Memory compiler tutorial – TSMC 40nm technology

2021/07/23

Ref: Shih Wei Hsieh

Outline

- Memory Compiler Introduction
- Memory Compiler Interface
  - Parameters
  - Files Generation
  - Using the Files
  - Memory Ports
- Lab #1 – Dual-Port SRAM
- Lab #2 – Asynchronous FIFO

Memory Compiler Introduction

- Memory compiler is the SRAM/ Register File/ ROM generator for ASIC design. The generated files include:
  - User guide
  - Verilog model (for simulation)
  - Timing information (for synthesis and APR)
  - Physical layout (for APR)
- We will focus on the Memory compiler for TSMC 40nm technology
  - Settings should be similar for other technology nodes
- Location:
  - 90nm: /cad/cell_library/CBDK_TSMC90G_Arm/CIC/Memory/
  - 40nm: /cad/cell_library/CBDK_TSMC40_Arm_f2.0/CIC/Memory/

Memory Compiler Introduction

- Folders:

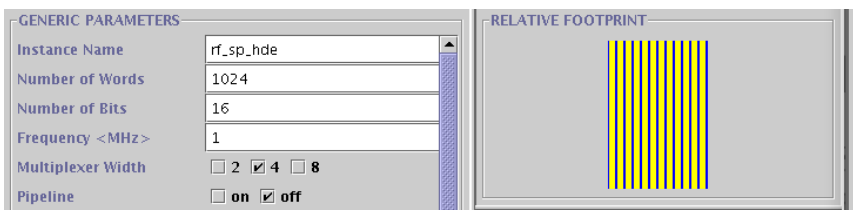
Directory name	Generate type
rf_2p_hse_rvt_hvt_rvt/	High-speed two-port register file
rf_sp_hde_rvt_hvt_rvt/	High-density single-port register file
rf_sp_hsd_rvt_rvt_hvt/	High-speed single-port register file
rom_via_rvt_hvt_rvt/	ROM
sram_dp_hde_rvt_hvt_rvt	High-density dual-port SRAM
sram_sp_hde_rvt_hvt_rvt/	High-density single-port SRAM
sram_sp_hsc_rvt_hvt_rvt/	High-speed single-port SRAM

- Single-port: 1 read or 1 write per clock
- Two-port: 1 read or 1 write or 1 read+ 1 write per clock
- Dual-port: 2 reads or 2 writes or 1 read+ 1 write per clock

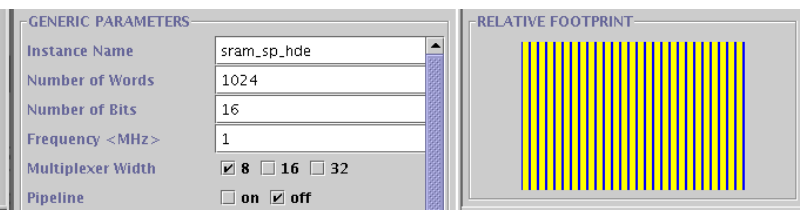
Memory Compiler Introduction

- Difference between SRAM and RF?
  - RF is usually used for smaller memory (lower address line bits)
- E.g. Single-port SRAM vs single-port RF
  - SRAM number of words range: [256, 16384]
  - RF number of words range: [16, 2048]
- For the number of words in the overlapped range (e.g. 1024 words)
  - Layout comparison:

RF



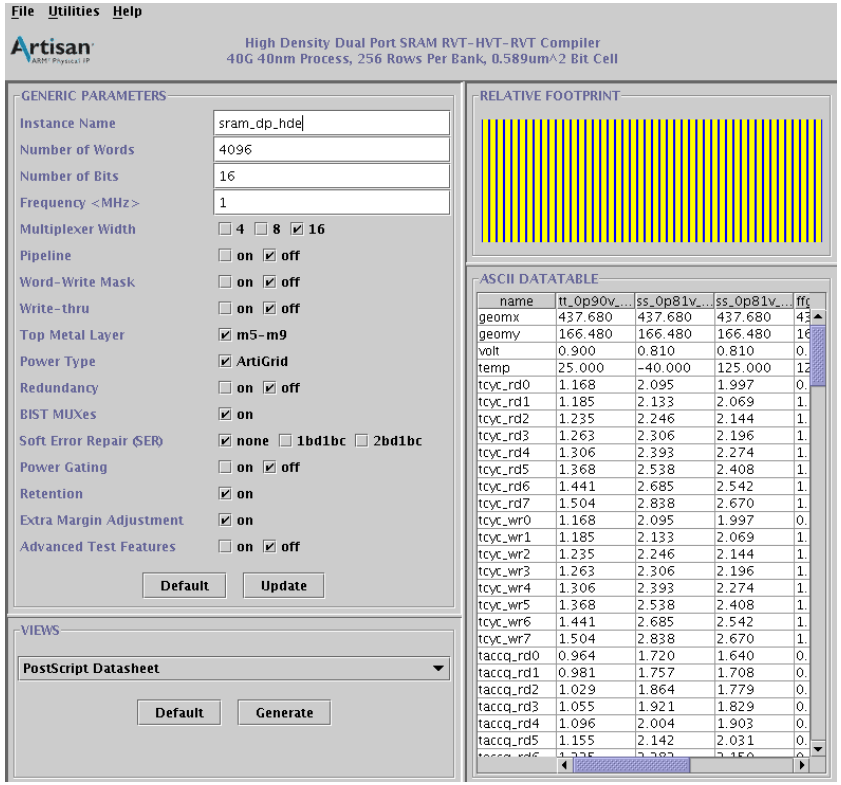
SRAM



  - Area comparison:
    - RF mux width (4, 8) : (10291, 10165) μm^2
    - SRAM mux width (8, 16, 32): (13346, 16651, 26693) μm^2

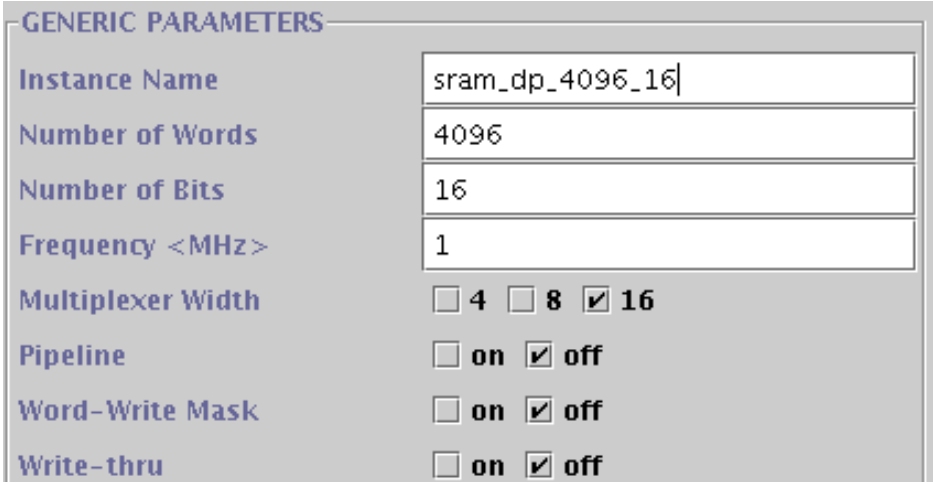
Memory Compiler – Parameters

- Interface (take dual-port SRAM for example):
  - execute the following command
    - /cad/cell_library/CBDK_TSMC40_Arm_f2.0/CIC/Memory/sram_dp_hde_rvt_hvt_rvt/r5p0/bin/**sram_dp_hde_rvt_hvt_rvt**



Memory Compiler – Parameters

- Usually we only modify:
  - Name
  - Number of words
  - Number of bits
  - Multiplexer Width
  - (Word-write mask)

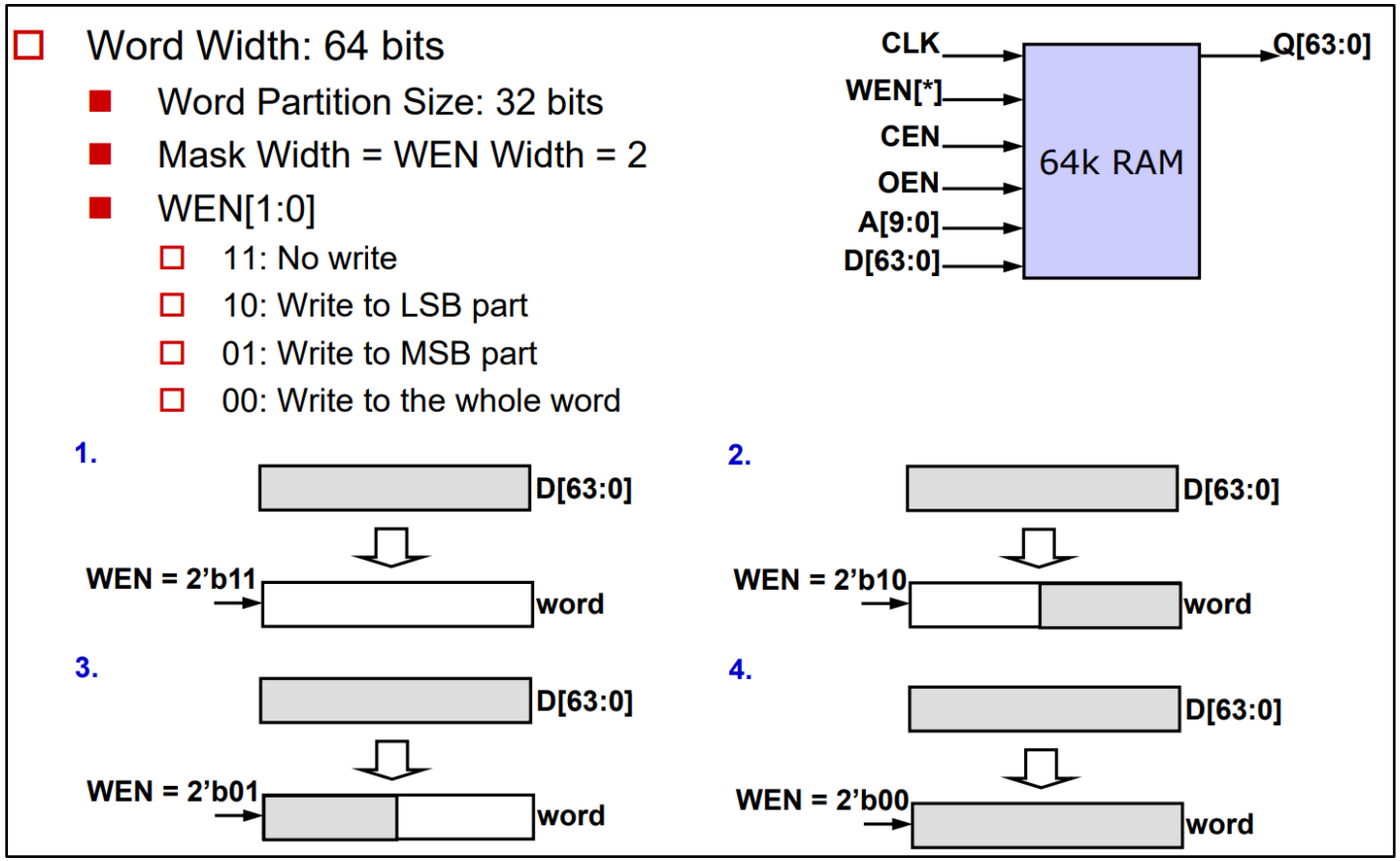


- Multiplexer width:
  - You should not read and write in the same clock where the address difference < multiplexer width to avoid row contention
  - Otherwise, the read or write operation might fail

```
row contention: in sram_tb.sram_unit.genBlock1[1].sbj_memory at 194000
contention: read B succeeds in sram_tb.sram_unit.genBlock1[1].sbj_memory at 194000
contention: write A fails in sram_tb.sram_unit.genBlock1[1].sbj_memory at 194000
```

Memory Compiler – Parameters

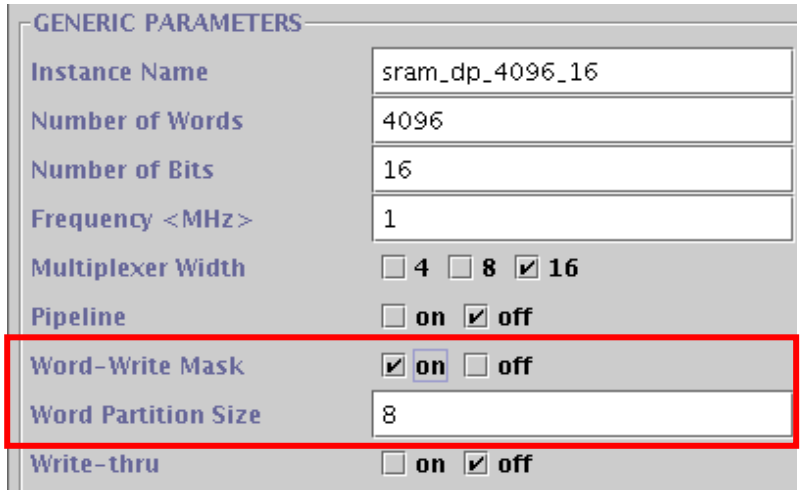
- Word-write mask:
  - Useful when you don't want to write the whole word but part of the word



Ref: http://www.ee.ncu.edu.tw/~jfli/vlsidi/lecture08/design-compiler

Memory Compiler – Parameters

- Word-write mask:
  - Useful when you don't want to write the whole word but part of the word
  - Word partition option will appear after you check the word-write box



- Without word-write mask, you have to do the following steps to achieve the same result:
  - Read the content first → change part of the data → write the whole word back

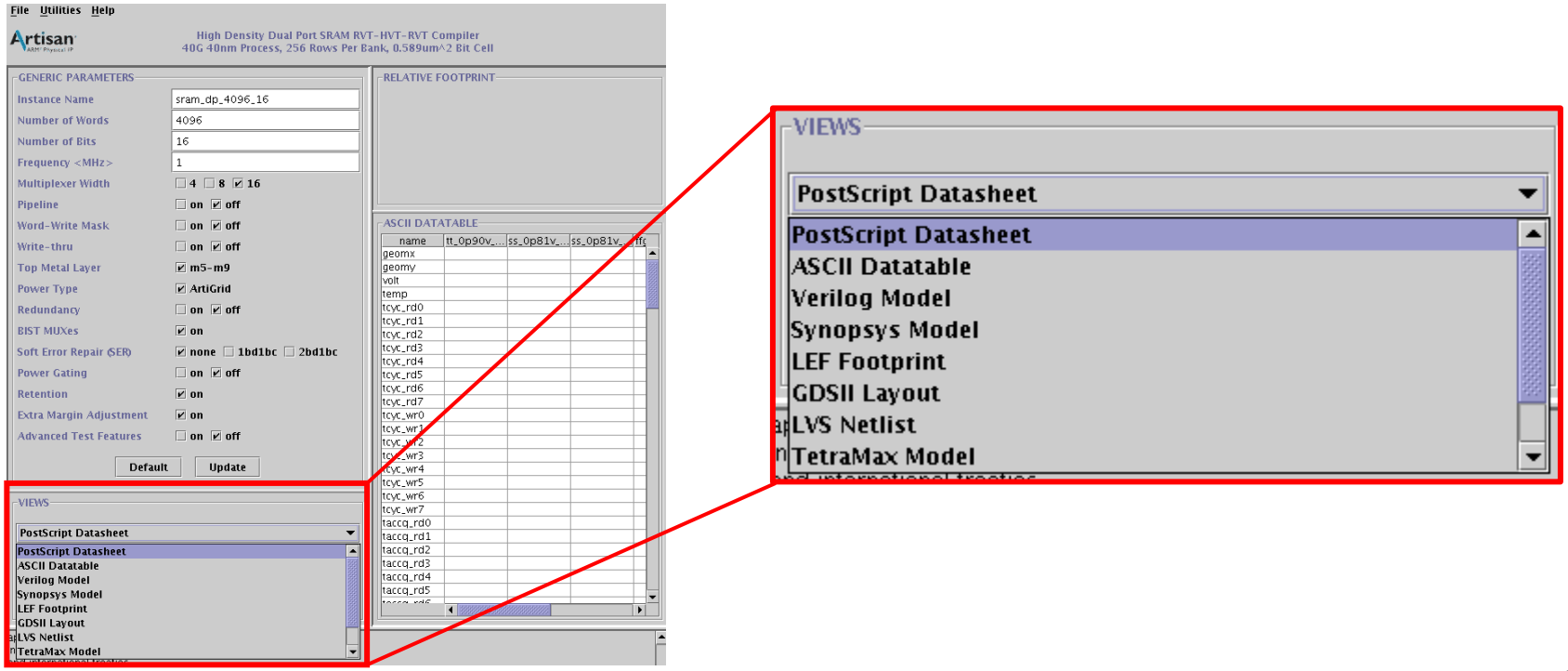
Memory Compiler – Parameters

- Pipeline:
  - A flip-flop is placed between the output of the memory and QA
- Write-through:
  - The data written into the memory is propagated through to the output port

- If you are interested in other options, you can find the detail explanation in the user guide document located at **/Memory/*/*/doc/*_userguide.pdf*
  - E.g. */cad/cell_library/CBDK_TSMC40_Arm_f2.0/CIC/Memory/sram_dp_hde_rvt_hvt_rvt/r5p0/doc/sram_dp_hde_rvt_hvt_rvt_userguide.pdf*

Memory Compiler – Files Generation

- You will only need:
  - Verilog model (.v): for ncverilog simulation
  - Synopsys model (.lib): timing information for synthesis and APR tool
  - LEF footprint (.lef/ .clf): physical layout for the APR tool
  - PostScript DataSheet (.ps) (optional): user guide



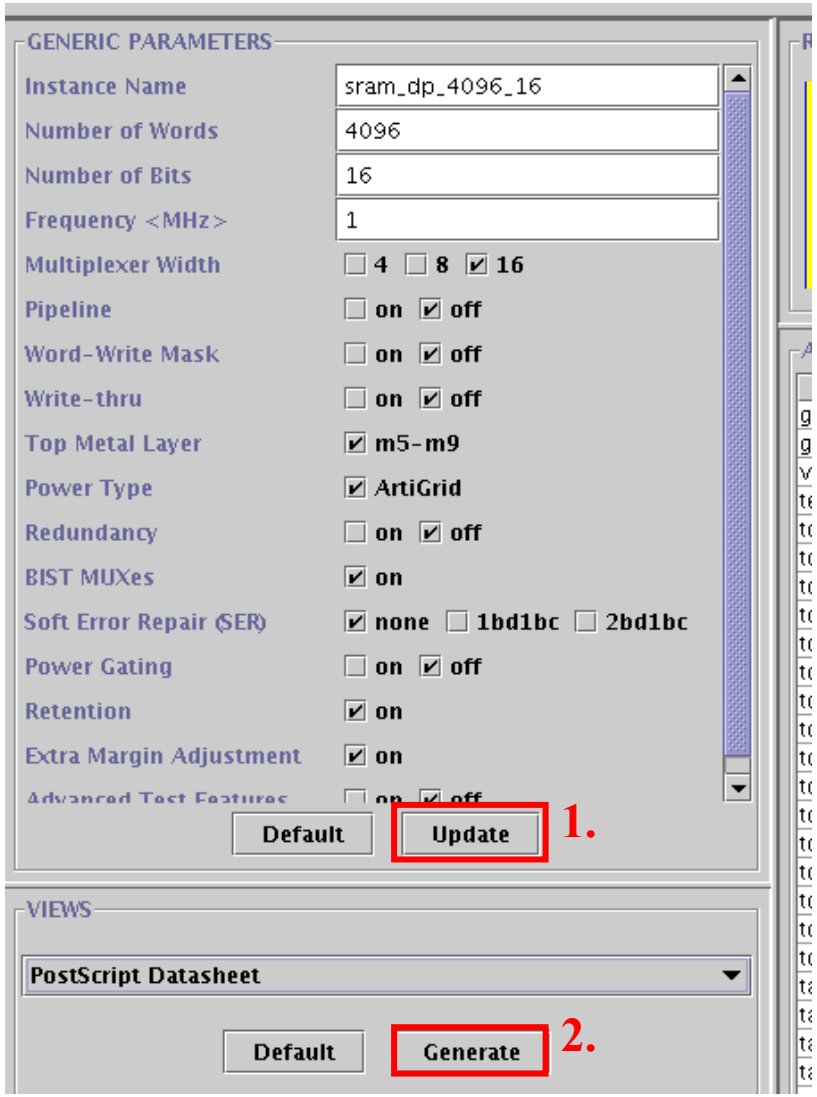
Memory Compiler – Files Generation

- PostScript DataSheet (.ps)
  - The generated files are *ps* files
  - Convert to PDF: ps2pdf *.ps
  - You can find the pin description in the document

Pin Description

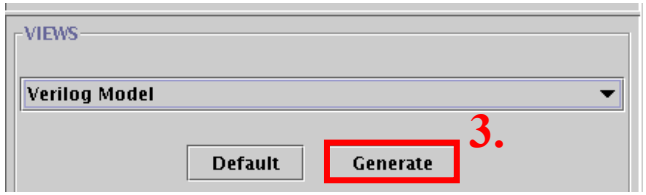
Pin	Description
AA[5:0], AB[5:0]	Port A & B Addresses (AA[0],AB[0] = LSB)
DA[127:0], DB[127:0]	Port A & B Data Inputs (DA[0],DB[0] = LSB)
CLKA, CLKB	Port A & B Clocks
CENA, CENB	Port A & B Chip Enables (active low)
WENA,WENB	Port A & B Write Enables (active low)
QA[127:0], QB[127:0]	Port A & B Data Outputs (QA[0],QB[0] = LSB)
EMAA[2:0], EMAB[2:0]	Port A & B Margin Adjustment (EMAA[0],EMAB[0] = LSB)
EMASA, EMASB	Port A & B Sense Amp Extra Margin Adjustment (EMASA,EMASB)
EMAWA[1:0], EMAWB[1:0] = LSB	Port A & B Write Extra Margin Adjustment (EMAWA[0],EMAWB[0])
TENA, TENB	Port A & B Test Mode Enables (active low)
TAA[5:0], TAB[5:0]	Port A & B Address Test Inputs (TAA[0],TAB[0] = LSB)
AYA[5:0], AYB[5:0]	Port A & B Address Mux Outputs (AYA[0],AYB[0] = LSB)
TDA[127:0], TDB[127:0]	Port A & B Data Test Inputs (TDA[0],TDB[0] = LSB)
DYA[127:0], DYB[127:0]	Port A & B Data Mux Outputs (DYA[0],DYB[0] = LSB)
TCENA, TCENB	Port A & B Chip Enable Test Inputs
CENYA, CENYB	Port A & B Chip Enable Mux Outputs
TWENA,TWENB	Port A & B Write Enable Test Inputs (active low)
WENYA,WENYB	Port A & B Write Enable Mux Outputs
BENA, BENB	Port A & B Bypass Mode Enables (active low)
TQA[127:0], TQB[127:0]	Port A & B Test mux QA,QB Inputs (TQA[0],TQB[0] = LSB)
COLLDISN	Allow the user to disable the internal collision detection circuitry(active low)
RET1N	Retention Input (active low)
STOVA, STOVB	Self timing override inputs

Pin description for the DP-SRAM

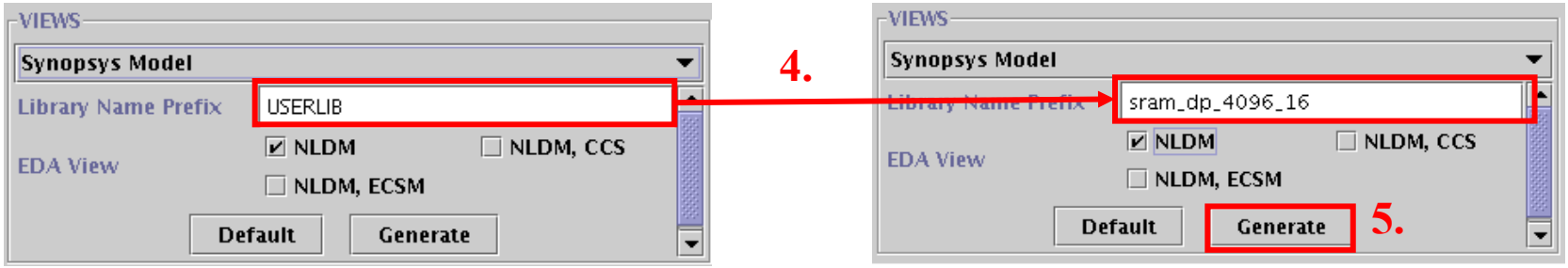


Memory Compiler – Files Generation

- Verilog model

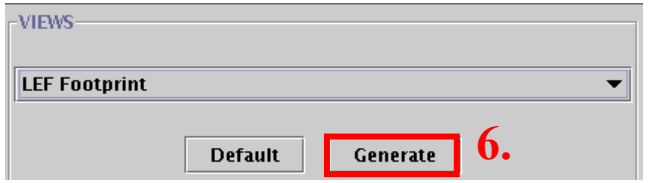


- Synopsis model
  - The default library name is "USERLIB"
  - It is recommended that you change the name to your instance name



  - Use lc_shell to convert .lib to .db (necessary)

- LEF Footprint



Memory Compiler – Files Generation

- Use lc_shell to convert .lib to .db (necessary)
  - Step.1 Open library compiler with lc_shell
  - Step.2 Read library (read_lib *.lib)

    >> read_lib sram_dp_64x128_nldm_tt_0p90v_0p90v_25c_syn.lib

    Library file

```
Technology library 'sram_dp_64x128_nldm_tt_0p90v_0p90v_25c' read successfully
1
lc_shell>
```

Library name (will be "USERLIB_..." if you didn't change the name)

  - Step.3 Convert to db (write_lib * -o *.db)

    >> write_lib sram_dp_64x128_nldm_tt_0p90v_0p90v_25c

    -o sram_dp_64x128_nldm_tt_0p90v_0p90v_25c_syn.db

    db file

- We will convert three different corners
  - Slow: ss + 125c
  - Typical: tt + 25c
  - Fast: ff + m40c

Memory Compiler – Files Generation

- Organizing your files
  - Move files to their respective folders (lib/ ps/ lef/ db)

Name	Size (KB
..	
RTL	
syn	
sram_dp_64x128	
sram_output_test	8
sram_test_gen.m	1

Name	Size (KB
..	
lib	
ps	
lef	
db	
sram_dp_64x128_tt_0p90v_0...	64
sram_dp_64x128.v	1 185

Name	Size (KB)
..	
sram_dp_64x128_nldm_ss_0p81v_0p...	5 868
sram_dp_64x128_nldm_ffg_0p99v_0p...	5 868
sram_dp_64x128_nldm_tt_0p90v_0p9...	5 868
sram_dp_64x128_nldm_ss_0p81v_0p...	5 868
sram_dp_64x128_nldm_ff_0p99v_0p9...	5 869
sram_dp_64x128_nldm_ff_0p99v_0p9...	5 868

Lib (6 files)

Name	Size (KB)
..	
sram_dp_64x128_nldm_ff_0p99v_0p9...	432
sram_dp_64x128_nldm_ss_0p81v_0p...	440
sram_dp_64x128_nldm_tt_0p90v_0p9...	440

db (3 files)

Name	Size (KB)
..	
sram_dp_64x128_antenna.clf	88
sram_dp_64x128.lef	494

lef (2 files)

Name	Size (KB)
..	
sram_dp_64x128_ff_0p99v_0p99v_m...	136
sram_dp_64x128_ss_0p81v_0p81v_1...	136
sram_dp_64x128_tt_0p90v_0p90v_2...	136

ps (3 ps/pdf files)

Memory Compiler – Using the files

- Verilog model: post rtl/syn/apr simulation
  - *ncverilog … sram_dp_64x128.v*

- DB files (Synthesis):
  - Add path to *.synopsys_dc.setup*

```
set search_path ".  /cad/cell_library/CBDK_TSMC40_Arm_f2.0/CIC/SynopsysDC/db/sc9_base_rvt/ \
                    /cad/cell_library/CBDK_TSMC40_Arm_f2.0/CIC/SynopsysDC/db/sc9_base_hvt/ \
                    /cad/cell_library/CBDK_TSMC40_Arm_f2.0/CIC/SynopsysDC/db/sc9_base_lvt/ \
                    ../sram_dp_64x128/db/ \
                    $search_path"
set target_library "sc9_cln40g_base_rvt_ss_typical_max_0p81v_125c.db \
                    sc9_cln40g_base_rvt_ff_typical_min_0p99v_m40c.db \
                    sc9_cln40g_base_hvt_ss_typical_max_0p81v_125c.db \
                    sc9_cln40g_base_hvt_ff_typical_min_0p99v_m40c.db \
                    sc9_cln40g_base_lvt_ss_typical_max_0p81v_125c.db \
                    sc9_cln40g_base_lvt_ff_typical_min_0p99v_m40c.db \
                    sram_dp_64x128_nldm_ss_0p81v_0p81v_125c_syn.db \
                    sram_dp_64x128_nldm_ff_0p99v_0p99v_m40c_syn.db \
                   "
```

Memory Compiler – Using the files

- Library files (APR) :
  - Add path to the *.view* file

```
create_library_set -name lib_max \
            -timing {/cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SynopsysDC/lib/sc9_base_rvt/sc9_cln40g_base_rvt_ss_typical_max_0p81v_125c.lib \
            /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SynopsysDC/lib/sc9_base_lvt/sc9_cln40g_base_lvt_ss_typical_max_0p81v_125c.lib \
            /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SynopsysDC/lib/sc9_base_hvt/sc9_cln40g_base_hvt_ss_typical_max_0p81v_125c.lib \
            ../blackScholes/sram_dp_1024_120/lib/sram_dp_1024_120_nldm_ss_0p81v_0p81v_125c_syn.lib \
            ../blackScholes/sram_dp_2048_120/lib/sram_dp_2048_120_nldm_ss_0p81v_0p81v_125c_syn.lib \
            ../blackScholes/sram_sp_256_120/lib/sram_sp_256_120_nldm_ss_0p81v_0p81v_125c_syn.lib \
            ../blackScholes/sram_sp_4096_120/lib/sram_sp_4096_120_nldm_ss_0p81v_0p81v_125c_syn.lib \
            ../blackScholes/rf_2p_16_240/lib/rf_2p_16_240_nldm_ss_0p81v_0p81v_125c_syn.lib \
            } \
            -si {/cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/celtic/sc9_base_rvt/sc9_cln40g_base_rvt_ss_typical_max_0p81v_125c.cdB \
            /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/celtic/sc9_base_lvt/sc9_cln40g_base_lvt_ss_typical_max_0p81v_125c.cdB \
            /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/celtic/sc9_base_hvt/sc9_cln40g_base_hvt_ss_typical_max_0p81v_125c.cdB \
            }

create_library_set -name lib_min \
            -timing {/cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SynopsysDC/lib/sc9_base_rvt/sc9_cln40g_base_rvt_ff_typical_min_0p99v_m40c.lib \
            /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SynopsysDC/lib/sc9_base_lvt/sc9_cln40g_base_lvt_ff_typical_min_0p99v_m40c.lib \
            /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SynopsysDC/lib/sc9_base_hvt/sc9_cln40g_base_hvt_ff_typical_min_0p99v_m40c.lib \
            ../blackScholes/sram_dp_1024_120/lib/sram_dp_1024_120_nldm_ff_0p99v_0p99v_m40c_syn.lib \
            ../blackScholes/sram_dp_2048_120/lib/sram_dp_2048_120_nldm_ff_0p99v_0p99v_m40c_syn.lib \
            ../blackScholes/sram_sp_256_120/lib/sram_sp_256_120_nldm_ff_0p99v_0p99v_m40c_syn.lib \
            ../blackScholes/sram_sp_4096_120/lib/sram_sp_4096_120_nldm_ff_0p99v_0p99v_m40c_syn.lib \
            ../blackScholes/rf_2p_16_240/lib/rf_2p_16_240_nldm_ff_0p99v_0p99v_m40c_syn.lib \
            } \
            -si {/cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/celtic/sc9_base_rvt/sc9_cln40g_base_rvt_ff_typical_min_0p99v_m40c.cdB \
            /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/celtic/sc9_base_lvt/sc9_cln40g_base_lvt_ff_typical_min_0p99v_m40c.cdB \
            /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/celtic/sc9_base_hvt/sc9_cln40g_base_hvt_ff_typical_min_0p99v_m40c.cdB \
            }
```

Memory Compiler – Using the files

- Lef files (APR) :
  - Add path to the *.globals* file

```
set init_lef_file {     /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/lef/sc9_tech.lef \
        /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/lef/sc9_cln40g_base_rvt.lef \
        /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/lef/sc9_cln40g_base_lvt.lef \
        /cad/CBDK/CBDK_TN40G_Arm/CBDK_TSMC40_core_Arm_v2.0/CIC/SOCE/lef/sc9_cln40g_base_hvt.lef \
        ../blackScholes/sram_dp_1024_120/lef/sram_dp_1024_120.lef \
        ../blackScholes/sram_dp_2048_120/lef/sram_dp_2048_120.lef \
        ../blackScholes/sram_sp_256_120/lef/sram_sp_256_120.lef \
        ../blackScholes/sram_sp_4096_120/lef/sram_sp_4096_120.lef \
        ../blackScholes/rf_2p_16_240/lef/rf_2p_16_240.lef \
        }
```

- APR for 40nm technology requires access to the EDA Cloud
  - We will not practice APR today

Memory Compiler – Ports

- Single-port SRAM:

Port name	Description
CLK	Clock
CEN	Active-low chip enable (0 to enable)
WEN	Active-low write enable (1 for read, 0 for write)
A	Address
D	Data input
Q	Data output
Ohers	Connect to 0 or 1

```
sram_sp_4096_120 UX_reg_mem4 (
    .CLK(CLK),
    .CEN( !(UX_w_en4||UX_r_en4) ),
    .WEN(!UX_w_en4),
    .A(UX_addr4),
    .D(UX_in4[((idx+1)*120-1)-:120]),
    // output
    //.QA(),
    .Q(UX_out4[((idx+1)*120-1)-:120]),
    .EMA(3'b0),
    .EMAS(1'b0),
    .EMAW(2'b0),
    .BEN(1'b1),
    .STOV(1'b0),
    .TEN(1'b1),
    .TCEN(1'b1),
    .TWEN(1'b1),
    .TA(12'b0),
    .TD(120'b0),
    .TQ(120'b0),
    .RET1N(1'b1)
);
```

Example

Memory Compiler – Ports

- Dual-port SRAM:

Port name	Description
CLKA/ CLKB	Clock
CENA/ CENB	Active-low chip enable (0 to enable)
WENA/ WENB	Active-low write enable (1 for read, 0 for write)
AA/ AB	Address
DA/ DB	Data input
QA/ QB	Data output
Ohers	Connect to 0 or 1

```
sram_dp_64x128 sbj_memory ( // can store at m
    .CLKA(CLK),
    .CLKB(CLK),
    .CENA( !sram_WEN ),
    .CENB( !sram_REN  ),
    .WENA(1'b0),
    .WENB(1'b1),
    .AA(sram_waddr),
    .AB(sram_raddr),
    .DA(sram_in[(128*(gen_idx+1)-1)-:128]),
    .DB(128'b0),
    // output
    //.QA(),
    .QB(sram_out[(128*(gen_idx+1)-1)-:128]),
    .EMAA(3'b0),
    .EMAB(3'b0),
    .EMASA(1'b0),
    .EMASB(1'b0),
    .EMAWA(2'b0),
    .EMAWB(2'b0),
    .BENA(1'b1),
    .BENB(1'b1),
    .STOVA(1'b0),
    .STOVB(1'b0),
    .TENA(1'b1),
    .TENB(1'b1),
    .RET1N(1'b1)
);
```

Example (always 1 read 1 write)

Memory Compiler – Ports

- 2-port RF:

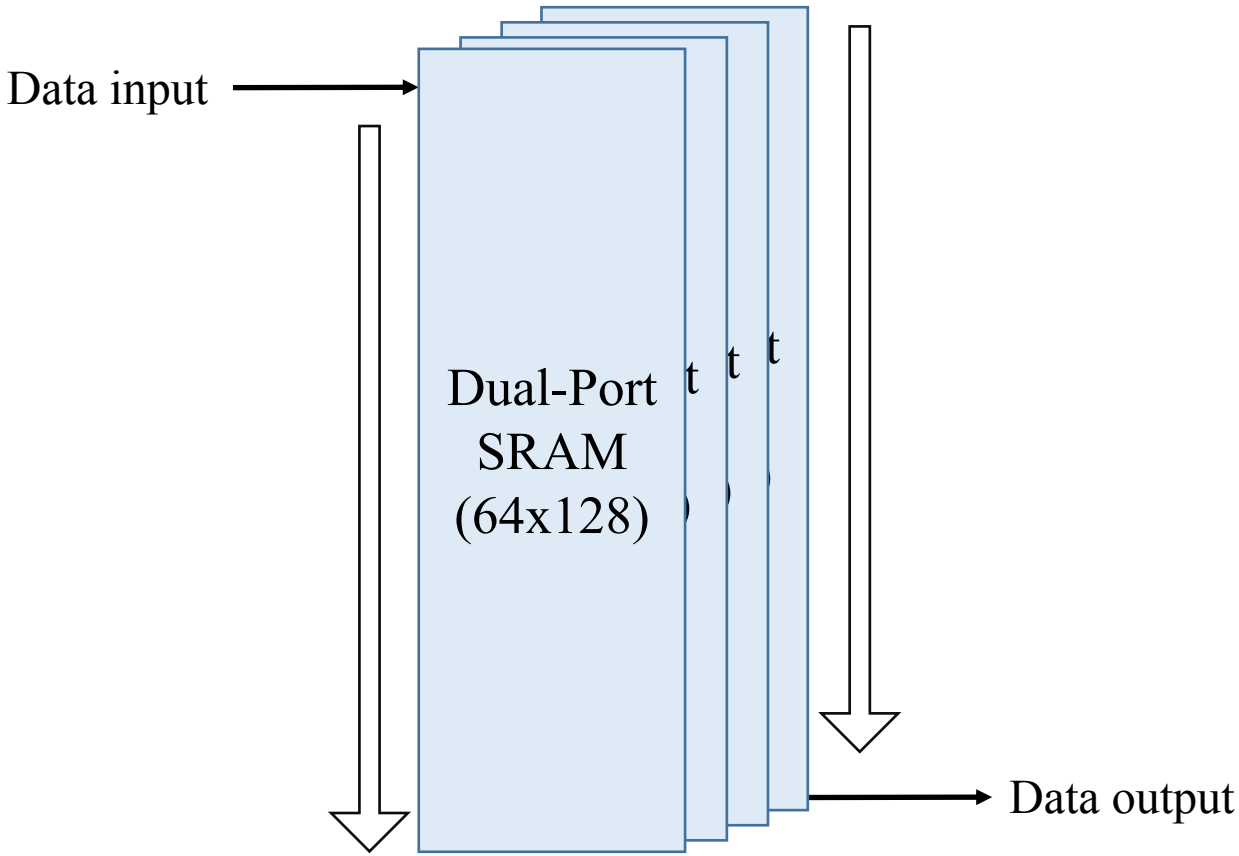
Port name	Description
CLKA/ CLKB	Clock
CENA/ CENB	Active-low chip enable (0 to enable)
AA/ AB	Address
DB	Data input
QA	Data output
Ohers	Connect to 0 or 1

```
rf_2p_16_240 last_UX_mem (
    .CLKA(CLK),
    .CLKB(CLK),
    .CENA( !last_UX_r_en ),
    .CENB( !last_UX_w_en  ),
    .AA(last_UX_out_addr),
    .AB(last_UX_in_addr),
    .DB(last_UX_in[idx]),
    .QA(last_UX_out[idx]),
    .EMAA(3'b0),
    .EMAB(3'b0),
    .EMASA(1'b0),
    .EMAWB(2'b0),
    .BENA(1'b1),
    .STOVA(1'b0),
    .STOVB(1'b0),
    .TENA(1'b1),
    .TENB(1'b1),
    .RET1N(1'b1),
    .TCENA(1'b1),
    .TAA(4'd0),
    .TQA(240'd0),
    .TCENB(1'b1),
    .TAB(4'd0),
    .TDB(240'd0),
    .COLLDISN(1'b1)
);
```

Example (A: read; B: write)

Lab #1 – Dual-Port SRAM

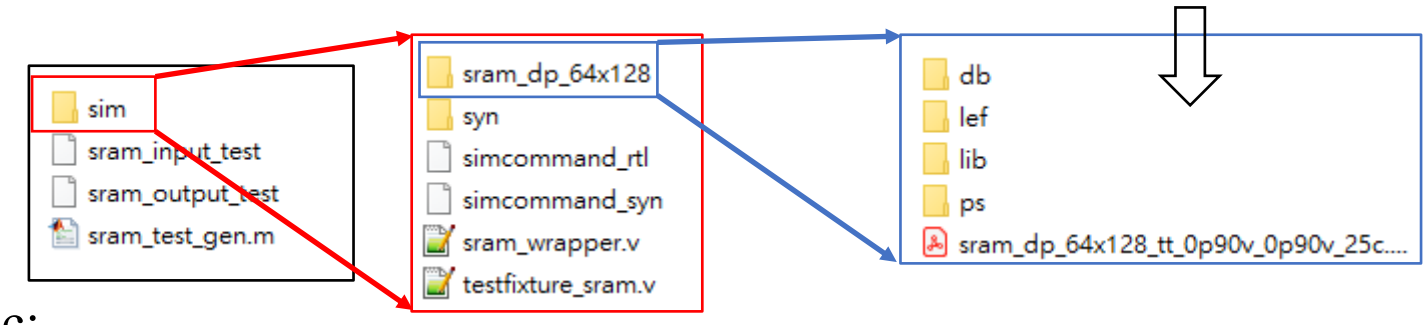
- Function:
  - Input → Store in SRAM → Read from SRAM → Output



Lab #1 – Dual-Port SRAM

- We have completed partial RTL files and synthesis script
- Your task:
  - Find some wrong connected nets in **sram_wrapper.v** and fix them
  - Use the memory compiler to generate the necessary files and put them in respective folders

Fill the blank folders with the *.db/ .lib/ .lef/ .ps/ .v* files



- Spec:
  - Instance name: sram_dp_64x128
  - Number of words: 64
  - Number of bits: 128
  - Multiplexer width: 4

Lab #1 – Dual-Port SRAM

- Dual-port SRAM:
  - always 1 read 1 write
  - find some bugs in **sram_wrapper.v**
    - take A as write port
    - take B as read port

Port name	Description
CLKA/ CLKB	Clock
CENA/ CENB	Active-low chip enable (0 to enable)
WENA/ WENB	Active-low write enable (1 for read, 0 for write)
AA/ AB	Address
DA/ DB	Data input
QA/ QB	Data output
Ohers	Connect to 0 or 1

```
generate
  for(gen_idx=0;gen_idx<4;gen_idx=gen_idx+1)begin: genBlock1
        sram_dp_64x128 sbj_memory (
                .CLKA(CLK),
                .CLKB(CLK),
                .CENA( !sram_REN ),
                .CENB( !sram_WEN  ),
                .WENA(1'b0),    // port A is always used to store
                .WENB(1'b1),    // port B provide the sequence data to register array/
                .AA(sram_waddr),
                .AB(sram_raddr),
                .DA(sram_in[(128*(gen_idx+1)-1)-:128]),
                .DB(128'b0),
                // output
                //.QA(),
```

Lab #1 – Dual-Port SRAM

- Steps:
  1. Generate all the files
  2. Find some bugs/RTL simulation: *bash simcommand_rtl*
  3. Synthesis (syn/): *dc_shell –f all_syn.tcl*
  4. Post-Synthesis simulation: *bash simcommand_syn*

```
-------------------------- Simulation Stops !!----------
============================================================

        ****************************
        **                        **        /|__/|
        **  Congratulations !!    **       / 0,0  |
        **                        **     /_____   |
        **  Simulation Complete!! **    /^ ^ ^ \  |
        **                        **   |^ ^ ^ ^ |w|
        *************** ************    \m___m__|_|

============================================================
Simulation complete via $finish(1) at time 408 NS + 0
./testfixture_sram.v:107                          $finish;
ncsim> exit
```

Pass simulation

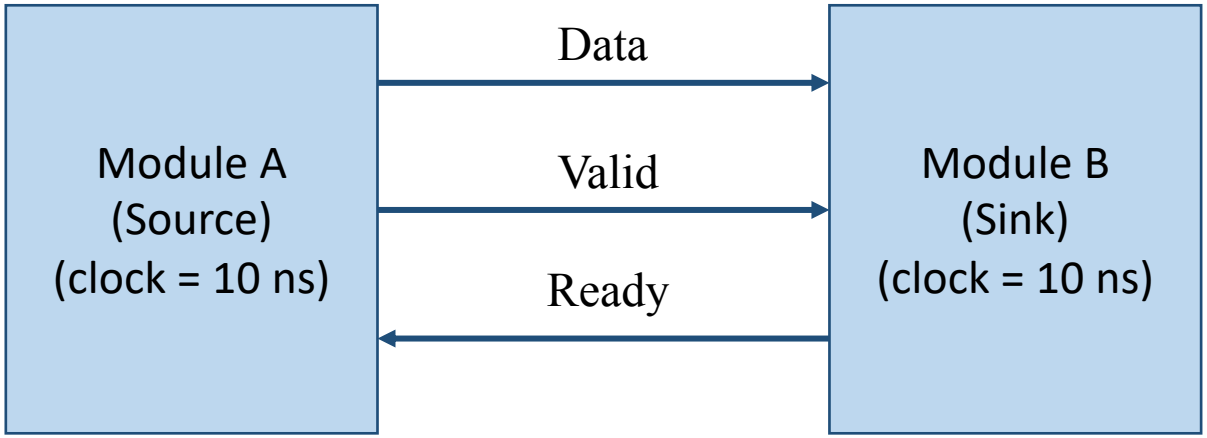
```
                                                  Weighted
Group (max_delay/setup)      Cost     Weight      Cost
-----------------------------------------------------
CLK                          0.00      1.00       0.00
default                      0.00      1.00       0.00
-----------------------------------------------------
max_delay/setup                                   0.00

                          Total Neg  Critical
Group (critical_range)      Slack    Endpoints    Cost
-----------------------------------------------------
CLK                          0.00         0       0.00
default                      0.00         0       0.00
-----------------------------------------------------
critical_range                                    0.00

                                                  Weighted
Group (min_delay/hold)       Cost     Weight      Cost
-----------------------------------------------------
CLK                          0.00      1.00       0.00
default                      0.00      1.00       0.00
-----------------------------------------------------
min_delay/hold                                    0.00
```

```
Constraint                                        Cost
-----------------------------------------------------
multiport_net                                     0.00 (MET)
min_capacitance                                   0.00 (MET)
max_transition                                    0.00 (MET)
max_fanout                                        0.00 (MET)
max_capacitance                                   0.00 (MET)
max_delay/setup                                   0.00 (MET)
sequential_clock_pulse_width                      0.00 (MET)
critical_range                                    0.00 (MET)
min_delay/hold                                    0.00 (MET)
min_period                                        0.00 (MET)
max_area                                     262186.38
                                                        (VIOLATED)
```

Synthesis result

Lab #2 – Asynchronous FIFO

- Data transfer between hardware modules
- The most frequently used handshake mechanism in digital design is the valid-ready protocol



  - Source asserts valid when it has data to transfer, and puts data on the line
  - Sinks asserts ready when it is ready to receive, and stores the data
  - Source acknowledges the ready signal and sends the next data
- What if the source generates the data faster than the sink can consume them?
  - E.g. Source generates data in a burst while sink receives them slow but steadily

Lab #2 – Asynchronous FIFO

- Data transfer between hardware modules
- A synchronous FIFO can be inserted that act as a buffer when the data transfer is not at the same rate

- Synchronous: the same clock is used for both reading and writing
- FIFO depth: the size of the buffer inside the FIFO
  - Calculating FIFO depth: http://www.asic-world.com/tidbits/fifo_depth.html
- What if source and sink operate at different frequency?

Lab #2 – Asynchronous FIFO

- Data transfer between hardware modules
- An asynchronous FIFO uses different clock for reading and writing

- The design tips of asynchronous FIFO can be found in paper:

Cummings, Clifford E. "Simulation and synthesis techniques for asynchronous FIFO design." *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*. 2002.

  - We will be using the implementation from https://github.com/dpretet/async_fifo for today's lab

Lab #2 – Asynchronous FIFO

- Asynchronous FIFO ports (2^8 words, 16 bits data):

```
module async_fifo
#(parameter DSIZE = 16, parameter ASIZE = 8, parameter FALLTHROUGH = "FALSE")
(
    input  wire              wclk,
    input  wire              wrst_n,
    input  wire              winc,
    input  wire [DSIZE-1:0]  wdata,
    output wire              wfull,
    output wire              awfull,
    input  wire              rclk,
    input  wire              rrst_n,
    input  wire              rinc,
    output wire [DSIZE-1:0]  rdata,
    output wire              rempty,
    output wire              arempty
);
```

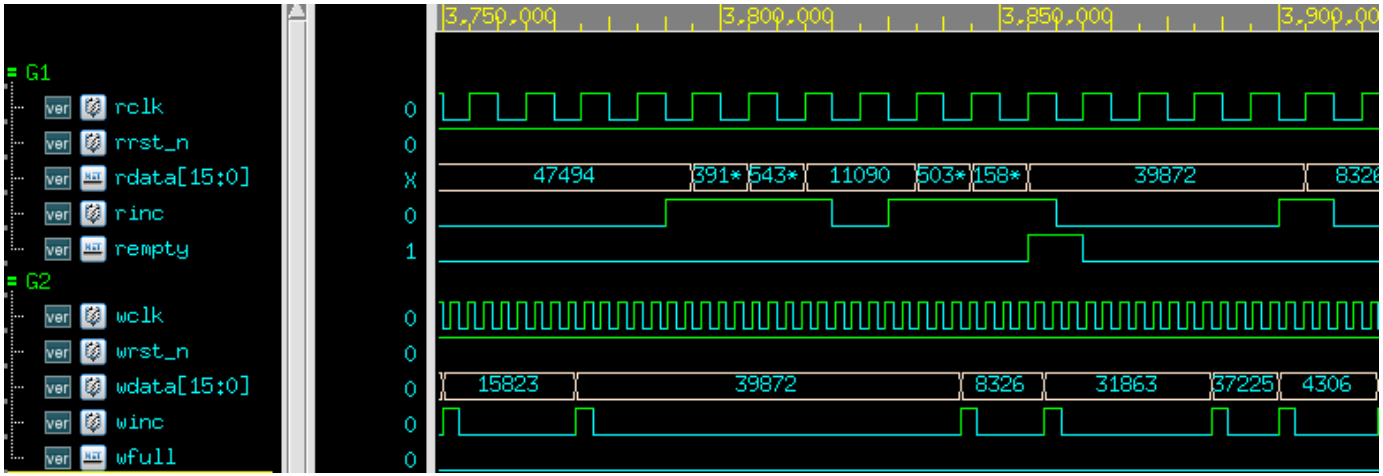
Read data will appear in the next clock

Port name	Description
wclk	Clock
wrst_n	Active low reset
winc	Write enable
wdata	Write data
wfull	Indicates FIFO is full
awfull	Almost full

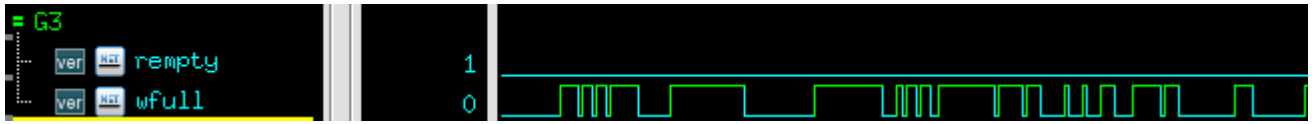
Port name	Description
rclk	Clock
rrst_n	Active low reset
rinc	Read enable
rdata	Read data
rempty	Indicates FIFO is empty
arempty	Almost empty

Lab #2 – Asynchronous FIFO

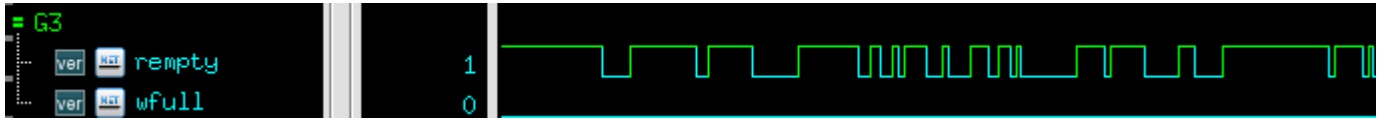
- Testbench: random reads and writes



- Default: balanced number of reads and writes
- +define+PAT2: increase number of writes (test full)



- +define+PAT3: decrease number of writes (test empty)



Lab #2 – Asynchronous FIFO

- Data memory inside the FIFO (fifo_2mem.v):
  - The provided file uses registers to compose the buffer
  - For FPGA synthesis, this will be synthesized as block RAM by inference
  - For ASIC design, however, this will be synthesized as registers (large area)
  - We can replace the memory with a 2 port RF to reduce the area

```
  reg [DATASIZE-1:0] mem [0:DEPTH-1];

  always @(posedge wclk) begin
      if (wclken && !wfull)
          mem[waddr] <= wdata;
  end

generate
  if (FALLTHROUGH == "TRUE")
    begin : fallthrough
      always @*
        rdata = mem[raddr];
    end
  else
    begin : registered_read
      always @(posedge rclk) begin
        if (rclken)
          rdata <= mem[raddr];
      end
    end
endgenerate
```

Lab #2 – Asynchronous FIFO

- You should be able to successfully run simulation and synthesis before any modification:
  - RTL simulation: *bash simcommand_rtl*
  - Synthesis (syn/): *dc_shell –f all_syn.tcl*
  - Post-Synthesis simulation: *bash simcommand_syn*

- The area of FIFO after synthesis is 26856

```
Combinational area:                10653.249416
Buf/Inv area:                        425.023203
Noncombinational area:             16202.819267
Macro/Black Box area:                  0.000000
Net Interconnect area:                 0.000000

Total cell area:                   26856.068682
Total area:                        26856.068682
```

Lab #2 – Asynchronous FIFO

- In this lab, You task is to replace the FIFO memory (fifo_2mem.v) with a 2-port register file to reduce the total cell area to < 10000
- Step-by-step hints:
  - Find the proper parameters (# of words, # of bits, …) for the 2-port RF
  - Generate all the files (.db, .lib, .v)
  - Replace the memory in *fifo_2mem.v* with the RF you generates
  - Modify *simcommand_rtl* and run RTL simulation
  - Add the path of the db files to *.synopsys_dc.setup*
  - Run synthesis (record total cell area)
  - Modify *simcommand_syn* and run post-synthesis simulation
  - Done

```
Combinational area:                 230.428797
Buf/Inv area:                        23.814000
Noncombinational area:              344.962783
Macro/Black Box area:              9221.239258
Net Interconnect area:                0.000000

Total cell area:                   9796.630839
Total area:                        9796.630839
```

Reference result

```
Pattern         998 is passed ! FIFO input = 41681, FIFO output = 41681 !!
Pattern         999 is passed ! FIFO input = 21433, FIFO output = 21433 !!
--------------------------- Simulation Stops !!---------------------------
==============================================================================

        ****************************
        **                        **        /|__/|
        **  Congratulations !!    **      / 0,0  |
        **                        **     /_____   |
        **  Simulation Complete!! **    /^ ^ ^ \  |
        **                        **   |^ ^ ^ ^ |w|
        *************** ************    \m___m__|_|


==============================================================================
Simulation complete via $finish(1) at time 21695 NS + 0
./testfixture_fifo.v:262                        $finish;
```