dumped to an error file (.err).
- A log file (.LOG) contains all the files used in synthesis, like "source file", "header file", and "various constraint files" etc, as well as "options" used dumped into one file.
- When "-OB" option is specified, the BDL file (-C.BDL,-9.BDL) of state transition description and BDL file (_E.BDL) of configuration description are output for the analysis of operation/behavior process.
- In the INF file, the resource allocation information (transfer information or input output information etc) of synthesized circuit is classified and outputted.
- The circuit quality report contains information pertaining to the quality of the synthesized circuit (number of variables/registers used, routability etc).
- Fan-in of register is the number of transfer source from the input port of register or primary input port till the time it gets connected.
- Fan-out of register is the number of transfer destination from the output port of register or primary output port till the time it get connected.
- The state transition diagram file (.dty) contains the state transition information of synthesis output in the "Graphviz" format.

38 CDFG Window

38.1 Overview

This section provides a list of options related to:

- State transition diagram
- Options related to control data flow graph (CDFG)
- Key operations

These options can be used in Solaris version and Linux version. However, some of the options are not supported in windows version.

38.1.1 Related Options

Option	Description
-Zcdfgdraw	Displays CDFG, State transition figure

38.2 Operations during Display

The CDFG display can be changed using the following mouse-based and menu-based operations:

- Mouse based operation
 - Information Display
Clicking mouse-button on a node will display information regarding the node in the activated console.
Clicking mouse-button on a whip will display information regarding the whip in the activated console.
 - Enlargement
A range can be specified by right clicking the mouse and dragging it to enclose the range.
The specified range will be enlarged and displayed.
 - Shifting the display range
Display range can be shifted by click and drag operation using mouse's middle button.
 - Highlighting the related nodes
If the left mouse button is clicked while the 'Shift' key is pressed, then all nodes and branch connected with that node are highlighted.
 - Highlighting the nodes allocated to same resource
If the left mouse button is clicked on a node while the 'Ctrl' key is pressed, all nodes allocated to that node and allocated resources (Input output, memory, register array, functional unit) are highlighted.

- Search based on node number
Input the node number to be displayed in the text box existing in the upper part of the windows and if the N button next to text box is pressed, then the corresponding node is displayed in the center of the window.
- Menu based operations

"File" Menu	Effect	Short cut key
Exit	CDFG display is terminated	Ctrl+Q

"Option" Menu	Effect	Short cut key
ZoomIn	Zoom in CDFG	Ctrl++
ZoomOut	Zoom out of CDFG	Ctrl+-
Fit	Whole CDFG is displayed in a size which fits the window	Ctrl+F

"Disp" Menu	Effect
NodeInfo	Displays node information on CDFG
Clear Bold	Restore the highlighted node, branch

Item displayable in "NodeInfo"	Information
Node No	Uniquely defined node number
NodeName	Node name
Node Type	Node type
bit width	Node bit width
Bit info	bit information
Register Name	Allocated register name
S Delay/T Delay	Start delay/Terminate delay of node
Ope Info	Functional unit information
Memory No	Memory number allocated in node
StepNo	Node allocating the state number
PF (priority Function)	PF value of node
DV (Distance Vector)	DV value of node
acv/ecv	acv/ecv value of node

38.3 Summary

The list of commands in refresh mode is as follows:

- o CDFG is refreshed when "D" or "d" and the CDFG display option is specified.
- o If a command starting with "H" or "h" is specified, help regarding available options will be displayed.
- o If a command starting with "Q" or "q" is specified, the refresh mode will be quit and synthesis will continue.
- o The CDFG display can be changed using mouse-based and key-based operations.

39 File specification

39.1 Overview

This section explains the formats and usage of the following 9 files. Below, extension of the files are shown in () .

- Functional unit constraint (LMT) file
- Memory constraint (MLMT) file
- Port constraint (PLMT) file
- Port relation (PREL) file
- Basic library (BLIB) file
- Functional unit library (FLIB) file
- Functional unit count (FCNT) file
- Lower module specification file (LMSPEC)
- Memory library (MLIB) file
- Memory count (MCNT) file

All of the above files can be specified with their respective options. The specified file needs to be separated by tab or space character. Multiple space character is allowed. In case the specified file name doesn't contain an extension at the end, it is assumed that the extension has been omitted and the extension is added automatically.

39.1.1 Related options

Option	Description
-lf <i>filename</i> [.LMT]	Specifies <i>filename</i> [.LMT] as first functional unit constraint file
+lf <i>filename</i> [.LMT]	Specifies <i>filename</i> [.LMT] as an additional functional unit constraint file
-ol	Do not change constraint count of functional units in manual scheduling (default)
-ou	In manual scheduling, if constraints count of a functional unit is found to be insufficient to generate desired number of units, it is increased to desired level
-lm <i>filename</i> [.MLMT]	Specifies <i>filename</i> [.MLMT] in first memory constraint file.

Option	Description
+lm <i>filename</i> [.MLMT]	Specifies <i>filename</i> [.MLMT] in additional memory constraint file
-lp <i>filename</i> [.PLMT]	Specifies <i>filename</i> [.PLMT] in port constraint file
-lpA	Do not use port constraint file.
-lpr <i>filename</i> [.PREL]	Specifies <i>filename</i> [.PREL] in a port relation file
-lb <i>filename</i> [.BLIB]	Specifies <i>filename</i> [.BLIB] in a basic library file
-lfl <i>filename</i> [.FLIB]	Specifies <i>filename</i> [.FLIB] in first functional unit library file
+lfl <i>filename</i> [.FLIB]	Specifies <i>filename</i> [.FLIB] in an additional functional unit library file
-lfc <i>filename</i> [.FCNT]	Specifies <i>filename</i> [.FCNT] in first functional unit count file
+lfc <i>filename</i> [.FCNT]	Specifies <i>filename</i> [.FCNT] in an additional functional unit count file
-ll <i>filename</i> [.LMSPEC]	Specifies <i>filename</i> [.LMSPEC] in first lower module specification file
+ll <i>filename</i> [.LMSPEC]	Specifies <i>filename</i> [.LMSPEC] in additional lower module specification file
-lml <i>filename</i> [.MLIB]	Specifies <i>filename</i> [.MLIB] in first memory library file
+lml <i>filename</i> [.MLIB]	Specifies <i>filename</i> [.MLIB] in an additional memory library file
-lmc <i>filename</i> [.MCNT]	Specifies <i>filename</i> [.MCNT] in first memory count file
+lmc <i>filename</i> [.MCNT]	Specifies <i>filename</i> [.MCNT] in an additional memory count file

39.2 Functional Unit Constraint (LMT) File

This file specifies the information and constraints regarding functional units used in the synthesis. Each line specifies one type of functional unit and contains the items specified below.

Item no.	Item name
1	Functional unit type number (IDX)
2	Functional unit name (NAME)
3	Constraint count (LIMIT)
4	Operator type (KIND)
5	Bit width (BITW)
6	Delay (DELAY)
7	Area (AREA)
8	Electric Power (POWER)

- Items are separated by one or more spaces / tabs.
- Text from # character till the end of line is treated as a comment.
- The comment whose line head starts from "#@" is treated as a library attribute for the functional unit constraint file.
- Empty lines are ignored.
- Delay unit is specified by the "#@UNIT 1/10ns" attribute in the functional unit constraint file. In case this attribute is not specified, a default value of "1/100ns" is assumed for the delay unit. Any of the following 18 types of values can be specified as a delay unit:

10000ps	10000ns
1000ps	1000ns
100ps	100ns
10ps	10ns
1ps	1ns
1/10ps	1/10ns
1/100ps	1/100ns
1/1000ps	1/1000ns
1/10000ps	1/10000ns

- The '@CLK' attribute displays the clock cycle

#IDX	NAME	LIMIT	KIND	BITW	DELAY	AREA	POWER
1	ADDSUB	3	+,-	8	10	100	
2	ADD	2	+	8	10	50	
3	CMP	4	<,>,<=,>=	8	10	30	
4	MUL	1	*	8	20	200	
5	MUL_S	1	(s) *	8	23	230	
6	MUL_SU	1	(s,u) *	8	24	240	
7	MOD	1	%	8	10	100	
8	DIV	1	/	8	10	100	
9	func1	1	@func	(32,32),32	50	500	

1. Functional unit type number (IDX)

Serial numbers are assigned to a functional unit and must be specified consecutively, like 1, 2, 3, 4, and so on.

2. Functional unit name (NAME)

Functional unit name is specified to a functional unit for identification purpose and is unique across the constraint file.

Further, in the output circuit, functional unit is generated based on these functional unit names. The name should start with an English alphabet and consist of an alphanumeric characters and “_”.

3. Constraint count (LIMIT)

Constraint count specifies the count restriction on functional units to be synthesized. In case the “-ou” option is specified in manual scheduling and if the constraint count of a functional unit is found to be insufficient to generate the desired number of units, the constraint count is increased to the desired level for synthesis.

4. Operator type (KIND)

Operator type specifies the type of operator and symbols denoting the function of the functional unit. Multiple operators can be specified by using commas “,” as a delimiter. Space or tab is not allowed around the “,” delimiter.

- Symbols can be specified using prefixes (u), (s) or (s,u).
- Prefix (u) denotes an unsigned functional unit,
- Prefix (s) denotes a signed functional unit.
- Prefix(s,u) denotes a functional unit that is capable of being operated as signed and unsigned unit depending upon the control signal.

Specification of symbol is optional. A functional unit is treated as an unsigned one if symbols are not specified.

When a function is being synthesized as a functional unit, it can be specified as “@<function_name>”. When an array is being synthesized as logic implemented, it can be specified as “@<array_name>”.

5. Bit width (BITW)

It specifies the bit-width of functional unit input. Output bit width is determined by the operation's carry over /carry truncation consideration (refer to **section 6.4**) during synthesis. Moreover, the output bit-width can be specified explicitly with the input bit-width by separating it with a "," delimiter like (8,9). Generally it is not recommended.

In case of functional unit being implemented by a function, bit-width of input arguments should be specified inside functional operation () by separating it with comma delimiter. Moreover, the bit-width of return value should be specified at the end of () (Example "(32,32),32"). Clock port, reset port, start signal, and stall signal can be specified by specifying "c", "r", "s", and "h" for a functional unit within "()" used for input bit-width specification. In this case, output will be in the order of terminal specified (for example (s,32,c,32,r),32).

In case of an array logic is realized, minimum bit width for the address specification is specified inside braces (), and after braces the bit-width of array which is output is specified (Example is "var (0:8) M[100]", which can be specified like "(7),8").

However, space or tab cannot be inserted in above specification.

6. Delay (DELAY)

It specifies maximum delay of a functional unit. Delay signifies time required by the functional unit to stabilize outputs after input is changed. Maximum delay refers to the maximum time required for the propagation of signal from each input bit to each output bit. In case the delay is different for rising and falling edge of a signal, the higher of the values is considered as a maximum delay.

Delay unit depends upon the value specified in the "#@UNIT" attribute of comment type. In case it is not specified, a default unit of "1/100ns" is assumed. During synthesis, it is converted into the delay unit of synthesis.

In delay specification, cycle specification and pipeline specification can be specified in addition to the absolute delay specification.

- **Absolute delay specification**

If only number has been specified in the delay field, it will be assumed as absolute delay. Absolute delay can be specified using an integer greater than 0.

- **Cycle specification**

Delay in cycles can be specified by suffixing "T" to a number representing the number of cycles, example "1T", which shows that the maximum delay is less than or equal to one clock cycle. A fractional number in decimal format can also be specified for cycle specification, like "0.5T".

- **Pipeline specification**

Pipeline specification can be specified in format:

"DELAYxSTAGENUM"

Example "300x3

Delay part is optional. Therefore, if it specified as "x3", it is assumed to be a three stage pipeline functional unit.

Functional unit chain effect can be specified for specifying delays. An example shown below demonstrates the functional unit chain effect.

#	IDX	NAME	LIMIT	KIND	BITW	DELAY	AREA	POWER
1		add32	2	+	32	28(14,20,x,+)	257	
2		sub32	2	-	32	29(20,17,x,+)	289	
3		shift32	1	<<	32	8(x,+,x,x)	300	
4		comp32	2	<,>	32	8(+,+,x,+)	293	

Functional unit chain effect can be specified after functional unit delay by enclosing delay specifications in braces () and separating with comma. No space or tab is permissible before and after parenthesis. In addition, space or tab cannot be inserted before and after comma.

In the above example, "(14,20,x,*)" specifies the delays for functional unit "add32" when it is chained to other functional units. Specifically, it is the delay when chained after the functional unit of functional unit type number starting from the left. (14,20,x,*) contains the following meaning.

- "14" shows that if functional unit "add32" is chained to output of "add32", then the delay of functional unit "add32" is "14"
- "20" shows that if functional unit "add32" is chained to output of "sub32" then the delay of functional unit "add32" is "20".
- "x" shows that functional unit "add32" can't be chained to the output of the functional unit "shift32".
- "+" shows that even if functional unit "add32" is chained to the output of the functional unit "comp32", delay will not change.

7. Area (AREA)

It specifies the area of the functional unit. This value is used in the calculation of the estimated area. In case omitted, it is assumed to be 0.

8. Power (POWER)

It specifies the power consumption of the functional unit. Its unit is mw. Its specification is optional. This item is for future function expansion and is not being used currently.

39.3 Memory Constraint (MLMT) File

This file specifies the information and constraints regarding memory used in synthesis.

Each line describes one type of memory using the following items in fields.

Item no	Item name
1	Memory type no.(IDX)
2	Memory name (NAME)
3	Constraint count (LIMIT)
4	Memory type (KIND)
5	Bit width (BITW)
6	Delay (DELAY)

- Items are separated by one or more spaces or tabs.
- Text after "#" character till the end of line is treated as a comment.
- Delay unit is specified by "#@UNIT" attribute of comment format. The description is same as described for functional unit constraint file. When nothing is specified, it is "1/100ns".
- Empty lines are ignored.

```
# IDX NAME LIMIT KIND BITW[xWORD] DELAY
1 memA 1 R1,W1 8 300
2 memB 2 RW1 8 400
3 memC 1 R1,W1 16x2000 500
4 memD 2 RW1 16x2000 600
5 memE 2 R1,W1 32 700
6 memF 2 RW1 32 800
```

1. Memory type number (IDX)

It refers to the serial numbers assigned to memory and must be specified consecutive numerical values like 1, 2, 3, 4, and so on.

2. Memory name (NAME)

Memory name is specified to the memory for identification purpose and is unique across the constraint file.

In addition, memory is generated based on this unique memory name in an output circuit. Therefore, it should start with an English alphabet and consist of an alphanumeric characters and "_".

3. Constrained count(LIMIT)

Constraint count specifies the count restriction on memory to be synthesized.

4. Memory type (KIND)

Memory type specifies the type and count of memory ports. It can be specified by specifying the number depicting port count along with the port type denoted by "R", "W", and "RW", such as "RW1". Here, "R" refers to the read-only port, "W" refers to the write-only port, and "RW" refers to the read-write port. The ports "R" and "W" can be specified in combination by separating them with comma, such as "R1,W1". No space or tab is allowed before and after comma.

5. Bit width and word count (BITW[x Word])

It specifies the bit width and word count of memory. Word count is specified by separating with "x". Word count specification is optional.

6. Delay (DELAY)

Delay can be specified in terms of absolute delay specification, cycle specification, or pipeline specification. Absolute delay specification or cycle specification is applicable for non pipeline memory. Similarly, pipeline specification is applicable for pipelined memory.

In certain cases, synchronous memories may require specification of appropriate options for synthesis depending on the method used for the specification of the delay (refer to **section 11.6**).

- **Non-pipelined memory**

The delay of memory is specified as shown below.

n
n "T"

Here, "n" should be an integer greater than zero. A specification using only number is assumed to be absolute delay specification. Delay unit depends on the value specified by "#@UNIT" attribute of comment format. If nothing is specified, it is "1/100ns". If a "T" is specified after the number, it is assumed that the delay is specified in the terms of cycle specification.

- **Pipelined memory**

Pipelined memory is specified using stage delays and number of stages of memory as shown below. Stage delay is specified by absolute delay. Delay unit follows the value specified by "#@UNIT" attribute of comment format. If nothing is specified, it is "1/100ns".

n "x" m
"x" m

(n: read delay. n is a natural number greater than 0. In case not specified, it is assumed to be 1)

(m: stage num. m is a natural number greater than 2)

This specification method is used when the pipelined memories support functionality is used.

- **Synchronous memory**
- Cycle Specification
“-Zmem_clocking” option, “mem_clocking” attribute should be specified.
- Pipeline specification
There is no need to specify any option, attribute for this.

39.4 Port Constraint (PLMT) File

This file specifies the information and constraints related to input output ports of a module that is used in synthesis. Each line specifies one Input Output port using the following items in fields.

Item no	Item name
1	I/O number (IDX)
2	Port name (NAME)
3	Constraint count (LIMIT)
4	Port type (KIND)
5	Bit width (BITW)
6	Delay (DELAY)

- Items are separated by one or more spaces or tabs.
- Text after “#” character till the end of line is treated as a comment.
- Delay unit is specified by “#@UNIT” attribute of comment format. The description is same as described for functional unit constraint file. When nothing is specified, it is “1/100ns”.
- Empty lines are ignored.

```
# IDX NAME LIMIT KIND BITW DELAY
1 portA 1 IN 8 300
2 portB 1 OUT 8 300
3 portC 1 INOUT 16 400
4 portD 1 IN 16 400
5 portE 1 OUT 32 400
```

1. I/O Number (IDX)

It refers to the serial numbers assigned to I/O port and must be specified with consecutive numerical value, such as 1, 2, 3, 4, and so on.

2. Port name (NAME)

Port name is given to ports for identification purpose and is unique across the constraint file. Based on this port name a port is generated in output circuit. However, port name should start with an English alphabet and consist of alphanumeric characters and “_”.

3. Constrained count (LIMIT)

Constraint count specifies the count restriction on I/O ports to be synthesized. However, currently it must have a value of 1.

4. Port type (KIND)

It specifies the types of module port. This port can be of three types, IN, OUT or inout. "IN" refers to input port, "OUT" refers to output port and "inout" refers to a bidirectional port.

5. Bit width (BITW)

It specifies the bit-width of a port.

6. Delay (DELAY)

It specifies the port delay in terms of absolute delay. Delay unit depends on the value specified by "#@UNIT" attribute of comment format. In case nothing is specified, it is "1/100ns".

39.5 Port Relation (PREL) file

This file specifies the correspondence between I/O ports of the module that are used in synthesis and I/O variables allocated to these ports. Each line specifies a relationship in terms of the following items:

Item number	Item name
1	Number (IDX)
2	Port name (NAME)
3	Signal name (SIG)

- Items are separated by one or more spaces or tabs.
- Text after "#" character till the end of line is treated as a comment.
- Empty lines are ignored.

```
# IDX  NAME    SIG
1      portA  xi
2      portC  yi,zi
3      portE  GCDout
```

1. Number (IDX)

It refers to the serial numbers assigned to allocated corresponding relationships and the number must be specified consecutive numerical values, such as 1, 2, 3, 4, and so on.

2. Port name (NAME)

It refers to a port name that is an allocation destination. In addition, it refers to a port name that has been specified in port constraint file.

3. Signal name (SIG)

It specifies the signals from the input specification that have been allocated to given port. While specifying multiple signals for one port, care should be taken so as to separate each signal by a comma. No space or tab is allowed before or after the comma.

39.6 Basic Library (BLIB) File

Basic library (BLIB) file is 1 of the constraint file required during behavioral synthesis. Currently, BLIB file format version 2.00 is supported. BLIB file contains the circuit (delay and area) information of constant table, decoder enabled multiplexer bit operation, multiplexer, decoder and register. BLIB file can be generated by using basic library file generating tool (BLIBgen), as given in **Appendix H**.

39.6.1 Basic Library (BLIB) file

39.6.1.1 Overview

- If a header line starts with "#@" and any of the following keyword is specified in continuation, the line will be treated as a library attribute that specifies the setting of the entire file:
 - VERSION, LIB, UNIT, END
- When # is specified, text after "#" character is treated as a comment and ignored till the end.
- By specifying "#@VERSION" in the first line of the BLIB file, BLIB format version can be specified. In case not specified, file will be assumed to be based on the latest format version.
- All the contents starting from keyword "#@LIB{name}" till keyword "#@END{name}" will be treated as valid specification and the name of the library is set to '*name*'.
- Contents starting from keyword "@BLIB{" till its matching parenthesis '}' are treated to be representing a single entry. Basic Library (BLIB) file consist of multiple entries of such type, where one entry represents one record.
- Starting from subsequent line to "#@LIB{name}" till first BLIB entry start (with keyword "@BLIB{}"), below specified things can be specified as library attribute format:
 - #@UNIT – specifies delay unit

Delay unit is specified by the "#@UNIT 1/10ns" attribute. In case not specified, a default value of "1/100ns" for delay unit is assumed. Any of the following 18 types of values can be specified as delay unit.

10000ps	10000ns
1000ps	1000ns
100ps	100ns
10ps	10ns
1ps	1ns
1/10ps	1/10ns
1/100ps	1/100ns
1/1000ps	1/1000ns
1/10000ps	1/10000ns

- As per the requirement, each information is described by the keyword and the corresponding value. Arbitrary number of spaces or tabs can be placed between them.

```
#@VERSION{2.00}
#@LIB{LIB}
#@UNIT 1/100ns
@BLIB {
    NAME      reg1
    KIND      reg
    BITWIDTH 1
    AREA     10
    NET      5
    PIN_PAIR 5
}
@BLIB {
    NAME      reg4
    KIND      reg
    BITWIDTH 4
    AREA     35
    NET      10
    PIN_PAIR 15
}
. . .
@BLIB {
    NAME      consttable256_8
    KIND      consttable
    SIZE     256
    BITWIDTH 8
    DELAY   1500
    AREA     600
    NET      450
    PIN_PAIR 1000
    LSTAGE   10
}
@BLIB {
    NAME      consttable256_32
    KIND      consttable
    SIZE     256
    BITWIDTH 32
    DELAY   1500
    AREA     1500
    NET      1000
    PIN_PAIR 3000
    LSTAGE   12
}
#@END{LIB}
```

39.6.1.2 Basic library element entry

1. Element name (NAME)

Element name is given for the identification purpose and must be unique across library files. Particularly it is not used.

2. Kind (KIND)

It specifies the kind of element. It can be any of the following type:

- Register : "reg"
- Decoder: "dec"
- Multiplexer: "nmux"
- Multiplexer provided with decoder: "dmux"
- Bit operation: "inv", "and", "or", "xor", "nand", "nor", "xnor"
- Constant table: "consttable"

3. Bit width (BITW)

It specifies the bit-width of an element.

In case of decoder, it refers to output bit-width. For example, when 10-bits are specified for a decoder, it refers to a decoder with 4-bit input and 10-bit output.

4. Delay (DELAY)

It specifies the delay of every element. This value is used in scheduling or delay analysis. Delay cannot be specified in a register.

5. Area (AREA)

It specifies area of every element. This value is used in the calculation of estimated circuit area. If the area of elements has not been specified in the BLIB, it is calculated using the area of related elements that have been included in specification. For example, the area of 16-bit register is estimated by doubling the area of 8-bit register.

6. NET count (NET)

It specifies the NET count for every element. This value is used for the evaluation of estimated NET count.

7. PINPAIR count (PINPAIR)

It specifies the PINPAIR count for every element. This value is used for the evaluation of estimated PINPAIR count.

8. Logical stages (LSTAGE)

It specifies logical stages of every element. This value is used for the evaluation of estimated number of logical stages. Logical stages cannot be specified in a register.

9. WAY count (WAY)

It specifies the WAY count of a multiplexer, decoder enabled multiplexer.

10. Number of inputs (INPUTNUM)

It specifies the number of inputs of bit operation other than the "NOT" operation. Using this value, the delay of reduction operation and estimated value of area can be computed.

11. Number of elements (SIZE)

It specifies number of elements of constant table.

Items (fields) that can be specified for an element kind have been summarized in the table shown below. In the following table, an entry symbol "◎" means that an item must be specified. Similarly, an entry symbol "○" means that the item can be specified. However, an entry symbol "—" means that the item cannot be specified.

	reg	dec	nmux	dmux	inv	and (⊗)	consttable
NAME	◎	◎	◎	◎	◎	◎	◎
KIND	◎	◎	◎	◎	◎	◎	◎
BITW	◎	◎	◎	◎	◎	◎	◎
DELAY	—	◎	◎	◎	◎	◎	◎
AREA	◎	◎	◎	◎	◎	◎	◎
NET	○	○	○	○	○	○	○
PIN_PAIR	○	○	○	○	○	○	○
LSTAGE	—	○	○	○	○	○	○
WAY	—	—	◎	◎	—	—	—
INPUTNUM	—	—	—	—	—	◎	—
SIZE	—	—	—	—	—	—	◎

Note: ◗ Same rule is applied to the "or", "xor", "nand", "nor, and"xnor" gates

It is mandatory to specify the following elements if version 2.00 of the BLIB file is being used:

- Register : (BITW=1)
- Decoder : (BITW=2)
- Multiplexer: (BITW=1, WAY=2)
- Multiplexer provided with decoder: (BITW=1, WAY=2), (BITW=1, WAY=16), (BITW=1, WAY=64), (BITW=1, WAY=256), (BITW=16, WAY=2), (BITW=16, WAY=16), (BITW=16, WAY=64), (BITW=16, WAY=256)
- Bit operation (inv): (BITW=1)
- Bit operation (other than inv): (INPUTNUM=2), (INPUTNUM=16), (INPUTNUM=64), (INPUTNUM=256)
- Constant table: (BITW=8, SIZE=2), (BITW=8, SIZE=16), (BITW=8, SIZE=64), (BITW=8, SIZE=256), (BITW=32, SIZE=2), (BITW=32, SIZE=16), (BITW=32, SIZE=64), (BITW=32, SIZE=256)

39.7 Functional Unit Library (FLIB) File

This file specifies the functional unit information that is used in synthesis. Functional unit library file has 2 types of format for basic functional unit and arithmetic macro functional unit (refer to [section 26.4](#)).

```

#@VERSION{1.00}
#@LIB{FOO_LIBRARY}
#@LIBTYPE{GENERATED}
#@CLK 1500
#@UNIT 10ps
@FLIB {
    NAME          ADD08           # this is comment
    KIND          +
    BITWIDTH     8
    DELAY         250
    CHAIN_FROM   ADD08 : 150
    CHAIN_FROM   ADDSUB08: 150
    CHAIN_FROM   LSHIFT08: +      # without chain effect
    CHAIN_FROM   RSHIFT08: x      # chain prohibited
    AREA          258
    SIGN          UNSIGNED
    SYN_TOOL     DC
    NET           30
    PIN_PAIR     35
    LSTAGE        8
    COMMENT       comment information regarding this function
}
@FLIB {
    NAME          ADDSUB08
    KIND          +,-
    BITWIDTH     8
    DELAY         270
    CHAIN_FROM   ADD08 : 160
    CHAIN_FROM   ADDSUB08 : 160
    CHAIN_FROM   DEFALUT : +
    AREA          300
    SIGN          SIGNED,UNSIGNED
    SYN_TOOL     DC
    NETLIST       EXIST
    NET           35
    PIN_PAIR     40
    LSTAGE        8
}
@FLIB {
    NAME          LSHIFT08
    KIND          <<
    BITWIDTH     8
    DELAY         400
    CHAIN_FROM   * : +
    AREA          500
}
@FLIB {
    NAME          RSHIFT08
    KIND          >>
    BITWIDTH     8
    DELAY         400
    CHAIN_FROM   * : x
    AREA          270
}
#@END{FOO_LIBRARY}

```

39.7.1 Overview

- In case a header line starts with "#@" and if any of following keyword is specified, the line will be treated as a library attribute that specifies setting of the entire file.
VERSION, LIB, LIBTYPE, KIND, CLK, UNIT, FPGA_FAMILY, FPGA_DEVICE,
FPGA_PACKAGE, FPGA_SPEED, END
- When # is specified, the text after "#" character till the end of line is treated as comment and is ignored. However, it excludes the following two cases:
 - # at the beginning of library attribute
 - # after comment information (COMMENT)
- By specifying "#@VERSION" in first line of header lines, FLIB format version can be specified. In case not specified, the file will be assumed to be based on the latest format version.
- All the contents starting from the "#@LIB{name}" keyword till the "#@END{name}" keyword will be treated as valid specification and the name of the library is set to 'name'.
- Contents starting from the "@FLIB{" keyword till the line having its matching parenthesis "}" are treated to be representing a single entry. A FLIB file can consist of multiple entries. However, one entry represents one functional unit type.
- Starting from subsequent line to the "#@LIB{name}" keyword till the first FLIB entry start with the "@FLIB{ }" keyword, the following information can be specified in the form of library format:
 - #@LIBTYPE – Specification of FLIB file format
#@LIBTYPE{STANDARD} is assumed to be specification of standard FLIB. In case of character string other than STANDARD, it is assumed as normal FLIB (FLIB other than standard FLIB).
 - #@KIND – Specification of functional unit type registered in FLIB file
#@KIND{BASIC_OPERATOR} indicates a FLIB for basic functional unit. Basic functional unit refers to a functional unit necessary in normal behavioral synthesis for addition or subtraction, etc. #@KIND{ARITH_MACRO_OPERATOR} indicates a FLIB for arithmetic macro functional unit. In case specification is omitted, then it is assumed to be FLIB for basic functional unit.
 - #@CLK – Specification of Clock cycle for FLIBgen
This is used to specify a clock cycle to "functional unit library delay area setting tool (FLIBgen)". (Refer Appendix I)
 - #@UNIT – Specification of delay unit
Delay unit is specified by attribute "#@UNIT 1/10ns". In case not specified, a default value of "1/100ns" for delay unit is assumed. Any of following 18-types can be specified as delay unit.

10000ps	10000ns
1000ps	1000ns
100ps	100ns
10ps	10ns
1ps	1ns
1/10ps	1/10ns
1/100ps	1/100ns
1/1000ps	1/1000ns
1/10000ps	1/10000ns

- #@FPGA_FAMILY— Specifies the target family of FPGA
 - #@FPGA_DEVICE— Specifies target device of FPGA
 - #@FPGA_PACKAGE— Specifies target package of FPGA
 - #@FPGA_SPEED— Specifies target speed grade of FPGA
- A keyword that indicates some information can be followed by specifying the setting after inserting spaces or tabs of arbitrary number. Multiple numbers of spaces or tabs are possible.
 - Only one type of information is specified in one line. A setting is specified in one line, and no carriage return or break is allowed in the middle.
 - There is no restriction in the order of information specification.

39.7.2 Functional Unit Entry

For every entry of the functional unit library (FLIB) file, the following information is specified:

1. Functional unit name (NAME)

This specifies the functional unit name of every entry. In one functional unit library (FLIB) file, duplicated names are not permitted. This is used while specifying the functional units in the functional unit count (FCNT) file. Moreover, as a module name is generated which is based on the functional unit name, it is necessary that the name should start with an English alphabet and use only alphanumeric characters and “_”.

2. Operator type (KIND)

This specifies the type of operator that exhibits function of the functional unit.

Regarding the FLIB for basic functional unit, multiple operators can be specified by using commas “,” as delimiter. Space or tab is not allowed around the “,” delimiter.

When a function is being synthesized as a functional unit, it can be specified as “@<function_name>”. When an array is being synthesized as a combinational logic, it can be specified like “@<array_name>”.

Operational expression can be specified regarding the FLIB for arithmetic macro functional unit. Currently, sign specifies the unsigned, signed by u, s respectively and bit width is specified by numbers. The priority of operations can be specified by (). For example the functional unit, with 4 units of variable of unsigned 8 bit generated by unsigned 8 bit is specified with $u8=u8+u8+u8+u8$.

3. Bit width (BITWIDTH)

This specifies the bit-width of the input side of the functional unit by FLIB for basic functional unit. Specification in FLIB for arithmetic macro functional unit is not possible.

Output bit-width is decided during synthesis based on the carry consideration of operation/mode specification of carry cut (refer to **section 6.4**). An output bit-width can be specified explicitly after input bit-width by using the comma separator "", such as "8,9". However, it is not recommended.

In case of functions being synthesized as a functional unit, the value inside () represents the input bit-width and the value after () denotes the bit-width of output return value (such as (32,32),32). The functional unit for which clock port, reset port, start signal, and stall signal are important, the following can be specified within the "()" of input bit-width.:

- "c" in the case of clock port
- "r" in the case of reset port
- "s" in the case of start signal
- "h" in the case of stall signal

During synthesis, the ports are outputted in the order of specification (example (s,32,c,32,r),32).

In case of specification for array logic being realized, minimum bit width for representing the address which changes to input bit width is specified inside braces ().The array bit width which changes to output is specified after the braces "()". (Example an array "var (0:8) M[100]", can be specified like "(7),8").

No space or tab insertion is allowed in above specification.

4. Delay (DELAY)

It specifies the maximum delay of functional unit.

Delay unit depends upon the value specified in the "#@UNIT". In case the value is not specified, a default unit of "1/100ns" is assumed. During synthesis, it is converted into the delay unit of synthesis.

In delay specification, cycle specification and pipeline specification can be specified apart from absolute delay specification.

- **Absolute delay specification**

In case only number has been specified at the delay field, it will be assumed as absolute delay. In Absolute delay, an integer greater than zero can be specified.

- **Cycle specification**

Delay in cycles can be specified by suffixing "T" to a number representing the "number of cycles", example "1T", which shows that maximum delay is less than or equal to 1 clock cycle. A smaller number in decimal format can also be specified in cycle specification, such as "0.5T".

- **Pipeline specification**

Delay based on pipeline can be specified in format:

"DELAYxSTAGENUM"

Example "300x3

Delay part specifies the maximum delay of functional unit. Therefore, if it is specified as "x3", it is assumed to be a 3 stage pipeline functional unit.

5. Input delay of pipeline functional unit (DELAY_INPUT) (optional)

It specifies the delay from input port till the flip flop corresponding to the pipeline functional unit.

6. Output delay of pipeline functional unit. (DELAY_OUTPUT) (optional)

It specifies the delay from flip flop till the output port corresponding to pipeline functional unit.

7. Area (AREA) (optional)

It specifies area of functional unit. This value is used for evaluation of the estimated area. In case not specified, it is set to 0.

8. Chain effect specification (CHAIN_FROM) (optional)

It specifies the chain effect in FLIB for basic functional unit. It cannot be specified in the FLIB for arithmetic macro functional unit.

Chain effect specification, specifies the delay including the chain effect delimited by a colon ":" when this functional unit is chained in name of functional unit of chain target (NAME) and output of functional unit of that chain target.

If "*" is specified to the functional unit name to be chained, this specification applies for all functional units. (In such a case, if there is any other "CHAIN_FROM" specification for this functional unit, it will be treated as an error.)

If "DEFAULT" is specified in the functional unit name of the chain target, then it changes to this uniform specification for chain effect from all the functional units for which there is no specification of CHAIN_FROM.

In delay, other than absolute delay, chain prohibition without chain effect can also be specified as follows:

- If only a number has been specified, it indicates an absolute delay of functional unit including the chain effect.
- "x" indicates that it cannot be chained.
- "+" indicates that even if it is chained, the delay will not change.

9. Power (POWER) (unused)

It specifies the power consumption of a functional unit and its unit is "mW". This specification is optional. It is an item for future function enhancing and is not being used in the present version.

10. Sign specification (SIGN) (optional)

In FLIB for basic functional unit, if the function of the functional unit is different based on the difference in signs (signed, unsigned), then the attribute value can be specified by "SIGNED", "UNSIGNED", or "SIGNED,UNSIGNED".

It cannot be specified in FLIB for arithmetic macro functional unit.

- "UNSIGNED" represents an unsigned functional unit.
- "SIGNED" represents a signed functional unit.
- "SIGNED,UNSIGNED" indicates that the functional unit is capable of being operated as either signed or unsigned unit depending upon the control signal.

The sign specification can be omitted. In case it is not specified, functional unit is treated as an unsigned functional unit.

11. Net list module specification (NETLIST) (optional)

Either "EXIST" or "EMPTY" can be specified for this. In case "EXIST" has been specified, it is assumed that the description of the net list module already exists. In addition, it is assumed that the module definition is not output during the RTL generation. If EMPTY is specified, module definition is output during RTL output. In case "EXIST" is not specified, it is assumed that "EMPTY" has been specified.

12. Logic synthesis specification tool to be used (SYN_TOOL) (optional)

It is an option to specify logic synthesis tool (used by functional unit delay setting tool) individually for every functional unit. (Refer to **appendix I** for details.)

13. NET count (NET) (optional)

Specify the NET counts for the functional unit. This value is used to compute the estimated number of NET. If NET count is not specified, it is set to zero.

14. PINPAIR count (PIN_PAIR) (optional)

Specify number of PINPAIR for functional unit. This value is used to compute the estimated number of PINPAIR. If PINPAIR count is not specified, it is set to zero.

15. Logical stages count (LSTAGE) (optional)

Specify logical stages count of functional unit. This value is used to compute the estimated logical stages count. If LSTAGE count is not specified, it is set to zero.

16. Macro block count (MACRO_BLOCK) (optional)

It specifies the macro block count of functional unit. This value is incorporated in the FPGA device, which specifies the count of multiplier or DSP block. Keyword, MUL when multiplier or DSP block is displayed, RAM when block RAM is displayed is used (such as MACRO_BLOCK MUL: 8). If it is not specified, then it is set to 0.

17. DSP block description specification (DSP_MACRO) (optional)

Description for the pipeline functional unit of FPGA is specified. In case of using automatic inference description for logical synthesis tool, IP_GEN is specified when INFER, Xilinx company CORE Generator IP, Altera company Megafunction is used.

18. Reset port specification (RESET_PORT) (optional)

Existence or nonexistence of reset port can be specified. YES or NO is specified in the value.

19. Enable (stall) port specification (PIPELINE_STALL) (optional)

Existence or nonexistence of enable (stall) port can be specified. YES or NO is specified in the value.

20. Pipeline stage number search specification (STAGE_SEARCH) (optional)

It is a specification for functional unit library delay area setting tool (FLIBgen) and the existence or nonexistence of the search for pipeline stage number is specified. YES or NO is specified in the value.

21. Pipeline stall compensated circuit specification (STALL_COMPENSATION) (optional)

In pipelined functional unit, it is possible to specify the requirement of generating a pipeline stall compensated circuit. "CREATE" or "IGNORE" can be specified as a value. In case CREATE is specified, a pipeline stall compensated circuit is generated and pipeline stall input is not used. (Refer to **section 23** for details)

22. Specification for Module name of IP (IP_NAME) (optional)

Module name of IP can be specified by specification for IP connection of pipeline functional unit.

23. Specification for functional unit mode of IP (IP_OPT_MODE) (optional)

Multiplier, divider, surplus calculator mode can be specified by specification for IP connection of pipeline functional unit. SPEED, AREA or DEFAULT is specified in the value.

24. Functional unit structure of IP (IP_CONSTRUCTION) (optional)

Structure of multiplier can be specified by specification for IP connection of pipeline functional unit. MUL or LUT is specified in the value.

25. Specification of division, surplus calculation mode of IP (IP_DIV_TYPE) (optional)

Divider or surplus calculation unit mode can be specified by specification for CORE Generator of pipeline functional unit. RADIX2 is specified in the value.

26. External functional unit specification (OUTSIDE) (Optional)

When functional unit is synthesized external to the circuit, specify YES. To synthesize functional unit internal to the circuit specify NO. In case this entry is not specified, a default value of 'NO' is assumed. Refer to section **12.5** for details on external functional unit.

27. Comment (COMMENT) (optional)

The character string specified here is retained even after the synthesis. Even if "#" is used in the comment, the whole comment till the end of the line is retained as a comment value.

39.8 Functional Unit Count (FCNT) File

This file specifies the constraint for all functional units used in synthesis. Functional unit count file consists of 2 types of format for Basic functional unit and for Arithmetic macro functional unit (refer to **section 26.4**).

```
#@VERSION{1.00}
#@CNT{CONSTRAINT1}
#@CLK 1500
#@UNIT 10ps
@FCNT {
    NAME          ADD08
    LIMIT         4
    COMMENT       Comment regarding functional unit count
}
@FCNT {
    NAME          ADD08
    ALIAS         add08a # ALIAS is used by attribute ope2fu or
                  # chain effect specification
    LIMIT         1
}
@
}
#@END{CONSTRAINT1}
```

39.8.1 Overview

- If a line starts with "#@" followed by any of following keyword, it is treated as a library attribute that specifies the setting of the whole file:
VERSION, CNT, LIB, KIND, CLK, UNIT, FPGA_FAMILY, FPGA_DEVICE, FPGA_PACKAGE, FPGA_SPEED, END
- When # is specified, the text after "#" character till the end of line is treated as comment and is ignored. However it excludes the following two cases:
 - "#" character at beginning of a library attributes.
 - "#" character after comment information (COMMENT)

- By specifying "#@VERSION" in first line of header lines, the FCNT format version can be specified. In case not specified, the file will be assumed to be based on the latest format version.
- All the contents starting from "#@CNT{name}" keyword till the "#@END{name}" keyword will be treated as valid specification and the name of constraint file is set to 'name'.
- Contents starting from the "@FCNT{" keyword }" keyword till the line having its matching parenthesis "}" are treated to be representing a single entry. A FCNT file consists of multiple entries of such type, where one entry represents one functional unit's constraint count.
 - For every entry starting with "@FCNT", a corresponding functional unit must exist in functional unit library file (FLIB). Otherwise, it will be treated as an error condition.
- From the next line after "#@CNT{name}" entry, and till start of first "@FCNT{" statement, the following information can be specified in library attribute form:
 - #@KIND — Specification of FCNT file format
 - ◊ #@KIND{BASIC_OPERATOR} shows the FCNT for basic functional unit.
 - ◊ #@KIND{ARITH_MACRO_OPERATOR} shows the FCNT for arithmetic macro functional unit.

If specification is omitted, FCNT is assumed to be for basic functional unit.
 - #@CLK- (Introduced for compatibility reason but not being used anymore)
 - #@UNIT- Specification of delay unit: same as described for functional unit library file (FLIB)
 - #@FPGA_FAMILY— Specification for target family of FPGA
 - #@FPGA_DEVICE— Specification for target device of FPGA
 - #@FPGA_PACKAGE— Specification for target package of FPGA
 - #@FPGA_SPEED— Specification for target speed grade of FPGA
- A keyword that indicates some information can be followed by setting value with minimum one space. Multiple numbers of spaces or tabs are possible.
- Only one type of information is specified on one line. A setting is specified in one line, and no carriage return or break is allowed in the middle.
- There is no restriction in the order of information specification.

39.8.2 Functional Unit Count Entry

In every entry of the FCNT file, the following information records are specified:

1. Functional unit name (NAME)

This specifies functional unit name of every entry. A corresponding functional unit entry must be specified in the functional unit library file (FLIB) or lower module specification file (LMSPEC).

When no 'ALIAS' is specified, a functional unit will be generated based on given functional unit 'name'. The functional unit 'name' must start with an English alphabet and use only alphanumeric characters and "_" .

2. Alias name (ALIAS) (Optional)

Alias name specifies an alternate name for an entry. The alias name is used as the functional unit name of the entry.

Therefore, the functional unit generated in output circuit will be based on this alias name. An alias must start with an English alphabet and use only alphanumeric characters and “_”.

3. LIMIT

This specifies the functional unit count that is used as constraint during synthesis.

In manual scheduling, the functional unit count is increased to desired level for synthesis if

- The “-ou” option has been specified.
- The constraints count of a functional unit is found to be insufficient to generate the desired number of units.

4. Chain effect specification (CHAIN_FORM) (Optional)

Description is same, as for functional unit library (FLIB) file (Refer to **section 39.7**).

However, alias name (ALIAS) can be specified for functional unit name of chain target.

When chain effect is specified both in the FLIB and FCNT file, the FCNT specification will be given priority.

5. Comment (COMMENT) (Optional)

39.9 Lower module specification file

This file is generated during synthesis. It specifies the generated lower module specification (LMSPEC) in synthesis of sub hierarchy of hierarchical description (refer to **section 24**) and user-defined functional unit (refer to **section 15.5**), in the synthesis of process function that calls parent hierarchy synthesis and user defined functional unit.

```

#@VERSION{3.00}
#@CNT{fnc1}
#@KIND{BASIC_OPERATOR}
#@CLK 1000
#@UNIT 1/100ns
@OPERATOR {
    NAME fnc1
    KIND @fnc1
    BITWIDTH (s,8,8,8,c,r),8
    DELAY 3T
    AREA 500
    AREA_REG 300
    AREA_COMB 200
    AREA_FU 100
    AREA_MUX 50
    AREA_DEC 0
    AREA_MISC 50
    AREA_MEM 0
    NET 100
    PIN_PAIR 110
    SPECULATION YES
    DEFMOD {
        in unsigned ter(0:1)      fnc1_start /* s, fnc1_start */;
        in unsigned ter(0:8)      a1 /* arg, $1 */;
        in unsigned ter(0:8)      b1 /* arg, $2 */;
        in unsigned ter(0:8)      c1 /* arg, $3 */;
        out unsigned ter(0:8)     V_fnc1 /* ret, $0 */;
        clock                      CLOCK /* c, CLOCK, @clock_period =
10ns */;
        reset                     RESET /* r, RESET */;
    }
}

```

39.9.1 Overview

- If a line starts with "#@" followed by any of following keyword, it is treated as a library attribute that specifies setting of the whole file:
VERSION, CNT, KIND, CLK, UNIT, END
- Text after "#" character till the end of line is treated as comment and is ignored. However, it excludes the below mentioned two cases:
 - # at beginning of a library attributes
 - # after comment information (COMMENT)
- By specifying "#@VERSION" in first line of header lines, the LMSPEC format version can be specified. In case not specified, the file will be assumed to be based on latest format version.
- All the contents starting from "#@CNT{name}" keyword till the "#@END{name}" keyword will be treated as valid specification and the constraint name of this file is set to 'name'.
- Contents starting from "@PROCESS" {or "@OPERATOR {" till the line having its matching parenthesis "}" are treated to be representing a single entry. "@PROCESS" is used If process function is synthesized and "@OPERATOR" is used if user defined function is defined.

- The following information can be specified in the formal comment format from next to "#@CNT{name}" till first @PROCESS {} or @OPERATOR {}.
 - #@CLK – Introduced for backward compatibility. However, this is not used currently.)
 - #@KIND-Specification of FCNT format. Possible to specify "#@KIND{BASIC_OPERATOR}" only.
 - #@UNIT –Specification of delay unit: Description is similar as for functional unit library (FLIB) file.
- A keyword that indicates some information can be followed by setting value with minimum one space. Multiple numbers of spaces or tabs are possible.
- Only one type of information can be specified on one line. A setting is specified in one line and no carriage return or break is allowed in the middle.
- There is no restriction in the order of information specification.

39.9.2 Entry Content

The following information is specified for the entry of lower module specification file (LMSPEC).

1. Process name or user defined functional unit name (NAME)
2. Operator type (KIND) (only "@OPERATOR")
3. Bit width (BITWIDTH) (only "@OPERATOR")
4. Delay (DELAY) (only "@OPERATOR")
5. Total area (AREA)

Register, combinational circuit and total memory area

6. Area of register (AREA_REG)
7. Area of combinational circuit (AREA_COMB)
- Functional unit, multiplexer, decoder and total of logic operation
8. Area of functional unit (AREA_FU) (ASIC synthesis only)
9. Area of multiplexer (AREA_MUX) (ASIC synthesis only)
10. Area of decoder (AREA_DEC) (ASIC synthesis only)
11. Area of logic operation (AREA_MISC) (ASIC synthesis only)
12. Area of memory (AREA_MEM) (ASIC synthesis only)
13. Memory bit count (FPGA_MEM_BIT) (ASIC synthesis only)
14. NET count (NET)
15. PINPAIR count (PIN_PAIR)
16. Speculative execution (SPECULATION) (only "@OPERATOR")

Species YES if speculative execution of user defined functional unit is allowed and if not, specifies NO. (Refer to **section 19.3** for speculation execution.)

17. Port information (DEFMOD)
18. The following specifications are possible. However, it is not generated in the automatically generated file LMSPEC.
 - Input delay of pipeline functional unit (DELAY_INPUT)
 - Output delay of pipeline functional unit (DELAY_OUTPUT)

- Macro block count (MACRO_BLOCK)
- DSP block description specification (DSP_MACRO)
- Net list module specification list (NETLIST)
- Logic synthesis specification tool to be used (SYN_TOOL)
- Enable port specification (stall) (PIPELINE_STALL)
- Pipeline stall compensated circuit specification (STALL_COMPENSATION)
- External functional unit specification (OUTSIDE)
- Logic stage (LSTAGE)
- Implicit Exclusion (IMPLICIT_EXCLUSION)
- Comment (COMMENT)

39.10 Memory Library (MLIB) File

This file specifies the memory information used during synthesis.

```

#@VERSION{1.00}
#@LIB{FOO_LIBRARY}
@MLIB {
    NAME          SRAM_A
    KIND          RW1
    BITWIDTH     8
    WORD          100
    DELAY         x2
    DEFMOD {
        # i/o t/r   bitw   name /* kind:port,pol, setup */ ;
        #first      port   (RW)
        in   ter   (0:8) A    /* ad:1, setup=100 */ ;
        out  ter   (0:8) DO   /* rd:1 */ ;
        in   ter   (0:8) DI    /* wd:1 */ ;
        in   ter   (0:1) BE    /* clk:1 */ ;
        in   ter   (0:1) WEB   /* we:1,neg */ ;
        in   ter   (0:1) CSB   /* cs:1,neg, setup=100 */ ;
        # Additional ports
        in   ter   (0:1) TBE   /* other */ ;
        in   ter   (0:1) TEST  /* other */ ;
        in   ter   (0:1) BUB   /* other:1,neg */ ;
    }
    COMMENT       comment related to this memory
}
@MLIB {
    NAME          SRAM_B
    KIND          R1,W1
    BITWIDTH     32
    WORD          256
    DELAY         200
    AREA          300
    PIN_EXPAND   yes
    DEFMOD {
        # i/o t/r   bitw   name /* kind:port,pol, setup */ ;
        #first port (R)
        out  ter   (31..0) DO   /* rd:1 */ ;
        in   ter   (7..0) AB   /* ra:1 */ ;
        in   ter   (0..0) CSB  /* rs:1,neg */ ;
        in   ter   (0..0) BEB  /* rclk:1 */ ;
        #second port (W)
        in   ter   (31..0) DI   /* wd:2 */ ;
        in   ter   (7..0) AA   /* wa:2 */ ;
        in   ter   (0..0) CSA  /* ws:2,neg */ ;
        in   ter   (0..0) BEA  /* wclk:2 */ ;
        #other ports
        in   ter   (0..0) TBEA /* other */ ;
        in   ter   (0..0) TBEB /* other */ ;
        in   ter   (0..0) TEST /* other */ ;
        in   ter   (0..0) BUB  /* other:1,neg */ ;
    }
}
#@END{FOO_LIBRARY}

```

39.10.1 Overview

- If a line starts with "#@" followed by any of following keyword, it is treated as a library attribute that specifies setting of the whole file:
VERSION, LIB, END
- Text after "#" character till the end of line is treated as comment and is ignored. However, it excludes the below mentioned two cases:
 - # at beginning of a library attributes
 - # after comment information (COMMENT)
- By specifying "#@VERSION" in first line of header lines, the MLIB format version can be specified. In case not specified, the file will be assumed to be based on latest format version.
- All the contents starting from "#@LIB{name}" keyword till the "#@END{name}" keyword will be treated as valid specification and the library name of this file is set to '*name*'.
- Contents starting from keyword "@MLIB{" till the line having its matching parenthesis "}" are treated to be representing a single entry. A MLIB file consists of multiple entries of such type, where one entry represents one memory type.
- The following can be specified in the formal comment format from row next to @LIB{name} till first @MLIB{ }.
 - #@UNIT — Specification of delay unit: Description is similar as for functional unit library (FLIB) file.
- A keyword that indicates some information can be followed by setting value with minimum one space. Multiple numbers of spaces or tabs are possible.
- Only one type of information can be specified on one line. Except DEFMOD, a setting is specified in one line and no carriage return or break is allowed in the middle.
- There is no restriction in the order of information specification.

39.10.2 Memory Entry

For every entry of the MLIB file, following information records can be specified:

1. Memory name (NAME)

NAME specifies memory name of every entry. Duplicated name entries are not permitted in one MLIB file. This name is used to specify memory type in the memory constraint (MCNT) file. A memory module or input port will be generated based on given memory name. The memory name must start with an English alphabet and use only alphanumeric characters and "_".

2. Type (KIND)

Memory type specifies the type and numbers of memory ports. It can be specified by specifying the number depicting port count along with the port type "R", "W", and "RW", such as "RW1".

- "R" refers to read-only port
- "W" refers to write-only port
- "RW" refers to read-write port

"R" and "W" can be specified in combination by separating them with comma (example "R1,W1"). No space or tab is allowed before and after comma.

3. Bit width of data (BITWIDTH)

It specifies the data bit-width of one word of memory.

4. Memory size (WORD) (Optional)

It specifies the word count of memory.

5. Access delay (DELAY)

Delay can be specified in terms of absolute delay specification, cycle specification or pipeline specification. Absolute delay specification or cycle specification is applicable in case of asynchronous read memory (asynchronous read and synchronous write memory, asynchronous read and write memory) and pipeline specification is applicable for pipelined memory (synchronous read and write memory).

- **Asynchronous read memory (asynchronous read-synchronous write memory and asynchronous read and write memory)**

Delay of memory is specified as shown below.

n

n"T"

Here, "n" should be an integer greater than zero. A specification using only number is assumed to be an absolute delay specification. Unit of delay follows the value specified by "#@UNIT" of comment format. If nothing is specified it is "1/100ns".

If a "T" is specified after number, it is assumed that the delay is specified in the terms of cycle specification.

- **Pipelined memory**

As given below, it specifies the stage delays and number of memory stages. Stage delay is specified by absolute delay. Unit of delay follows the value specified by "#@UNIT" of comment format. If nothing is specified it is "1/100ns".

n "x" m

"x" m

(n: read delay. n is a natural number greater than 0. In case not specified, it is assumed to 1)

(m: number of stages. "m" is a natural number greater than 2)

This specification method is used when pipelined memories support functionality is used.

- **Synchronous memory**

- Cycle Specification

"-Zmem_clocking" option, "mem_clocking" attribute should be specified. Further, clock port must be specified in port information.

- Pipeline specification

There is no need to specify any option for this.

6. Synchronous write access specification (WRITE_SYNC)

For writing acces of memory, you can specify sysnchronous (YES) or asynchronous write (NO). Specify YES for asynchronous read-synchronous write memory and specify NO for asynchronous read- write memory.

7. Area (AREA) (can be omitted)

Specifies the area of memory.

8. Memory type (MEM_SYNTH) (omittable, effective only in FPGA mode)

It specifies the types of memory.

- "LUT" when realized with LUT
- "BLOCK" when realized with block memory
Or block memory name (Example, M9K block of Altera company is "M9K")
- "NONE" in case of others
- When not specified, it is assumed as block memory

9. Port information (DEFMOD) (Optional)

It specifies information regarding memory ports. Only one port information should be specified in a single line.

```
DEFMOD {
#      in/out ter/reg      bitw   name   /* pin_kind:port,pol, setup */ ;
    in     ter           (0:8)  AD     /* ad:1, setup=100          */ ;
    in     ter           (0:1)  REB    /* re:1,neg, setup=100      */ ;
    out    ter           (0:8)  RD     /* rd:1                      */ ;
}
```

One port information line consists of following 10 fields. The detail of each field is given below.

Field 1: I/O attribute: "in", "out", or "inout" can be specified.

Field 2: "ter" / "reg" attribute: Currently only "ter" can be specified.

Field 3: Bit information: Can be specified in ascending or descending order. A space is must before the parenthesis.

Field 4: port name

Field 5: separator "/"

Field 6 Port type and port number

The following keywords can be used as identifier for port types:

ad	:	address	(RW# port)
ra	:	read address	(R# port)
wa	:	write address	(W# port)
<hr/>			
dt	:	data	(RW# simultaneous reading and writing by port)
rd	:	read data	(R# port, RW# port)
wd	:	write data	(W# port, RW# port)
<hr/>			
re	:	read enable	(RW# port, R# port)
we	:	write enable	(RW# port, W# port)
<hr/>			
cs	:	chip select	(RW# port)
rs	:	read port select	(R# port)
ws	:	write port select	(W# port)
<hr/>			
be	:	byte enable	(RW# port, W# port)
<hr/>			
clk	:	memory clock	(RW# port)
		memory read	
rclk	:	clock	(R# port)
		memory write	
wclk	:	clock	(W# port)
<hr/>			
rst	:	memory reset	(RW# port)
rrst	:	memory read reset	(R# port)
		memory write	
wrst	:	reset	(W# port)

req	:	request	(RW# port)
rreq	:	read request	(R# port)
wreq	:	write request	(W# port)

ack	:	acknowledge	(RW# port)
rack	:	read acknowledge	(R# port)
wack	:	write acknowledge	(W# port)

other	:	others
-------	---	--------

The numbering rules for port number are same as specified in section "Specifying allocated destination memory type for array" (Refer to **section 20.9**). For example, in "R1,W2" one read and two write ports are generated for memory, with read port being assigned to port 1 and write ports assigned to port 2 and port 3. In case of read/write dual use port like "RW3", port number is allocated in sequence.

In a port where 'rst' is specified, system reset is connected. In this case, polarity of reset is considered.

In case "other" is specified for port type, ports are clamped to '0' in case of 'in' port, and are left open in case of 'out' port.

Field 7: separator "," (It must be omitted when 8th field is omitted)

Field 8: polarity: "pos" (active high), "neg" (active low) can be specified. Its specification is optional. In case not specified, a polarity value of "pos" is assumed.

Field 9: separator "*/" (It should be omitted when 10th field is omitted)

Field 10: setup time: Setup time of pipelined memory port can be specified in the format called "setup=#" with absolute time#. It can be omitted and when it is omitted, it is considered as 0.

Delay unit follows the value specified by "#@UNIT" attribute of comment format. When nothing is specified, it will be "1/100ns".

Field 11: separator "*/"

Field 12: separator ";"

10. Byte Enable Pattern specification (ENABLEPTN) (optional)

Specifies data bit width corresponding to 1 bit of byte enable starting from MSB.

N,N,N,N: Data is partitioned into units of N-bits each starting from MSB and it is indicated that one-bit enable (total 4-bit) exists for respective data. Bit width of total bits and data port should match.

Example:

8,8,8,8 : 32 bit is divided into 4 parts of 8 bit each and control is done by enabling 4 parts
16,16 : 32 bit is divided into 2 parts of 16 bit each and control is done by enabling 2 parts

N1,N2,N3,N4 : Data is divided in sequence as N1 bit, N2 bit, N3 bit, N4 bit, from MSB, and it is indicated that 1 bit (total 4 bits) enable exists for respective data. Bit width of data port and total bits should match.

Example:

10,6,10,6 : 32 bit is divided into 4 parts of 10 bit, 6 bit, 10 bit, 6 bit each and control is done by enabling 4 parts.

16,8,8 : 32 bit is divided into 3 parts of 16 bit, 8 bit, 8 bit each and control is done by enabling 3 parts.

N+ : Data is divided into N bit each from MSB and it is indicated that 1 bit enable exists for respective data. Bit width of data port should be N times.

Example:

8+ : M bit is divided into 8 bit each and control is done by enabling M/8 (M is 8 times)

1+ : M bit is divided into 1 bit each and control is done by enabling M.

(N1,N2)+ : Data is repeatedly divided in the sequence of N1 bit and N2 bit from MSB. It is indicated that 1 bit enable exist for respective data. Bit width of data port should be (N1+N2) times.

Example:

(10,6)+ : M bit is divided into 10 bit and 6 bit each in sequence from MSB, and control is done by (M/16 * 2) enable (M is 16 times)

(6,1,1)+ : M bit is divided into 6 bit, 1 bit, 1 bit each in sequence from MSB, and control is done by (M/8 * 3) enable (M is 8 times)

In the value of byte enable pattern, the nesting of brackets is prohibited. Further, repeated specification "+" should be only once.

Following result will be obtained, if the grammar of byte enabled pattern specification value is indicated by BNF.

Pattern specification: Bit width specification line

```
| Bit width specification "+"
| "(" bit width specification line")"""+"
```

Bit width specification line: Bit width specification

```
| bit width specification "," Bit width specification line
```

Bit width specification: [0-9]+

11. Wiring harness wrapper generation specification (PIN_EXPAND) (optional)

Specify 'YES' or yes when wrapper needs to be generated that fragments memory port from wiring harness into bit single track. Name of the bit single track is defined by appending bit-number to the port name of wiring harness.

When it is omitted and NO or no is specified, wrapper is not generated. However, if this is specified both in MLIB file as well as MCNT file, MCNT specification is given priority.

12. Comment (COMMENT) (Optional)

The character string specified here is retained even after the synthesis. Even if "#" is used in the comment, the whole comment till the end of the line is retained as comment value.

39.11 Memory Count (MCNT) File

This file specifies the constraint regarding memory that is used in synthesis.

```

#@VERSION{1.00}
#@CNT{MEM_CONSTRAINT1}
@MCNT {
    NAME      SRAM_A
    LIMIT     5
    COMMENT   Comment related to this memory
}
@MCNT {
    NAME      SRAM_A
    ALIAS    memA
    LIMIT     1
}
@MCNT {
    NAME      SRAM_A
    RESOURCE  MEM
    INSTANCE  shared.M
    ADDRESS_CALC CONCAT
}
#@END{MEM CONSTRAINT1}

```

39.11.1 Overview

- If a line starts with "#@" followed by any of following keyword, it is treated as a library attribute that specifies setting of the whole file:
VERSION, CNT, END
- The text after "#" character till the end of line is treated as comment and is ignored. However, it excludes the below mentioned two cases.
 - # character at beginning of a library attribute
 - # character after comment information (COMMENT)
- By specifying "#@VERSION{}" in first line of header lines, the MCNT format version can be specified. In case not specified, the file will be assumed to be based on latest format version.
- All the contents starting from keyword "#@CNT{name}" till keyword "#@END{name}" will be treated as valid specification and the name of this constraint file is set to 'name'.
- Contents starting from the "@MCNT{" keyword till the line having its matching parenthesis "}" are treated to be representing a single entry. A memory count (MCNT) file consists of multiple entries of such type, where one entry represents one memory type.
- A keyword that indicates some information can be followed by setting value with minimum one space. Multiple numbers of spaces or tabs are possible.
- Only one type of information can be specified on one line. A setting is specified in one line and no carriage return or break is allowed in the middle.
- There is no restriction in the order of information specification.

39.11.2 Memory count entry

In every entry of memory count (MCNT) file, there are two prominent types of specification method. First method specifies the type and constraint count of memory and specifies the type and count of memory which can be used. The other method specifies the signal in behavioral description and implementation method and explicitly specifies the allocation destination resource of particular signal.

1. In the method, where type and constraint count of memory is specified and type and count of memory which can be used is specified, following 5 types of information records can be specified.

1. Memory name (NAME)

NAME specifies memory name of every entry. Memory name existing in the memory library (MLIB) file must be specified.

2. Alias name (ALIAS) (Optional)

Alias name can be specified in an entry.

Entry specified with alias can be used as an allocated destination memory name in attribute "sig2mu" attribute of allocated resource destination of behavioral description for that alias.

As there are cases when, a memory instance or I/O port is generated in output circuit based on this alias name, then the alias name must start with an English alphabet and use only alphanumeric characters and "_".

3. Constraint count (LIMIT)

Constraint count specifies number of memories that can be used.

4. Wiring harness wrapper generation specification (PIN_EXPAND) (Optional)

Specify 'YES' or yes when it is desired to generate wrapper that fragments memory port from wiring harness into bit single track. Name of the bit single track is defined by appending bit-number to port name of wiring harness.

In case NO or no is specified, wrapper is not generated. In case it is omitted, it follows the specification of MLIB.

5. Comment (COMMENT) (Optional)

Description is same as for the memory library (MLIB) file. (Refer to **section 39.10**). In case specified in both the MLIB file and MCNT file, the specification in the MLIB is ignored and the one specified in the MCNT file is used.

2. In the method, where signal in behavioral description and implementation method is specified and allocation destination resource of particular signal is explicitly specified, the following 8 types of information record can be specified.

3. As this usage method is for defmod abstract interface, refer to "User manual hierarchical setting, Connection method of multiple module addition" for details on usage method and restrictions etc.

1. Memory name (NAME)

It specifies the memory name of entry. Memory name existing in MLIB file must be specified.

2. Instance (INSTANCE)

It specifies the signal name in behavioral description. When specified signal is shared variable, "shared" must be specified before the array name. For example, when shared array foo is specified, shared.foo must also be specified.

3. Resource (RESOURCE) (optional)

Synthesis method for signals specified with INSTANCE can be specified. You can specify any of the below. In case it is omitted, it is handled in the same way when memory (MEM) is specified.

- MEM : Synthesized as memory
- REG : Synthesized as register
- REGARY : Synthesized as registered array
- VAR : Synthesized as VAR type
- VARARY : Synthesized as array of VAR type
- TER : Synthesized as wire
- TERARY : Synthesized as wire array

4. Address calculation method (ADDRESS_CALC) (optional)

In case the variable specified in instance specification is multi-dimensional array, then the method of unrolling an array into 1 multi-dimensional array is specified.

When address calculation is performed with concatenation operation, CONCAT is specified and when address calculation is performed with multiplication, MULT is specified. (Refer to **section 11.4.3** for details on address calculation).

In case it is not specified, it is handled in the same way when address calculation (MULT) is specified through calculation.

5. Specification for priority order of port (PORTPRIORITY) (optional)

Port priority order specification can be used for 3 objectives.

First objective: It can be used for specifying which port is used in which low order hierarchy, when it is accessed to memory from low order hierarchy.

Second objective: It can be used for specifying which is to be given priority when simultaneous access occurs when written in same register array or register with multiple low order hierarchy and same memory port with multiple low order hierarchy is used.

Third objective: It can be used for restricting the use of only particular port when there are multiple memory ports.

```

PORTPRIORITY {
# port      instance      instnce-port      priority
  rw:2      inst1.ary      rw:1                  0
  rw:2      inst2.ary      rw:1                  1
}

```

```

PORTPRIORITY {
# port      instance      instnce-port      priority
  rw:1      @this          -                   0
}

```

Field 1: Memory port number

It specifies the port number of target connection.

In case of memory which is divided with read port and write port like "R1,W2", read port being assigned to port 1 (r:1) and write ports assigned to port 2 (w:2) and port 3 (w:3).

In case of register array or register, '-' is specified with a meaning of "No specification".

Field 2: Connecting instance name

It specifies the connecting instance.

When low order hierarchy instance is stated, instance name is specified.

When instance stated in INSTANCE of MCNT entry is to be specified, "@this" is specified.

Field 3: Port number of connecting instance

It specifies which port of instance specified in field 2 is to be connected.

In case of register array, register or in case of "@this" specified in field 2, '-' must be specified with a meaning of "No specification".

Field 4: Priority order

Priority order is specified with integer of 0 or more. Smaller value is given higher priority.

6. Port Connection Specification (PORTCONNECT) (optional)

```

PORTCONNECT {
#   variable expression
  @CLK1    clkA      #Connect input clkA to memory port CLK1
  @rstB    inst1.M@RST #Connect memory port RST of memory
            #M of low order hierarchy inst1 to memory
            port rstB
  @TEST    0         #Connect 0 to memory port TEST
  outB     @STATE    #Connect memory port STATE to output outB
}

```

Signal connected explicitly to clock port, reset port, other ports of memory can be specified.

In case of no specification, clock signal of circuit is connected to the clock port of memory, reset signal of circuit is connected to the reset port of memory, inactive value is input to the other input port and nothing is connected to the other output port.

Field 1: Connecting signal

It specifies the connecting signal.

In memory port name, '@' must be put before the port name.

When memory port of array allocated to memory of low order hierarchy is specified, '@' and port name is specified after signal name. For example, reset port RST of shared memory M of low order hierarchy inst1 is specified as inst1.M@RST".

Field 2: Connecting expression

It specifies the connecting expression.

Not only signal but constant number and operation can also be specified.

4 types of operation "&", "|", "^", "~" can be described.

As restriction, it is not supported in signal of large bit, structured variable, array, variables other than input output variable.

7. Wiring harness wrapper generation specification (PIN_EXPAND) (optional)
 8. Comment (COMMENT) (optional)
- 1.(NAME), 7. (PIN_EXPAND), 8. (COMMENT) are similar to method specifying the type and constraint count of memory.

39.12 Summary

This topic covered the following:

- The LMT file specifies the information and constraints regarding the functional units used in synthesis.
- The MLMT file specifies the information and constraints regarding the memory used in synthesis.
- The PLMT file specifies the information and constraints related to input-output ports of a module that is used in synthesis.
- The PREL file specifies the correspondence between I/O ports of the module that are used in synthesis and I/O variables allocated to these.
- The BLIB file specifies the information regarding registers, multiplexers and decoders etc. that are used in synthesis.
- The FLIB file specifies the functional unit information that is used in synthesis.
- The FCNT file specifies the resource constraint for all functional units used in synthesis.
- The MLIB file specifies memory information used during synthesis.
- The MCNT file specifies the constraint regarding memory that is used in synthesis.

A. Logical Synthesis Script Generation Tool

A.1. Functional Overview

LSscrgen is a script generation tool which accepts “_E.IFF” files as an input and generates a template of logical synthesis script. At the same time, any error detected during the generation will be available in the “_E.lserr” file. Currently supported logical synthesis tools are DC (Design Compiler) of Synopsys, Synplify, ISE of Xilinx, Synplify Pro and Quartus II of Altera. Further, script for PowerTheater of Apache Design Solution can also be generated.

A.2. Outline flow

LSscrgen is always executed⁸ after veriloggen⁹ or vhdlgen¹⁰. **Figure 130** shows the flow.

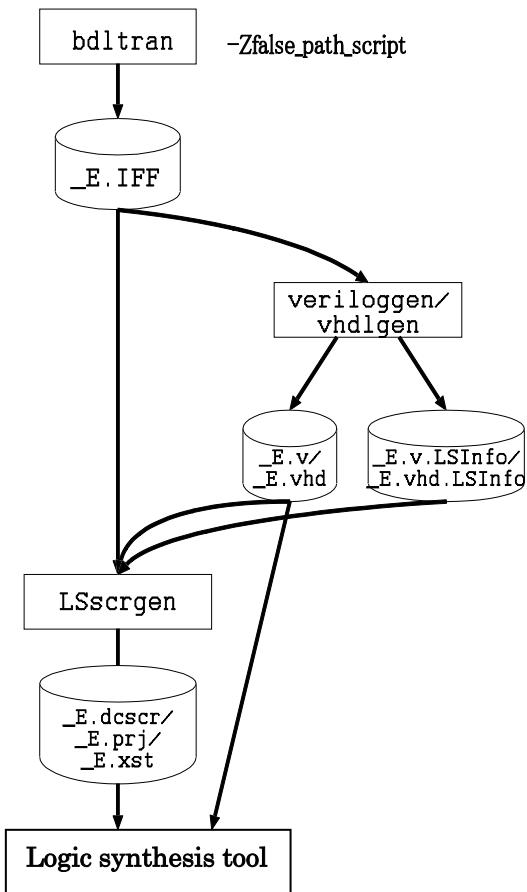


Figure 130: Logical synthesis script generation flow using LSscrgen

⁸ LSscrgen checks the Verilog-HDL/VHDL description generated by veriloggen or vhdlgen, and generates a script (template) that includes the description as the input for logic synthesis tool.

⁹ Refer to Appendix D for Veriloggen usage.

¹⁰ Refer to Appendix C for vhdlgen usage.

A.3. Usage

- Format

LSscrGen[options] _E.IFF [rtl_files]

- By default, rtl file is assumed from input _E.IFF.
- It is possible to specify rtl_file also. Multiple rtl files must be specified in argument for LSscrGen tool, as multiple rtl_file are set in logical synthesis script.

- Option

- Common options

Option	Description
-syn_tool=dc -syn_tool=sf -syn_tool=sf_pro -syn_tool=ise -syn_tool=vivado -syn_tool=qts -syn_tool=pt	Specifies a target logic synthesis tool. dc : Design Compiler sf : Synplify sf_pro : Synplify Pro ise : ISE vts : Vivado qts : Quartus II pt: PowerTheater
-vhdl	Target of generated script is assumed to be VHDL. (By default, Verilog-HDL description)
-h -H -v -V	Outputs help. Outputs version number.
-EF0	Do not generate error file (.lserr).
-lic_wait=#	As license is already obtained through another process. However when license is not obtained, repeat retry within the specified time (Time specified: minutes) [0 minutes]
-EJ -EE	Display error messages in Japanese language. Display error messages in English.
-pfile	Specify the arguments in the parameter file at the time of execution. (By default, LSscrGen.lgprm file is used)
-f=file	Specify the list file by RTL file list.
-o filename	Specifies name of output file.
-proc=name	Specifies top module name. (By default, it is name specified by process statement)
-check_rtl =[YES NO]	Check the existence of specified RTL file/s. (By default, -check_rtl=YES)

Option	Description
-comp	Assumes RTL file from _E.IFF
-out_file_name =prefix_string	Specify the prefix of file name generated at the time of logic synthesis execution. (By default it is name specified by process statement)

- o **Options for -syn_tool=dc**

Option	Description
-dc_mode=[db xg]	Specify the start mode of Design Compiler. (By default, it is -dc_mode=xg)
-dc_scr=[sh tcl]	Specify the script of Design Compiler. (By default it is -dc_scr=tcl)
-dc_ultra=[YES NO]	Use the Design Compiler Ultra.
-power_compiler=[YES NO]	Add the option for Power Compiler (-power_compiler=NO is default)
-effort=[low medium high]	Specify the degree of optimization of logic synthesis tool (By default, it is -effort=medium)
-syn_delay_unit=unit	Specify delay unit in logic synthesis library. (By default, it is 1ns)
-clock_period=#	Specify the clock period.
-timing=[YES NO]	Generate the delay constraint. (By default, it is -timing=YES)
-max_area=#	Specify the area constraint.
-max_fanout=#	Specify max. fanout constraint. (By default, it is 10)
-max_transition=#	Specify the constraint for transition time. (By default, it is 1/4th of clock period constraint during bdltran execution)
-bus_naming_style ="format"	Specify bus expansion rules. (Example -bus_naming_style="%s_%d")
-ungroup_before_rpt	Output ungroup command before report command.

Option	Description
-syn_tool_setup =filename +syn_tool_setup =filename	Specify setup file for logic synthesis tool. In multiple specification, specify +syn_tool_setup.
-syn_module_setup =filename +syn_module_setup =filename	Specify single external user script file that includes additional synthesis commands for logic synthesis. Specify multiple external user script file that includes additional synthesis commands for logic synthesis.
-area_report_file_name =filename -timing_report_file_name =filename -net_report_file_name =filename	Specify file name of area report file. Specify the file name of delay report file. Specify the file name of wiring report file.
-tx_out -tx_out=hierarchy +tx_add=filename -tx_pinform="format"	Generate timing exception path file. Generate timing exception path file with high order hierarchy specified. Add the timing exception path file. Specify port name format in timing exception path file. (By default, it is -tx_pinform="%s[*]")

- o Options for **-syn_tool=sf, -syn_tool=sf_pro**

Option	Description
-fpga_family =NAME	Specify the FPGA family name.
-fpga_device =NAME	Specify the FPGA device name.
-fpga_speed_grade =NAME	Specify the speed grade of FPGA.
-fpga_package =NAME	Specify the package name of FPGA.
-effort=[low medium high]	Specify the degree of optimization for

Option	Description
	logic synthesis. (By default, it is –effort=medium)
-syn_delay_unit=unit	Specifies the delay unit in logic synthesis library. (By default, it is 1ns)
-clock_period=#	Specify the clock period.
-timing=[YES NO]	Generate the delay constraint. (By default, it is –timing=YES)
-syn_tool_setup= <i>filename</i> +syn_tool_setup= <i>filename</i>	Specify the setup file for logic synthesis tool. In multiple specification, +syn_tool_setup is specified.
-syn_module_setup= <i>filename</i> +syn_module_setup= <i>filename</i>	Specify single external user script file that includes additional synthesis commands for logic synthesis. Specify multiple external user script file that includes additional synthesis commands for logic synthesis.
-insert_pad=[YES NO]	Insert IO pad. (By default, it is–insert_pad=NO)
-tx_out -tx_out= <i>hierarchy</i> +tx_add= <i>filename</i>	Generate timing exception path file. Generate timing exception path file after high order hierarchy is specified. Add the timing exception file named as < <i>filename</i> >.

- o **Option for –syn_tool=ise**

Option	Description
-fpga_family= <i>NAME</i> -fpga_device= <i>NAME</i> -fpga_speed_grade= <i>NAME</i> -fpga_package= <i>NAME</i>	Specify the FPGA family name. Specify the FPGA device name. Specify the FPGA speed grade Specify the package name of FPGA.
-ise_scr=all -ise_scr=synth	Generate script for ISE and CORE Generator. Generates script for ISE.

Option	Description
-ise_scr=coregen	Generate script for CORE Generator. (By default, it is -ise_scr=all)
-syn_delay_unit= <i>unit</i>	Specify the delay unit in logic synthesis library. (By default, it is 1ns)
-clock_period=#	Specify the clock period.
-timing=[YES NO]	Generate delay constraint. (By default, it is -timing=YES)
-insert_pad=[YES NO]	Insert IO pad. (By default, it is -insert_pad=NO)

- o **Options for -syn_tool=vivado**

Option	Description
-fpga_family= <i>NAME</i>	Specify the FPGA family name
-fpga_device= <i>NAME</i>	Specify the FPGA device name
-fpga_speed_grade= <i>NAME</i>	Specify the FPGA speed grade
-fpga_package <i>NAME</i>	Specify the FPGA package name.
-vivado_scr=all	Generate the scripts for vivado and IP Catalog
-vivado_scr=synth	Generate the script for vivado
-vivado_scr=ip_catalog	Generate the script for IP catalog. (by default -vivado_scr=all)
-import_ip=# +import_ip=	Specify xci file of IP that imports. (In case of multiple specifications + import_ip)
-project_mode=[YES NO]	Generate project (by default YES)
-syn_delay_unit= <i>unit</i>	Specify the delay unit of logic synthesis library (by default 1ns)
-clock_period=#	Specify the clock frequency
-timing=[YES NO]	Generate the delay constraints (by default YES)

- o **Option for -syn_tool=qts**

Option	Description
-fpga_family= <i>NAME</i> -fpga_device= <i>NAME</i> -fpga_speed_grade= <i>NAME</i> -fpga_package= <i>NAME</i>	Specify the FPGA family name. Specify the FPGA device name. Specify the FPGA speed grade Specify the package name of FPGA.
-syn_delay_unit= <i>unit</i>	Specify the delay unit in logic synthesis library. (By default, it is 1ns)
-clock_period=#	Specify the clock period.
-timing=[YES NO]	Generate delay constraint. (By default, it is timing=YES)
-timing_report_file_name= <i>filename</i>	Specify the file name of delay report file.

- o **Option for -syn_tool=pt**

Option	Description
-sim_file_name =file	Specify the activity file.
+synlib_file_name =library	Specify the library name.
-clockbuf_lib =library	Specify the library name including the buffer model.
-clockbuf_leaf =buffer_name	Specify the buffer model name (leaf).
-clockbuf_branch =buffer_name	Specify the buffer model name (branch).
-clockbuf_root =buffer_name	Specify the buffer model name (root).
-gated_clock_style =gating_cell_type	Specify the value of integrated_clock_gating_cell attribute.
-topinst= <i>name</i>	Specify the top module name of simulation hierarchy.
-opt_file_name =filename	Specify the option file name.

- Example

```
% LSscrgen top_E.IFF  
% LSscrgen -proc=main top_E.IFF main.vhd top_IP.v -comp  
% LSscrgen top_E.IFF -effort=low -syn_delay_unit 1ns  
% LSscrgen top_E.IFF -max_fanout=20 -max_transition=25.225  
% LSscrgen top_E.IFF -bus_naming_style="%s_%d"
```

A.4. Logical synthesis method with Generated file

- Logical synthesis method with script files for Design Compiler

File name	Description
<i>basename</i> .dcscr	Script file for Design Compiler

Example) Design Compiler execution method

```
% dc_shell-xg-t -f foo_E.dcscr
```

- Logical synthesis method with script files for Synplify, Synplify Pro

File name	Description
<i>basename</i> .prj	Script file for Synplify, Synplify Pro
<i>basename</i> .prj.sdc	Constraint file

Example) Synplify execution method

```
% synplify -batch -tcl foo_E.prj
```

Example) Synplify Pro execution method

```
% synplify_pro -batch -tcl foo_E.prj
```

- Logical synthesis method with script files for ISE

File name	Description
<i>basename</i> .tcl	Script file for ISE
<i>basename</i> .tcl.ucf	Constraint file
<i>basename</i> .cgp.bat	Script file for CORE Generator
<i>basename</i> .cgp	Project file for CORE Generator
<i>basename</i> .xco	Command file for CORE Generator

Example) CORE Generator execution method

```
% sh foo_E.cgp.bat
```

Example) ISE execution method

```
% xtclsh foo_E.tcl
```

- Logical synthesis method with script files for vivado

File name	Description
<i>basename.tcl</i>	Script file for ISE
<i>basename.tcl.xdc</i>	Constraint file

Example) Vivado execution method

```
% vivado -mode batch -source foo_E.tcl
```

- Logical synthesis method with script files for Quartus II

File name	Description
<i>basename.tcl</i>	Script file for ISE
<i>basename.sta.tcl</i>	Script file for timing analysis
<i>basename.sdc</i>	Constraint file

Example) Quartus II execution method

```
% quartus_sh -t foo_E.tcl
```

- Execution method with script files for PowerTheater

File name	Description
<i>basename.scr</i>	Script file for PowerTheater
<i>basename.scr.vfiles</i>	RTL file list
<i>basename.scr.clk</i>	Clock constraint file
<i>basename.scr.opt</i>	Option file

Example) PowerTheater execution method

```
% sh foo_E_pt.scr
```

A.5. Timing Exception Path

Generally, logic synthesis generates timing exception path in generated circuit. Usually, in synchronous circuit, all paths between register should be synchronized to clock cycle. However, there may be paths that need not be synchronized to clock. These types of paths are called timing exceptional paths and are categorized into multicycle paths and false paths.

Multicycle path is a path that operates at integral multiples of clock cycle. The multipliers are fixed for each path. Since, false path is the non-functional path but in case it contains many logical stages, it may become critical path.

Normally, in synthesis procedure, timing exception paths are also logically synthesized as normal clock synchronous paths, with timing constraint unnecessarily strict and lead to increase in circuit area.

In this section, timing exception path is explained. Based on this, in the next section, procedure of synthesizing appropriate circuit using logical synthesis will be explained.

A.5.1 Synthesis Procedure

Cyber can output timing exception path¹¹ that enables it to generate logical synthesis scripts. Synthesis flow considering timing exception path is shown in **Figure 130**.

- Add “-Zfalse_path_script” in “bdltran” option

This option enables the detection of timing exception path during synthesis and generates timing exception path specification script in E.IFF file.

Example)

```
% bdltran -p -Zfalse_path_script foo.IFF
```

Moreover, during the retrieval of timing exception path information, the information gathered pertaining to loops and constraints is output to the “.FPI” file. Therefore, this check is recommended.

Further, when this option is specified, information excluding the false path detected in delay period, delay path of delay information, detailed delay information of circuit quality report is output. However, in case false loop exists, restriction with non exclusion of false path exists.

- Execution of **veriloggen/ vhdlgen**

.LSInfo file is generated along with RTL file, when **veriloggen** or **vhdlgen** is executed by using _E.IFF file that includes the information of timing exceptional path. This .LSInfo file becomes the input file of **Lssrcgen**.

Example)

```
% veriloggen foo_E.IFF
```

It generates the file named as foo_E.v.LSInfo, which includes timing exception path information.

Example)

```
% vhdlgen foo_E.IFF
```

¹¹ When the “-Zzf_u_noshare_states” option is specified, this function is disabled.

It generates the file named as foo_E.vhd.LSInfo, this file includes timing exception path information

- Execution of **LSscrGen**

Logic synthesis script with E.dcsr or E.prj extension is generated when this tool is executed. At present, it is possible to generate the script that includes the specification of timing exception path for Design Compiler and Synplify.

Example)

```
% LSscrGen foo_E.IFF
```

Template of logic synthesis script is generated in the file named as foo_E.dcsr or foo_E.prj

- Editing of synthesis script if required.

Synthesis script generated by **LSscrGen** is only a template and contains a minimum set of required commands. Therefore, in case of actual application, it is recommended to change a generated script according to designing rules.

Further, synthesis script file generated by **LSscrGen** is overwritten on re-execution of LSscrGen. Hence, it is recommended to change the file name at the time of editing.

- Execution of synthesis script with logic synthesis tool

Run the tool and whenever synthesis script is executed, a logical synthesized circuit is generated by considering the timing exception path.

Example) In case of Design Compiler

```
% dc_shell-xg-t -f foo_E.dcsr
```

Example) In case of Synplify

```
% Synplify -batch foo_E.prj
```

A.5.2 Limitations

False path script generation function has the following limitations:

- Analysis Target Resources

Analysis target resource of false path script output function is limited only to the functional unit, decoder, memory, and I/O port. Registers are not covered under its objectives. Among the false path that originate from sharing of certain resources (functional unit, decoder, memory, I/O port), path in which analysis target resource does not exist at the input side or output side of corresponding source is not outputted in a false path script.

For example, in circuit of **Figure 131**, false path arising from sharing of resource F1 is

-a →F1 →F2→e

However, when "a" is not the analysis target, it is omitted from the false path specification due to this limitation of false path specification.

Whereas,

-b→F1→F2→e

is true path and hence the effect of this limitation is limited if path delay of

-b→F1

is more than path delay of

-a→F1

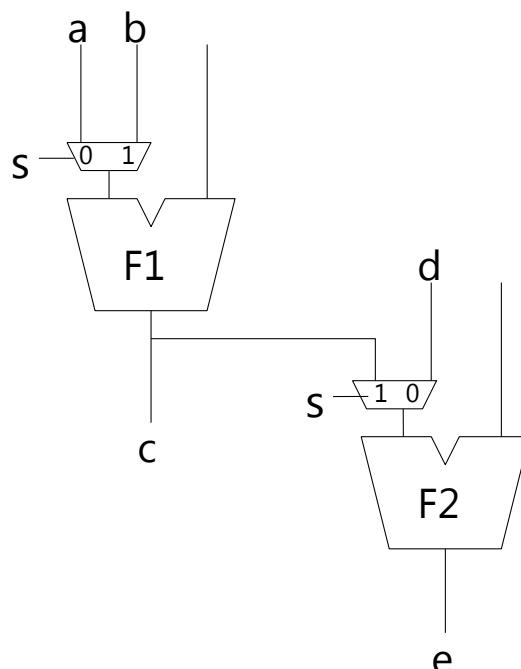


Figure 131: Example of false path

- Continuous Assignment

During the false path script generation, functional description of analysis target is considered. Thus, it does not support false path that passes through continuous assignments (:=). However, the path defined by false operation is generated only by the functional description of previous and next stage of continuous assignment. This means that path performing exclusive operation before -stage and after-stage of continuous assignment fall under this limitation.

- "-ZZfu_noshare_states" option

False path script generation function is not supported with the "-Zzfu_nonshare_states" option. Hence, it may generate undefined scripts.

A.6. Points to be noted while editing the synthesis script (Template)

While modifying the synthesis script template generated by **LSscrGen**, following points should be taken care of:

1. Description sequence of the synthesis script

Description generated as a template has a strong dependency with each other; therefore, basically it will not function if this sequence is not maintained. Causing template sequence to move out of order might produce unexpected result; therefore inserting/editing of any commands have to be done with caution.

A.7. Other Points to be noted

- According to the Design Compiler version, if bus_naming_style is specified in %s_%d format, there are cases when name collision occurs with the variable name that is auto-generated by Cyber and logic synthesis finishes with an error. Such cases can be avoided by setting bus_naming_style in the format other than %s_%d. Further, in case timing exception path is specified, it becomes necessary to avoid such cases similarly by specifying -tx_pinform option of LSscrGen.
- The following options must be specified while using the script generation for PowerTheater.

- sim_file_name
- +synlib_file_name
- clockbuf_lib
- clockbuf_leaf

Example)

```
% LSscrGen -syn_tool=pt -sim_file_name=foo_E.vcd \
+synlib_file_name=LIB_A.lib +synlib_file_name=LIB_B.lib \
-clockbuf_lib=LIB_A -clockbuf_leaf=CELL_A foo_E.IFF
```

B. BDL Input Command

B.1. Functional Overview

The BDL input command bdlpars, reads the description of BDL and C languages and analyzes the language syntax. It is a command that outputs an internal format file (IFF) which acts as an input for the "bdltran" synthesis command.

B.2. Usage

- **Format**

`bdlpars [option] filename(.bdl | .c)`

- **Options**

<code>-Dname[=def_name]</code>	<i>name</i> is assigned to <i>def_name</i> just like the preprocessing directive <code>#define</code> execution. In case <i>def_name</i> is not specified, <i>name</i> is assigned to 1.
<code>-Ipathname</code>	When searching an include file whose name does not begin with slash (/), search the directory <i>pathname</i> before searching it in the system include directory.
<code>-o filename[IFF]</code>	Make output file name as <i>filename.IFF</i>
<code>-W</code> <code>-Wall</code>	Do not output warning Output all warnings
<code>-WBDL3.0=no_warning</code> <code>-WBDL3.0=warning</code> <code>-WBDL3.0=error</code>	Generate warning regarding the modified description of the specifications in BDL version 3.0 Do not warn (default) Generate a warning message. Finish with an error when any warning is generated
<code>-Zmix_signed</code> <code>-Zunsigned</code>	Set the sign for signals (signed/unsigned) as per declaration (default) Set all signals as unsigned signals
<code>-p</code> <code>-p: filename</code>	Read the file 'bdlpars.bpprm' as parameter file. Read the ' <i>filename</i> ' as parameter file.

-Zdefault_bit_order=up	Set the variable and constant bit order whose bit ordering is not specified in ascending or descending order
-Zdefault_bit_order=down	[related] 'default_bit_order' attribute
-Zchk_bit_description	Alarm in case of bit description like (m:n)
-Zno_check_concat	Do not produce an error for concatenation of constant whose operation or bit-width is not specified.
-assert_on -assert_off	Enable assert function Disable assert function (default)
-ignore_encryption_attr	Do not produce an error even if encryption_start attribute and encryption_end attribute are left.
-OB	Output BDL file after syntax analysis (i.e. parsing).
-EFO	Do not output error file (.bperr)
-lic_wait=#	As license is already obtained through another process. However, when license is not obtained, repeat retry within the # minutes. (Time specified: minutes) [0]
-EE -EJ	Show error message in English Show error message in Japanese
-h, -H	Output command option help
-v, -V	Output version number

B.3. Alarm Output Control Option (-w, -Wall)

The BDL input command has two levels of warnings: strong and weak. Normally, only strong warnings are displayed and no weak warnings are displayed. If the “-Wall” option is specified, all warnings of both levels are displayed. No warnings are displayed if “-w” option is specified.

B.4. BDL Output Option (-OB)

-OB option is an option that outputs the BDL file after analysis. A BDL file named *filename*_

0. BDL is generated after syntax analysis (i.e. parsing)

Command line execution example:

```
% bdlpars -OB data.bdl
```

Generated BDL file name:

```
data_0.BDL
```

B.5. Error File Control Option (-EFO)

Warning, error and log are output in error file with name "circuit name.bper". If "-EFO" option is specified, the error file generation is suppressed.

B.6. Code Related Option (-Zmix_signed, -Zunsigned)

Variable sign type (signed, unsigned) is decided by signal declaration.

According to the option, the signal code can be altered forcefully. The "-Zunsigned" option is a mode where all variables are synthesized as 'unsigned'. (For details refer to **section 6.3**).

B.7. Ascending/ Descending Bit Order Specification Option (-Zdefault_bit_order)

In BDL, "int" or "char" type are simple representation of bit specification, showing 32-bit width or 8-bit width respectively. In the description "int a"; a bit width of 32 bits is assigned to variable "a" which does not have any ascending or descending bit order specification.

Therefore, the "-Zdefault_bit_order" option is used to specify the ascending/ descending order of the variables or constants for which no ascending/descending order of bits has been specified. To specify the ascending or the descending order, "up" or "down" is specified after the "=" of the option respectively.

```
Example)
In case -Zdefault_bit_order=down option is specified
int x; -----> ter(31..0) x;
char x; -----> ter(7..0) x;
char x; -----> ter(7..0) x;
struct {
    reg int is_keyword : 4; -----> reg(3..0) is_keyword;
    var int is_extern : 2; -----> var(1..0) is_extern;
    ter int is_static : 2; -----> ter(1..0) is_static;
} flags;
enum abc {A, B, C, D, E} x; -----> ter(2..0) x;
x(0:4) = 0xA; -----> x = 14(3..0);
```

B.8. Ascending/ Descending Bit Order Specification Attribute (**default_bit_order**)

This attribute is same as the "-Zdefault_bit_order" option. It is an attribute that specifies the ascending/ descending order in the middle of description.

```
int a;
/* Cyber default_bit_order = down */
process main()
{
    int b;
}
int function()
{
    int c;
}
```

If the above attribute is described just before process, it gets specified for the whole description. Thus this attribute will specify ascending or descending bit order for all the variables and constants whose ascending/descending bit order specification has not been provided. In this example, bit order of all the variables a, b and c gets specified to be of descending type. However, in case the option of **Appendix B.7** and this attribute is set at the same time, attribute is given priority. However, if there is a contradiction of specification, a warning is displayed.

B.9. Checking option of Descending Bit Order Description (- **Zchk_bit_description**)

This option is for the descending order description check in case of a description for any ascending order bit reference such as (2:4) or declaration existing in the variable reference or variable declaration in the input description, a warning is generated.

B.10. Parameter File Input Option (-p, -p:)

By specifying parameter file using the "-p" option, the first line of parameter file can be added as an option.

If only "-p" option is specified, file name with the name of "bdlpars.bpprm" of the current directory is considered as a parameter file.

The "-p:" option can specify the "filename" file as a parameter file.

Example1:

```
[Parameter file bdlpars.bpprm]
-DBDL -DDUMP -OB -Zdefault_bit_order=up abc.bdl
[In case it is executed at command line.]
%bdlpars -p
```

Example2:

```
[Parameter file abc.bpprm]
-DBDL -DDUMP -OB -Zdefault_bit_order=up abc.bdl
[In case it is executed at command line]
%bdlpars -p: abc.bpprm
```

B.11. assert function option (-assert_on, -assert_off)

The case where assert condition turns false can be confirmed in formal verification, by describing void assert (int expression) function in BDL. This option interchanges, whether the assert function described in BDL is to be made verifiable with formal verification or not.

In case of verification with formal verification, it is enabled with -assert_on, and when not detected, it is disabled with -assert_off

Operations having side effect cannot be described in the assert condition.

- Operator having side effect
 - ++(Frontend/Backend), --(Frontend/Backend)
 - + =, - =, * =, / =, % =, & =, |=, ^ =, << =, >> =, =
- User function
- input/output function
- Input variable (in ter/in reg), output variable (out ter), inout variable (inout) and RESET/CLOCK signal

B.12. Error check control of attribute made for source code encryption tool

It is a functionality which encrypts a portion of the source code. (See **Section N**). Portion of source code can be encrypted by specifying encryption_start and encryption_end attribute and using the encryption tool.

To remove both attributes after encryption, below error will appear asking for encryption as attributes encryption_start and encryption_end are found in the BDL input command.

```
"foo.c", line 6: It can not be encrypted in the source code.
Attribute encryption_start/encryption_end are found
[Counter measure] Do you want to encrypt the source code or
specify the option -ignore_encryption_attr<FF013>
```

If you want to avoid above error when testing the description before encryption, specify ignore_encryption_attr option in the BDL input command.

B.13. include Path

In bdlpars, current and system include directory (\$CYBER_PATH/include) are included in the include search paths.

In case it is specified with < >, the target file is searched in system in the order of user specified –I directory, system include directory, and current directory respectively.

In case it is specified with " ", the target file is searched in the order of directory with source code, user specified –I directory, system include directory and current directory respectively.

B.14. Multiple Process and fragmented compilation

In BDL, a circuit synthesized by unique control sequence (one FSM) is called a process. Currently, multiple processes cannot be specified in one file, so each process should be specified in a different file and then synthesized.

However, in case one process is fragmented into multiple files and described, it is important to "#include" the files from the main file. This is because fragmented compilation (synthesis) can't be done. Include every file and the entire circuit can be synthesized by specifying the main file in the circuit name becoming the synthesis target (Refer to **Figure 132**).

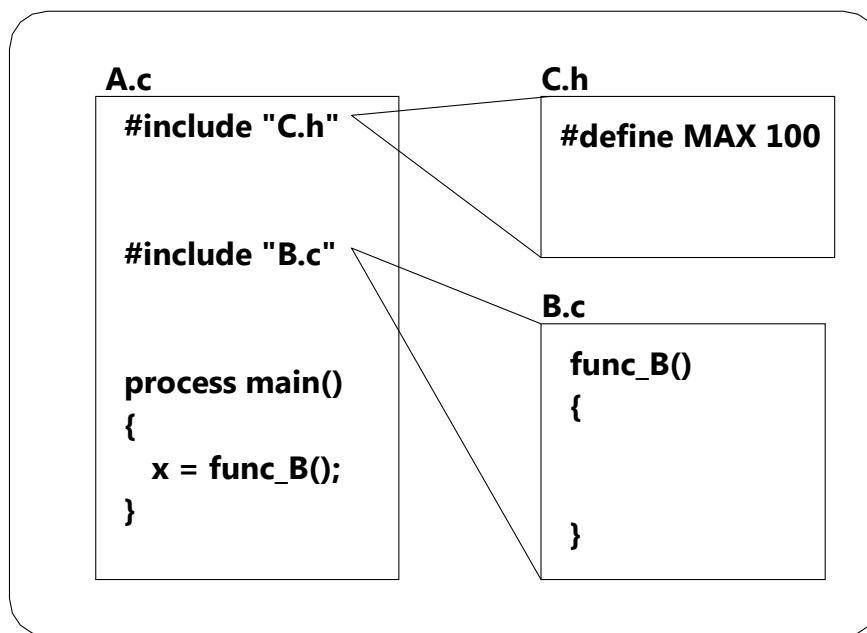


Figure 132: "Include" method of multiple files

B.15. Character code Limitations

The characters with a code similar to "/" for 2nd byte of Shift-JIS, cannot be used at the end of '//' comment.

Following example explains the characters with limited usage.

```
-ソル尊涅欺圭構蚕十申曾簞貼能表暴予禄兔喀媾彌拿朽歟濬眷秉綵臀藹觸體鐸饅鶴
BDL example :
in ter a, b;
out ter o;
#define ABC a
#define DEF b
process
main()
{
    o = ABC;
    // Not possible
    o = DEF;
    return 0;
}
```

To avoid the limitation, following methods are recommended.

- Character targeted for limitation will not be used at the end of the comment.
- Add a blank row under the comment row.

C. VHDL Output Command

C.1. Functional Overview

The VHDL output command vhdlgen is a command which generates logically synthesizable RTL VHDL from the result of behavioral synthesis obtained from synthesis command bdltran.

C.2. Usage

- Format

`vhdlgen [option] filename_E[.IFF]`

In the input file "`filename_E[.IFF]`", _E.IFF file is specified.

In case the extension is not ".IFF", it will be assumed that the extension has been omitted and it will be suffixed to the filename.

- Options

Various command line options are as described below (options specified within [] are default).

Options for Target CAD Specification

Option	Description
-OS	Specify target CAD [L]
-OL	S: Use standard package to generate VHDL for simulation.
-OD	L: Use standard package to generate synthesizable VHDL (for normal logic synthesis). D: Generate synthesizable VHDL for Design Compiler

License Related options

Option	Description
-lic_wait=#	As license is already obtained through another process. In case license cannot be obtained, repeat retry between the specified time. (Time specified: minutes) [0 minutes]

Memory Related options

Option	Description
-mem_clkcell	When using synchronous SRAM, input clock for memory is supplied through CLOCK_INV module. [clock is supplied directly, in the same hierarchy]
-memcomN	Don't give common reference to internal memory of the same type. [Do]
-mem_pin_ex	Unroll the multi bit port of memory module into single bit. [Do not unroll]
-sim_mem	Unroll the simulation model description of memory generated into single bit [Do not generate]
-mem_black_box= YES -mem_black_box =NO	Generate the black box description of memory YES : Generate NO : Generate the memory description of FPGA support target [ASIC mode is YES] [FPGA mode is NO]

Clock, Reset Related options

Options	Description
-FcE	Generate a separate clock pin for registers that are synchronized on negative phase of clock. (This option is planned to be obsolete. -FcE option of bdltran is recommended.)
-Zreset_macro=individual -Zreset_macro=inverter	Reset of register with initial value is assumed to be an asynchronous reset Reset of register with initial value is assumed to be an asynchronous reset and it is achieved by the method using the inverter.

Hierarchy Related Options

Option	Description
-inst_by_name -inst_by_order	Describe the entity interface with port name connection. Describe the entity interface with port sequence connection. [-inst_by_name]
-mux_module[:N:M] -mux_module=NO	Make hierarchies for multiplexer with N or more bits and M or more branches. -mux_module will become -mux_module:4:8 (default value) In case of -nmux=case, it is -mux_module:1:65 In case of -nmux=andor, it is -mux_module:4:8 Do not make hierarchies for multiplexer.
-ffmux_module	Make hierarchies for multiplexer and register.
-inst_by_name -inst_by_order	entity instance is described by port name connection entity instance is described by port order connection [-inst_by_name]
-mux_module[:N:M] -mux_module=NO	Multi plexer of N bits or more and M branch or more are converted into hierarchy -mux_module becomes -mux_module:4:8 (default value) -mux_module:1:65 in case of -nmux=case -mux_module:4:8 in case of -nmux=andor Multi plexer is not converted in hierarchy
-decoder_hier=YES -decoder_hier=NO	Specify hierarchy generation of decoder circuit [YES] YES: generate as lower hierarchy NO: generate within the same hierarchy
-top_name= <i>name</i> -top_name	Change the process name to " <i>name</i> ". [Do not] Change the process name to "process name_ref".
-module_prefix -module_prefix= <i>prefix</i>	Add "process name_" as prefix of lower hierarchy name. [default] Add " <i>prefix_</i> " as prefix of lower hierarchy name.

-module_prefix=	Do not add prefix to lower hierarchy name.
-new_opfunc_name -new_memfunc_name -new_decoder_name	Add "process name_" to the functional unit hierarchy name excluding user defined operator. Add "process name_" to internal memory hierarchy name. Add "process name_" to decoder hierarchy name. (valid only when -module_prefix= is added)
-opout	Generate functional unit hierarchy in a separate file [Generate in the same file]
-GT	Do not generate the files for black box hierarchy.[Generate]
-empty_module[=YES] -empty_module=NO	Generate lower module of black box hierarchy in a file Do not generate lower module of black box hierarchy in a file. [default is -empty_module=YES]
-ignore_flib_netlist	Generate module definition ignoring the net list module Specification (NETLIST) of functional unit library file

Output Format Options

Option	Description
-nmux=andor -nmux=case	Output format of multiplexer [ASIC mode follows -O option (-OL: andor, Other: case)] [FPGA mode follows target device (Xilinx: andor, Altera: case)] andor: Format using "and" and "or" case: Format using case statement
-case_default_x=(YES NO)	"when others" section of case statement [YES] YES : Add "when others" section of case statement when required NO : Change ending conditions of case statement to "when others"

-dec=case -dec=index -dec=loop	Output format of decoder[case] case : Format using <i>case</i> statement index : Format using bit selection loop : Format using <i>for</i> loop
-D0 -DX	Format in case the "when others" section of <i>case</i> statement is dont care [X] 0 : 0 is output X : X is output
-reg_feedback	Value of register maintained is implemented in feedback format
-addsub=select -addsub=share	addsub, addcsubc functional unit format specification [ASIC mode follows -O option (-OS : select, other: share)] [FPGA mode is share] select : Both addition, subtraction is described after HDL description and shared by logical synthesis tool share : Addition, subtraction is shared after HDL description
-dw -nec_dw	Use Synopsys DesignWare for division/surplus calculator/comparator Use DesignWare of NEC for comparator
-Zpipeline_func =(YES NO) ONLY_UNSUPPORTED_FU	Output format of pipeline functional unit [NO] YES : Generates simulation model NO : Do not generate simulation model ONLY_UNSUPPORTED_FU : Generates simulation model only for unsupported operations
-GO1 -GO2 -GO3	Specifies the OUTPUT type of signal[3] 1 : BUFFER 2 : OUT (Only external output port is converted in OUT) 3 : OUT (Output port also including all the internal output ports are converted in OUT)

Other Options

Option	Description
-arch_name=name	Change architecture name into "name"
-register_suffix=suffix -register_suffix=	Add "suffix" to register name Do not add suffix to register name (default)
-chk_case_mult_asgn =(YES NO)	In nmux description of case format Specify yes or no for multiple assignment check [YES] YES : Check NO : Do not check
-dec=(case index loop)	Format of generated decoder [case] case: format using case statement index: format using bit select loop: format using for loop
-D(0 X)	If all cases are listed except the default, in "dmux", then this option specifies the value generated in 'others' of case statement [X] 0: Output 0 X: Output X
-addsub=(select share)	Specify addsub, addcsub functional unit format [In ASIC mode, depend on -O option (-OS → select, Other → share) In FPGA mode, it is share] select : Describe both addition and subtraction in HDL description and share in logic synthesis tool share: Share addition, subtraction in HDL description
-dw	Use Synopsis' pure DesignWare for comparator and adder with carry-in.
-nec_dw	Use DesignWare of NEC for comparator and adder with carry-in.
-GC	Forcibly change variable name which does not fulfill VHDL syntax. [Treat this condition as an error]
-GP	Do not generate parenthesis during concatenate of signal (concat) [Generate]

-fp_pinform" <i>format</i> "	Specify output format of multi bit port name for false path script ["%s[*]"] (%s must be included during <i>format</i>)
-fp_highername= strings	Specify higher hierarchy names to false path script. (This option adds the specified hierarchy name before pin name, in the false path script)
-dump_on	Enable dump attribute. [default]
-dump_off	Disable dump attribute.
-dump_stdout 1 -dump_stdout 2	Specify destination for dump output. [1] 1: "/dev/stdout" 2: "STD_OUTPUT"
-n -n1	Generate all corresponding line numbers. [default] Do not generate line numbers.
-option_info=YES -option_info=NO	Output the options of each tool in comment of generated RTL. [default] Do not output the options of each tool in comment of generated RTL.
-p -p <i>filename</i> -p: <i>filename</i>	Read file "vhdlgen.vhprm" as a parameter file. <i>Read filename</i> as a parameter file <i>Read filename</i> as a parameter file
-o <i>filename</i> [.vhd][1]	Change output file name into <i>filename</i> [.vhd][1]
-EE -EJ	Change the message display into in English. Change the message display in Japanese.
-h, -H -v, -V	Generate command option help Generate version number

C.3. Circuit Configuration

The circuit structure generated by "vhdlgen" is as follows:

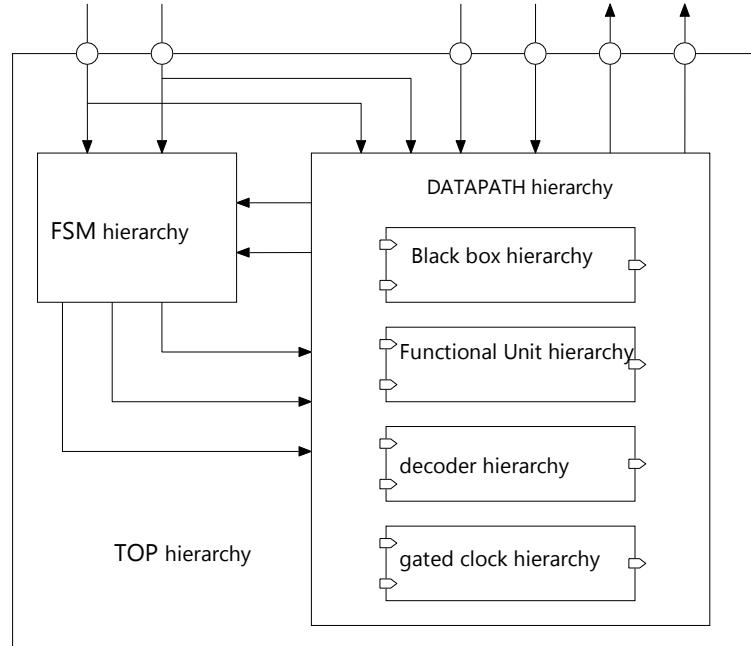


Figure 133: Circuit configuration

Decoder hierarchy, gated clock hierarchy can change the existence or non-existence of hierarchization by options. At the same time, hierarchization is performed by default.

C.4. Output Files

The "vhdlgen" tool reads the "*filename_E.IFF*" file and generates the below files for each hierarchy. As shown in the table below, *file1* shows the output file name *file1.[(vhdl | vhd)]* specified by option. If output file name is not specified by option, name used for *file1* will be "*filename_E*" for input file name "*filename_E.IFF*". If "-GT" option is specified, the "vhdl" file for black box hierarchy is not created. In other words, *file1_IP.vhd* is not generated.

Filename	Generated Hierarchy	Option
<i>file1.vhd</i>	TOP hierarchy, FSM hierarchy, DATAPATH hierarchy	default
<i>file1_IP.vhd</i>	Black box hierarchy	default
<i>file1_SIM.vhd</i>	Simulation model	-sim_mem

<i>file1_OP.vhd</i>	Functional unit hierarchy, decoder hierarchy	-opout
<i>file1_GCLK.vhd</i>	gated clock hierarchy	default
<i>file1.vhd.scuba</i>	Information file	default
<i>file1.vhd.LSInfo</i>	Information file for LSscrge	default

C.5. Regarding “–nmux” Option

The “vhdlgen” tool can modify the description format of multiplexer using the “–nmux” option. Two types of description format can be generated: one using “case” and other using “andor”.

Description when “case” is Specified

The description generated using “case” can simulate faster as compared to that generated by “andor”. In addition, multi assignment which was not possible using control signal in 1-hot, can now be taken care during the “RTL” simulation using the value X. However, the simulation pattern of X value may possibly differ between RTL simulation and gate level simulation.

```
VHDLgen_Label1004 : process(ST1_02d,ST1_01d,VHDLgen_tmp5001,subplot,addlot)
begin
    tmp5001 <= ST1_02d & ST1_01d;
    case tmp5001(0 to 1) is
        when "10" =>
            tmp5000 <= subplot;
        when "01" =>
            tmp5000 <= addlot;
        when "00" =>
            tmp5000 <= '0';
        when others =>
            tmp5000 <= 'X';
    end case;
end process;
```

Moreover, if –case_default_x=NO is specified, description with multiple assignment not detected can be generated. Coverage of RTL description can be expected to enhance.

```
VHDLgen_Label1004 : process(ST1_02d,ST1_01d,tmp5001,subplot,addlot)
begin
    tmp5001 <= ST1_02d & ST1_01d;
    case tmp5001(0 to 1) is
        when "10" =>
            tmp5000 <= subplot;
        when "01" =>
            tmp5000 <= addlot;
```

```

when others =>
    tmp5000 <= '0';
end case;
end process;

```

Description when "andor" is Specified

If the description is generated using "andor", the results of the simulation between RTL and gate level simulation match with each other. However, in this case the simulation is slower. Also, the multi assignment, which was not possible using control signals in 1-hot, cannot be detected during the RTL simulation.

```

tmp5000 <= --- NMUX
    VHDLGEN_AND(sublot, ST1_02d) or
    VHDLGEN_AND(addlot, ST1_01d) or
    VHDLGEN_AND('0', tmp5001);
c <= tmp5000;

```

C.6. Regarding -case_default_x=YES|NO Option

Implementation of "when others" section can be controlled by option at the time of generating *case* statement.

-case_default_x=YES (default setting)

Dont care condition is assumed as "when others" section and X is output. Input beyond expectation is detected and X is created for knowing the differences and X can be transmitted. It is conventional description.

(Ex In case of multiplexer with decoder)

```

case c is
when "00" =>
    b <= a1;
when "01" =>
    b <= a2;
when "10" =>
    b <= a3;
when "11" =>
    b <= a4;
when others =>
    b <= (others => 'X');
end case;

```

-case_default_x=NO

Final branch is changed in "when others" section. It is a notation which prioritizes the prevention of unnecessary decline of code coverage.

(Ex In case of multiplexer with decoder)

```

case c is
when "00" =>
    b <= a1;
when "01" =>

```

```

        b <= a2;
when "10" =>
        b <= a3;
when others =>
        b <= a4;
end case;

```

C.7. Regarding “-dec” Option

The “vhdlgen” tool can modify the description format of decoder using the “-dec” option. It can be described using three techniques: case, index, and loop statements. The result of the logical synthesis can vary by description format. However, the circuit quality after the logical synthesis depends upon the logical synthesis tool. Appropriate format depending on the CAD tool used is specified.

Description when “case” is Specified

```

VHDlGen_Label1002 :process(i1) -- decoder
begin
    case i1 is
        when '0' => o1 <= "10";
        when '1' => o1 <= "01";
        when others => o1 <= "00";
    end case;
end process;

```

Description when “Index” is specified

```

in1_int <= CONV_INTEGER(in1);
process(in1_int)
begin
    out1 <= (others => '0');
    out1(in1_int) <= '1';
end process;

```

Description when “loop” is specified

```

in1_int <= CONV_INTEGER(in1);
process(in1_int)
begin
    out1 <= (others => '0');
    for i in 0 to N-1 loop
        if (in1_int = i) then
            out1(i) <= '1';
        end if;
    end loop;
end process;

```

C.8. Black Box Hierarchy

The black box hierarchy, as a portion of hierarchy is a module that consists of only port declarations. During simulation, it should be replaced by module implementing the appropriate simulation model. During logic synthesis, it should be replaced by a logically synthesizable model or an already designed IP etc.

The hierarchies which are often considered as black box are as follows:

- Internal memory
- Pipelined functional unit
- Functional unit converted into operational functional unit
- Module which specifies "EXIST" within netlist module specification (NETLIST) of the functional unit library (FLIB) file.

C.9. Register Output Format

Output format of register maintaining the value is output in clock enable format which easily recognizes the clock enable (Refer to **section 30.3**).

Specify option -reg_feedback when it is to be changed in feedback format which uses feedback in old version.

C.10. Pipeline Functional Unit Simulation Model

Regarding pipeline functional unit, model for simulation can be generated. As for the functional unit supporting the description for logical synthesis of pipeline functional unit, description for logical synthesis is generated by default and black box is generated for the functional unit not supporting the logical synthesis description (Refer to **section 26.2**).

If -Zpipeline_func=YES is specified, simulation model is generated for the entire pipeline functional unit. If -Zpipeline_func=ONLY_UNSUPPORTED_FU is specified, simulation model having only functional unit which does not support logical synthesis description is generated.

Pipeline functional unit supporting the simulation model in vhdlgen are multiplier, divider, surplus calculator (However, mixed sign functional unit is excluded).

D. Verilog-HDL Output Command

D.1 Functional Overview

The Verilog-HDL output command veriloggen generates Verilog-HDL of register transfer level from structure level internal format file *_E.IFF*.

D.2 Usage

- **Format**

`veriloggen [option] filename [_E][.IFF]`

In the input file "*filename_E.IFF*", extension ".IFF" is optional and *_E.IFF* file is specified.

In case filename ends with *_E* at the end, it will be assumed that the extension has been omitted and it will be suffixed to the filename.

- **Option**

Various command line options are as described below (options specified within "[]" are set by default).

- **Options for Target CAD Specification**

Option	Description
-O(S L D C E)	Specify the target CAD specification [L] L: for Logic Synthesis S: for simulation D: for Design Compiler C: for Certify/Synplify E: for Leonardo Exemplar

- **License related options**

Option	Description
-lic_wait=#	License is already obtained through another process. However, in case license is not obtained, repeat retry during the given time (Time specified: minutes) [0 minutes]

– **Memory Related Options**

Option	Description
-mem_clkcell	When using synchronous SRAM, input clock for memory is supplied through CLOCK_INV hierarchy. [clock supplied directly in the same hierarchy]
-memcomN -memcomM	Do not give common reference to internal memory of the same type. Internal memory of same type is assumed as separate module and common reference to same module from all such modules. [direct common reference]
-mem_pin_ex	Unroll the multi bit port of memory module into single bit. [Do not unroll]
-sim_mem	Generate simulation model description of memory. [Do not generate]

– **Clock, Reset Related Options**

Optional	Description
-FcE	Generate a separate clock pin for registers that are synchronized on negative phase of clock. (This option is planned to be obsolete. –FcE option of bdltran is recommended)
-Zreset_macro=individual -Zreset_macro=inverter	Reset of register with initial value is assumed to be an asynchronous reset Reset of register with initial value is assumed to be asynchronous reset and achieved by method using the inverter.

– **Hierarchy Related Options**

Option	Description
-inst_by_name -inst_by_order	Describe module instance with port name connection. Describe module instance with port sequence connection. [-inst_by_name]
-mux_module[:N:M] -mux_module=NO	Make hierarchies for multiplexer with N or more bits and M or more branches. -mux_module will be –mux_module:4:8 (default value) Further, in case expression format of –nmux= case or casex, then -mux_module:1:65 will be used as default. In case –nmux=andor, elseif, then - mux_module:4:8 will be used. Do not make hierarchies for multiplexer.
-ffmux_module	Make hierarchies for multiplexers and registers as module.
-top_name= <i>name</i> –top_name	Change process name to " <i>name</i> ". [Do not] Change process name to "process name_ref".
-module_prefix -module_prefix= <i>prefix</i> -module_prefix=	Add "process name_" to prefix of lower hierarchy name. [default] Add " <i>prefix</i> " to the prefix of lower hierarchy name. Do not add prefix to lower hierarchy name.
-new_opfunc_name -new_memfunc_name -new_decoder_name	Add "process name_" to the Functional unit hierarchy name excluding user defined operator. Add "process name_" to internal memory hierarchy name. Add "process name_" to decoder hierarchy name. (Valid only when –module_prefix= is added)
-opout	Generate functional unit hierarchy in a separate file

Option	Description
	[Generate in the same file]
-empty_module[=YES] -empty_module=NO	Generate lower module of black box hierarchy in a file Do not generate lower module of black box hierarchy in a file. [default is -empty_module=YES]
-ignore_flib_netlist	Generate module definition ignoring the net list module Specification (NETLIST) of functional unit library file.

– **Output Options**

Option	Description
-nmux= case -nmux= casex -nmux= andor -nmux= ifelse	Format of generated multiplexer [In ASIC mode, follow –O option] [-OL→andor, others→case] case: Format using case statement casex: Format using casex statement andor: Format using and & or ifelse: Format using if statement
-case_default_x=(YES NO)	Default section of <i>case</i> statement [YES] YES : Add the default section of <i>case</i> statement when required NO : Change final condition of <i>case</i> statement to default
-DX -D0	When default clause of <i>case</i> statement / <i>casex</i> statement is “Don’t Care”. [-DX] X: Add undefined value (X) as default value 0: Add 0 as default value
-keep_starc_rule	Representation system design rule of STARC RTL style guide is agreed forcefully
-reg_feedback	Value of register maintained is represented in feedback format

-addsub=select -addsub=share	Specify the addsub,addcsubc functional unit format [In ASIC mode, follow -O option (-OS,-OC → select, other→ share) In FPGA mode, it is share] select : Describe both addition, subtraction in HDL description and share in logical synthesis tool share : Share addition, subtraction in HDL description
-dw -nec_dw	Use Synopsis' pure Design Ware for comparator and adder with carry-in. Use NEC's Design Ware for comparator and adder with carry-in.
-name_upper_case	Convert identifiers into upper case
-dec= index -dec= loop -dec= case	Format of generated decoder [index] index: Format using bit select loop : Format using "for" statement case: Format using "case" statement
-div	Output unsigned, pipelined unsigned division(/), surplus calculation(%) functional units by operator symbol
-signed_fu=(logic systemfunc)	Output format of signed functional unit [systemfunc] logic : Specify sign bit explicitly and operation with unsigned multiplier (Corresponds to Verilog 1995) systemfunc : Use \$signed system task (Corresponds to Verilog 2001)
-Zpipeline_func =(YES NO) ONLY_UNSUPPORTED_FU	Output format of pipeline functional unit [NO] YES : Generates simulation model NO : Do not generate simulation model ONLY_UNSUPPORTED_FU : Generates simulation model having only unsupported operation

– Other Options

Option	Description
-register_suffix = <i>suffix</i> -register_suffix=	Add suffix " <i>suffix</i> " to register name Do not add suffix to register name [default]
-param(1 2 N)	Specify format for state numbers output [1] 1: Use parameter with no bit width 2: Use parameter provided with bit width N: Do not use parameters at all, directly use constant values.
-base(16 10 8 2 H D O B N)	Specify output format for a constant [16] 16,H : hexadecimal number 10,D : decimal 8,O : octal number 2,B : binary N: Follow description
-NB -B	Assignment symbol in always block of a combinational circuit should be '<='. Assignment symbol in always block of a combinational circuit should be '='. [Default]
-LNB -LB	Assignment symbol in always block of a latched circuit should be '<='.[default] Assignment symbol in always block of a latched circuit should be '='.
-2process	Generate FSM with two process representation (This option is valid only for circuit synthesized with -Zfsm_st=NO option)
-fsm_dup	Generate a circuit to detect exception, after duplicating FSM.
-fp_pinform " <i>format</i> "	Specify output format of false path script multi-bit port name ["%s[*]"] (%s has to be included in the <i>format</i>)

-fp_highername=strings	Specify name of higher order hierarchy in false path script. (This option adds the specified hierarchy name "strings" before pin name in the false path script)
-dump_on -dump_off	Enable dump attribute [default] Disable dump attribute.
-dump_ref= "name"	Specify destination hierarchy name for dump reference. ["testbench.INST_0."]
-n -n1	Generate all corresponding line numbers. [default] Do not generate corresponding line numbers.
-option_info=YES -option_info=NO	Output the options of each tool in comment of generated RTL. [default] Do not output the options of each tool in generated RTL comment.
-regdelay_file=name[:#] -regdelay_file_def=name -regdelay_file_name=filename	Insert the delay value # in register assignment statement as parameter name. Default delay value is 1. Define the delay value by name "name". Set the name of delay value definition file as filename.
-regdelay_file_gen=YES -regdelay_file_gen=NO	Generate delay value definition file. Do not generate delay value definition file [default].
-regdelay_async_reset=YES -regdelay_async_reset=NO	Insert delay at asynchronous reset also. Do not insert delay at asynchronous reset. [default]
-p -pfilename -p: filename	Read file "veriloggen.vrprm" as parameter file Read filename as parameter file Read filename as parameter file

-o <i>filename</i> [.v]	Change output file name to <i>filename</i> [.v]
-EE -EJ	Display message in English language. Display message in Japanese language. [Follow environment variable LANG]
-h, -H -v, -V	Output command option help Output the version number

D.3 Circuit Configuration

1. When a circuit consists of FSM:
 - TOP hierarchy of synthesized circuit (hierarchy name is "process function name"), which further consists of
 - FSM hierarchy (hierarchy name is "process function name_fsm")
 - Data path hierarchy (hierarchy name is "process function name_dat"), which has following hierarchies:
 - + -- Black box hierarchy
 - + -- Functional unit hierarchy
 - + -- Decoder hierarchy
 - + -- Internal memory hierarchy
 - + -- Register, multiplexer hierarchy (When -ffmux_module option etc. are used)
 - + -- Gated clock circuit hierarchy

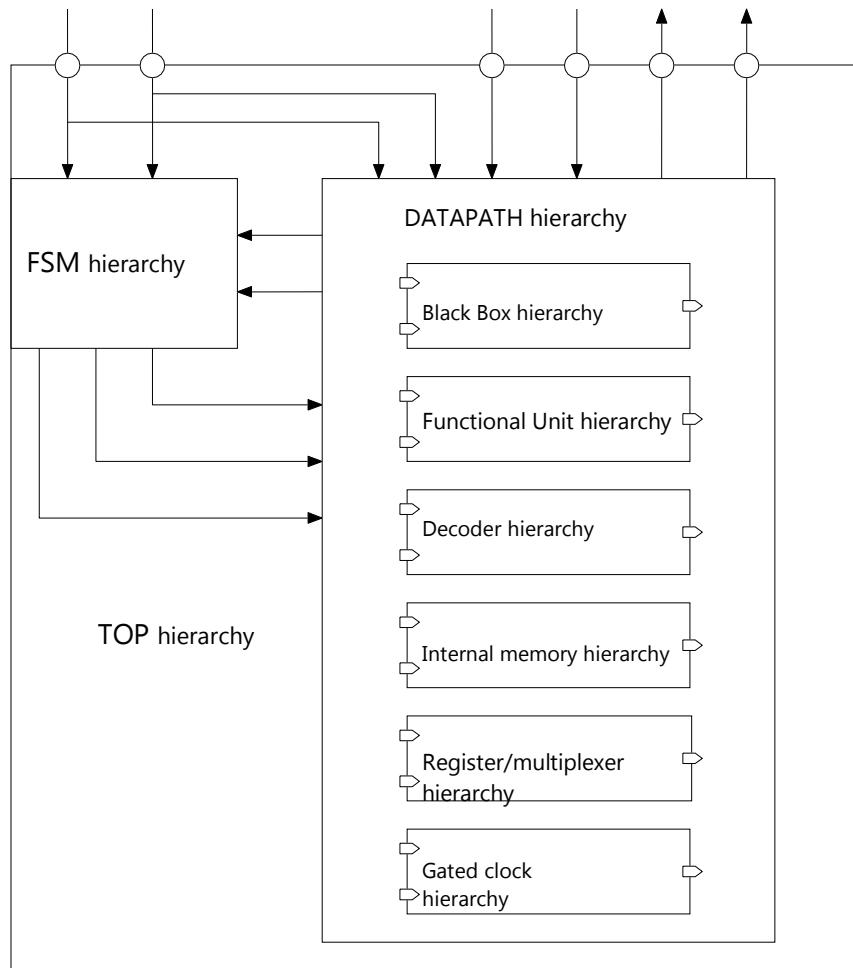


Figure 134: Circuit configuration (with FSM)

2. When the circuit does not contain a FSM (when circuit is combinational or is synthesized with “-ZZpipeline” option).

In this case, the top hierarchy of the synthesized circuit becomes data path hierarchy described above.

- Top hierarchy of synthesized circuit (hierarchy name is “process function name”)
 - + -- Functional unit hierarchy
 - + -- Decoder hierarchy
 - + -- Internal memory hierarchy
 - + -- Register, multiplexer hierarchy
 - + -- Gated clock hierarchy

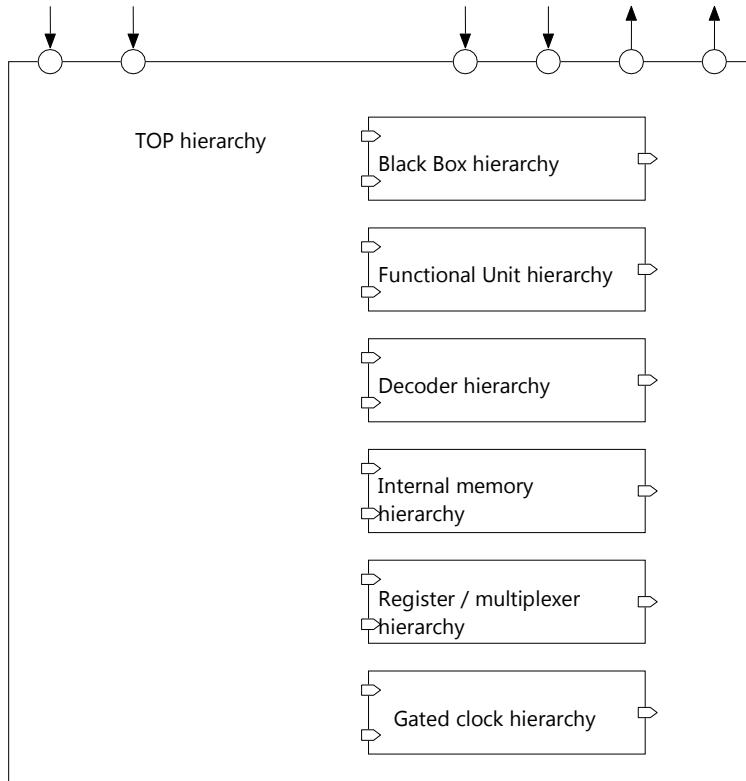


Figure 135: Circuit configuration (without FSM)

D.4 Output file

veriloggen reads _E.IFF file and outputs files as shown in the table. Here, *filename* represents the generated file name, *filename* [.v], that was specified with the "-o *filename*[.v]" option. In case the output file is not specified with option, the name of the generated file *filename* becomes "*filename*_E" when input file name is "*filename*_E.IFF".

Veriloggen [option] filename1_E.IFF

Output file name	Description
<i>filename</i> . v	Verilog-HDL file of top hierarchy, FSM hierarchy, data path hierarchy
<i>filename</i> _IP . v <i>filename</i> _SIM.v	Verilog-HDL file of black box module Simulation model
<i>filename</i> _OP. v	Verilog-HDL file of functional unit hierarchy, decoder hierarchy
<i>filename</i> _DUMP. V	Verilog-HDL file of dump hierarchy
<i>filename</i> . v. scuba	Information file

<i>filename.v.LSInfo</i>	Information file for LSscrge
--------------------------	------------------------------

D.5 Regarding “–nmux” option

The “*veriloggen*” tool can modify the specification of multiplexer when using the “–nmux” option. Specifications which can be generated are “case”, “casex”, “andor” and “ifelse” statements.

- **Description when “case” has been Specified**

The description generated by specifying “case” can simulate faster as compared to that generated by “andor”. In addition, multiple assignments are detectable in RTL simulation by observing X value even when the control signal is not given in 1-hot.

However, the simulation pattern of the X value may possibly differ in the results of simulation between RTL / gate.

```
always @ ( ST1_01d or addlot or ST1_02d or subplot )
    case ( { ST1_02d , ST1_01d } )
        2'b10 :
            tmp2 = subplot ;
        2'b01 :
            tmp2 = addlot ;
        2'b00 :
            tmp2 = 1'h0 ;
        default :
            tmp2 = 1'bx ;
    endcase
```

Moreover, if –case_default_x=NO is specified, the description with multiple assignment not detected can be generated. Coverage of RTL description can be expected to enhance.

```
always @ ( ST1_01d or addlot or ST1_02d or subplot )
    case ( { ST1_02d , ST1_01d } )
        2'b10 :
            tmp2 = subplot ;
        2'b01 :
            tmp2 = addlot ;
        default :
            tmp2 = 1'h0 ;
    endcase
```

- **Description when “casex” has been Specified**

The description generated by specifying “casex” can simulate faster as compared to that generated by “andor”. Moreover, as compared to “case” circuit, here a decoder is not required. Hence, a circuit with better execution performance and smaller area is synthesized. However, limitation of this approach is that multiple assignments are not detectable. Further, sometimes undefined value cannot be detected by RTL simulation because undefined value, which was input to control signal of multiplexer, is not propagated to output.

```

always @ ( ST1_01d or add1ot or ST1_02d or subplot )
  casex ( { ST1_02d , ST1_01d } )
    2'b1x :
      tmp2 = subplot ;
    2'bx1 :
      tmp2 = add1ot ;
    default :
      tmp2 = 1'h0 ;
  endcase

```

If –OD option for Design Compiler is specified, parallel_case directive is added to casex statement for improving the optimization of logical synthesis. Using this directive, delay and area can be improved. However, it is possible that the results for multiple assignments during the RTL/ gate level may differ.

Further, if -keep_starc_rule option is used concurrently, parallel, full_case directive are controlled.

```

always @ ( ST1_01d or add1ot or ST1_02d or subplot )
  casex ( { ST1_02d , ST1_01d } )
    // synopsys parallel_case full_case
    2'b1x :
      tmp2 = subplot ;
    2'bx1 :
      tmp2 = add1ot ;
    default :
      tmp2 = 1'h0 ;
  endcase

```

- **Description when “andor” has been Specified**

If the description is generated using ‘andor’, the results for the RTL simulation/ gate level simulation match with each other. However, the simulation is slower in this case. Also the multi assignment, which was not possible using control signals in 1-hot, cannot be detected during the RTL simulation. Further, there are cases when undefined value cannot be detected by RTL simulation because undefined value which is input to control signal of multiplexer, is not propagated in output depending upon input value.

```

always @ ( add1ot or ST1_01d or subplot or ST1_02d )
  tmp2 = ( ( { 1{ ST1_02d } } & subplot )
    | ( { 1{ ST1_01d } } & add1ot )
    | ( { 1{ ( (~ST1_02d ) & ( ~ST1_01d ) ) } } & 1'h0 ) ) ;

```

- **Description when “ifelse” has been Specified**

If the description is generated using 'ifelse, the results for the RTL simulation/gate level simulation match each other. However, it is not recommended as conditional branches result in the generation of multiplexers, which results in a number of branches and hence results in longer delay and performance deterioration.

```
always @ ( ST1_02d or subplot or ST1_01d or add1ot )
    if ( ST1_02d )
        tmp2 = subplot ;
    else if ( ST1_01d )
        tmp2 = add1ot ;
    else
        tmp2 = 1'h0 ;
```

D.6 Regarding -case_default_x=YES|NO Option

Representation of "default" section can be controlled by option when *case* statement is output.

-case_default_x=YES (default setting)

Dont care condition is assumed to be "default" section and X is output. Input beyond expectation is detected and X is created for knowing the differences and X can be transmitted. It is a conventional description.

(Ex In case of multiplexer with decoder)

```
case ( c )
2'h0 :
    b = a1 ;
2'h1 :
    b = a2 ;
2'h2 :
    b = a3 ;
2'h3 :
    b = a4 ;
default :
    b = 8'hx ;
endcase
```

-case_default_x=NO

The final branch is changed into "default" section. It is a notation which prioritizes the prevention of unnecessary decline of code coverage.

(Ex In case of multiplexer with decoder)

```
case ( c )
2'h0 :
    b = a1 ;
2'h1 :
    b = a2 ;
2'h2 :
    b = a3 ;
default :
    b = a4 ;
endcase
```

D.7 Regarding -keep_starc_rule Option

-keep_starc_rule option of veriloggen is an option which agrees compulsorily to the main representation system rules in the RTL style guide (design rule) which calls STARC.

Regarding the details of STARC RTL setting style guide, refer to concerned book of STARC publications.

- Basic RTL setting style guide VerilogHDL edit, STARC editorial, Baifukan of LSI setting are compatible with the following rules.
- Target rule list

2.8.1.5 Required Not changed to fullacase forcefully by directive of *case* statement which exists in specific logical synthesis tool

2.8.5.1 Recommended 1 Not changed to parallel forcefully by directive of *case* statement which exists in specific logical synthesis tool

-case_default_x=NO option violates the policy (Transmission of X etc.) of rule related to *case* statement of many low recommendation and -case_default_x=NO is not disabled in -keep_starc_rule.

D.8 Regarding “-dec” option

The “veriloggen” tool can modify the decoder specification using the “-dec” option. It can be described using three techniques: index, loop and case statements. The logic synthesis result may vary for different specifications techniques. However, the quality of logically synthesized circuit depends upon the logic synthesis tool used. Appropriate format depending on the CAD tool used is specified.

- **Description when “index” is Specified**

```
always @ (DECODER_in_tmp0 )
begin
    DECODER_out_tmp0 = 2'b0 ;
    DECODER_out_tmp0 [VerilogOut_DECODER_in_tmp0] = 1'b1 ;
end
```

- **Description when “loop” is Specified**

```
always @ (DSEL0s)
begin
    for(i=0;i<OBW;i=i+1)
        NSEL0o[i] = (DSEL0 == i);
end
```

- **Description when “case” is specified**

```
always @ ( VerilogOut_DECODER_in_tmp0 )
begin
    case ( DECODER_in_tmp0 )
```

```

0 :
    DECODER_out_tmp0 = 8'b10000000 ;
1 :
    DECODER_out_tmp0 = 8'b01000000 ;
2 :
    DECODER_out_tmp0 = 8'b00100000 ;
3 :
    DECODER_out_tmp0 = 8'b00010000 ;
4 :
    DECODER_out_tmp0 = 8'b00001000 ;
5 :
    DECODER_out_tmp0 = 8'b00000100 ;
6 :
    DECODER_out_tmp0 = 8'b00000010 ;
7 :
    DECODER_out_tmp0 = 8'b00000001 ;
default :
    DECODER_out_tmp0 = 8'b00000000 ;
endcase
end

```

D.9 Black Box Hierarchy

The *black box hierarchy* is a hierarchy, which consists of port declarations only and is generated in the file named "*filename_IP.v*".

During simulation, it must be replaced by a module implementing the appropriate simulation model. During logic synthesis, it should be replaced by a model for appropriate logical synthesis or an already designed IP etc.

The hierarchies which become black box hierarchies are given below:

- Internal memory
- Division, surplus calculation which can be signed and sign conversion at the time of –div specification
- Pipeline functional unit (excluding unsigned division and surplus calculation at the time of –div specification)
- Functional unit converted into function functional unit
- Module that specifies, "EXIST" within netlist module specification (NETLIST) of functional unit library (FLIB) file.

D.10 Regarding Register Output Format

The output format of register maintaining the value is output by clock enable format which easily recognizes clock enable (Refer to **section 30.3**).

Specify option -reg_feedback when it is to be changed in feedback format which uses the feedback in old version.

D.11 Pipeline Functional Unit Simulation Model

Model for simulation can be generated for pipeline functional unit. Regarding functional unit supporting the description for logical synthesis of pipeline functional unit, description for logical synthesis is generated by default and black box is generated for the functional unit which does not support the logical synthesis description (Refer to **section 26.2**).

If -Zpipeline_func=YES is specified, simulation model is generated for the entire pipeline functional unit. If -Zpipeline_func=ONLY_UNSUPPORTED_FU is specified, simulation model having only the functional unit which does not support the logical synthesis description is generated.

Simulation model for pipeline functional unit besides divider, surplus calculator of mixed sign functional unit is supported in veriloggen.

D.12 Regarding Absolute Delay Insertion in Register Assignment Statement

Absolute delay, as shown in figure below, can be inserted to the register using assignment statement written in always statement (always statement which describes the combinatorial circuit, is out of scope).

Therefore, there is a need to specify three options. The point to be noted here is that inserted delay appears in parameter statement and the value itself is defined in a separate include file without being written in parameter statement. First option is -regdelay_file, and it specifies the parameter name (DLY as given below) and delay value (integer). Second is -regdelay_file_def, and it specifies the define name (DEFVAL as given below). Third is -regdelay_file_name, and it specifies the file name to define delay value.

For example, the following RTL is generated, if -regdelay_file = DLY:3, -regdelay_file_name ="include_file.v", -regdelay_file_def = DEFVAL, are specified.

```
"include "include_file.v" // Specify the include file name
module ...

parameter DLY = 'DEFVAL; // Numeric value will be written in
include file

always @(posedge clk or negedge rst_x)
  if (!rst)           RG <= 0;
  else if (...)      RG <= #DLY exp1; // insert delay
  else               RG <= #DLY exp2; // insert delay
endmodule
```

Further, if -regdelay_async_reset is used as an additional option, it can be decided whether delay will be inserted even in asynchronous reset part or not. If regdelay_file_gen is used, it can be decided whether above mentioned include file will be generated or not. The contents of this file are as follows.

```
'define DEFVAL 3
```

E. Combinational Loop Detection Command

E.1. Functional Overview

Combinational loop detection command "check_E" takes the structural level in the internal file format (_E.IFF) as input and detects combinational loops (including the "false" loop).

E.2. Usage

- **Format**

check_E [option] *filename*[.IFF]

Only "_E.IFF" type of file can be specified for "*filename*".

If the extension of the file specified is not ".IFF", it will be assumed that the extension has been omitted and it will be suffixed to the filename.

- **Options**

Options	Description
-EF0	Do not generate error and log information file (.ceerr)
-lic_wait=#	License is already obtained through another process. However, in case license is not obtained, repeat retry during given time (Time specified: minutes) [0 minutes]
-h, -H	Generate command options help
-v, -V	Generate version number

E.3. Functional Description

This command detects the combinational loops present inside the synthesized result and shows them. Circuits that have loops formed without involving registers are termed as combinational loops. Such loops may prove to be hazardous during logical synthesis.

The combinational loops detected may also include the "false" loops that are logically not active.

In case no combinational loop is detected, the command will display the following message after execution:

```
**** loop not found ****
check_E Finish
```

When a combinational loop is detected, the signal name and structure description that comprise the combinational loop will be displayed, as shown in the following example.

At that time, the corresponding source file name and line number will also be displayed. In order to avoid loop formation, either description can be modified based on a file name and a line number specified or constraints / options can be changed.

For each combinational loop, the following two types of information are displayed:

- **List of signal names comprising the loop**

no	signal number
signal name	signal name
destination signal no	signal number of assignment destination

- **List of structural description comprising the loop**

[Signal no. and signal name of assignment destination]	[Signal no. and signal name of assignment destination]
---	---

↓ ↓
No.14 :sub081ot <--- No.15 :sub081i1
8(test.c): sub081ot(0:8) ::= sub08@1(sub081i1(0:8), sub081i2(0:8));

↑
[Structural description configuring loop]

Functional unit library file func.FLIB

```

#@VERSION{1.00}
#@LIB{sample}
@FLIB{
    NAME      add08
    KIND      +
    BITWIDTH 8
    DELAY    200
    SIGN     SIGNED,UNSIGNED
    AREA     100
}
@FLIB{
    NAME      sub08
    KIND      -
    BITWIDTH 8,8
    DELAY    200
    SIGN     UNSIGNED
    AREA     100
}
#@END{sample}
$
```

Source file test.c

```

in ter(0:8) a,b,c;
out reg(0:8) o;
process main(){
    o = (a + b) - c ;
    $                                }
    o = (a - b) + c ;               #@END{sample}
    $
```

}

Functional unit count file func.FCNT

```

#@VERSION{1.00}
#@CNT{sample}
@FCNT{
    NAME add08
    LIMIT 1
}
@FCNT{
    NAME sub08
    LIMIT 1
}
#@END{sample}
```

Command option

>bdltran -lfl func.FLIB -lfc func.FCNT -c1000 test.IFF

Attention

- As detection is done in signal units, a false error is generated even when no loop is there when checked in bit width.
- It does not support functions being converted into functional units and modules declared as "defmod". Hence, it checks internals of all respective modules assuming the existence of path across all inputs and outputs.
- When memory and functional units are kept externally, path within respective modules will not be checked.
- Assuming the existence of the following paths, the internal memory other than pipelined memory will be checked:
 - Path from the address port to the output data port
 - Path from the read enable port to output data port
 - Path from the port select port to output data port.

Example of output result

```
%check_E test_E.IFF
Copyright (C) 1988-2004 NEC Corporation. All rights reserved.
check_E version : 1.51 Thu Dec 2 14:51:10 JST 2004
          (BIF version : 2.97h)
<W>5901: loop detected
  loop exist among the following signals
  -----
    no signal name |-> destination signal no
    15 sub081i1   |-> 14,
    14 sub081ot   |-> 9,
    9 add081i1    |-> 8,
    8 add081ot    |-> 15,
  -----
  No.14 :sub081ot <--- No.15 :sub081i1
  8(test.c): sub081ot(0:8) ::= sub08@1(sub081i1(0:8),
  sub081i2(0:8));
  -----
  No.9 :add081i1 <--- No.14 :sub081ot
  add081i1(0:8) ::= nmux {
    when (ST1_02d(0:1)) : sub081ot(0:8); /* line# test.c:10 */
    when (ST1_01d(0:1)) : a(0:8); /* line# test.c:8 */
  };
  -----
  No.8 :add081ot <--- No.9 :add081i1
  8(test.c): add081ot(0:8) ::= add08@1(add081i1(0:8),
  add081i2(0:8), add081_s(0:1));
  -----
  No.15 :sub081i1 <--- No.8 :add081ot
  sub081i1(0:8) ::= nmux {
    when (ST1_01d(0:1)) : add081ot(0:8); /* line# test.c:8 */
    when (ST1_02d(0:1)) : a(0:8); /* line# test.c:10 */
  };
  -----
<W>5902: 1 loop(s) detected
!!!! 1 loop(s) detected !!!!
log and error file generated [test_E.ceerr]
check_E ABNORMAL Finished
```

F. Overflow Detection Tool

F.1. Functional Overview

Overflow detection command "check_overflow" detects the overflow (overflow/underflow) by taking input from internal format file (IFF) after BDL input.

F.2. Usage

- **Format**

`check_overflow [option] filename[.IFF]`

Only the ".IFF" type of file can be specified for "*filename*".

If the extension of the file specified is not ".IFF", it will be assumed that the extension has been omitted and it will be suffixed to the filename.

- **Options**

Option	Description
+w#	Enable the checking of warning number#. (default)
+w-#	Disable the checking of warning number#.
-f	Generate the report file (cof)
-s directory	Specify the directory that executed BDL input command.
-EFO	Do not output the error and log information file (.coerr)
-lic_wait=#	License is already obtained through another process. However, in case license is not obtained, repeat retry during given time (Time specified: minutes) [0 minutes]
-h , -H	Output command option help
-v , -V	Output version number

F.3. Functional Description

This command takes the input of IFF after BDL input, detects and displays the description in which overflow may occur. If overflow is detected, the types of overflows, their description and detailed information are displayed as given below.

```
%bdlpars foo.c
%check_overflow foo.IFF

[WARNING:3001] possible overflow in assignment
[Source Lines]
9(foo.c): s9 = s8 + u8;
[analysis result]
(signed 9 bitwidth) = (signed 10 bitwidth);
[ERROR:3000] 1 warning is detected
check_overflow finished. See the generated error file
foo.coerr.
```

<information displayed in warning>

In [WARNING:????], warning number and description of warning is displayed.

In [Source Lines], relevant source line is displayed.

In [analysis result], the detailed information i.e. finally, how many bits of variable is assigned to how many bits of variable is displayed.

There can be 3 types of completion

- Normal completion, not detected → return value0
Overflow Warning is not detected
check_overflow finished successfully.
- Normal completion, detection → return value 2
[ERROR:3000] # warning is detected
check_overflow finished. See the generated error file foo.coerr.
- Abnormal completion → return value 1
[ERROR:1006] input file (foo.IFF) cannot open
check_overflow aborted due to errors. See the generated error file foo.coerr.

There are following 12 types of warnings.

- [WARNING:3001] possible overflow in assignment
When there is a possibility of overflow occurring during assignment
- [WARNING:3002] possible underflow in assignment
When there is a possibility of assigning negative value to unsigned variable
- [WARNING:3003] array index range exceeds array boundary.
Possibility of out of bound access
There is a possibility that array subscript will exceed the number of elements present during array declaration
- [WARNING:3004] array index of type signed. possibility of negative index
When there is a possibility that negative value can be assigned to array subscript
- [WARNING:3005] possible overflow in return value
When return value is inserted in return value type of function by 'return' and there is a possibility of overflow

- [WARNING:3006] possible underflow in return value
When return value of function is unsigned type and there is possibility that negative value can return by 'return'.
- [WARNING:3007] possible overflow in functional call
When there is a possibility of overflow while inserting a value of actual argument by calling function in parameter type of function.
- [WARNING:3008] possible underflow in functional call
When function parameter is unsigned type and there is a possibility that negative value can assign to actual argument
- [WARNING:3009] possible overflow in cast
When there is a possibility of overflow occurring due to type casting.
- [WARNING:3010] possible underflow in cast
When there is a possibility that negative value is type casted in unsigned
- [WARNING:3011] possible overflow/underflow due to '++'
When there is a possibility of overflow occurring when increment, decrement and complex assignment operator is used.
- [WARNING:3012] cannot detect overflow/underflow in pointer operation
In case pointer operation is used, warning is generated as detection cannot be done because pointer is used for judging whether array subscript exceeds the number of elements present during array declaration.

F.4. Warning Control Options

Option	Description
+w#	Enable the checking of warning number # (default)
+w-#	Disable the checking of warning number #

```
check_overflow +w-3005 +w-3006 foo.IFF
```

As mentioned above, by using option +w-#, checking of #number can be disabled. By default, all warnings are enabled.

F.5. Report File Output Option

Option	Description
-f	Generate the report source file (cof).
-s <i>directory</i>	Specify the directory, where BDL input command is executed.

If -f is specified, report file, where marking is done on line corresponding to statement on which warning is given in source file, is generated with 'cof' extension in same directory as input IFF.

At the starting of line corresponding to statement on which warning is given, !! is added. The total number of warnings on statement included in that line is displayed as the numerical character after !!.

(Example) input source file test.c

```
in ter(0:9) a;
out ter(0:8) b;
process foo() {
    b = a;
}
```

Output report file test.c.cof

```
in ter(0:9) a;
out ter(0:8) b;
process foo() {
    !! 1
    b = a;
}
```

Input source file is searched assuming that the directory where bdlpars, scpars and check_overflow is executed, are same. If the check_overflow execution directory is different, then directory, where bdlpars, scpars were executed through -s *directory option* must be specified.

Remarks:-

- Since checking process is carried out on the basis of bit width, sometimes pseudo warning is generated. The actual range of fixed value will become even narrower than bit width, therefore, there is more possibility that pseudo warning will come.
 - If 0 and HiZ is assigned to 1 bit signed variable, pseudo warning will come.
- Since the checking process is carried out for every statement one by one, therefore, even when an 8 bit variable has only 2 bit value, it will be treated as 8 bit.
- The line, which is marked in the report file, becomes the entire statement including relevant locations. In the given below example, there are 2 lines to specify condition of if statement, therefore, both 2 lines will be marked even if the location, where overflow will occur is only at assignment statement of 1st line.

```
if( (u8 = u9) > 10
    && flag ) {
    .
}
!! 1 if( (u8 = u9) > 10
    && flag ) {
    .
}
```

- Encrypted description is out of scope of checking and all other descriptions are in scope.

G. Signal Display Tool

G.1. Functional Overview

This tool reads the text file with numbers enlisted and arranges them in an order and displays it. It specifies pin list file, which contains a list of text files where numbers are enlisted and described as input.

Execution Example

```
%signallist -l -w 8 pin.txt
signal name | 1 2 3 4 5 6 7 8
-----
a | 10 20 30 40 50 60 80 90
b | 1 0 1 0 0 0 1 0
```

G.2. Usage

- **Format**

`signallist [option] pin_list_file [pin_list_file ..]`

Specifies pin list file in "*pin_list_file*". Multiple files can be specified.

- **Options**

Options	Description
<code>-1</code>	Displays horizontally
<code>-s Num</code>	Specifies first element number to be displayed
<code>-e Num</code>	Specifies last element number to be displayed
<code>-w Num</code>	Specifies number of elements to be displayed
<code>-i Num</code>	Specifies the display interval
<code>-ix</code>	Input file form is in hexadecimal number
<code>-io</code>	Input file form is in octal number
<code>-ox</code>	Output file form is in hexadecimal number
<code>-oo</code>	Output file form is in octal number

-h	Display the description of option
-hj	Explain the description of option in Japanese

G.3. Functional description

- **Form of Pin List File**
 - o Ignore blank lines and line starting with "#".
 - o One file name is specified in one line.
 - o There are two formats for specifications:
 - Specify only a file name in one line.
 - Specify "I" or "O" at the start of the line followed by a file name after a space
 - o Presence of Japanese character, space, or tab in the path name is not supported.

Example of a pin list file

```
pin1
# Comment here
I pin2
O pin3
```

The decimal values that are written in file and specified by the pin-list file are displayed one by one. There are two formats for output: one is a vertical format and the other is horizontal format. By default, the output is in the vertical format and if '-I' option is specified, the output is displayed in horizontal format. File name is assumed to be signal name and then displayed.

Vertical type

		a	b
1		10	1
2		20	0
3		30	1
4		40	0
5		50	0
6		60	0
7		80	0
8		90	1
9			0
10			

Horizontal type

	a	b	signal name	1	2	3	4	5	6	7	8	9
				1	0	30	40	50	60	80	90	
					0	1	0	0	0	0	0	
						1	0	0	0	0	0	
							1	0	0	0	0	

In case there are lot of elements displayed, then display can be controlled using the “-s”, “-e”, and “-w” options. The specifications of these options are as follows:

- The “-s” option can be used to specify the first element number to be displayed.
- The “-e” option can be used to specify the last element number to be displayed.
- The “-w” option can be used to specify the number of elements to be displayed.

By default, all files from beginning are displayed till the point where no more value (number) can be read. Moreover, displayed interval can be specified by the “-i” option. By default, it is displayed horizontally in 4 line intervals.

Numerical value written in file specified by pin list file must specify “-ix” option at the time of hexadecimal number and “-io” option at the time of octal number. Display is shown by decimal. Specify “-ox” option when it is to be displayed by hexadecimal and “-oo” option when it is to be displayed by octal.

H. Basic Library Generation Tool

H.1. Functional Overview

Basic library generation tool "BLIBgen" generates the Basic Library (BLIB) file using the logic synthesis tool. The quality of synthesis improves by using the generated BLIB file in the behavioral synthesis execution command "bdltran". Also, the quality of circuit quality report (QOR) improves.

H.2. Usage

- **Format**

BLIBgen [option]

library.BLIB is generated.

library: Used library name.

- **Options**

- **Common option**

Option	Description
-vhdl	Change generated RTL to VHDL specification (Default is Verilog-HDL specification)
-c #	Set # as clock period
-ext	Provide additional script in logical synthesis tool
-save	Save netlist.
-d#	Specifies the output bit width of decoder The following numbers can be specified. 2, 4, 8, 16, 32, 64, 128, 256
-dec=(case index loop)	Specifies the output format of decoder case : Format using the case statement index : Format using the bit selection (Default) loop : Format using the "for" loop
-nmux=(case casex andor ifelse)	Specify the output format of multiplexer case : Format using the case statement casex : Format using the casex statement andor : Format using the "and" & "or" (Default) ifelse: Format using the if statement (andor or case can be specified at the time of combined usage with -vhdl option)

Option	Description
-D(0 X)	Specify default output value of multiplexer
-syn_tool dc sf sf_pro ise vivado qts	Specify tool used for logical synthesis dc : use Design Compiler sf : use Synplify sf_pro : use Synplify Pro ise : use ISE vivado: use vivado qts : use Quartus II
-syn_tool_setup dc= <i>file</i> sf= <i>file</i> sf_pro= <i>file</i>	Specify the setup file of logical synthesis tool Spaces are not allowed before and after "="
-syn_module_setup dc= <i>file</i> sf= <i>file</i> sf_pro= <i>file</i>	Specify the file for additional command at the time of logic synthesis. Spaces are not allowed before and after "=".
-syn_delay_unit unit dc= unit sf= unit sf_pro= unit qts= unit	Specify delay units value for the logical synthesis Following units can be specified 10000ns, 1000ns, 100ns, 10ns, 1ns, 1/10ns, 1/100ns, 1/1000ns, 1/10000ns, 10000ps, 1000ps, 100ps, 10ps, 1ps, 1/10ps, 1/100ps, 1/1000ps, 1/10000ps By default, a value of 1ns is set When specifying different units for each synthesis tool, specify "dc=", "sf=" before the unit. However, spaces are not allowed before and after "=".
-lib_delay_unit unit	Specify delay unit value generated in the BLIB file The following units can be specified. 10000ns, 1000ns, 100ns, 10ns, 1ns, 1/10ns, 1/100ns, 1/1000ns, 1/10000ns, 10000ps, 1000ps, 100ps, 10ps, 1ps, 1/10ps, 1/100ps, 1/1000ps, 1/10000ps By default, a value of 1/100ns is set.
-syn_dir <i>directory</i>	Specify directory of logical synthesis tool
-syn_retry -syn_retry_num# -syn_retry_interval#	When license of logical synthesis tool is not obtained, retry until it is obtained When license of logical synthesis tool is not obtained, retry # times until it is obtained When license of logical synthesis tool is not obtained,

Option	Description
	retry in # seconds interval
-o filename.BLIB	Change output file name to <i>filename.BLIB</i>
-work_dir directory	Specify work directory
-EF0	Do not generate error file
-lic_wait=#	When license is already obtained through another process. However, when license is not obtained, repeat retry during given time [0 minutes]
-H	Output command option help
-V	Output version number

– Option for Design Compiler

Option	Description
-dc_mode [db xg]	Specify Design Compiler mode db: DB mode xg : XG mode
-dc_scr [sh tcl]	Specify script language of Design Compiler sh : dcsh mode tcl : tcl mode
-64bit	Execute in 64 bit mode

– Option for Synplify/Synplify Pro/ISE/Vivado/Quartus II

Option	Description
-fpga_family NAME	Specify FPGA family
-fpga_device NAME	Specify FPGA device
-fpga_speed_grade NAME	Specify speed grade of FPGA
-fpga_package NAME	Specify FPGA package
-fpga_characterize (auto quick full)	Specify the estimated method of ISE/Quartus II auto : Automatic judgment (default) [ISE is quick, Quartus II is full] quick : Adopt estimation after logical synthesis full : Adopt estimation after arrangement wiring
-check_family_name (YES NO)	Check the FPGA family name [YES]

-64bit	Execute in 64 bit mode (Options for Quartus II)
--------	--

H.3. Functionality Description

Basic library generation tool, BLIBgen, generates Basic library file (BLIB) using logic synthesis tool. Generated Basic Library file (BLIB) contains specification of delay and area etc. of bit operations or multiplexers etc. Using these values Cyber generates optimized circuit. Hence, it is recommended to use basic library file (BLIB) in behavioral synthesis execution command (bdltran). Also, by using basic library file during behavioral synthesis, the estimated quality of Circuit quality report (QOR) generated by Cyber improves.

Basic library generation tool "BLIBgen" uses logic synthesis tool. Therefore, it is necessary that environment setting is done such that logic synthesis tool can be used. Logical synthesis tool and operation check version supported by BLIBgen are same as functional unit library delay area setting tool. (Refer to **section I.3**).

By default, every basic library element is generated in verilog-HDL description and synthesized. "-vhdl" option is specified when it is generated in VHDL description. Further, "/tmp" is used for work directory. To change the work directory, the "-work_dir" option is specified.

The format version of the BLIB file generated by "BLIBgen" is 2.00. The BLIB elements are as follows.

- Register
- Decoder
- Multiplexer
- Multiplexer provided with decoder
- Bit operations ("INV", "AND", "OR", "XOR", "NAND", "NOR", "XNOR")
- Constant table

H.4. User Script File Specification

As in the case of functional unit library delay area setting tool "FLIBgen", basic library generation tool BLIBgen can add command corresponding to logical synthesis tool by specifying -syn_module_setup option. (Refer to **I.10.2**).

H.5. Setting regarding Logical Synthesis Tool

Basic library generation tool "BLIBgen" uses logical synthesis tool. Settings and precautions related to logic synthesis tool are similar to the functional unit library delay and area setting tool "FLIBgen" (Refer to **I.10**).

I. Functional Unit Library Delay and Area Setting Tool

I.1. Functional Overview

Functional unit library delay and area setting tool, FLIBgen, takes functional unit library file (FLIB) as input. Each described functional unit or functional module is synthesized by using logical synthesis tool. It searches delay, area, net count, pin pair count and logical stage count, and overwrites the same in the input file.

Input file:	Output file:
<pre> #@VERSION{1.00} #@LIB{foo_library} #@CLK 2000 #@UNIT 1/100ns @FLIB { NAME ADD KIND + BITWIDTH 8 SIGN SIGNED, UNSIGNED } #@END{foo_library} </pre>	 <pre> #@VERSION{1.00} #@LIB{foo_library} #@CLK 2000 #@UNIT 1/100ns @FLIB { NAME ADD KIND + BITWIDTH 8 DELAY 100 AREA 150 SIGN SIGNED, UNSIGNED SYN_TOOL DC NET 30 PIN_PAIR 40 LSTAGE 10 } #@END{foo_library} </pre>

I.2. Usage

- **Format**

FLIBgen [option] "*filename.FLIB*"

Here, "*filename.FLIB*" file is specified as input file.

The area and delay are set in "*filename.FLIB*" as output.

- **Options**

– Common options

Option	Description
-vhdl	Change generated RTL to VHDL description (Default is Verilog-HDL description)
-characterize quick -characterize full	Set to high speed estimation mode Set to high accuracy estimation mode
-ch (YES NO) -ch_replace type [:type...] -noch_replace type [:type...]	Compute the chain delay also Specify the type of operations for computation of chain delay Specify the type of operations to be ignored for computation of chain delay
-c #	Set clock cycle to #
-ignore_clk -ignore_delay	Do not use clock cycle specified in functional unit library file Do not use delay value of every functional unit specified in functional unit library file
-ignore_netlist -ignore_syn_tool	Functional unit whose netlist exists also becomes the target Ignore the specification in SYN_TOOL and follows option -syn_tool
-lib_delay_unit unit	Specify the unit of delay value generated in FLIB [1/100ns].
-full_search_limit mul=# -full_search_limit div=# -full_search_limit mod=#	Specify bit width for high accuracy estimation Change maximum bit width of multiplication to # bit [24] Change maximum bit width of division to # bit [12] Change maximum bit width of remainder to # bit [12]
-syn_tool dc -syn_tool sf -syn_tool sf_pro -syn_tool ise -syn_tool vivado	Specify the tool used in logic synthesis Use Design Compiler Use Synplify Use Synlify Pro Use ISE

Option	Description
-syn_tool qts	Use Vivado Use Quartus II
-syn_tool_setup dc= <i>file</i> -syn_tool_setup sf= <i>file</i> -syn_tool_setup sf_pro= <i>file</i>	Specify the setup file of logic synthesis Specify the script for Design Compiler Specify the script for Synplify Specify script for Synplify Pro
-syn_module_setup dc= <i>file</i> -syn_module_setup sf= <i>file</i> -syn_module_setup sf_pro= <i>file</i>	Specify the script at the time of functional unit synthesis Specify the script for Design Compiler Specify the script for Synplify Specify script for Synplify Pro
-syn_delay_unit <i>unit</i> -syn_delay_unit dc= <i>unit</i> -syn_delay_unit sf= <i>unit</i> -syn_delay_unit sf=unit -syn_delay_unit qts= <i>unit</i>	Specify the delay value unit for logical synthesis [1ns] Specify the delay value unit for Design Compiler Specify the delay value unit for Synplify Specify the delay value unit of Synplify Pro Specify the delay value unit of Quartus II
-syn_dir <i>directory</i>	Specify directory of logical synthesis tool.
-syn_retry -syn_retry_num# -syn_retry_interval#	Retry until the license of logical synthesis tool is not acquired Retry at maximum # number of times till license of logical synthesis tool is acquired Set retry interval as # seconds when license logic synthesis is not acquired
-stage_search (AUTO YES NO)	Specify the search method of stage count of pipeline functional unit [AUTO] AUTO : Automatic judgment YES : Search appropriate stage count NO : Estimation with specified stage count
-stage_search_limit #	Specify the upper value of stage count search of pipeline functional unit [10]
-signed_fu (logic systemfunc)	Specify the description format of signed functional unit [systemfunc] logic : Explicitly specify sign bit Operate with multiplier with no sign (corresponds to Verilog 1995)

Option	Description
	systemfunc : use \$signed system task (corresponds to Verilog 2001)
-D[0 X]	Specify the output value of shifter, adder, subtracter functional units
-ext	Provide an additional script to logical synthesis tool
-save	Save net list
-max_bitw #	Specify the maximum value of the bit width targeted for estimation.
-sub_module_file <i>file</i>	Specify sub module when FCNT for functions that are synthesized as functional unit is input.
-replace_1T	Replace to delay value when delay is within 1 cycle in cycle specification
-replace_nT	Replace to delay value when delay is cycle specification
-work_dir <i>directory</i>	Specify work directory
-lic_wait=#	License is already obtained through another process. However, when license cannot be obtained, repeat retry during given time (Time specified: minutes) [0 minutes]
-o <i>filename.FLIB</i>	Set output file name to <i>filename.FLIB</i>
-EF0	Do not generate error file
-h -H	Output command options help
-v -V	Output version number

– Options for Design Compiler

Option	Description
	Specify Design Compiler mode
-dc_mode db	db: DB mode
-dc_mode xg	xg: XG mode
	Specify script language of Design Compiler
-dc_scr sh	sh: dcsh mode
-dc_scr tcl	tcl: tcl mode

-dc_ultra (YES NO)	Use Design Compiler Ultra [NO]
-dw	Synthesize using DesignWare for pipeline multiplier, divider, surplus multiplier, and comparator.
-64bit	Execute in 64 bit mode

- Options for Synplify/Synplify Pro/ISE/Vivado/Quartus II

Options	Description
-fpga_family <i>name</i>	Specify FPGA family
-fpga_device <i>name</i>	Specify FPGA device
-fpga_speed_grade <i>name</i>	Specify FPGA speed grade
-fpga_package <i>name</i>	Specify FPGA package
-fpga_characterize (auto quick full)	Specify the estimation method of ISE/Quartus II auto : Automatic judgment (default : quick) quick : Adopt estimation after logical synthesis full : Adopt estimation after arrangement wiring
-check_family_name (YES NO)	Check FPGA family name [YES]
-64bit	Execute in 64 bit mode (Options for Quartus II)

I.3. Functionality Description

FLIBgen which is functional unit library delay area setting tool, sets delay and area etc. by using logic synthesis tool for the functional unit library file. In order to use FLIBgen, it is necessary to set the environment in a way so that logic synthesis tool can be used. Logical synthesis tool supported at present are as follows.

Logical Synthesis Tool	Operation Check version	OS
Design Compiler	H-2013.03-SP2	LINUX
Synplify	E-2011.03-SP2	LINUX, Windows
Synplify Pro	H-2013.03-SP1	LINUX, Windows
ISE	14.4	LINUX, Windows
Vivado	2014.2	LINUX, Windows
Quartus II	13.1	LINUX, Windows

By default, logic synthesis for each functional unit of FLIB file is done with verilog-HDL description. -vhdl option is specified while doing the logical synthesis with VHDL description. Further, "/tmp" is used in work directory. -work_dir option is specified while changing the work directory. Types of operations supported by FLIBgen are as under.

Operation type	KIND	Number of input ports	Number of output ports
Addition	+	2	1
Subtraction	-	2	1
Addition Subtraction	+ or -	2	1
Increment operation	++	1	1
Decrement operation	--	1	1
Multiplication	*	2	1
Division	/	2	1
Remainder Calculation	%	2	1
Comparator (basic functionality)	Either one of <,>,<=,>+	2	1
Comparator(complex functionality)	All combinations of comparator (basic functionality). Number of combinations, 2, 3, 4 all possible.	2	1
Right shift	>>	2	1
Left shift	<<	2	1
Left right shift	>>,<< or <<,>>	2	1

Information of functional units not supported by "FLIBgen" is not replaced.

I.4. Standard Functional Unit Library (Standard FLIB)

Standard functional unit library (standard FLIB) is a library which includes functional unit required in normal behavioral synthesis. In the behavioral synthesis system Cyber, formal comment #@LIBTYPE{STANDARD} within the functional unit library (FLIB) file treats specified FLIB as standard FLIB and usage of this standard FLIB is recommended.

When architecture search is performed, normally, appropriate functional unit library (FLIB) for each of the architecture differs. However, it is necessary to modify the FLIB file every time. If

standard FLIB is used, then without modifying the FLIB file at the time of architecture search, it will improve the flow if only functional unit count (FCNT) file is set and efficiency of design increases. Standard FLIB can be generated by using the template for standard FLIB that is included in the installation directory.

Standard FLIB can be generated by changing template-standard_fpga.FLIB for FPGA and template-standard_asic.FLIB for ASIC to an appropriate name in \$CYBER_PATH/lib directory and considering it as an input file of FLIBgen.

```
> cp $CYBER_PATH/lib/template-standard_asic.FLIB project_standard.FLIB  
> FLIBgen project_standard.FLIB
```

It is assumed that standard FLIB is specified in behavioral synthesis option along with FLIB and FCNT files generated by using FLIB, FCNT file auto generation of behavioral synthesis system Cyber. Please refer to **section 25.3.1** for the details of automatically generated files of FLIB and FCNT.

In case of using functional unit other than functional unit included in above template (for example, functional unit with large bit width or pipeline functional unit), FLIB file as per automatically generated FLIB and FCNT files is added and either FLIBgen is executed or a separate FLIB file must be prepared.

Further, CyberWorkBench provides standard functional unit library (FLIB) and Basic library (BLIB) are provided below by \$CYBER_PATH/lib. The usage is assumed in case target technology is not decided.

```
$CYBER_PATH/lib/asic_130.FLIB      : FLIB for 130nm  
asic_130.BLIB        : BLIB for 130nm  
asic_90.FLIB        : FLIB for 90nm  
asic_90.BLIB        : BLIB for 90nm  
...  
fpga_siii.FLIB      : FLIB for fpga  
fpga_siii.BLIB      : BLIB for fpga  
fpga_siv.FLIB       : FLIB for fpga  
fpga_siv.BLIB       : BLIB for fpga  
fpga_aiigx.FLIB    : FLIB for fpga  
fpga_aiigx.BLIB    : BLIB for fpga  
...  
fpga_s6.FLIB        : FLIB for fpga  
fpga_s6.BLIB        : BLIB for fpga  
fpga_v4.FLIB        : FLIB for fpga  
fpga_v4.BLIB        : BLIB for fpga  
fpga_v5.FLIB        : FLIB for fpga  
fpga_v5.BLIB        : BLIB for fpga  
fpga_v6.FLIB        : FLIB for fpga  
fpga_v6.BLIB        : BLIB for fpga  
...
```

I.5. Estimation Mode

The two types of FLIBgen estimation modes are explained below:

- High Speed Estimation Mode
- High Accuracy Estimation Mode

High speed estimation mode is the mode assuming the estimation of registered standard functional unit library by various functional units. By default high speed estimation mode is set in FLIBgen if the library name (#@LIB{CWBSTDLIB}) indicating standard functional unit library is specified in FLIB file. In this mode, logic synthesis is done by extracting minimum required functional unit from the functional unit entry in FLIB; therefore, high speed estimation can be done with high accuracy.

High accuracy estimation mode is the mode which computes suitable constraint values for the entire functional unit present in FLIB file and conducts synthesis by setting that constraint value. Caution is required when estimation is done for the registered FLIB file, having numerous functional units similar to standard functional unit library, as in these cases execution time of FLIBgen is long. By default, high accuracy estimation mode is set, when the library name of FLIB file is other than the standard functional unit library.

Further, in spite of these estimation modes, there are cases when some functional units are estimated according to the presumed value. For details please refer to **Section I.11**.

I.6. Delay Specification

Functional unit delays can be specified in the following three ways:

- Absolute delay specification
- Cycle specification
- Pipeline specification

There are four patterns which are included when delay is not included. By default, functional units with absolute delay and functional unit with "NO" delay are considered for delay computation, whereas functional units with cycle specification and pipeline specification remains as it is.

When the "-replace_1T" option is specified, functional units specified with one cycle in cycle specification are also considered and delay is computed that overrides the absolute delay. When the "-replace_nT" option is specified, all functional units with cycle specification are considered and delay is computed that overrides the absolute delay.

Functional unit with pipeline delay specification remains as it is.

Above description is summarized in the table below. Function units whose delay can be overridden with a given option are marked with "O".

Delay type of Functional unit	Default	-replace_1T	-replace_nT
No delay	O	O	O
Absolute delay specification	O	O	O
Cycle specification (1 cycle)	—	O	O
Cycle specification	—	—	O
Pipeline specification	—	—	—

I.7. Functional Unit Chain Effect Specification

By default, FLIBgen sets the chain effect delay between functional units. Because of this, it is possible to generate FLIB with high delay accuracy. When chain effect delay is not required, execution time of FLIBgen can be fasten by specifying NO in "-ch" option.

When chain effect between particular functional unit is not required, '+' or "x" can be specified by chain effect specification (CHAIN_FROM) in FLIB file. In case "+" or "x" specification exists in chain effect specification, the chain effect of corresponding portion is not set by default. It should be explicitly specified by -ch_replace option in setting the chain effect of these functional unit.

It is possible to control the estimation target for each type of chain effect specification method through -ch_replace, -noch_replace. At that time, following type name can be specified more than once as a separator by ":".

Chain effect delay specification	Type name specified in option	Target types at the time of -ch

CHAIN FROM <i>name</i> : delay	DD	○
CHAIN FROM <i>name</i> : +	D+	—
CHAIN FROM <i>name</i> : x	Dx	—
CHAIN FROM * : delay	AD	○
CHAIN FROM * : +	A+	—
CHAIN FROM * : x	Ax	—
CHAIN FROM DEFAULT : delay	FD	○ (Note1)
CHAIN FROM DEFAULT : +	F+	—
CHAIN FROM DEFAULT : x	Fx	—

Note: CHAIN_FROM DEFAULT specifies the chain effect when chain source functional unit is not specified; therefore it remains there even after chain effect of all functional units is calculated.

I.8. Delay Constraints in Functional Unit Synthesis

By default suitable constraint value is computed and specified for every functional unit at the time of their synthesis.

In case delay is set as absolute delay in FLIB file, synthesis is done by using the specified delay value as the delay constraint. If the -ignore_delay option is specified, FLIBgen is synthesized by ignoring the specified delay constraint.

In case delay is set with cycle specification in FLIB file, synthesis is done by setting clock cycle as upper limit of the delay constraint. If -ignore_clock option is specified, synthesis is done when FLIBgen is specified and ignoring the clock cycle.

Clock cycle can be specified in FLIB file by using "#@CLK". Further, clock cycle can be specified in option -c #. Option has high priority when both are specified.

Delay unit of FLIB file is 1/100 ns (Nano Second) by default. Delay unit can be set by using "#@UNIT" in FLIB file. If delay unit is set, FLIBgen uses the specified unit for estimation. However, the decimal point is omitted.

I.9. Pipeline Functional Unit Support

It supports the following pipeline functional units.

Type	Synthesis tool
Multiplication Division, Surplus calculation	Design Compiler, ISE, Quartus II Design Compiler, ISE, Quartus II

Pipeline functional unit is estimated using the DesignWare for Design Compiler. For ISE, it is estimated using automatic recommendation description or CORE generator. For Quartus II, it is estimated using automatic recommendation description or Megafunction. Hence, pipeline functional unit which can be estimated in FLIBgen depends on the DesignWare or CORE generator or Megafunction specification. For example, as the stage count of pipeline multiplier prepared in DesignWare is from 2 to 6 stages. Therefore, even in condition where estimation is possible in FLIBgen, stage count changes to pipeline multiplier from 2 stages till 6 stages.

In FLIB for pipeline functional unit, in addition to information which can be specified in functional unit of normal combinatorial circuit, more detailed information can be specified.

- **DELAY**

Delay information can be specified in functional unit of combinatorial circuit. However, pipeline functional unit can be specified in the format of "Delay" x "Stage count". At this time, value of "Delay" specified the maximum delay value between below **DELAY_INPUT**, **DELAY_OUTPUT**, flip flop.

- **DELAY_INPUT**

Delay from input port till filp flop can be specified.

- **DELAY_OUTPUT**

Delay from flip flop till output port can be specified.

- **RESET_PORT**

Existence or non-existence of reset port can be specified.

- **PIPELINE_STALL**

Existence or non-existence of enable port can be specified.

- **DSP_MACRO**

Description of functional unit can be specified. IP_GEN which indicates IP connection or INFER of automatic recommendation description can be specified.

- **IP_NAME**

In specification for IP connection, IP module name can be specified.

- **IP_OPT_MODE**

In specification for IP connection, SPEED or AREA or DEFAULT can be specified.

- **IP_CONSTRUCTION**

In specification for IP connection, configuration of multiplier can be specified from MUL or LUT.

- **IP_DIV_TYPE**

In specification for CORE Generator, division or surplus calculation mode can be specified. At present, only RADIX2 is supported.

- STAGE_SEARCH

YES or NO can be specified. In the condition of set frequency, it can be specified whether search for appropriate pipeline stage count is to be performed or not. It is possible to specify with -stage_search option. In case of Design Compiler, it is applicable for multiplication, division and surplus calculation. In case of ISE, it is applicable for multiplication.

Input file: Example of Design Compiler

```
#@VERSION{2.00}
#@LIB{library}
#@CLK 500
#@UNIT 1/100ns
@FLIB {
  @FLIB {
    NAME      mul8
    KIND      *
    BITWIDTH  8
    DELAY    x3
    SIGN     UNSIGNED
  }
}
#@END{library}
```

Output file: Example of Design Compiler

```
#@VERSION{2.00}
#@LIB{library}
#@CLK 500
#@UNIT 1/100ns
@FLIB {
  NAME      mul8
  KIND      *
  BITWIDTH  8
  DELAY    295x3
  DELAY_INPUT 15
  DELAY_OUTPUT 290
  AREA     2300
  SIGN     UNSIGNED
  SYN_TOOL DC
  NET      225
  PIN_PAIR 350
  LSTAGE   15
  RESET_PORT NO
  PIPELINE_STALL NO
}
#@END{library}
```



Input file: Example of ISE

```

{@VERSION{2.00}
{@LIB{library}
{@CLK 1000
{@UNIT 1/100ns
@FLIB {
    NAME      mul8
    KIND      *
    BITWIDTH 8
    DELAY    x3
    SIGN     UNSIGNED
    IP_NAME   mul8_ip
    IP_OPT_MODE SPEED
    IP_CONSTRUCTION MUL
    RESET_PORT YES
    PIPELINE_STALL YES
}
{@END{library}}

```

**Output file: Example of ISE**

```

{@VERSION{2.00}
{@LIB{library}
{@CLK 1000
{@UNIT 1/100ns
{@FPGA_FAMILY{family}
{@FPGA_DEVICE{device}
{@FPGA_PACKAGE{package}
{@FPGA_SPEED{speed}
@FLIB {
    NAME      mul8
    KIND      *
    BITWIDTH 8
    DELAY    267x3
    DELAY_INPUT 0
    DELAY_OUTPUT 0
    MACRO_BLOCK MUL:1
    SIGN     UNSIGNED
    SYN_TOOL  ISE
    IP_NAME   mul8_ip
    IP_OPT_MODE SPEED
    IP_CONSTRUCTION MUL
    RESET_PORT YES
    PIPELINE_STALL YES
}
{@END{library}}

```

I.10. Settings related to Logical Synthesis Tool

I.10.1 Delay Unit of Report Result of Logical Synthesis Tool

By default, the delay unit in result report of logical synthesis tool is 1ns. However, in case this delay unit is not 1ns, delay unit must be specified using the "-syn_delay_unit *unit*" option.

The following are the 18 possible units which can be specified as *unit*:

10000ps	10000ns
1000ps	1000ns
100ps	100ns
10ps	10ns
1ps	1ns
1/10ps	1/10ns
1/100ps	1/100ns
1/1000ps	1/1000ns
1/10000ps	1/10000ns

I.10.2 64 bit mode specification of logic synthesis tool

If you want to use logic synthesis tool in 64 bit mode, be cautious as specification method may vary as per logic synthesis tool.

- It is necessary to specify -64 bit option for FLIBgen option for Design compiler or Quartus II.
- It is necessary to perform a setting so that it operates 64 bit binary in user environment settings for ISE.
- It is not necessary to specify any option as logic synthesis tool automatically determines the specification, as per the environment for Synplify or Synplify Pro.

I.10.3 User Script File Specification

In order to estimate with high accuracy by using FLIBgen, it is necessary to execute FLIBgen in the environment that is almost similar to the actual execution environment of logic synthesis.

Commands such as wire_load and dont_use can be set in FLIBgen by specifying -syn_module_setup option. FLIBgen conducts the synthesis by adding the file specified in -syn_module_setup option to script file.

Example of user script

```
update_lib LIBRARY_NAME -overwrite WIRE_LOAD_FILE
set_wire_load "WIRE_LOAD_NAME"
set_dont_use standard.sldb/DW01_add/rpl
set_dont_use standard.sldb/DW01_sub/rpl
set_dont_use standard.sldb/DW01_addsub/rpl
set_dont_use standard.sldb/DW01_inc/rpl
set_dont_use standard.sldb/DW01_dec/rpl
set_dont_use standard.sldb/DW01_incdec/rpl
```

I.10.4 Obtaining License of Logical Synthesis Tool

By default, if the license of logical synthesis tool is not acquired, it terminates with error.

In case the license is not available, the "-syn_retry" option can be specified to retry till it is acquired once. In order to limit the number of retrials, "-syn_retry_num#" option is specified, which retries for a maximum of "#" times. In case license is still not available, it terminates with error. By default, retrial interval is two seconds, which can be changed using the "-syn_retry_interval#" option. The interval is specified to "#" seconds.

I.10.5 Settings related to Logic Synthesis Tool for FPGA

In case Synplify/Synplify Pro of Synopsys, ISE of Xilinx, Quartus II of Altera are used as logic synthesis tool, -fpga_family/-fpga_device/ -fpga_speed_grade/-fpga_package options are provided. Among these, it is necessary to specify -fpga_family option and in case other options are abbreviated, default device of logic synthesis tool will be set. Option can be specified as shown in the example below:

```
% FLIBgen ... -fpga_family spartan3 -fpga_device XC3S1500
-fpga_speed_grade -4 -fpga_package FG320
```

The supported target device can be checked by "Functional unit library (FLIB) Delay area setting" of FLIB context menu of project management window for integrated development environment CWB-gui or by "Basic library automatic generation" of context menu of project management window process.

I.11. Points to be noted

FLIBgen sets area, delay etc. for the below functional unit according to the estimated value.

- Multiplication of 25 bit or more
- Division/Surplus calculation of 13 bit or more

In case estimation is done as per the estimated value, the message specified below will be generated.

W_FL0316: div32 is characterized without full search.

It is necessary to specify -full_search_limit option, in order to do high accuracy estimation for the functional unit estimated according to the presumed value. However, there is a possibility that logic synthesis time may increase, hence caution is required.

I.12. Limitations

- Limitations when Synplify/Synplify Pro is used as logic synthesis tool
FLIBgen uses the batch mode of Synplify/Synplify Pro. Therefore, floating license that supports the batch mode will be required.
- Limitations regarding the estimate of function synthesized as functional unit
Estimate regarding the functions that include the below functionalities is not supported.
 - Including internal memory
 - Including gated clock circuit

J. Memory library delay area setting tool

J.1. Functional overview

Memory library delay area setting tool, MLIBgen is a tool which allows you to set area and access delay of memory library file (MLIB) by using logical synthesis tool. You can carry out synthesis for area and access delay of the memory by specifying the generated memory library file (MLIB) in synthesis option of behaviour synthesis tool (bdltran). Hence, it is an effective tool for delay convergencenet etc. and quality of synthesis.

Example of output file:

```

#@VERSION{1.00}
#@LIB{foo_library}
@MLIB {
    NAME MEMB32W128
    KIND R1,W1
    BITWIDTH 32
    DELAY 400x3 # Read access delay (1/100ns)
    WORD 128
    AREA 1 # Block RAM usage count
    MEM_SYNTH BLOCK
    DEFMOD {
        in ter (0:5) ral /* ra:1 */;
        in ter (0:1) rclk1 /* rclk:1 */;
        out ter (0:32) rd1 /* rd:1 */;
        in ter (0:5) wa2 /* wa:2 */;
        in ter (0:32) wd2 /* wd:2, setup=300 */; # Write access
        delay(1/100ns)
        in ter (0:1) we2 /* we:2 */;
        in ter (0:1) wclk2 /* wclk:2 */;
    }
}
#@END{foo_library}

```

J.2. Usage

- Format

MLIBgen [option] *filename.MLIB*

Here *filename.MLIB* file is specified as input file.

The area and access delay of memory are set in *filename.MLIB* as output.

(you can also change the output file name by specifying the option)

- Option

Option	Description
-syn_tool (ise vivado qts)	Specifies the tool used for logical synthesis. ise : Uses ISE vivado: Uses vivado qts : Uses Quartus II
-vhdl -c #	Set the RTL to be generated as VHDL description. (By default it is Verilog-HDL) Set the clock cycle to #. (unit 1/100ns)
-lib_delay_unit unit	Seficies unit of delay value output in MLIB. You can specify below units. 10000ns,1000ns,100ns,10ns,1ns,1/10ns,1/100ns, 1/1000ns,1/10000ns,10000ps,1000ps,100ps,10ps, 1ps,1/10ps,1/100ps,1/1000ps,1/10000ps In short it will be 1/100ns.
-syn_delay_unit unit	Specifies unit of delay value of logical synthesis. You can specify below units. 10000ns,1000ns,100ns,10ns,1ns,1/10ns,1/100ns, 1/1000ns,1/10000ns,10000ps,1000ps,100ps,10ps, 1ps,1/10ps,1/100ps,1/1000ps,1/10000ps In short it will be 1ns.
-syn_dir <i>directory</i>	Specify directory of logical synthesis tool.
-work_dir <i>directory</i>	Secify work directory.
-o <i>filename.MLIB</i>	Set output file name as <i>filename.MLIB</i> .
-lic_wait=#	License is already obtained through another process. However, when license cannot be obtained, repeat retry during given time (Time specified: minutes) [0 minutes]
-EF0	Do not generate error file
-EJ	Display the error message in Japanese language.
-EE	Display the error message in English language.
-h -H	Output command options help
-v -V	Output version number
-fpga_family	Specify FPGA family

Option	Description
<i>NAME</i> -fpga_device <i>NAME</i> - fpga_speed_grade <i>NAME</i> -fpga_package <i>NAME</i>	Specify FPGA device Specify FPGA speed grade Specify FPGA package
-64bit	Enables 64 bit mode of Quartus II. (option for -syn_tool=qts)

J.3. Functionality Description

MLIBgen which is a memory library delay area setting tool, sets access delay and area etc. of memory by using logical synthesis tool for memory library (MLIB) file. In order to use MLIBgen, it is necessary to set the environment where logical synthesis tool can be used. Logical synthesis tool supported at present are Quaruts II of Altera and ISE of Xilinx. Refer **section I.3**, to know about the version on which behaviour check is carried out. You can specify the logical synthesis tool by specifying option syn_tool.

By default, logic synthesis for memory description for logical synthesis is done with verilog-HDL description. -vhdl option is specified while doing the logical synthesis with VHDL description. Further, in Linux environment "/tmp" is used in work directory. In Windows environment, directory specified with environment variable TEMP, is used. -work_dir option is specified for changing the work directory.

Clock cycle must be specified when executing MLIBgen. Option -c (unit is 1/100ns) is specified in order to specify the clock cycle.

J.4. Settings related to logic sysnthesis tool

By default, the delay unit in result report of logical synthesis tool is 1ns. However, in case this is not 1ns, delay unit must be specified using "-syn_delay_unit unit" option.

Installation location for logical synthesis tool must be specified by "-syn_dir" option.

This option is assumed to be used in order to specify version of the logical synthesis tool which is executed when tools of multiple versions are installed or when program of logical synthesis tool has not been set as PATH environment variable.

MLIBgen is a tool which helps to create estimation for FPGA.

Hence for functer details on setting methods and limitations when using logical synthesis tool made FPGA, see **section I.10** which can also be reffere for FLIBgen.

K. BDL Output Tool

K.1. Functional Overview

BDL output tool "iff2bdl" is a tool that takes internal format file (IFF) as input and generates BDL file.

BDL input command can convert and look IFF file of internal format of "_C.IFF" of state transition level generated as result in the middle of behavioral synthesis or generated IFF file, "_9.IFF" of resource sharing, "_E.IFF" of structural level or "_E.IFF" generated by top hierarchy output tool, etc into a BDL format text file.

K.2. Usage

- **Format**

`iff2bdl [option] filename [.IFF]`

Specify the IFF file in "*filename*".

In case the extension of specified file is not ".IFF", it will be assumed that the extension has been omitted and will be suffixed to filename.

If extension is ".bdl" or ".BDL", it is changed to the extension ".IFF".

- **Option**

Option	Description
-O1	Display the first line number of the corresponding source at the start of line
-Oo	Display the line number and file name of all the corresponding sources as comments at the end of line (default)
-O-o	Do not display the line number and file name of all the corresponding sources as comments at the end of line
-Ot	Display attributes (default)
-O-t	Do not display attributes
-Oe	Display the bitw attribute (default)
-O-e	Do not display the bitw attribute
-Oy	Display typecasting (default)
-O-y	Do not display typecasting

Option	Description
-Ow	Display bit information of signals and constants (default)
-O-w	Do not display bit information of signals and constants
-O2	Display constant in binary format
-OI	Output IFF file
-h , -H	Outputs command line option help (English version)
-hj, -Hj	Outputs command line option help (Japanese version)
-V	Outputs version number

K.3. Function Description

This tool generates text file in the .BDL format from ".IFF" files (internal format files).

- **-O1 Display the first line number of the corresponding source at the start of line.**
As shown in the example below, the line number of the corresponding source is specified only in one line head. In case there are multiple lines associated with it, only one will be displayed.

```
33 : add081i1(0:8) ::= a(0:8); /* line# test.bdl:33 test2.bdl:7 */
```

- **-O2 Display constants as binary numbers.**
By default, constants are displayed in decimal format. In case this option is specified, they are displayed in binary format. However, digits whose bit-width is undefined are displayed as decimals.
- **-O-w Do not display bit width information of signals and constants.**
By default, the bit-width information of signals and constants is displayed as shown below. In case this option is specified, this information is not displayed.

$a_1(0:8) ::= a(0:8);$ $r(0:8) = a_1(0:8) + 1(0:8);$ $o_1(0:8) = r(0:8);$	$a_1 ::= a;$ $r = a_1 + 1;$ $o_1 = r;$
--	---

Displays bit information

Do not display bit information

NOTE:

- This tool does not support the initial value specified at the time of signal declaration. Therefore, the initial value assignment is displayed at function start.
- Function declaration is not generated.
- Outside, shared, and static are replaced by attributes.
- There is no surety that the generated file can be used as input for behavioral synthesis but it can be used for analysis as it is.

L. Top Hierarchy Generation Tool

L.1. Functional Overview

Top hierarchy generation tool “topmodgen” accepts multiple internal format file (_E.IFF) of behavioral level and structural level and generates the circuit description of higher hierarchy (top module) that connects these modules (which are retained at lower level) in structural level internal format description (_E.IFF) or behavioral level BDL description (.BDL). The mode, in which structural level internal format description is generated, is called as **structural level mode** and the mode in which behavioral level BDL description is generated is called **behavioral level mode**. The structural level internal format file output in structural level mode can be used to obtain the RTL description of top hierarchy by giving it as an input to Verilog-HDL output command or VHDL output command. BDL description file output in behavioral level mode can be used to obtain the RTL description of top hierarchy by executing behavioral synthesis.

L.2. Usage

- **Format**

`topmodgen [option] filename[_E.IFF]`

One or more files can be specified for “*filename* [_E.IFF]”.

In case the extension of specified file is not “.IFF”, it will be assumed that the extension has been omitted and will be suffixed to filename.

- **Options**

Option	Description
<code>-o filename[IFF]</code>	Set output IFF file name to <i>filename</i> [.IFF]
<code>-name name</code>	Set top module name as <i>name</i>
<code>-behav [=YES]</code> <code>-behav=NO</code>	Execute in behavioral level mode (default) Execute in structural level mode
<code>-lic_wait=#</code>	License is already obtained through another process. However, in case license is not obtained, repeat retry within given time (Time specified: minutes) [0]
<code>-EE</code> <code>-EJ</code>	Display messages in English Display messages in Japanese (Default setting is as per environment variable LANG)
<code>-h, -H</code> <code>-v, -V</code>	Display help message Display version

Option	Description
(Structural level mode options)	
-e	In case the extension of specified file is not ".IFF", it will be assumed that the extension has been omitted and will be suffixed to filename.
-mon <i>filename</i> [.mon]	Specify the monitor tap definition file
-force_monitor	A shared register, that is not written, can be monitored by generating a monitor port by specifying the monitor tap.
-FcE	Generate the clock pin for synchronous register separately at negative edge. (This option is planned to be discontinued, and -FcE option of bdltran is recommended)
-Fcts	Add FCTS module to clock wiring.
-no_state	Do not generate the state.
(Behavioral level mode options)	
-mem_kind=[RnWm RWn]	Specify the number of ports of shared mem used for referencing and assignment. (n,m are constant numbers and by default, it is R1W1)
-mem_kind_ro=[RnWm RWn Rn]	Specify the number of ports used for referencing shared mem only. (By default, R1)
-mem_kind_wo=[RnWm RWn Wn]	Specify the number of ports used only for the assignment to shared mem. (By default, W1)
-mem_delay=[# #T x#]	Specify the delay in generated MLIB template. In case of #, #T, asynchronous memory and in case of x, synchronous memory

-Zmlib_mcnt_out	Generate MLIB, MCNT template Insert the clock port and connect to lower hierarchy which does not have clock type input signal before synthesis.
-Fcl[:NAME1:NAME2:...]	Specify the process name, in which the clock is to be inserted, by NAME. When NAME is omitted, all processes are targeted.
-FrI[:NAME1:NAME2:...]	Insert reset port and connect to lower hierarchy, which does not have reset type input signal before synthesis. Specify the process name, in which reset is inserted, by NAME When NAME is omitted, all the processes are targeted.
-Fc[:NAME]	Generate clock port in top module without fail Name of clock signal can be specified by NAME.
-Fr[:NAME]	Generate reset signal in top hierarchy without fail Name of reset signal can be specified by NAME

- **Attribute**

Attribute	Description	Settings Target
(Valid in all behavioral level mode) /* Cyber inst_name = name */	Specify the interface name of corresponding module.	Process function
/* Cyber con_name = name */	Specify the alias name of corresponding signal	Input output variable, shared variable

L.3. Generated File Name and Higher Hierarchy

The top hierarchy process name generated by default is the name connected in alphabetical order of "two characters_ of lower process name" after "Top_". For example, in case process names of input lower hierarchy are foo, bar and baz, top hierarchy name will become "TOP_ba_ba_fo". Top hierarchy process name can be specified using –name option.

In structural level mode, output file name is process name_E.IFF of top hierarchy, and in case of behavioral level mode, it becomes process name .BDL of top hierarchy. Output file name can be specified by using –o option.

L.4. Port Connections between Lower Hierarchy

Connections between respective I/O ports of lower hierarchy are made based on name, bit-width, and input/output direction. While focusing on connection of an I/O port of lower hierarchy, an external I/O port is generated in top hierarchy for connection if similar I/O ports are not available in separate lower hierarchy. In case another lower hierarchy has an I/O port with the same name and bit-width, then it is divided into following 3:

1. When all ports are input ports

Generates an input port in the top hierarchy for external connection and connects it to all input ports. (e.g., clock port or reset port etc.)

2. When there is only one output port and the rest are all input ports

No I/O port is generated in the top hierarchy externally. All input ports are connected to the one output port.

3. When two or more than two output ports are present

In case valid signal of output port exists, then optional connection is done by nmux. However, if it does not exist, it is connected by OR logic. For details, refer to **section L.6.3**.

Further, while focusing on the I/O ports of lower hierarchy, there might be a similar port in a different lower hierarchy but with different bit-width. Such a case will result in an error.

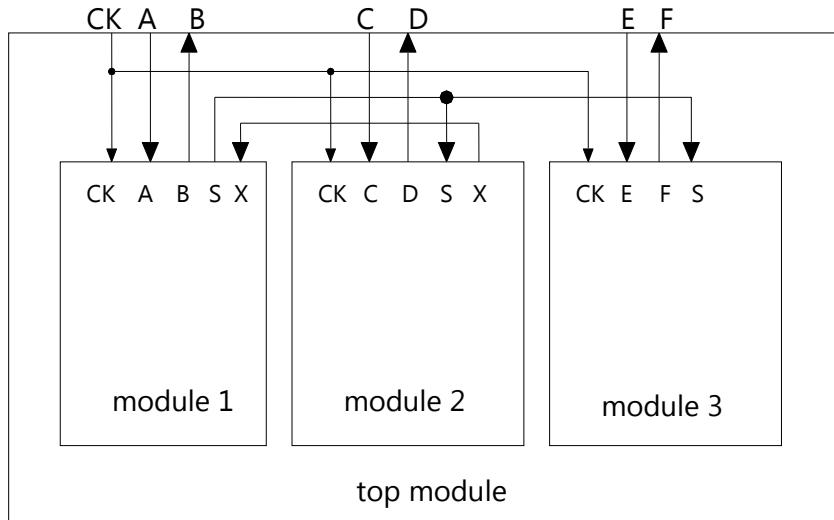


Figure 136: Basic concept diagram of topmodgen

L.5. Variable Connection in Behavioral Level Mode

This functionality reads multiple description (both behavioral level and structural level are possible) including in/out variables and shared variables, and automatically generates behavioral level BDL.

L.5.1 Functional Description

Any one of behavioral level or structural level input IFF file can be specified. However, in case behavioral level IFF file is specified, and port count other than clock and reset gets changed by behavioral synthesis, then it is necessary to update the generated top module description. In case clock and reset gets added owing to behavioral description, clock and reset port name of top module and lower module can be specified by -Fc, -Fr and -Fcl, -Frl options respectively. Further, description that uses pointers for I/O port are not supported therefore, structural level IFF should be used for input when such description are the input.

In case in/out ports between modules have same name automatic connection is carried out. In case of shared variable, each process is connected by using defmod abstract interface.

Further, template description of MCNT and MLIB files for top module and lower module can be generated by specifying options. For Behavioral synthesis system, CyberWorkBench Reference Manual (defmod abstract Interface) describes about the defmod abstract interface.

Specify the option "-behav" along with input IFF file to use this mode.

RTL description of top hierarchy can be acquired by implementing the behavioral synthesis for BDL files generated by specifying the following command.

```
> topmodgen -behav a.IFF b.IFF c.IFF" (-behav omission is possible)
```

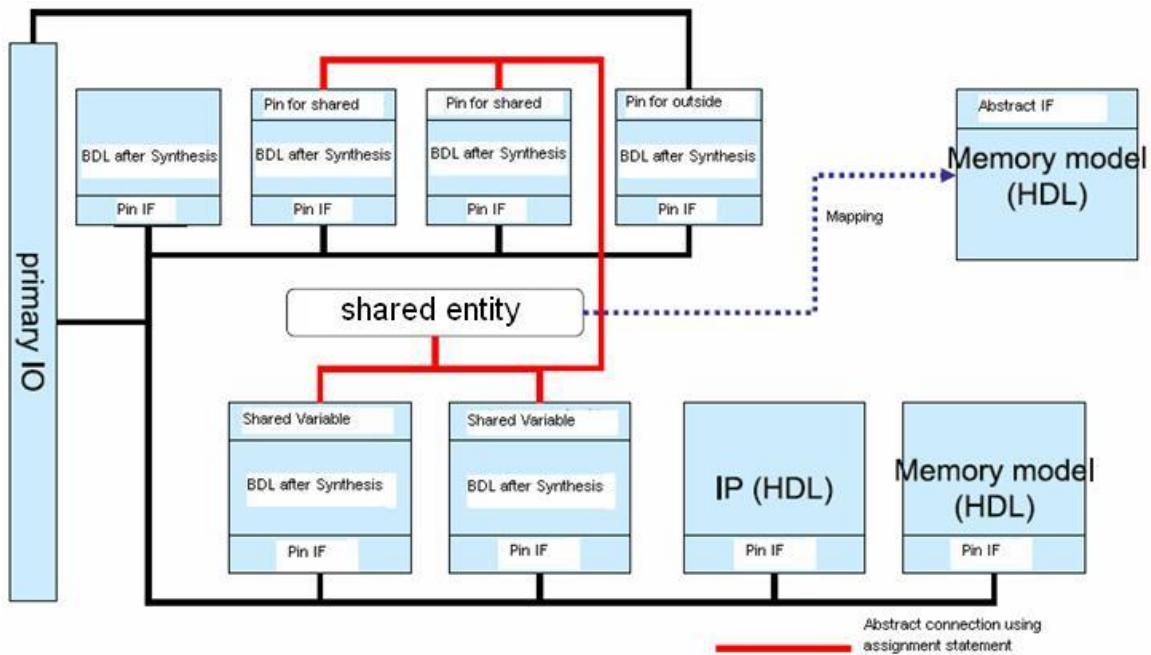


Figure 137: Top module generated in behavioral level mode

Given below is the detailed description about the interconnection of in/out variables and shared variables in behavioral level model.

L.5.2 Connection of in/out variable

L.5.2.1 Connection of Non Structural Variables

Given below is the description about connection of nonstructural variable.

As mentioned in the example below, in case name of the variable inside structural level description matches with name of the variable inside behavioral level description, the basic operation will be as under:

- (1) Both will be connected if one I/O type is "in" and other I/O type is "out",
- (2) I/O port will be output to top hierarchy if both the I/O types are "in".
- (3) In case I/O type for both is "out", then both are connected using OR logic and output to top hierarchy when valid signal of corresponding signal does not exist. And, it is output to top hierarchy using the NMUX logic when valid signal exists.

module 1 (behavioral level description):

```
in ter (0:8)x;
out ter (0:8)y;
in ter (0:8)z[2];
out ter (0:8)v
/*Cyber
valid_sig_gen=v_valid1*/;
out ter (0:8)w;
process module1( ){
    .
    .
}
```

module 2 (Structural level description):

```
in unsigned ter(0:8) x;
in unsigned ter(0:8) y;
out unsigned ter(0:8) z_a00
/*Index is 0 Array variable z*/;
out unsigned ter(0:8) z_a01
/*Index is 1 Array variable z*/;
out unsigned ter(0:8) v;
out unsigned ter(0:8) v_valid2
/*v !valid signal*/;
out unsigned ter(0:8) w;
clock           CLOCK;
reset           RESET;
process module2() {
    .
    .
}
```

Example is shown below.

Output top hierarchy file:

```

defmod module1 {
    in unsigned ter(0:8) x;
    out unsigned ter(0:8) y;
    in unsigned ter(0:8) z[2];
    out unsigned ter(0:8) w;
} INST_module1;
defmod module2 {
    in unsigned ter(0:8) x;
    in unsigned ter(0:8) y;
    out unsigned ter(0:8) z[2];
    out unsigned ter(0:8) w;
    clock      CLOCK;
    reset      RESET;
} INST_module2;
    in unsigned ter(0:8) x;
    unsigned ter(0:8) z[2];
    out unsigned ter(0:8) v;
    out unsigned ter(0:1) v_valid1;
    out unsigned ter(0:8) w;
    unsigned ter(0:8) y;
    out unsigned ter(0:1) v_valid2;
    unsigned ter(0:8) TG_0;
    unsigned ter(0:1) TG_1;
    unsigned ter(0:8) TG_2;
    unsigned ter(0:8) TG_3;
    unsigned ter(0:1) TG_4;
    unsigned ter(0:8) TG_5;
process Top_mo_mo(){
    INST_module1.x ::= x;
    y ::= INST_module1.y;
    INST_module1.z[0] ::= z[0];
    INST_module1.z[1] ::= z[1];
    TG_0 ::= INST_module1.v;
    TG_1 ::= INST_module1.v_valid1;
    TG_2 ::= INST_module1.w;
    INST_module2.x ::= x;
    INST_module2.y ::= y;
    z[0] ::= INST_module2.z[0];
    z[1] ::= INST_module2.z[1];
    TG_3 ::= INST_module2.v;
    TG_4 ::= INST_module2.v_valid2;
    TG_5 ::= INST_module2.w;
    v ::= nmux {
        when (TG_1) : TG_0;
        when (TG_4) : TG_3;
    };
    v_valid1 ::= TG_1;
    v_valid2 ::= TG_4;
    w ::= TG_5 | TG_2;
    return;
}

```

L.5.2.2 Connection of Structural Variable

Below is the explanation regarding the basic operation when structural description is included in the input file:

- (1) In case of 2 different processes, if structure variable with identical name and in identical format are present, and when order/name/type of the structure members is uniform, in/out structure variables will be connected.
- (2) In case of 2 different processes, if structure type is different, structure variable name is same, and when order/name/type of the structure members is uniform (However, regarding I/O type, it is inserted when it changes to any one of the "one is in and other is out", "both in", "both out"), each structure members of both the processes will be connected.

Given below is the example of typical input description related to the operation, mentioned in (1), and it shows the top hierarchy description that is considered as output by "topmodgen". Same name structural variables of ST1 structure are connected and output.

module 1 (behavioral level description):

```

struct ST2{
    var (0:8)x[2][3];
    var (0:16)y;
};

struct ST1{
    var (0:32)x;
    var (0:8)y;
    struct ST2 st2[4];
};
in ter struct ST1 st1_a;
out ter struct ST1 st1_b;
out ter struct ST1 st1_c;
process module1( ){
    .
    .
}

```

module 2 (Behavioral level description):

```

struct ST2{
    var (0:8) x[2][3];
    var (0:16) y;
};

struct ST1{
    var (0:32) x;
    var (0:8) y;
    struct ST2 st2[4];
};
out ter struct ST1 st1_a;
in ter struct ST1 st1_b;
out ter struct ST1 st1_c;
process module2( ){
    .
    .
}

```

Output top hierarchy file:

```

struct ST2 {
    unsigned var(0:8) x[2][3];
    unsigned var(0:16)y;
};

struct ST1 {
    unsigned var(0:32)x;
    unsigned var(0:8) y;
    struct ST2 st2[4];
};

defmod module1 {
    in ter struct ST1 st1_a;
    out ter struct ST1 st1_b;
} INST_module1;

defmod module2 {
    out ter struct ST1 st1_a;
    in ter struct ST1 st1_b;
} INST_module2;

ter struct ST1 st1_a;
ter struct ST1 st1_b;

process
Top_mo_mo(){
    INST_module1.st1_a ::= st1_a;
    st1_b ::= INST_module1.st1_b;
    st1_a ::= INST_module2.st1_a;
    INST_module2.st1_b ::= st1_b;
    return;
    $
}

```

Below is the example of typical input description related to operation in (2), and it shows the top hierarchy description that is considered as output by "topmodgen". In top hierarchy, each member in structure is connected and output.

module 1 (behavioral level description):

```

struct STB1{
    out ter (0:8)z[2];
};

struct STA1{
    out ter (0:8)x;
    in ter (0:16)y;
    struct STB1 stb;
};
struct STA1 sta;
process module1( ) {
    .
    .
}

```

module 2 (Behavioral level description):

```

struct STB2{
    in ter (0:8)z[2];
};

struct STA2{
    out ter (0:8)x;
    in ter (0:16)y;
    struct STB2 stb;
};
struct STA2 sta;
process module2( ) {
    .
    .
}

```

Output top hierarchy file:

```

struct STB1 {
    out unsigned ter(0:8) z[2];
};
struct STA1 {
    out unsigned ter(0:8) x;
    in unsigned ter(0:16) y;
    struct STB1 stb;
};
struct STB2 {
    in unsigned ter(0:8) z[2];
};
struct STA2 {
    out unsigned ter(0:8) x;
    in unsigned ter(0:16) y;
    struct STB2 stb;
};
defmod module1 {
    ter struct STA1 sta;
} INST_module1;
defmod module2 {
    ter struct STA2 sta;
} INST_module2;
out unsigned ter(0:8) sta_x;
in unsigned ter(0:16) sta_y;
    unsigned ter(0:8) TG_0[2];
    unsigned ter(0:8) TG_1;
    unsigned ter(0:8) TG_2;
process Top_mo_mo() {
    TG_1 ::= INST_module1.sta.x;
    INST_module1.sta.y ::= sta_y;
    TG_0[0] ::= INST_module1.sta.stb.z[0];
    TG_0[1] ::= INST_module1.sta.stb.z[1];
    TG_2 ::= INST_module2.sta.x;
    INST_module2.sta.y ::= sta_y;
    INST_module2.sta.stb.z[0] ::= TG_0[0];
    INST_module2.sta.stb.z[1] ::= TG_0[1];
    sta_x ::= TG_2 | TG_1;
    return;
}

```

L.5.3 Connection of “shared” variable

L.5.3.1 Connection of unstructured variable

The shared variable that exists in lower module, are instantiated as global variable in top module and an assign statement is generated that connects these shared variables of lower hierarchy. In case same name variables have different bit width, word length, and bit declaration order etc, an error is generated similar to that in structural level mode. Regarding the shared reg variable, error is generated even when clock signal name or reset signal name are different. Given below is template of the top hierarchy description generated.

```

Declaration of entity of shared variable;
defmod lower process name{
    I/O port declaration
    shared variable declaration
} INST_lower process name;
top hierarchy I/O port declaration;
process process_name ()
{
    ...
    ...
    allstates {
        assign (variable name of top module, INST_lower process
        name.sharred variable name );
        ...
    }
    INST_lower process name port name::= INST_lower process name. port
    name; ...
    External port name::= INST_lower process name. port name; ...
    return ;
}

```

L.5.3.2 Connection of Structured Variable

Explanation regarding the basic operation when structure description is included in input file.

1. In 2 different processes, assign statement in the format specifying the structure variable in both arguments of assign statement is output when shared structured variable with same name exists in similar type structure.
2. In 2 different processes, assign statement in the format specifying these structured member in both argument of assign statement is output when unshared structured variable (shared variable is included in structured member) with same name exists in similar type structure.

Also, typical input description example related to operation (2.), or top hierarchy description output by topmodgen is shown below. Same name structure variable of ST1 structure is connected and output to respective same entity.

module1(behavioral level description):

```
struct ST2{
    var (0:8)x;
    var (0:16)y;
};

struct ST1{
    var (0:32)x;
    var (0:8)y;
    struct ST2 st2;
};

shared mem struct ST1
st1_a[32][8];
shared reg struct ST1 st1_b;
.

.

process module1( )
.
.
}
```

Module2(behavioral level description):

```
struct ST2{
    var (0:8)x;
    var (0:16)y;
};

struct ST1{
    var (0:32)x[2];
    var (0:8)y;
    struct ST2 st2;
};

shared mem struct ST1
st1_a[32][8];
shared reg struct ST1 st1_b;
.

.

process module2( )
.
.
```

Output top hierarchy file

```

struct ST2 {
    unsigned var(0:8) x;
    unsigned var(0:16) y;
};

struct ST1 {
    unsigned var(0:32) x[2];
    unsigned var(0:8) y;
    struct ST2 st2;
};

defmod module1 {
    shared mem struct ST1 st1_a[32][8];
    shared reg struct ST1 st1_b;
    .
    .
} INST_module1;
defmod module2 {
    shared mem struct ST1 st1_a[32][8];
    shared reg struct ST1 st1_b;
    .
    .
} INST_module2;
mem struct ST1 st1_a[32];
reg struct ST1 st1_b;
.

process
Top_mo_mo()
{
    allstates {
        assign(st1_a, INST_module1.st1_a);
        assign(st1_a, INST_module2.st1_a);
        assign(st1_b, INST_module1.st1_b);
        assign(st1_b, INST_module2.st1_b);
    }
    .
    .
}

```

Also, typical input description example related to operation (2), or top hierarchy description output by topmodgen is shown below. Each member of structure defined by sub hierarchy becomes the output connected to respective same entity.

**module1(behavioral level
description**

```
struct ST2{
    shared reg (0:8)x;
    shared mem (0:16)y[64][4];
};

struct ST1{
    shared reg (0:32)x;
    shared mem (0:8)y[64];
    struct ST2 st2;
};

struct ST1 st1;
.

.

process module1( )
.
.
}
```

**module2(behavioral level
description**

```
struct ST2{
    shared reg (0:8)x;
    shared mem (0:16)y[64][4];
};

struct ST1{
    shared reg (0:32)x;
    shared mem (0:8)y[64];
    struct ST2 st2;
};

struct ST1 st1;
.

.

process module2( )
.
.
}
```

Output top hierarchy file

```

struct ST2 {
    shared unsigned reg(0:8) x;
    shared unsigned mem(0:16) y[64][4];
};

struct ST1 {
    shared unsigned reg(0:32) x;
    shared unsigned mem(0:8) y[64];
    struct ST2 st2;
};

defmod module1 {
    struct ST1 st1;
    .
    .
} INST_module1;
defmod module2 {
    struct ST1 st1;
    .
    .
} INST_module2;
    unsigned reg(0:32) st1_x;
    unsigned mem(0:8) st1_y[64];
    unsigned reg(0:8) st1_st2_x;
    unsigned mem(0:16) st1_st2_y[64][4];
process
Top_mo_mo()
{
    allstates {
        assign(st1_x, INST_module1.st1.x);
        assign(st1_x, INST_module2.st1.x);
        assign(st1_y, INST_module1.st1.y);
        assign(st1_y, INST_module2.st1.y);
        assign(st1_st2_x,
               INST_module1.st1.st2.x);
        assign(st1_st2_x,
               INST_module2.st1.st2.x);
        assign(st1_st2_y,
               INST_module1.st1.st2.y);
        assign(st1_st2_y,
               INST_module2.st1.st2.y);
    }
    .
    .
}

```

L.5.4 MLIB/MCNT Template Generation

By specifying the option -Zmlib_mcnc_out at the same time with -behav option, MLIB file for shared memory and MCNT file template for respective top hierarchy and lower hierarchy can be generated simultaneously.

> topmodgen -behav -Zmlib_mcnc_out a.IFF b_E.IFF c.IFF

Behavioral synthesis can be executed, by synthesizing the top hierarchy and each hierarchy, after changing DELAY, KIND, DEFMOD (pin information) etc of generated MLIB file, MCNT file, as per the memory used.

Port counts of generated memory are automatically decided according to number of read ports and write ports of memory of lower modules. However, it can be controlled by using execution option (-mem_kind, -mem_kind_ro, -mem_kind_wo) or attribute (-rw_port). In case port count is specified for entire memory, execution option is used, and in case specified separately for respective memory, attribute is used.

```
shared mem(0:8) m [100] /* Cyber rw_port = r1 */;
```

In case different attribute values are specified for a shared variable with same name, maximum port count is calculated considering the minimum write and read ports required to avoid any port conflict. However, error will be generated, when both rw? and r?w? are specified as rwport attribute value for the shared variable with same name.

L.5.5 Connection between Ports with Different Name

This tool basically performs connection related to port with similar names. However, connection between ports with different name is also possible by user specification.

By specifying alias name of corresponding port in con_name attribute, port between each process based on that alias name can be connected automatically.

In the below example, in each port x,y,z in process foo, it is connected in a format same as respective ports a,b,c by adding con_name attribute.

**module1(behavioral level
description):**

```
in ter (0:8) a
/*Cyber con_name=x*/;
out ter (0:16) b[2]
/*Cyber con_name=y*/;
shared reg c
/*Cyber con_name=z*/;
process module1() {
    .
    .
}
```

**module2(behavioral level
description):**

```
out ter (0:8) x;
out ter (0:16) y[2];
shared reg z;
process module2() {
    .
    .
}
```

Output top hierarchy file

```

defmod module1 {
    in unsigned ter(0:8) a;
    out unsigned ter(0:16) b[2];
    shared unsigned reg(0:1) c;
} INST_module1;
defmod module2 {
    out unsigned ter(0:8) x;
    out unsigned ter(0:16) y[2];
    shared unsigned reg(0:1) z;
} INST_module2;
    unsigned ter(0:8) x;
    out unsigned ter(0:16) y[2];
    unsigned reg(0:1) z;
    unsigned ter(0:16) TG_0[2];
    unsigned ter(0:16) TG_1[2];
process Top_mo_mo() {
    allstates {
        assign(z, INST_module1.c);
        assign(z, INST_module2.z);
    }
    INST_module1.a ::= x;
    TG_0[0] ::= INST_module1.b[0];
    TG_0[1] ::= INST_module1.b[1];
    x ::= INST_module2.x;
    TG_1[0] ::= INST_module2.y[0];
    TG_1[1] ::= INST_module2.y[1];
    y[0] ::= TG_1[0] | TG_0[0];
    y[1] ::= TG_1[1] | TG_0[1];
    return;
}

```

Furthermore, con_name attribute can implement topology with lots of variation by combining with multiple instance generation functionality. For details, refer to **Section L.5.6**.

L.5.6 Multiple Instance Generation Functionality

L.5.6.1 Overview

Top hierarchy retaining the instance of input lower hierarchy more than once can be generated by specifying the inst_name attribute in the description of target input lower hierarchy. Further, topology with lots of variations can be implemented by combining with con_name attribute explained in **Section L.5.5**.

L.5.6.2 Related attributes

Attribute	Description	Settings Target
<code>/* Cyber inst_name = name */</code>	Specify the instance name of corresponding module	Process function
<code>/* Cyber con_name = name */</code>	Specify the alias name of corresponding signal	Input output variable, Shared variable

Below, format of inst_name attribute is shown.

In addition, format of con_name attribute is also same as inst_name attribute.

Format:

```
/*Cyber inst_name=(character string) (scope specifier) (character
string) (scope specifier)...*/
Further   /*Cyber inst_name={ (character string) (scope specifier) (character
string)...(scope specifier), (character string)(scope
specifier) (character string)...(scope specifier),...}
```

- Usable identifier in character string section are ? and _ and numeric character and alphabets. However, ? is a metacharacter and corresponding process name is displayed when used with inst_name attribute.
- In the scope specification part, the 3 types of from to type, width specification type and comma specification type can be used.
 - from to specification
ex. foo{0..3}{0..2} → foo00,foo01,foo02,foo10,...,foo32
 - Width specification
ex. foo{0:10} → foo0,foo1,...,foo9
 - Comma specification
ex. {foo,bar,baz} → foo,bar,baz

L.5.6.3 Description

- Lower hierarchy instance can be generated more than once by specifying the inst_name attribute in process function part of input lower hierarchy.
- Input output port name of every instance of generated lower hierarchy module can be specified by specifying the con_name attribute in input output port declaration part of input lower hierarchy.
- Specification method of inst_name attribute, con_name attribute along with 3 types (from to specification, width specification, comma specification) can be used. Element count specified in inst_name attribute and instance element count specified in con_name attribute should be the same (However, like foo1..6 and bar1..21..3, there is no problem if dimension between the two even if it differs).

Below, it is further explained by using examples.

In the below input BDL, the value of inst_name attribute added to process function is ?{1..3}. Here, metacharacter ? represents the process name foo. Hence, generated instance consists of 3, foo1,foo2,foo3.

Further, con_name attribute {a,s_{1..2}} is added in input port a. In top hierarchy, input port a is unrolled with "a" related to instance foo1 , "s_1" related to instance foo2, "s_2" in instance foo3. Also, con_name attribute ?{1..3} is added in port b. Here, metacharacter ? shows the corresponding port name (here it is b). Therefore in top hierarchy, input port b is unrolled with "b1" related to instance foo1, "b2" related to instance foo2, "b3" in instance foo3.

Regarding other input output port x,y also, each will be unrolled according to attribute in every instance in the same way.

Regarding this description,

if > topmodgen -behav foo.IFF -name top

is specified, instance of element count specified by attribute in generated in top hierarchy. Top hierarchy having connection relation as shown in **Figure 138** is generated.

Input BDL

```
#define ATTR_VALUE ?{1..3}
in var(0..0) a/*Cyber con_name={a,s_{1..2}}*/;
in var(7..0) b/*Cyber con_name=ATTR_VALUE*/;
out var(0..0) x/*Cyber con_name={s_{1..2},x}*/;
out var(7..0) y/*Cyber con_name=ATTR_VALUE*/;
/* Cyber inst_name=ATTR_VALUE*/
process foo(){
    .
    .
}
```

Output file

```

defmod foo {
    in ter (0..0) a;
    in ter (7..0) b;
    out ter (0..0) x;
    out ter (7..0) y;
} fool,foo2,foo3;
in ter (0..0) a;
in ter (7..0) b1;
in ter (7..0) b2;
in ter (7..0) b3;
out ter (0..0) x;
out ter (7..0) y1;
out ter (7..0) y2;
out ter (7..0) y3;
ter (0..0) s_1;
ter (0..0) s_2;
process top() {
    fool.a ::= a;
    fool.b ::= b1;
    s_1 ::= fool.x;
    y1 ::= fool.y;
    foo2.a ::= s_1;
    foo2.b ::= b2;
    s_2 ::= foo2.x;
    y2 ::= foo2.y;
    foo3.a ::= s_2;
    foo3.b ::= b3;
    x ::= foo3.x;
    y3 ::= foo3.y;
    return;
}

```

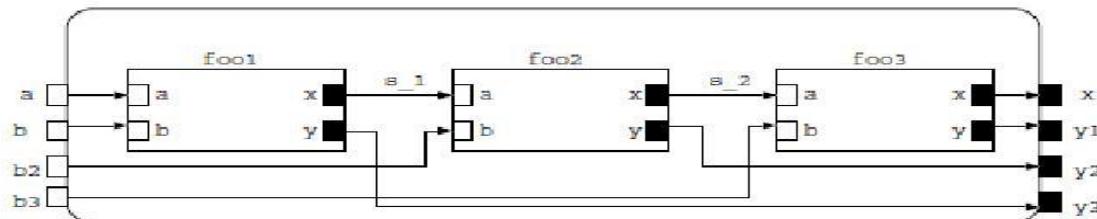


Figure 138: Connection relation diagram of every instance in top hierarchy

Also, reference is given regarding usage of this functionality in SystemC. User can add `inst_name` attribute in `SC_MODULE` or member function definition part for the target to be duplicated. Below, examples are given with explanation.

- In case input SystemC description is output in multiple hierarchy format
As shown below, in the description in which multiple `SC_CTHREAD` exists, sub hierarchy corresponding to top hierarchy and each `SC_CTHREAD` is generated.

```

SC_MODULE(test) {
    sc_in<clk>      clk;
    sc_in<bool>      rst;
    sc_in<int>        din;
    sc_out<int>       dout;
    sc_signal<int>    x;
    SC_CTOR(test) {
        SC_CTHREAD(entry, clk.pos());
        watching(rst.delayed() == true);
        SC_CTHREAD(entry2, clk.pos());
        watching(rst.delayed() == true);
    };
    void entry() {
        wait();
        while (1) {
            wait();
            dout = x;
            wait();
        }
    }
    void entry2() {
        wait();
        while (1) {
            x = din;
            wait();
            wait();
        }
    }
}
;

```

In such a case, in cases where input SystemC description is output in multiple hierarchy format, when top hierarchy is to be duplicated, inst_name attribute must be added in SC_MODULE definition part as shown below.

```

/*Cyber inst_name=?{0..2}*/
SC_MODULE(test) {

```

Also, when sub hierarchy is to be duplicated, inst_name attribute must be added in member function definition part as shown below.

```

/*Cyber inst_name=?_{0..2}*/
void entry() {

```

- In case input SystemC description is output in single hierarchy format

For example, in the description in which below mentioned single SC_CTHREAD exist is output in single hierarchy format.

```

SC_MODULE(test) {
    sc_in<clk>      clk;
    sc_in<bool>      rst;
    sc_in<int>        din;
    sc_out<int>       dout;
    SC_CTOR(test) {
        SC_CTHREAD(entry, clk.pos());
        watching(rst.delayed() == true);
    };
    void entry() {
        int tmp;
        wait();
        while (1) {
            wait();
            tmp = din.read();
            wait();
            dout.write(tmp);
            wait();
        }
    }
};

```

When input SystemC description is output in single hierarchy format, duplication of corresponding process is possible by adding `inst_name` attribute for member function definition part as shown below.

```

/*Cyber inst_name=?_{0..2}*/
void entry() {

```

L.5.6.4 Usage example

Designer can carry out connection of various types by properly combining `con_name` attribute and `inst_name` attribute.

1. 1:N Connection

4 unit generation of process `foo` instance and a descriptive example is shown when it is to be connected between in port `x` of instance for `i` unit ($i=0,1,2,3$) of process `foo` and out port `x[i](i=0,1,2,3)` of process `bar`

```
> topmodgen -behav foo.IFF bar.IFF -name top
```

Process foo description:

```

in ter (0:8)x
/*Cyber con_name=x_{0..3}*/;
.

/*Cyber inst_name=INST_{1..4}*/
process foo(){
.
.
}

```

Process bar description:

```

out ter (0:8)x[4];
process bar(){
.
.
}

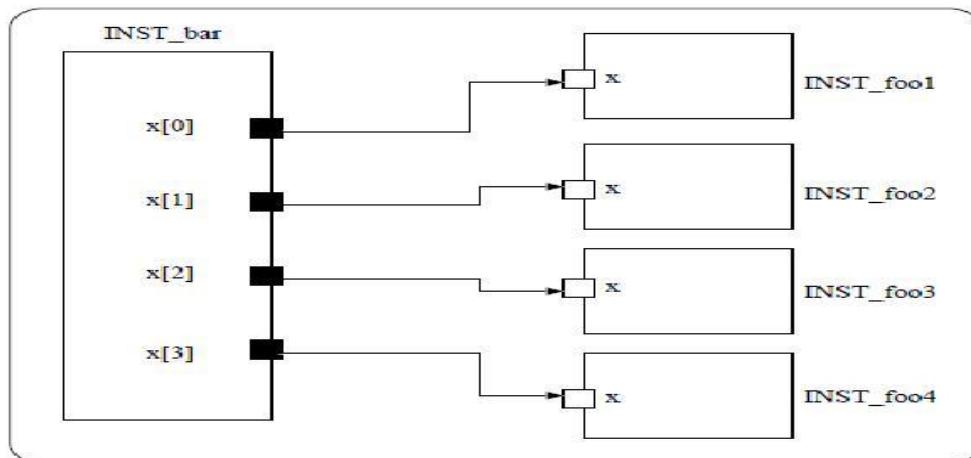
```

top.BDL(generated top hierarchy description):

```

defmod foo {
    in ter(0:8) x;
    .
    .
} INST_foo1,INST_foo2,INST_foo3,INST_foo4;
defmod bar {
    out ter(0:8) x[4];
} INST_bar;
    ter(0:8) x[4];
process top(){
    INST_foo1.x ::= x[0];
    INST_foo2.x ::= x[1];
    INST_foo3.x ::= x[2];
    INST_foo4.x ::= x[3];
    x[0] ::= INST_bar.x[0];
    x[1] ::= INST_bar.x[1];
    x[2] ::= INST_bar.x[2];
    x[3] ::= INST_bar.x[3];
    return;
}

```

**Figure 139:** Connection relation diagram between process foo – bar**2. M × N connection**

Process operates parallel to M unit and descriptive example and top hierarchy generated in case of already connected straight to N unit is shown. Example in case M=4, N=3.

```
> topmodgen -behav foo.IFF -name top
```

Process foo description:

```
in ter (0:8) x
/*Cyber con_name={t{1..2}{1..4},?{1..4}}*/;
out ter (0:8) a
/*Cyber con_name={?{1..4},t{1..2}{1..4}}*/;
/*Cyber inst_name=?{1..3}{1..4}*/
process foo(){}
}
```

top.BDL(generated top hierarchy description):

```
defmod foo {
    in ter(0:8) x;
    out ter(0:8) a;
} foo11,foo12,foo13, foo14,
    foo21,foo22,foo23,foo24,
    foo31,foo32,foo33,foo34;
in ter(0:8) x1,x2,x3,x4;
out ter(0:8) a1,a2,a3,a4;
    ter(0:8) t11,t12,t13,t14;
    ter(0:8) t21,t22,t23,t24;
process top(){
    foo11.x ::= t11;
    a1 ::= foo11.a;
    foo12.x ::= t12;
    a2 ::= foo12.a;
    foo13.x ::= t13;
    a3 ::= foo13.a;
    foo14.x ::= t14;
    a4 ::= foo14.a;
    foo21.x ::= t21;
    t11 ::= foo21.a;
    foo22.x ::= t22;
    t12 ::= foo22.a;
    foo23.x ::= t23;
    t13 ::= foo23.a;
    foo24.x ::= t24;
    t14 ::= foo24.a;
    foo31.x ::= x1;
    t21 ::= foo31.a;
    foo32.x ::= x2;
    t22 ::= foo32.a;
    foo33.x ::= x3;
    t23 ::= foo33.a;
    foo34.x ::= x4;
    t24 ::= foo34.a;
    return;
}
```

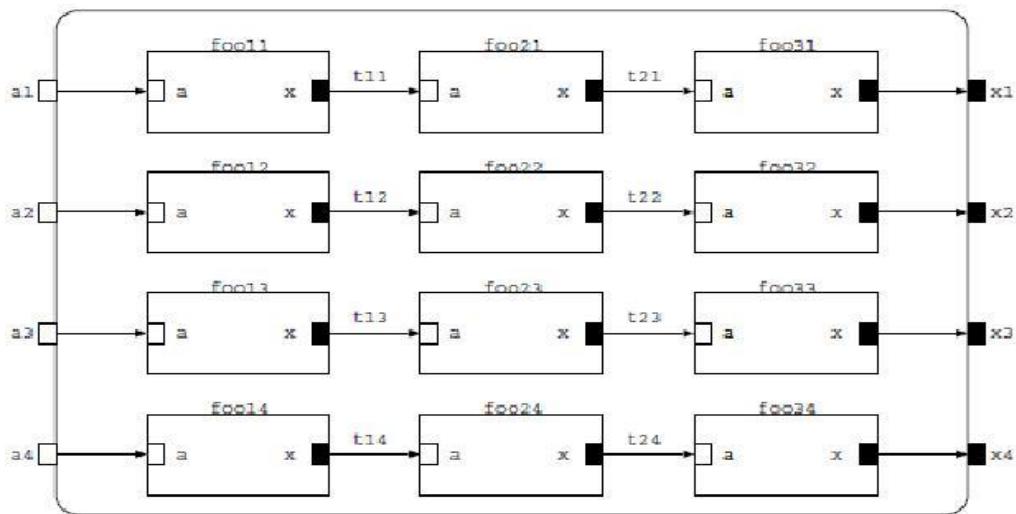


Figure 140: M × N connection relation diagram for process foo

L.6. Variable connection in structural level Mode

L.6.1 Connection of shared reg

A register that implements the “shared reg” variable, specified in the lower level modules, will be generated in the top hierarchy. If the “shared reg” variable with the same name exists in different lower hierarchy, a multiplexer will be added as shown in **Figure 141**. The addition of the multiplexer facilitates the sharing of the shared register by these modules. When a shared reg variable with same name but different bit-widths exists in separate lower hierarchy, it will result in an error. In a similar fashion, when a “shared reg” array with the same name but different number of elements exists in different lower hierarchy, it results in an error.

```
shared reg(0:8) r;
process module1( ) {
    .
    .
    .
}
```

```
shared reg(0:8) r;
process module2( ) {
    .
    .
    .
}
```

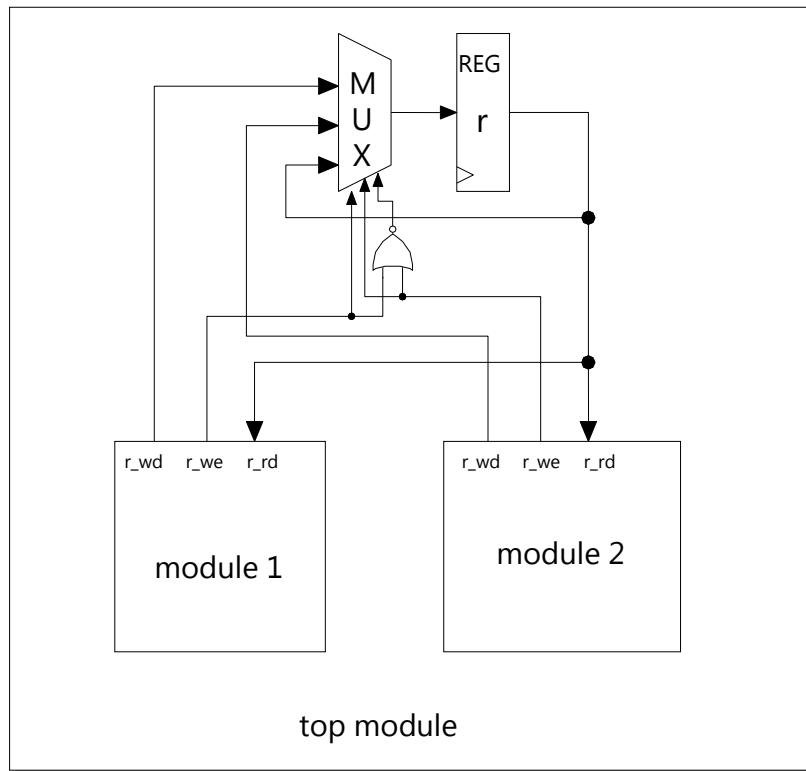


Figure 141: Connection of “shared reg” variable

L.6.2 Connection of “outside mem” and “shared mem” variables

Memories specified in lower hierarchy and that implements the “shared mem” and “outside mem” variables are assumed to be implemented outside the “top hierarchy” generated using topmodgen tool. Accordingly, the I/O ports are generated in the top hierarchy for connecting to this lower hierarchy.

Note:

If a shared mem and an outside mem, which has both read-write port, is generated by using option –Zrw1=1port, then read-write data port is generated as bi-directional port which is not supported by top hierarchy output tool, therefore, it is necessary to synthesize by two ports namely, read data input port and write data output port, without specifying the option –Zrw1=1port (Refer to **section 11.5.1.1**).

L.6.2.1 “outside mem” variable

Memory that implements the “outside mem” variable generates separate I/O port assuming they will be used in different memories, as shown in **Figure 142**.

Module1.c:

```
outside mem(0:8) memA[100] /* Cyber sig2mu = memoryA */;
process module1( ) {
    .
    .
    .
}
```

Module2.c:

```
outside mem(0:8) memB[200] /* Cyber sig2mu = memoryB */;
process module2( ) {
    .
    .
    .
}
```

mem.MLIB

```

#@LIB { module }
@MLIB {
    NAME      memoryA
    KIND      RW1
    BITWIDTH 8
    DELAY    1T
    WORD     100
    ...
}
@MLIB {
    NAME      memoryB
    KIND      RW1
    BITWIDTH 8
    DELAY    1T
    WORD     200
    ...
}
#@END { module }

```

mem.MCNT

```

#@CNT { module }
@MCNT {
    NAME      memoryA
    LIMIT    1
}
@MCNT {
    NAME      memoryB
    LIMIT    1
}
#@END { module }

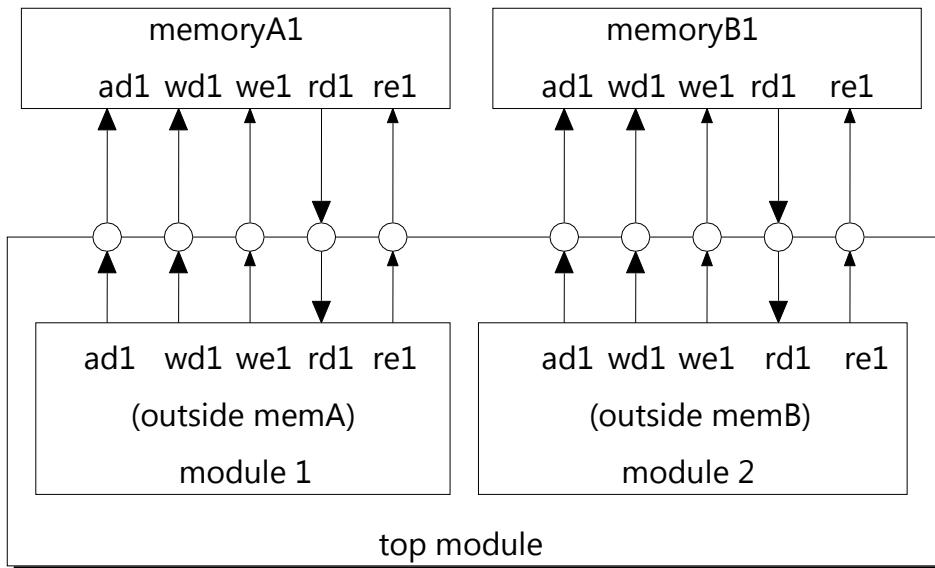
```

In case same number with same memory type name exists, it will result in an error.

L.6.2.2 “Shared mem” Variable

With regards to shared mem variable, if the memory type name specified in the memory library file and memory number is same, then I/O port are generated to use memory by sharing (This is regardless of the variable name used in specification).

Figure 143 shows an example where different ports are used by respective lower hierarchy.

**Figure 142:** Connection of "outside mem" variable**Module1.c:**

```
shared mem(0:8) memC[200] /* Cyber sig2mu_port = memory.r1w1 */;
process module1( ) {
    .
    .
    .
}
```

Module2.c:

```
shared mem(0:8) memC[200] /* Cyber sig2mu_port = memory.r2w2 */;
process module2( ) {
    .
    .
    .
}
```

```

mem.MLIB

#@LIB { module }
@MLIB {
    NAME memory
    KIND RW2
    BITWIDTH 8
    DELAY 1T
    WORD 200
    ...
}
#@END { module }

mem.MCNT

#@CNT { module }
@MCNT {
    NAME memory
    LIMIT 1
}
#@END { module }

```

When lower hierarchy is synthesized, following notes are necessary.

Notes:

In order to make memory ports independent with respect to the lower hierarchy, the memory ports should be specified in such a way that a different port of same memory is used by the respective lower hierarchy. Additionally, the “-ap?:o” option should be specified so that I/O ports for unused ports are not generated (Refer to **Section 11.5.1.6**).

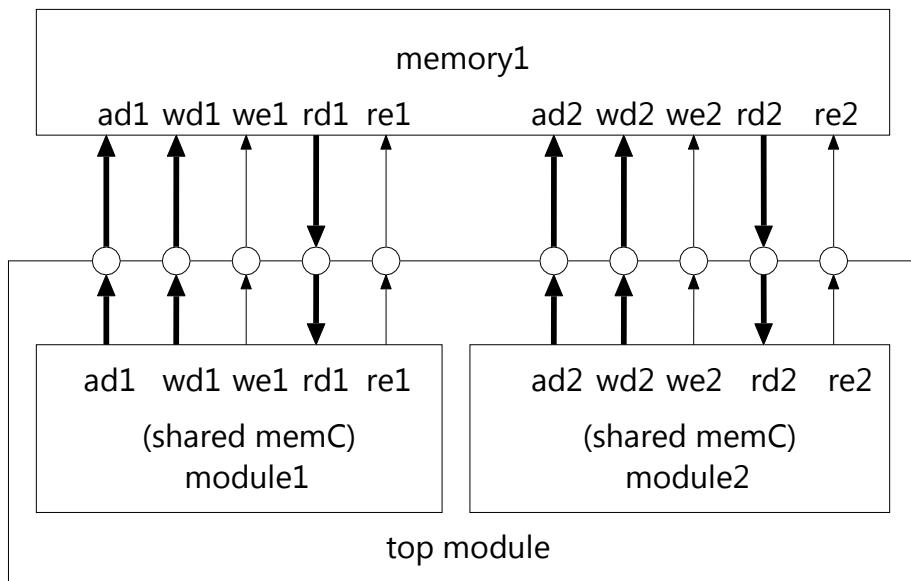


Figure 143: The “Shared mem” variable with independent memory ports

When referencing and assigning the “shared mem” variable by sharing and using a port by multiple lower hierarchies, a multiplexer will be added so that the same port can be shared and used at top hierarchy, as shown in **Figure 144**. Hence, there are two restrictions, which are as follows:

Note:

1. Since chip-select signal or read-enable signal is necessary as multiplexer selection signal, it is important to generate the following during synthesis of lower hierarchy with shared mem having read-write port:

- Generate read-enable signal by adding the “-Zmem_re” option or the “mem_re” attribute.
- Generate chip-select signal by specifying the “-Zmem_cs” option.

(Refer to **sections 11.5.1.1 and 11.5.1.2**)

2. An error occurs if multiple lower hierarchies write to asynchronous SRAM. This is because there will be a timing problem regarding write-enable signal.

In this case, if synthesis is performed using synchronous SRAM by specifying “-syncRAM” option of topmodgen, synthesis will be possible as there will be no timing problem.

When synthesizing lower hierarchy using synchronous SRAM, memory reference or assignment timings should be synthesized for read/write in later half of clock cycle (Refer to **Section 11.5.1.5**).

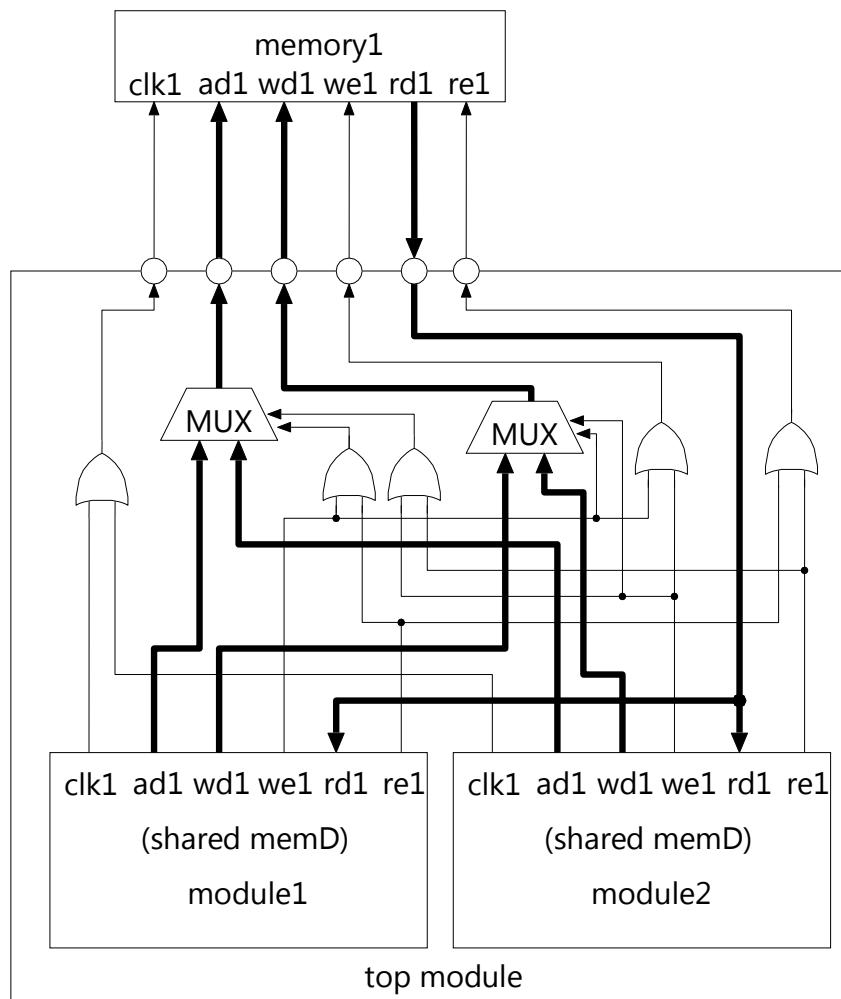


Figure 144: “Shared mem” sharing a memory port

Module1.c:

```
shared mem(0:8) memD[200] /* Cyber sig2mu = memory*/;
process module1( ) {
    .
    .
}
```

Module2.c:

```
shared mem(0:8) memD[200] /* Cyber sig2mu = memory*/;
process module2( ) {
    .
    .
}
```

mem.MLIB

```
#@LIB { module }
@MLIB {
    NAME memory
    KIND RW1
    BITWIDTH 8
    DELAY 1T
    WORD 200
    ...
}
#@END { module }
```

mem.MCNT

```
#@CNT { module }
@MCNT {
    NAME memory
    LIMIT 1
}
#@END { module }
```

L.6.3 Connection of “out” port of same name

In this section, reference is given for connection among “out” port.

- In n unit ($n \geq 2$) process, when same name “out” port exists, it is connected with OR logic and below warning is generated.

W_TG0176: External output port w (module module1 and module2) of same name is connected with OR.

In case of competitive timing, there is a possibility that output value becomes non assumed value.

- In n unit ($n \geq 2$) process, same name "out" port exists. When that valid signal exists, nmux is generated and it is selectively connected for "out" port.

Explanation is given using example regarding connection by OR logic.

Below, it is the top hierarchy output description when out port dout is declared in sub module module1 and module2.

Out port of sub module module1 and module2, INST_module1.dout and INST_module2.dout is connected with OR logic. dout port is assumed in top hierarchy and generated.

module1(Structure level description):

```
out ter (0:8) dout;
.
.
process module1( ) {
.
.
}
```

module2(Structure level description):

```
out ter (0:8) dout;
.
.
process module2() {
.
.
}
```

Output top hierarchy file:

```
defmod module1 {
.
.
    out unsigned ter(0:8) dout;
} INST_module1;
defmod module2 {
.
.
    out unsigned ter(0:8) dout;
} INST_module2;
out unsigned ter(0:8) dout;
process Top_mo_mo(){
    unsigned ter(0:8) TG_0;
    unsigned ter(0:8) TG_1;
    ST1_01 :
.
.
    TG_0 ::= INST_module1.dout;
    TG_1 ::= INST_module2.dout;
    dout ::= TG_1 | TG_0;
    goto ST1_01;
}
```

Explanation is given by using example regarding connection by NMUX logic.

Below, out port dout is declared in sub module module1 and module2. It is a top hierarchy output description when valid signal is generated after adding valid_sig_gen attribute.

In sub module "moudule1", attribute valid_sig_gen=dout_v1 is added to out port dout and valid signal dout_v1 is generated. In sub module "moudule2", attribute valid_sig_gen=dout_v2 is added to out port dout and pay attention to the generation of valid signal dout_v2.

In output top hierarchy, with regards to sub module "module1" and out port of module2, INST_module1.dout and INST_module2.dout, selective connection is done by NMUX and out port dout is output externally.

module1(Structure level description):

```
out ter (0:8)dout
out ter (0:1)dout_v1;
.
.
process module1( ){
.
.
}
```

module2(Structure level description):

```
out ter (0:8)dout
out ter (0:1)dout_v2
/*dout のvalid 信号*/;
.
.
process module2() {
.
.
}
```

Output top hierarchy file:

```

defmod module1 {
    .
    .
    .
    out unsigned ter(0:8) dout;
    out unsigned ter(0:1) dout_v1;
    /* In sub hierarchy, synthesize and add
     * attribute valid_sig_gen=dout_v1
     * in dout
     */
} INST_module1;
defmod module2 {
    .
    .
    .
    out unsigned ter(0:8) dout;
    out unsigned ter(0:1) dout_v2;
    /* In sub hierarchy, synthesize and add
     * attribute valid_sig_gen=dout_v2
     * in dout
     */
} INST_module2;
out unsigned ter(0:8) dout;
out unsigned ter(0:1) dout_v1;
out unsigned ter(0:1) dout_v2;
process Top_mo_mo() {
    unsigned ter(0:8) TG_0;
    unsigned ter(0:1) TG_1;
    unsigned ter(0:8) TG_2;
    unsigned ter(0:1) TG_3;
    .
    .
    .
    ST1_01 :
    .
    .
    .
    TG_0 ::= INST_module1.dout;
    TG_1 ::= INST_module1.dout_v1;
    TG_2 ::= INST_module2.dout;
    TG_3 ::= INST_module2.dout_v2;
    dout ::= nmux {
        when (TG_1) : TG_0;
        when (TG_3) : TG_2;
    };
    dout_v1 ::= TG_1;
    dout_v2 ::= TG_3;
    goto ST1_01;
}

```

L.6.4 Monitor tap Output Specification

If an output variable of given module and an input variable with the same name are there in different modules, then these are connected and I/O ports for external connection are not generated (**Figure 145**, left-hand side diagram). ‘topmodgen’ observes this type of variable from outside, therefore, it can generate output port (**Figure 145**, right-hand side diagram). These output ports generated for external connection are referred to as “monitor tap output port”. The variables to be linked to monitor tap port can be specified by listing the variable names in monitor tap definition file (.mon) and specifying this file using “-mon” option.

Example of monitor tap definition file.

```
A B
M[1]
```

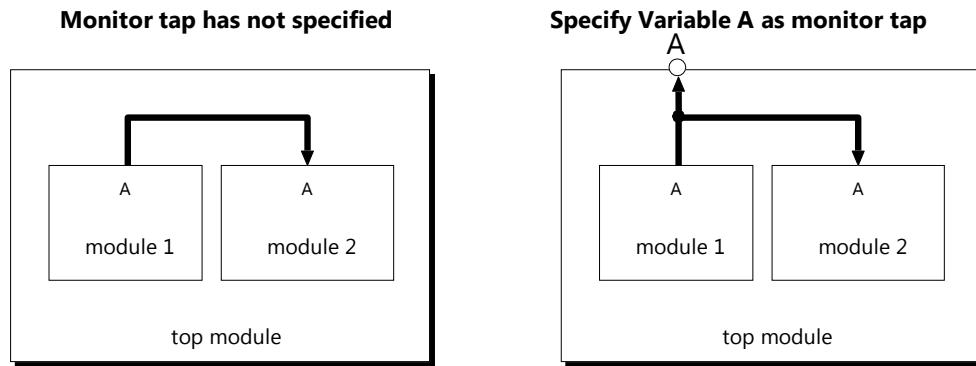


Figure 145: Monitor tap output specification

Monitor tap definition file contains signals or special formats delimited by linefeeds or spaces, as specified in the table below. However, non-existent variable names or unused variable names will be ignored even if specified in the definition file. Also, shared registers and shared register arrays, which are not read, will be ignored even if they are specified. If monitor ports for such shared registers and shared register arrays are to be generated, then these variables should be specified in the monitor tap definition file along with specification of the “-force_monitor” option.

- Output variable
- Shared register variable
- Shared register array

Special formats

*	All output variables and shared register variables (excluding shared register array)
array_name [1]	Specifies specific element of shared register array <i>array_name</i>
array_name	Specifies all elements of shared register array <i>array_name</i>
array_name [*]	Specifies all elements of shared register array <i>array_name</i>
* [*]	Specifies all elements of all shared register array

L.7. Limitations

L.7.1. Bidirectional ports are not supported

An input “.E.IFF” file, having bidirectional ports is not supported.

For the same reason, if option –Zrw1=1port is specified and memory with both read – write port is synthesized (with shared mem and outside mem), read-write data port becomes bidirectional "dt" (dual port) port which is not supported.

For the "topmodgen" tool, option –Zrw1=1port is not specified and read-write data ports must be divided into two ports: a read data port "rd" and a write data port "wd.

L.7.2. shared var/ ter variable in structural level mode are not supported

In the _E.IFF generated by declared description of shared var/ter variable, shared var/ ter array variable, it is not supported when shared var/ter port or shared var/ter array port declared by BDL/SystemC exists.

M. Topology checker among hierarchies (Connection system design rule check)

M.1. Functional overview

Internal format (_E.IFF) file group of design of multiple hierarchies or single hierarchy is input and topology checker is done. In the STARC RTL style guide (design rule), it is design rule checker command which carries out inspection of main connection system rules.

M.2. Usage

- Format

`pathcheck [option] filename[.IFF] ...`

One or more `filename[_E.IFF]` can be specified.

In case extension is not ".IFF", it is considered as omitted and added.

- Option

Option	Description
-p	File "pathcheck.pkprm" is read as Parameter file
-pfilename	<code>filename</code> is read as parameter file
-p: filename	<code>filename</code> is read as parameter file
-EE	Message display is changed to English
-EJ	Message display is changed to Japanese [Depend on environment variables LANG]
-h, -H	Command option help output
-v, -V	Version number output
-ignoreN.N.N.N	Ignore the rule of specified number

M.3. Functional description

Internal format (_E.IFF) file group of design of multiple hierarchies or single hierarchy is input and topology checker is done. In the design rule proposed in STARC RTL setting style guide, it is design rule checker command which carries out inspection of main connection system rules.

Violation point can be reported by line number information attached/file name of behavioral description before synthesis.

Rules for inspection are created in command beforehand and user cannot add rules by themselves.

Separate rules can be excluded from inspection by option.

- List of rules for inspection

1.2.1.3 Required Required combinatorial loop is restricted (2.4.1.4 Loop via latch is also included)

1.3.1.6 Required Synchronous asynchronous reset are not mixed in similar reset line

1.4.3.2 Recommended 1 Output pint of register (FF) is not input in clock pin of other register

1.4.3.4 Recommended 1 Clock signal is not provided besides clock input port (D Input etc.) of register

Important points

- Refer to appropriate book of STARC publication regarding details of STARC RTL setting style guide.
 - Basic RTL design style guide VerilogHDL edit, STARC editorial, Baifukan of LSI design
 - Basic RTL design style guide VHDL edit, STARC editorial, baifukan of LSI design
- Regarding representation system rule of STARC RTL guide, Compulsorily agreed option (-keep_starc_rule option) is provided in RTLgen (veriloggen).
- In inspection of 1.2.1.3, loop through asynchronous reset port of register is not checked.

N. Source Code Encryption Tool

In this chapter, functionality and usage of tools for distribution after encryption of source code is explained.

Source code encryption tool changes into separate package; therefore enquire until the support is provided for release.

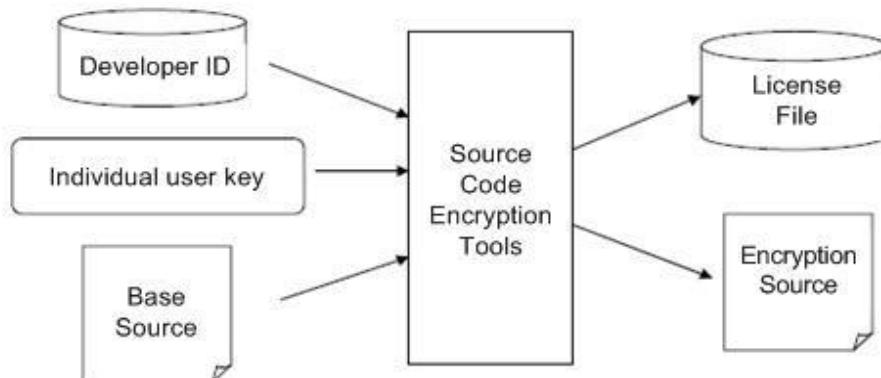
N.1. Source Code Encryption Tool Functionality

Source code encryption tool is a function which creates library, etc. and when it is used by a different user, a part of source is encrypted and source cannot be seen by that user.

The portion of source which is to be encrypted is encircled by /* Cyber encytion_start */ and /* Cyber encytion_end */ and when it is input in encrypted tool, encrypted source is output. At that time, besides the developer ID file and individual user key (any positive number) which is issued at the time of source code encryption tool release must be specified in encryption tool. Source is encrypted for individual user and the license file entered by the license key for individual user is output.

This license file is distributed to user along with encryption source. If license file is specified in environment variable for user, encrypted source can be entered. Multiple files can be specified in environment variable CYBER_IP_LICENSE_FILE (Separated with ; in windows and : in Linux). Further, multiple license keys can be specified in single license file (1 line 1 license key).

Source before encryption	Source after encryption
<pre>in ter(0:8) a, b; out ter(0:8) c; process foo() { /* Cyber encryption_start */ var(0:8) v; v = a + b; c = v; /* Cyber encryption_end */ }</pre>	<pre>in ter(0:8) a, b; out ter(0:8) c; process foo() { /* Cyber ciphertext 8348114950ef238e824513e395b07342 88c80927920a41fdb3e2476c83225bec 60f28739d73e651c45c0d3329e6639f4 aca9bb1a10b969c2394efa8b65347925 065d067e20f39c70f54a94a29f777a55 5ce985412b6f389c6a454a0e784fdc41 a43edaaa369395dfa3242ad962cb3695 5a74971c7b0c4160ddf91336de6900be f1441903e4203cd61c603d32c71f7ffb 29c39a696400c32bc42fac15dbb15cf 44888d77030a86cb53da03d83302db65 5428280971d2227a89be4cee2557edd3 */ }</pre>



When source code having /* Cyber_encytion_start */ and /* Cyber_encytion_end */(which enclose the target portion of the source code before encryption), input in the behaviour synthesis, it will generate an error. Error can be avoided by specifying ignore_encryption_attr option in the input command. (See **Section B.12**).

O. Attribute List

O.1 Attributes added in items

/* Cyber array_index = const */	11.4.8
/* Cyber bitw = # */	6.6.1
/* Cyber dec_port = # */	11.9.2
/* Cyber dump = {"format",arg} */	35.1.1
/* Cyber ex_group = # */	11.4.4
/* Cyber folding_ex_group = #[:#...] */	11.4.5,17.3.1
/* Cyber func = goto */	15.1.2
/* Cyber func = inline */	15.1.2
/* Cyber id = <i>id</i> */	21.6.1
/* Cyber implicit_exclusion = YES */	19.4.1
/* Cyber implicit_exclusion = NO */	19.4.1
/* Cyber inst_head = <i>name</i> */	15.1.2
/* Cyber mu_port = # */	20.9
/* Cyber ope2fu = <i>name</i> */	20.10.1
/* Cyber ope2fu_ex = <i>name</i> */	20.10.1
/* Cyber ope_synth = AUTO */	20.10.1
/* Cyber ope_synth = FU_IP */	20.10.1
/* Cyber ope_synth = FU_COMB */	20.10.1
/* Cyber ope_synth = FORCE_1CYCLE */	20.10.1
/* Cyber redundant_ope = asitis */	27.3.2
/* Cyber redundant_ope = optimize */	27.3.2
/* Cyber speculation = NO */	11.4.7,19.3.3
/* Cyber speculation = YES */	11.4.7,19.3.3

O.2 Attributes added in process function

/* Cyber commentword = comment */	6.8.1
/* Cyber default_bit_order = down */	6.3.2
/* Cyber default_bit_order = up */	6.3.2
/* Cyber inst_name = name */	1.5.6.2
/* Cyber interval_id = interval */	21.6.1
/* Cyber path_interval_id = interval */	21.6.1

O.3 Attributes added in sentence

/* Cyber arith_macro */	26.4.1
/* Cyber arith_macro = <i>name</i> */	26.4.1
/* Cyber array_index = const */	11.4.8
/* Cyber chain_group = #[#...] */	18.1.1,19.10.1
/* Cyber chain_group = all */	18.1.1,19.10.1
/* Cyber dec_port = # */	11.10.2
/* Cyber exit = reset_state */	10.4.4
/* Cyber exit = st_1_state */	10.4.4
/* Cyber folding = # */	17.1.2
/* Cyber latency_loop = <i>range</i> */	21.4.2
/* Cyber latency_loop_ext = <i>range</i> */	21.4.2
/* Cyber latency_set = <i>num</i> */	21.3.1
/* Cyber latency_set = <i>num:num</i> */	21.3.1
/* Cyber loop_merge_ex = <i>name</i> */	16.4.2
/* Cyber loop_merge_ex_top = <i>name</i> */	16.4.2
/* Cyber loop_merge_ex_bottom = <i>name</i> */	16.4.2
/* Cyber loop_merge_ex_force = <i>name</i> */	16.4.2
/* Cyber loop_merge_ex_top_force = <i>name</i> */	16.4.2
/* Cyber loop_merge_ex_bottom_force = <i>name</i> */	16.4.2
/* Cyber nmux_multi_asgn = check */	36.6
/* Cyber nmux_multi_asgn = ignore */	36.6
/* Cyber thr_unroll = NO */	15.3.2

/* Cyber thr_unroll = YES */	16.3.2
/* Cyber unnesting = AUTO */	17.2.10
/* Cyber unnesting = YES */	17.2.10
/* Cyber unnesting = NO */	17.2.10
/* Cyber unroll_folding = # */	17.1.2
/* Cyber unroll_times = #[:#] */	16.1.2

O.4 Attributes added in signal declaration

/* Cyber address_calc = CONCAT */	11.4.3
/* Cyber address_calc = MULT */	11.4.3
/* Cyber array = EXPAND */	11.1.2
/* Cyber array = LOGIC */	11.1.2
/* Cyber array = RAM */	11.1.2
/* Cyber array = REG */	11.1.2
/* Cyber array = ROM */	11.1.2
/* Cyber array_data_hazard = AVOID */	17.2.2,22.6.2
/* Cyber array_data_hazard = CHECK */	17.2.2,22.6.2
/* Cyber array_data_hazard = NO */	17.2.2,22.6.2
/* Cyber array_hazard = AVOID */	17.2.6
/* Cyber array_hazard = CHECK */	17.2.6
/* Cyber array_hazard = IGNORE */	17.2.6
/* Cyber array_index = const */	11.4.8
/* Cyber array_index_suffix = word */	11.4.10
/* Cyber array_merge = name */	11.4.6
/* Cyber array_order = AUTO */	17.2.6
/* Cyber array_order = IGNORE */	17.2.6
/* Cyber array_refer = asitis */	27.3.2
/* Cyber array_refer = optimize */	27.3.2
/* Cyber bitw_opt = NO */	27.5.2
/* Cyber bitw_opt = YES */	27.5.2
/* Cyber chain_group = #[:#...] */	18.1.1,19.10.1
/* Cyber chain_group = all */	18.1.1,19.10.1

/* Cyber clock_edge = neg */	8.1.2
/* Cyber clock_edge = pos */	8.1.2
/* Cyber clock_pol = neg */	8.1.2
/* Cyber clock_pol = pos */	8.1.2
/* Cyber clock_sig = name */	8.1.2
/* Cyber con_name = word */	L5.6.2
/* Cyber decoder_div = #:#[#:...]* /	11.10.4
/* Cyber default_value = value */	7.9
/* Cyber gated_clock = YES */	30.1.2
/* Cyber gated_clock = NO */	30.1.2
/* Cyber group_interval_array = val */	17.2.2, 21.5.1
/* Cyber hazard_avoid_sche = NO */	17.2.7
/* Cyber hazard_avoid_sche = YES */	16.2.7
/* Cyber init_reg = asitis */	27.6
/* Cyber init_reg = optimize */	27.6
/* Cyber interval_array = array_interval */	21.5.1
/* Cyber mem_be = create */	11.5.1.4
/* Cyber mem_be = ignore */	11.5.1.4
/* Cyber mem_clocking = always */	11.5.1.5
/* Cyber mem_clocking = enable */	11.5.1.5
/* Cyber mem_cs = create */	11.5.1.3
/* Cyber mem_cs = ignore */	11.5.1.3
/* Cyber mem_init = create */	10.2.3
/* Cyber mem_init = ignore */	10.2.3
/* Cyber mem_re = create */	11.5.1.2
/* Cyber mem_re = ignore */	11.5.1.2
/* Cyber mem_reg = in */	11.7
/* Cyber mem_reg = in:out */	11.7
/* Cyber mem_reg = none */	11.7
/* Cyber mem_reg = out */	11.7
/* Cyber multi_cycle_input = true */	26.3.3.2
/* Cyber no_share */	20.8

/* Cyber pipeline_forwarding = NO */	17.2.2.21.1.2
/* Cyber pipeline_interval_check = NO */	17.2.2.21.1.2
/* Cyber pipemem = x# */	11.6.1.1
/* Cyber pipemem_io */	11.6.1.4
/* Cyber port_hazard = AVOID */	17.2.5
/* Cyber port_hazard = CHECK */	17.2.5
/* Cyber port_hazard = IGNORE */	17.2.5
/* Cyber port_reg_stage = # */	7.15
/* Cyber port_synchronizer = # */	7.14
/* Cyber port_type = REG */	7.13
/* Cyber port_type = TER */	7.13
/* Cyber port_type = VAR */	7.13
/* Cyber read_timing = ALAP */	19.2.3
/* Cyber read_timing = ASAP */	19.2.3
/* Cyber reg_bind */	20.8
/* Cyber reg_hazard = AVOID */	17.2.4
/* Cyber reg_hazard = CHECK */	17.2.4
/* Cyber reg_hazard = IGNORE */	17.2.4
/* Cyber reg_reset_cone = YES */	9.1.2
/* Cyber reg_reset_cone_valid_sig = YES */	9.1.2
/* Cyber reset_active = high */	9.1.2
/* Cyber reset_active = low */	9.1.2
/* Cyber reset_macro = individual */	9.1.2
/* Cyber reset_macro = inverter */	9.1.2
/* Cyber reset_mode = async */	9.1.2
/* Cyber reset_mode = sync */	9.1.2
/* Cyber reset_sig = name */	9.1.2
/* Cyber reset_sig = void */	9.1.2
/* Cyber reset_synchronizer */	9.1.2
/* Cyber reset_synchronizer = # */	9.1.2
/* Cyber rw_delay = # */	11.10.3
/* Cyber rw_port = RW# */	11.10.1

/* Cyber rw_port = R#.W# */	10.9.1
/* Cyber rw_port = R# */	10.9.1
/* Cyber rw_port = W# */	10.9.1
/* Cyber rw_timing = r?w? */	10.6 , 10.6.2.1
/* Cyber self_chain = YES */	18.10.1
/* Cyber self_chain = NO */	18.10.1
/* Cyber share_name = <i>name</i> */	19.8
/* Cyber sig2dec_port = r#w# */	10.9.2
/* Cyber sig2mu = <i>name</i> */	19.9
/* Cyber sig2mu_port = <i>name</i> */	19.9
/* Cyber sig_pol = neg */	6.4
/* Cyber speculation = YES */	10.4.7 , 18.3.3
/* Cyber speculation = NO */	10.4.7 , 18.3.3
/* Cyber stall_control = high */	22.1.2
/* Cyber stall_control = low */	22.1.2
/* Cyber synchronizer */	7.14
/* Cyber synchronizer = #1 : #2 */	7.14
/* Cyber univ_mem_if = create */	10.10.2
/* Cyber univ_mem_if = ignore */	10.10.2
/* Cyber valid_sig_bind = <i>name</i> */	6.11.1
/* Cyber valid_sig_gen = <i>name</i> */	6.10
/* Cyber valid_sig_neg_bind = <i>name</i> */	6.11.1
/* Cyber valid_sig_neg_gen = <i>name</i> */	6.10
/* Cyber value_range =#1 - #2 */	26.5.2
/* Cyber write_timing = ASAP */	18.2.3
/* Cyber write_timing = ALAP */	18.2.3

O.5 Attributes added in Function definition, Function declaration

/*Cyber func=combinational_operator*/	14.5.3
/* Cyber func = goto */	14.1.2

/* Cyber func = inline */	14.1.2
/* Cyber func = operator */	13.1.2, 14.5.3
/* Cyber func = operator[:c][:r][:s][:e][:a][:h] */	14.5.3
/* Cyber func = pipeline_operator */	14.5.3
/* Cyber func = sequential_operator */	14.5.3
/* Cyber func_clock_sig = name */	14.5.3
/* Cyber func_inst_num = # */	14.5.5
/* Cyber func_out_sig = name */	14.5.3
/* Cyber func_outside = yes */	12.5
/* Cyber func_reset_sig = name */	14.5.3
/* Cyber func_static = asitis */	14.5.8
/* Cyber func_static = asglobal */	14.5.8

O.6 Attributes added in Block

/* Cyber operator_block */	14.5.3
/* Cyber reset_block */	9.3.2
/* Cyber scheduling_block */	20.2.3
/* Cyber scheduling_block = transparent */	20.2.3
/* Cyber scheduling_block = non-transparent */	20.2.3
/* Cyber thr_block */	26.4.2

O.7 Attributes for SystemC

/* Cyber const_variable */	
/* Cyber immediate_access */	
/* Cyber immediate_access = R */	
/* Cyber immediate_access = W */	
/* Cyber immediate_access = RW */	
/* Cyber module_boundary = keep */	
/* Cyber module_name = name */	
/* Cyber signal_type = pulse[:0][:1] */	
/* Cyber signal_type = store[:0][:1] */	

Index

Sign / Numeric Value

\$	31
.cyber	48
1port	179
2port	179

A

-a	153
-Aa	415
Active High	107
Active Low	107
-Ad	415
add_dflt	485
address_calc	164
-ae	159
-aeu	159
-A-f	415, 569
-AF	415
-Af	415, 569
-al	229
ALAP	378
all	300, 417
allstates statement	580
restrictions	581
-am	153
-aN	153
-ap#:o	178
-ap?:o	75
-apA	153
-apA:o	178
-apI	153
-apI:o	178

-ar	153
-area_report_file_name	747
area priority macro option.....	580
arith_macro	518,531
arithmetic macro functional unit	531
array	107
subscript specification-Array reference/assignment..	138
speculative execution reference.....	138
handling of initial value.....	138
allocated destination memory type specification.....	368
array	154–158
array_data_hazard.....	455
array_index	173
array_merge.....	173
array_refer	531
array information	630
count	630
ASAP	352
asis	531, 557
asynchronous SRAM.....	160, 813
asynchronous reset	107
-assert_off	764
-assert_on	764
-A-t	415, 569
-At	415, 569
Attribute.....	37
value.....	37
relation with option.....	43
priority order with option.....	43
location description.....	37
check.....	603
constant substitution	30

display	603
macro substitution	29
priority order	49
name.....	37
-au	234
Automatic scheduling.....	25, 352
Register sharing	246
ave	234

B

Basic library delay consideration	
scheduling.....	367
Basic library file.....	701, 712
BDL	26, 53, 635
bdlcmp.....	62, 64
bdldraw.....	64
bdlpars	26, 53, 607, 765
bdltran	26
bdltran.prm	47
BIST.....	566
binary encode.....	570
bit width check of allocated functional unit	
.....	592
bitw.....	48
Attribute display.....	621
bitw_opt	544
BLIB	690, 703
BLIBgen.....	23, 792
bmcppgen.....	27
-bus_naming_style	737
byte enable port.....	138

C

-c	86
----------	--------------------

upper case letter/lower case letter . . . 22
careful [607](#)
cast display..... [633](#)
CDFG. [698](#)
 key operation. [698](#)
 mouse operation [698](#)
circuit quality report. [647](#)
chain_group [378, 397](#)
check [607](#)
check_E [27, 796](#)
check_overflow. [27, 796](#)
-check_rtl [747](#)
check nmux statement of continous
assignment [607](#)
chip select port. [178](#)
character code limitations. [764](#)
clock. [101, 102](#)
clock [101](#)
 cycle. [101](#)
 unit. [101](#)
 port name. [102](#)
 rising/falling edge [101, 102](#)
clock cycle boundary specifier [24](#)
clock_edge. [101, 102](#)
-clock_period [747–749](#)
clock_pol [178](#)
clock_sig [178](#)
-clockbuf_branch [747](#)
-clockbuf_leaf. [747](#)
-clockbuf_root.. [747](#)
-clockbuf_lib [747](#)
cmscgen [27](#)

cmspeccgen	27
cmveriloggen	27
combinatorial circuit.....	153
specification method	159
combinatorial loop	563
detetction functionality..	569, 796
comment	62
common Sub-Expressions Elimination	531
controller data flow graph.....	701
-comp	747
CONCAT.....	164
Condition information.....	633
Conditional Branch Scheduling.....	397
Connection system rule check	876
connection between ports with different name.....	873
continuous assignment	759
const	173
CPI.....	614
create	71, 132, 153, 178, 179
CSV.....	633
Synthesis Info File of CSV format....	633
-cu	101
CWB-gui	26
CYBER_ENV.....	48
cybus	26
CybusM.....	26
D	
dcscr	747
delay	698
violation.....	614
cycle specification.....	698, 708

absolute delay specification[698, 708](#)
unit.....[698, 708](#)
pipeline specification ... [698, 708](#)
delay information [645](#)
Synthesis script for DC
Notes during Editing [747](#)
decoder_div [234](#)
default_bit_order [48, 764](#)
default_value..... [75](#)
default option specification file..... [48](#)
default output value [75](#)
descending order [48](#)
allocation destination registers specification
for variables..... [415](#)
deletion of unreferenced registers... [557](#)
detailed delay information..... [660](#)
DII [299](#)
DIM [132, 182](#)
dty [680](#)
dump..... [580](#)

E

-EFO..... [747](#)
EFCNT [633](#)
-effort [747, 748](#)
err..... [633](#)
error..... [586](#)
 file..... [633](#)
ex_group..... [164](#)
exit [132](#)
EXPAND..... [153](#)
expand_dim [144, 145](#)

External functional unit. [248, 249](#)
External memory [32, 153, 154, 229](#)

F

false path [747](#)
 restriction. [748](#)
false loop. [557](#)
 avoidance scheduling. [569](#)
 detection functionality [557](#)
fast. [531](#)
-fc [272](#)
-Fc? [107](#)
FCNT. [698](#)
-fg [272](#)
-fi [272](#)
file output destination directory. [698](#)
first state [132](#)
FLIB. [698](#)
FLIBgen [26, 796](#)
floating port. [75](#)
flattening of multistage multiplexer. [557](#)
folding. [300](#)
folding_ex_group. [168, 345, 356](#)
for loop with fixed repeat count. [299](#)
for loop [299](#)
 Optimization of operation in
 unrolled loop [299](#)
 unrolling [298](#)
 specification [299](#)
 partial unrolling [298](#)
 loop folding and
 loop unrolling. [345](#)

exclusive subscript during array reference, assignment in	345
Generation of memory description for FPGA logical synthesis.	234
-fpga_device	747, 748
-fpga_family	747, 748
-fpga_package	747, 748
-fpga_speed_grade	747, 748
FPI	756
-Fr?	94
-fs	266
FSM.	560
division of state register	566
decoder delay	650
transfer destination of invalid state	586
func	672
function	764
goto conversion	272
inline expansion.	273
functional unit conversion	262
return value type.	289
parameter type	266
return value.	268
initialization of local variable.	266
functional unit	
shared.	282
type number	703
operator type.	704
maximum delay.	705
usage information	624

number of constraints	513
chain effect.	706
delay	706
output delay of pipeline	
functional unit.	722
input delay of pipeline functional	
unit.	730
Bit width.	720, 734
Area	696, 707
name.	690
functional unit information.	654
functional unit count file	702, 725
template generation.	852
functional unit constraint file.	701, 703
functional unit library file	717, 727
template generation	852
-fx	246
G	
-gated_clock_style	752
gated clock	577
goto	272
Graphviz	689
group_interval_array.	456
GUI	27
H	
hierarchical_design	452
higher hierarchy.	839
high impedance.	31, 32, 168
hint file.	624
HiZ.	76
hold.	452
HOME	49

I

id [442, 445](#)
id used for time constraint specification [417](#)
iff2bdl [28, 834](#)
if statement's comprehensive
parallelization [329](#)
ignore [606](#)
ignore [29, 72, 131, 149, 180, 606](#)
implicit_exclusion [387](#)
important warning file. [633](#)
in [220](#)
in:out. [220](#)
individual. [124](#)
individual format. [100](#)
INF [639](#)
information file. [639](#)
interrupt. [581](#)
initial value provided
 Variable declaration. [849](#)
 register. [99](#)
init_reg [553](#)
inline [271](#)
input output
 information. [647](#)
 signal name [713](#)
 port. [70](#)
 type [711](#)
 delay [711](#)
 alignment sequence. [69](#)
 bit width. [711](#)
 name. [711, 712](#)
 timing specification [379](#)

array	166
input output/memory information.	654
input file specification	50
-insert_pad	739
inside_mem	165
internal memory	165
interval_id	447
interphase.	69
inverter format	100
inverter	125
io.	150
-ise_scr	739

L

Language level conversion functionality...	
.....	516
LATCH	519
latch	519
latency_loop	437
latency_loop_ext	437
latency_set.	434
-lb	703
-lfc.....	703
+lfc.	703
-lfl.	703
+lfl.	703
-lf	703
+lf	703
-lic_wait	42
license retry.	42
line number.	626
-lm	155, 687
-lmc	140, 155, 687

+lmc	140 , 155 , 687
-lml	140 , 155 , 687
+lml	140 , 155 , 687
+lm	140 , 155 , 687
LMT	704
LOG	613
log file.....	613
LOGIC.....	142
LOGIC	554
logical operation delay based scheduling.	
.....	369
logical level conversion functionality...	
.....	540
loop avoidance in resource allocation.. .	
.....	570
loop unrolling	300
loop latency.....	437
information.....	645
loop latency constraint specification.	
.....	408
loop folding.....	280
loop merging.....	275
loop_merge_ex.....	309
loop_merge_ex_bottom.....	309
loop_merge_ex_top.....	309
-lp	75 , 687
-lpA	75 , 687
-lpr	75 , 687
LSscrGen.....	28 , 522
-lt	452
M	
-MA	590

macro option	590
Manual scheduling.....	32
multiple assignment check ...	379
register sharing	409
-max_area.....	748
-max_fanout	748
-max_transition	748
-mc	98
-mcD	98
-mcN	98
MCNT.....	69, 175, 687
mem_be	183, 711
mem_clocking.....	711
mem_cs.....	137, 168, 711
mem_init.....	131
mem_re.....	29, 178, 510
mem_reg.....	221
mem_synth	245
memory.....	69, 140
access timing specification....	140
type	711
type number	711
specification method	212
usage information.....	625
constraint count.....	702
port.....	166
delay.....	711
bit width.....	711
port type.....	166
word count.....	711
port	711
name	711

allocation.....	393
memory count file	69, 184, 689, 704
template generation.....	853
memory constraint file.....	140, 703, 690
memory library file.....	69, 140, 703, 726
template generation.....	853
merging multiple arrays	171
merging similar condition of multiplexer.	
.....	553
message	
type	596
number	596
mkmfsim.....	28
MLIB	69, 140, 702
MLMT.....	140, 687, 709
-module_prefix	49, 51
module information	643
module definition file	623
monitor tap.....	874
-MPD.....	590
-MPL.....	590
-mr?w?	170
-MS	562
mu_port	408
MULT.....	132
multi cycle	
operation.....	519
path.....	678
memory	218, 519
multi_cycle_input.....	519
multidimensional array	
1 dimension.....	172

multiple memory partition
allocation.....[172](#)
unrolling of partial dimension ..[172](#)
multi assign.....[603, 604, 601](#)
multiple assignment..... [603, 597, 602](#)
multiple functionality of process.....[857](#)

N

neg[69, 86](#)
negFF[566](#)
negative logic[69, 94](#)
-net_report_file_name[739](#)
nmux_multi_asgn.....[592](#)
NO[163, 329, 329](#)
no[213](#)
no_dflt[566](#)
none.....[204](#)
no_share[375, 395](#)

O

-o[737](#)
-O-C.....[622](#)
-O-e.....[624](#)
-O-o.....[623](#)
-o-R.....[390](#)
-O-T.....[622](#)
-O-t.....[623](#)
-O-w[623, 827](#)
-O-y.....[623](#)
-O2[623, 827](#)
-OA[615](#)
-OB[597, 612, 615, 500](#)
-OC[622](#)
-Oe[624](#)

-OF	615
-o-H.....	274
-oH	274
-OI	615
-o-I.....	324
-oI	324
-o-K.....	520
-oK	520
-Ol	623, 827
-o-L.....	519
-oL	519
-ol	687
-OM	615
-Oo	623
-OP	615
one hot encoded.....	566
ope_synth	387, 389, 393
ope2fu.....	408
ope2fu_ex.....	408
operation	
reduction in number of stages of tree.	
.....	524
output bit width.....	47, 49
allocated destination operation type	
specification.....	393
operator	246
-opt_file_name.....	739
optimize	520, 543
optimization of redundant logical functions.	
.....	540
option.....	41
priority.....	41
option related to circuit quality report.	

.....	670
-OR	615
-oR	390
-OS	615
-OT	620
-Ot	623
-OU	615
-ou	687
out	204
-out_file_name..	736
output variable reference	69
outside.....	140, 229
outside_mem	150
Overflow detection functionality	792
-OW	615
-Ow	623
P	
-p	40, 736, 754
parameter file.....	40
partial loop unrolling.....	270
path_interval_id	425
pathdraw	23
pipeline_forwarding	435
pipeline_interval_check	435
pipemem	180
pipemem_io.....	185
PLMT	30, 689, 695
pointer	234
port constraint file.....	30, 689, 695
port_reg_stage.....	86
port relation file.....	30, 689, 695
port_synchronizer	84

pos	86
positive logic	94
pre	213
prefix.....	49
PREL.....	30, 689, 696
prm	42
-proc	483
process	24
process function	24
end time.....	117
variable handling.....	118
parameter.....	69

Q

-QDm.....	670
-QDn.....	670
-QDt.....	670
-QFn.....	670
-QMa.....	670
QOR.....	640
QOR.HTML.....	640
-QRi.....	670
-QRn.....	670
-QRo.....	670
-QTn.....	670

R

RAM.....	140
read_timing	329
read only port	166
read write port.....	165
data port	166
read enable port	164
recursive function	213

redundant_ope	520
REG	140
reg	513
-reg_feedback	556
reg_file	150
reg_reset_cone	103
reg_reset_cone_valid_sig	103
reg_bind	369
register/multiplexer/decoder information.	
.....	649
register based scheduling.....	369
register transfer information.....	627
register feedback loop.....	544
register array usage information.....	627
register fan in/fan out information.....	651
register variable conversion into constant	
.....	543
register array with decoder.....	138, 164
specification method	141
decoder count	210
decoder number specification.	211
decoder separation	212
decoder delay.....	212
register generation to input output port..	
.....	90
reset.....	114
reset.....	96, 58
reset.....	94
port name.....	96
logic.....	96
reset block.....	116
reset_block.....	116

reset_active	102
reset_macro	101
reset_mode	94 , 102
reset_sig	101
reset_state	114
restriction	49 , 55 , 114 , 212 , 217 , 332 , 395 , 555 , 566 , 575 , 595
resource allocation.....	385
rf	170
rf0.....	170
rl	170
rl0.....	170
ROM.....	147
rtlpars	22
rw_delay	213
rw_timing	170 , 180

S

-s	24
-S-M.....	371
-S0	329
-S1	329
-S2	329
-S3	329
same name out port.....	836
scheduling_block	403
scheduling_block	402
scpars.....	22
setup	106 , 483
-SF	554
shared	144 , 229
shared_mem.....	150
shared_reg	150

shared subexpression elimination	520
shared ter	229
array	229
shared var	229
shared ter.	229
shared var	229
array.	229
shared memory	144, 229
shared register.	69, 229
array	208
share_name	393, 396
sig_pol	69
sig2mu	396
sig2mu_port.	396
sign.	47
signallist	23, 792
signed	47, 388
-sim_file_name.	739
Simplification of logic condition of multiplexer	
.	540
slow.	525
-SM	371
-sN	24
-SP	365
software reset.	560
speculation	160, 362
speculative execution	362
-S-R.	369
-SR	369
-SS	262
st_1	114
st_1_state	117

ST1out	69
stall.....	452
stall_control.....	452
state transition description	613
state transition figure	687
file	680
statement count	215
structure description	613
SUMM.....	615
Summary file	615
-syn_delay_unit	737 – 738
-syn_module_setup	737 , 738
-syn_tool_setup..	737 , 738
-synlib_file_name	739
synchronization circuit between out-of-phase clocks.....	84
synchronous SRAM.....	832
enable port, clock port	168
synchronous reset	96
synthesis execution speed priority level macro option	570
SystemC.....	22
T	
tbgen	23
ter variable division.....	541
terminal transfer information	617
text version circuit quality report	656
thr_block	516
thr_unroll	273
time constraint	393
time constraint violation.....	433
time constraint specification file	430

time constraint specification for port.	408
time out	55
-timing	737 – 739
-timing_report_file_name	737 , 739
tips	620
TLMT	433
-topinst	739
topmodgen	23 , 827
top hierarchy	827
true	516
true loop	540
+tx_add	737 , 738
-tx_out	737 , 738
-tx_pinform	737

U

-u0	272
-uA	272
-UB	272
-UL	272
-ungroup_before_rpt.....	736
univ_mem_if.	212 , 516
unroll_folding	320
unroll_times.....	271
unsigned.....	47 , 244
-UN	387
unused port.....	69 , 168
universal memory interface specification	
	213

v

valid signal	69
variable unrolling	139 , 164
specification method	142

value_range [525](#)
veriloggen [1](#), [106](#), [764](#)
-vhdl [735](#), [797](#), [802](#)
vhdlgen [22](#), [106](#), [757](#)
void [94](#)

W

-w [752](#)
wait_valid signal [83](#)
warning [583](#)
-Wall [751](#)
-Wandor_side_effect [605](#)
warn [620](#)
-Was_C_lang [598](#)
watch statement [329](#)
-Wattr [596](#)
-WBDL3.0=error [602](#)
-WBDL3.0=no_warning [602](#)
-WBDL3.0=warning [602](#)
-Wcritical_path [601](#)
-wd [686](#)
-Wexit [551](#), [552](#), [586](#)
wf [170](#)
wf0 [170](#)
-Wfalse_loop [552](#)
-Wfalse_loop_max [552](#)
wl [170](#)
wl0 [170](#)
wiring characteristics/routability information
..... [649](#)
count [654](#)
-Wmulti_asgn [576](#)
-Wope_asgn_bitw [577](#)

write_timing	325
write only port.....	166
wrapper generation bypassing internal memory	568
-Wxor_bitw	586

X

XOR operation bit width check.....	590
------------------------------------	---------------------

Y

YES	160, 331, 333
-----------	-------------------------------

Z

-Zaddress_calc_concat	150
-Zaddress_calc_mult	150
-Zarray_data_hazard.....	435
-Zarray2multi_mem	163
-Zarray2multi_mem=NO.....	163
-Zassign=1	393
-Zassign=2	567
-Zasync	90
-Zbase_unit	87
-Zbdlcmp	66
-Zbit_order_force=down	55
-Zbit_order_force=NO.....	55
-Zbit_order_force=up.....	56
-Zbitw_opt=ALL.....	545
-Zbitw_opt=INTERNAL.....	545
-Zbitw_opt=LOGIC_TYPE	545
-Zbitw_opt=NO	545
-Zcfgdraw	687
-Zchk_bit_description.....	16, 749
-Zcse_limit	516
-Zcsv_lst.....	625
-Zdecoder_div	235

-Zdecoder_div_over	213
-Zdecoder_div_unit	213
-Zdefault_bit_order.....	764
-Zdefault_bit_order=down	55
-Zdefault_bit_order=up	55
-Zdotty_out	615
-Zfalse_path_script.....	654
-Zfcnt_arith_macro_out	490
-Zfcnt_out	490
-Zflatten_nmux=ALL	545
-Zflatten_nmux=DP	545
-Zflatten_nmux=FSM	545
-Zflatten_nmux=NO	545
-Zplib_fcnt_arith_macro_out	490
-Zplib_fcnt_out	490
-Zplib_out	490
-Zplib_out_limit	490, 516
-Zplib_out_multi=NO	490, 516
-Zplib_out_multi=YES.....	490, 516
-Zplib_out_sign=EACH.....	490, 516
-Zplib_out_sign=MIX	490, 516
-Zplib_out_sign=NONE.....	490, 516
-Zfsm_dec.....	519
-Zfsm_st	566
-Zfu_total_limit	55
-Zgated_clock	573
-Zgen_stateno_out	69
-Zin_port_reg_stage=#	100
-Zin_port_synchronizer=#	100
-Zinit	131
-Zinit_local_sig	275
-Zinit_reg=asitis	552

-Zinit_reg=optimize..... [552](#)
-Zinternal_valid_sig_gen [91](#)
-Zmcnt_out..... [497](#)
-Zmcnt_out [497](#)
-Zmem_be..... [28](#), [166](#), [497](#), [516](#)
-Zmem_be=NO [166](#)
-Zmem_bitw_opt..... [546](#)
-Zmem_bitw_opt=NO [546](#)
-Zmem_clocking [185](#)
-Zmem_clocking=always [492](#), [510](#)
-Zmem_clocking=enable [492](#), [510](#)
-Zmem_cs..... [28](#), [167](#), [492](#), [510](#)
-Zmem_kind..... [492](#), [510](#)
-Zmem_re..... [28](#), [166](#), [492](#), [510](#)
-Zmember_name_length [260](#)
-Zmix_bitw_reg_share [404](#)
-Zmix_bitw_reg_share=0_EXPAND_ONLY
..... [413](#)
-Zmix_bitw_reg_share=NO [413](#)
-Zmix_sign_share=ALL [404](#)
-Zmix_sign_share=NO..... [404](#)
-Zmix_sign_share=OPE [404](#)
-Zmix_sign_share=REG [404](#)
-Zmix_signed [42](#), [754](#)
-Zmlib_mcmt_out..... [492](#), [509](#)
-Zmulti_cycle_mem [509](#)
-Zmulti_cycle_ope [509](#)
-Zname_sensitive..... [59](#), [63](#)
-Zno_aryinit [166](#)
-Zope_stall [293](#), [295](#)
-Zopt_loop_lvl..... [537](#)
-Zopt_loop_merge [275](#)

-Zopt_loop_merge=NO.....	297
-Zopt_redundant_logic	553
-Zopt_redundant_logic=NO.....	553
-Zopt_redundant_ope.....	540
-Zopt_reg_feedback	555
-Zopt_reg_feedback=NO	560
-Zopt_ter_rename	541
-Zopt_ter_rename=NO.....	552
-Zout_port_reg_stage=#	71
-Zpointer_analyze_effort.....	257
-Zport_asitis	77
-Zport_optimize.....	77
-Zport_optimize_auto.....	77
-Zport_valid_sig_gen.....	80
-Zprefix	49
-Zprefix_outside_fu.....	255
-Zreg_asitis	542
-Zreg_head_len.....	393
-Zreg_min_reset.....	106
-Zreg_name_func	393
-Zreg_name_len.....	393
-Zreg_optimize.....	552
-Zreg_orig_name	393
-Zreg_reset_cone	127
-Zreg_reset_cone_valid_sig	127
-Zreset_macro	127
-Zreset_states=AUTO	102
-Zreset_states=NO	102
-Zreset_states=YES	102
-Zrw1	28, 166
-Zrw1=1port	180, 516
-Zrw1=2port	180, 516

-Zsame_cycles	392
-Zsche_logic_delay=NO	398
-Zsche_logic_delay=YES	398
-Zshare_nmux_cond	350
-Zshared_mem_re-gen	135, 205
-Zshared_reg_reset	228, 470
-Zsig_name1state	60
-Zsimplify_nmux_cond	550
-Zsimplify_nmux_cond=NO	550
-Zstreg	565
-Zstruct_name_length	260
-Zsync	99
-Ztimeout	58
-Ztop	278, 276
-Zundisp_fname..	626
-Zunsigned	44, 761
-Zwait_valid	82
-ZZfu_noshare_states.....	403, 759
-ZZlogicbist	566
-ZZpipeline	446