ASIP Meister Tutorial

Revision 1.1.2
PEAS Project

December 13, 2004

Table of Contents

List of Figures

List of Tables

1. Tutorial convention

This tutorial is written according to the following rules.

- User interfaces such as menus, buttons, text boxes and check boxes are enclosed with brackets, like [Save Design As …].
- “%” symbol means command prompt.
- The name of files and directories are enclosed with double quotation marks, like “/usr/local/ASIPmeister”.

2. Before using ASIP Meister

2.1. Installation and setup

This section briefly explains how to install and set up ASIP Meister. For details, please refer to ASIP Meister 1.1 install guide.

2.1.1. Set the service port number

Add the following lines to "/etc/services".

```
# port for fhm (ASIP Meister)
fhm 55555/tcp
```

2.1.2. Make setup file for ASIP Meister

Make a file that sets the path for executable file for ASIP Meister and JAVA™2. If you have followed ASIP Meister 1.1 install guide, make a setup file with like the following lines;

```
PATH=/usr/java/jre1.3.1_04/bin/:$PATH
PATH=/usr/local/ASIPmeister/bin/:$PATH
export PATH
```

You can use the sample setup file, "/usr/local/ASIPmeister/share/ASIPmeister.setup.sample", to make your setup file.

2.1.3. Design Sample for Tutorial

When ASIP Meister is installed, the design sample specification used in this tutorial is installed in "/usr/local/ASIPmeister/share/tutorial/dlx_integer/" directory.

● dlx_integer.pdf	(English)
● dlx_integer_jpn.pdf	(Japanese)

3. Common operations of ASIP Meister

3.1. Start ASIP Meister

3.1.1. Set the path

Please open a terminal and set the path for JAVA™2 and ASIP Meister. This tutorial assumes that users use the sample ASIP Meister setup file. Run the following command in the directory where “ASIPmeister.setup” file is located.

```
% source ASIPmeister.setup
```

3.1.2. Start ASIP Meister

Please enter the following command.

- To start the new design

```
% ASIPmeister
```

- To update the design

```
% ASIPmeister            ... (1) Select the data name from the [File]>[Open Design] menu.
% ASIPmeister xxx.pdb    ... (2) Specify the design data file name.
```

".pdb" is the extension of the design data file of ASIP Meister.

3.2. Exit ASIP Meister and save the design data

3.2.1. To save a new design data or save the design data under the different name

In [ASIP Meister Main Menu] window, click [File] > [Save Design As …] to open [Save] window (Figure 1).

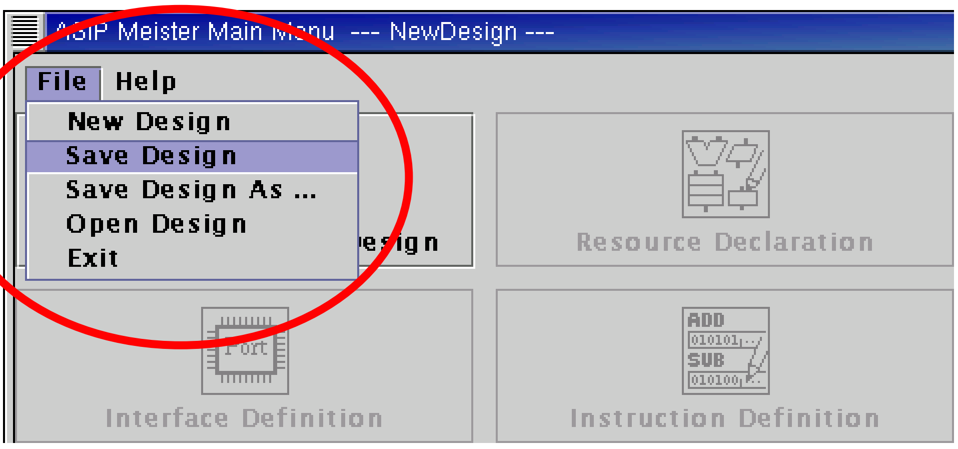
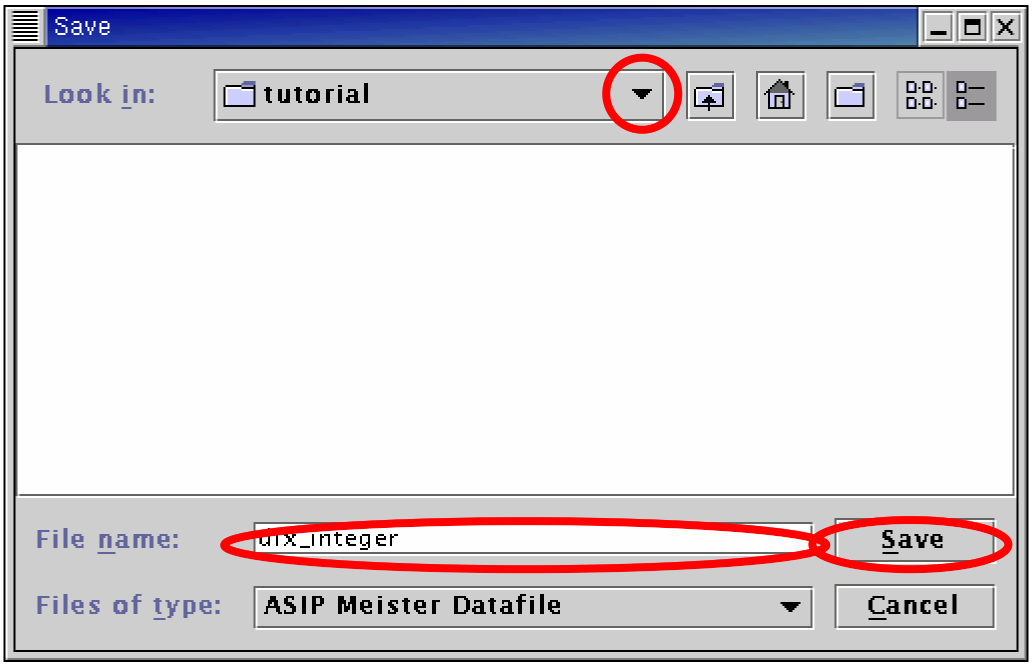


Figure 1: [File] menu

In the [Save] window, click [Look in] pull down menu to select the directory to save the design data (Figure 2).

Enter the design data name in [File name] field. Click [Save] button to save the data and close the window. The extension ".pdb" follows this name.



Look in: tutorial

File name: dlx_integer

Files of type: ASIP Meister Datafile

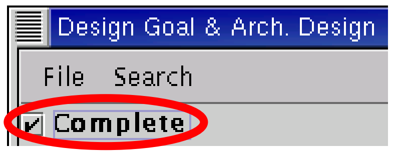
Figure 2: Data save dialog

3.2.2. To update the design data without changing the file name

In [ASIP Meister Main Menu] window, click [File]>[Save Design] menu.

3.3. Close sub windows and return to the main window

After you have finished the design at the sub window, click [Complete] check box and click [File] > [Close] menu to return to the main window (Figure 3). The check box cannot be checked if the information required for the completion of the sub window is missing. Once you check the [Complete] checkbox, the design at the sub window is regarded as having been completed. Then you can proceed the next sub window. If you close the sub window without checking the [Complete] check box, you cannot proceed the next sub window.



✓ Complete

Figure 3: [Complete] check box in sub windows

3.4. Working directory and design data file

3.4.1. Directory "meister"

ASIP Meister creates a working directory named "meister" in the current directory. Working files of ASIP Meister and generated HDL description files are stored in "meister" directory.

3.4.2. Design data

The extension of the design data file of ASIP Meister is “.pdb”. To save the data with ASIP Meister, please specify the design data name. The extension “.pdb” follows this name and the design data will be saved.

4. Processor design flow on ASIP Meister

Open the each sub window for ASIP Meister in the order in Figure 4 and enter information as required.

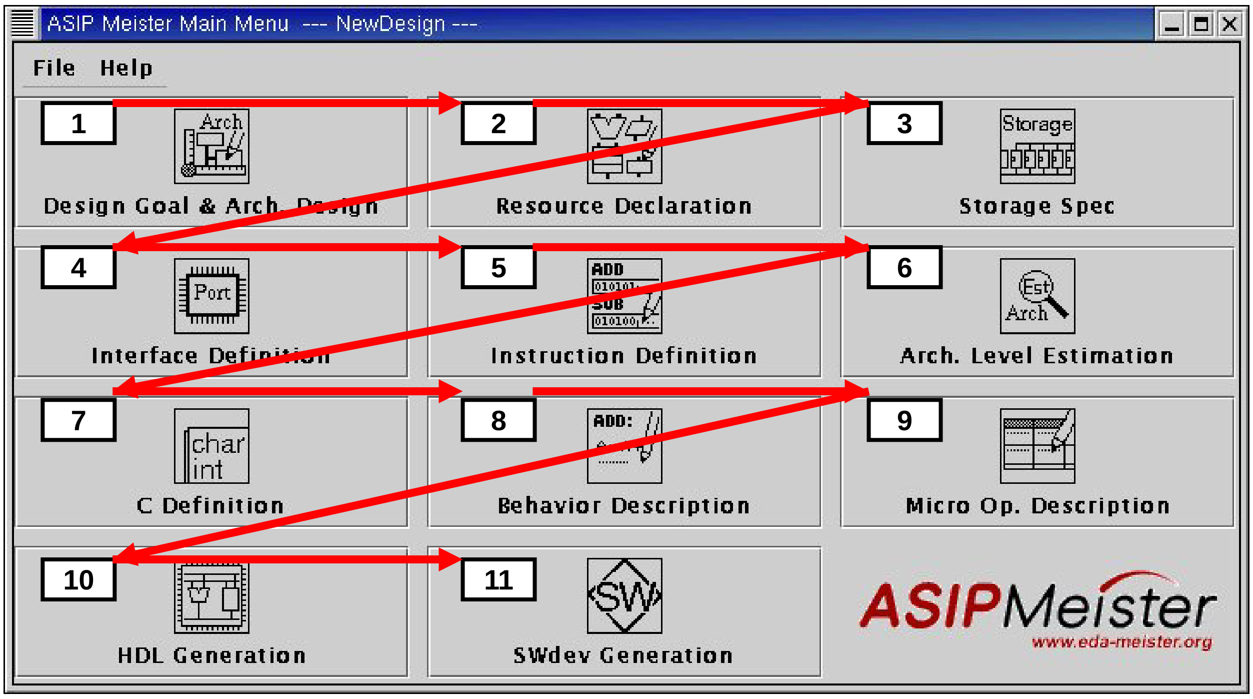


Figure 4: Design flow on ASIP Meister

1) **Design Goal & Arch. Design**

Entering design target values and architecture parameters

2) **Resource Declaration**

Declaring resources

3) **Storage Spec**

Specifying storages

4) **Interface Definition**

Defining I/O interfaces of the target processor

5) **Instruction Definition**

Defining instruction types and instructions

6) **Arch. Level Estimation**

Estimating design qualities of the target processor

7) **C Definition**

Defining C language interfaces

8) **Behavior Description**

Describing behavior of instructions

9) Micro Op. Description

Describing micro operations

10) HDL Generation

Generating HDL

11) SWdev Generation

Generating descriptions for software development tools

5. Confirm the operation with pre-installed tutorial data

The tutorial data is installed in "/usr/local/ASIPmeister/share/tutorial/dlx_integer/" directory.

- Copy tutorial directory

```
% cp -r /usr/local/ASIPmeister/share/tutorial/dlx_integer
```

- Move to the directory

```
% cd dlx_integer
```

- Setup the working environment

```
% source ASIPmeister.setup
```

- Start ASIP Meister with design data

```
% ASIPmeister dlx_integer.pdb
```

6. Design the new DLX integer processor

This section explains how to design a new processor. The sample design is DLX integer processor. Before starting this section, please open "dlx_integer.pdf" for your reference.

First, start ASIP Meister with following procedures;

- Create a working directory (here we will name the directory as “tutorial”)

  ```
  % mkdir tutorial
  ```

- Move to the directory.

  ```
  % cd tutorial
  ```

- Set up the working environment

  ```
  % source ASIPmeister.setup
  ```

- Start ASIPmeister.

  ```
  % ASIPmeister
  ```

If you have proceeded correctly, the main window [ASIP Meister Main Menu] will open (Figure 5).

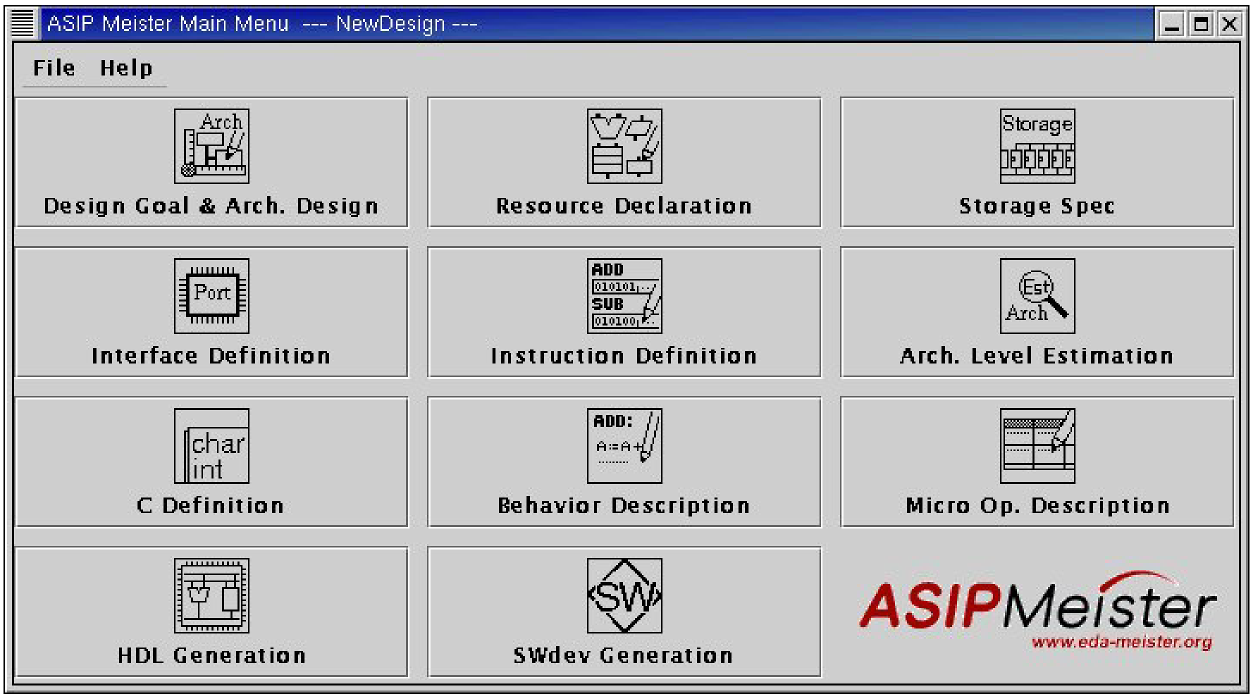


Figure 5: Main window of ASIP Meister

Now, let’s begin the tutorial lesson.

6.1. [Design Goal & Arch. Design] window: setting the design goal value and architecture parameters

This section explains how to use [Design Goal & Arch. Design] window to set the processor type and architecture parameters.

Click [Design Goal & Arch. Design] in [ASIP Meister Main Menu] window to open the sub window.

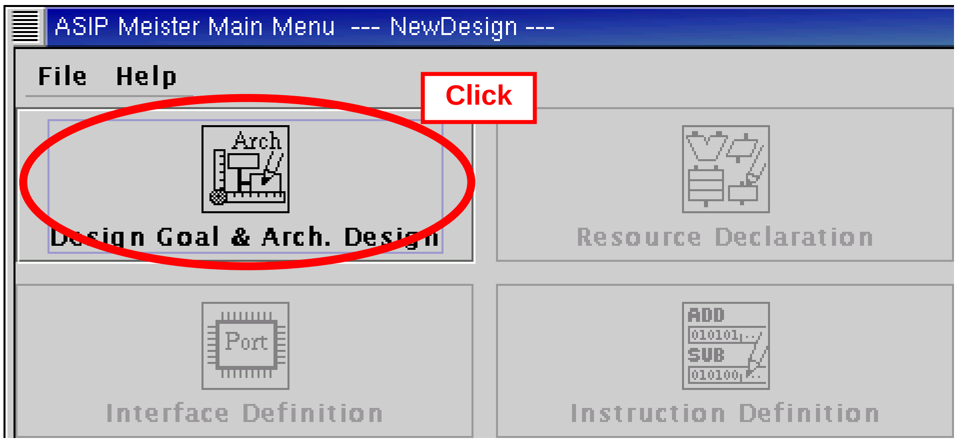


Figure 6: [Design Goal & Arch. Design] in main menu

According to “Processor DLX Integer Specification”, enter parameters to [Design Goal & Arch. Design] window. The default values for some parameters are set to the recommended values in ASIP Meister. Here only change parameters whose values are not set to default.

First, input the following information for managing design data;

- **Project name**

Enter the design project name. Any letters can be used.

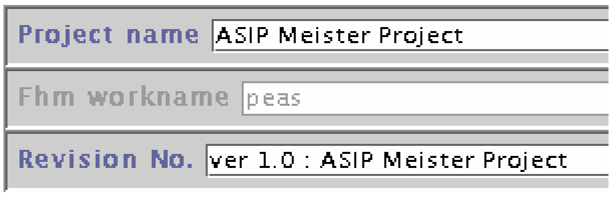
- **Fhm workname**

Current version does not support this parameter.
FHM (Flexible Hardware Model) is a resource data base.

- **Revision No**

You can manage the version of the design data by entering information to this field. Any letters may be used.

Figure 7 shows DLX integer processor setting of management information.



Project name	ASIP Meister Project
Fhm workname	peas
Revision No.	ver 1.0 : ASIP Meister Project

Figure 7: Management information for design data

And then, input the following design goals and the priority of design quality;

- **Goal Area**

Area [gates]

- **Goal Delay**

Maximum delay of internal circuits [ns]

- **Goal Power S**

Static power consumption [uW/MHz]

- **Design Pripríty**

Select the design goal that has the highest priority from [Area], [Performance] and [Power]. According to this value, the typical value in the architecture-level estimation step changes.

Figure 8 shows DLX integer processor setting of design goals and the priority.

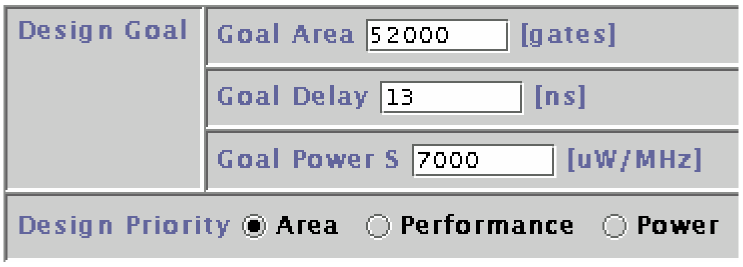

Design Goal
Goal Area 52000 [gates]
Goal Delay 13 [ns]
Goal Power S 7000 [uW/MHz]
Design Priority [✓ Area] Performance Power

Figure 8 Design goal

Next, input the following architecture parameters;

- **CPU Type**

Current version only supports the pipeline processor.

- **Pipeline**

Specify the parameters for the pipeline processor.

- **Number of stages**

Enter the number of pipeline stages.

- **Stage**

Enter the name, attribute and number of cycles for each pipeline stage.

- **Stage name**

Enter the name of stages.

- **attribute**

Select from [fetch], [decode], [register read], [exec], [memory read], [memory write] and [register write]. Multiple attributes could be selected.

- **Delayed branch**

Enable/disable the delayed branch function.

- **Num. Of delayed slot**

Enter the number of delay slots when [Delayed branch] is enabled.

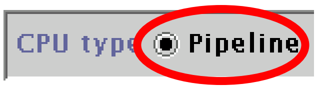
- **Max inst. bit width**

Enter the maximum bit width of instruction.

- **Max data. bit width**

Enter the maximum bit width of data.

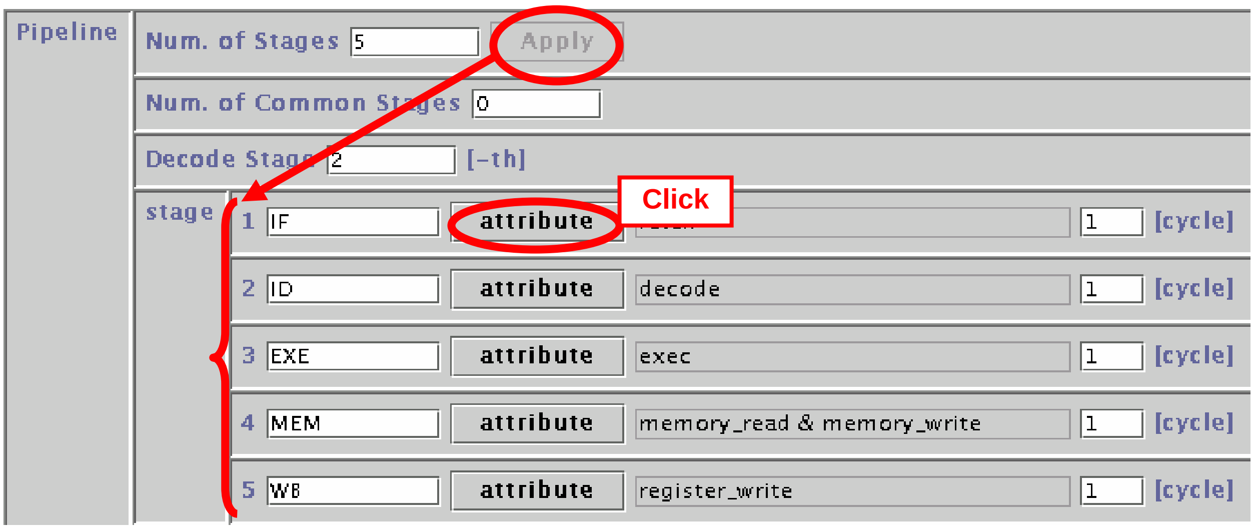
Figure 9 shows DLX integer processor setting of the CPU type.



CPU type [✓ Pipeline]

Figure 9: Architecture parameter of CPU type

Enter "5" to [Num. of Stages] field and click [Apply] button (Figure 10). The [stage] fields will be enabled according to the number entered in [Num. of Stages] field. In this case, five rows will be enabled in [stage] fields.



Pipeline

Num. of Stages 5

Num. of Common Stages 0

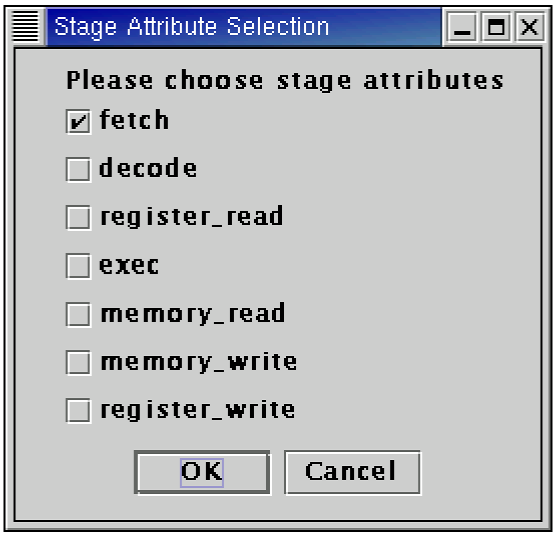
Decode Stage 2 [-th]

stage					
1	IF	attribute	[illegible]	1	[cycle]
2	ID	attribute	decode	1	[cycle]
3	EXE	attribute	exec	1	[cycle]
4	MEM	attribute	memory_read & memory_write	1	[cycle]
5	WB	attribute	register_write	1	[cycle]

Figure 10: Architecture parameter about stages

And then, follow these procedures;

- Enter each stage name. This tutorial uses default names.
- When [attribute] button is clicked, the following window opens to select the stage attribute. Click necessary checkboxes and click [OK] button (Figure 11).


Stage Attribute Selection

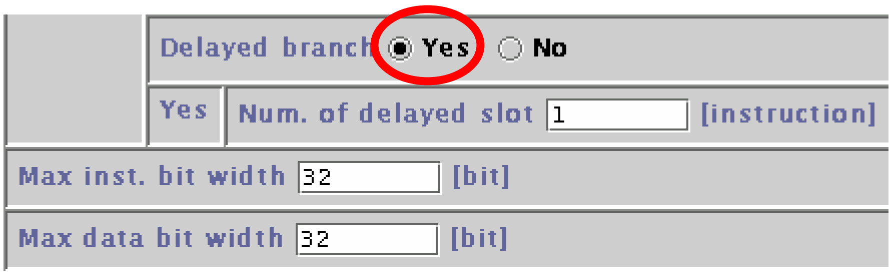
Please choose stage attributes

- [✓] fetch
- [ ] decode
- [ ] register_read
- [ ] exec
- [ ] memory_read
- [ ] memory_write
- [ ] register_write

OK Cancel

Figure 11: [Stage Attribute Selection] window

- Set the number of cycle of pipeline stages to "1" (Figure 12).
- Click [Yes] in [Delayed branch] field to enable [Num. of delayed slot] field. Enter "1" in [Num. of delayed slot] field.
- Enter "32" to the [Max inst. bit width] and [Max data bit width] fields.


Delayed branch [✓ Yes] No

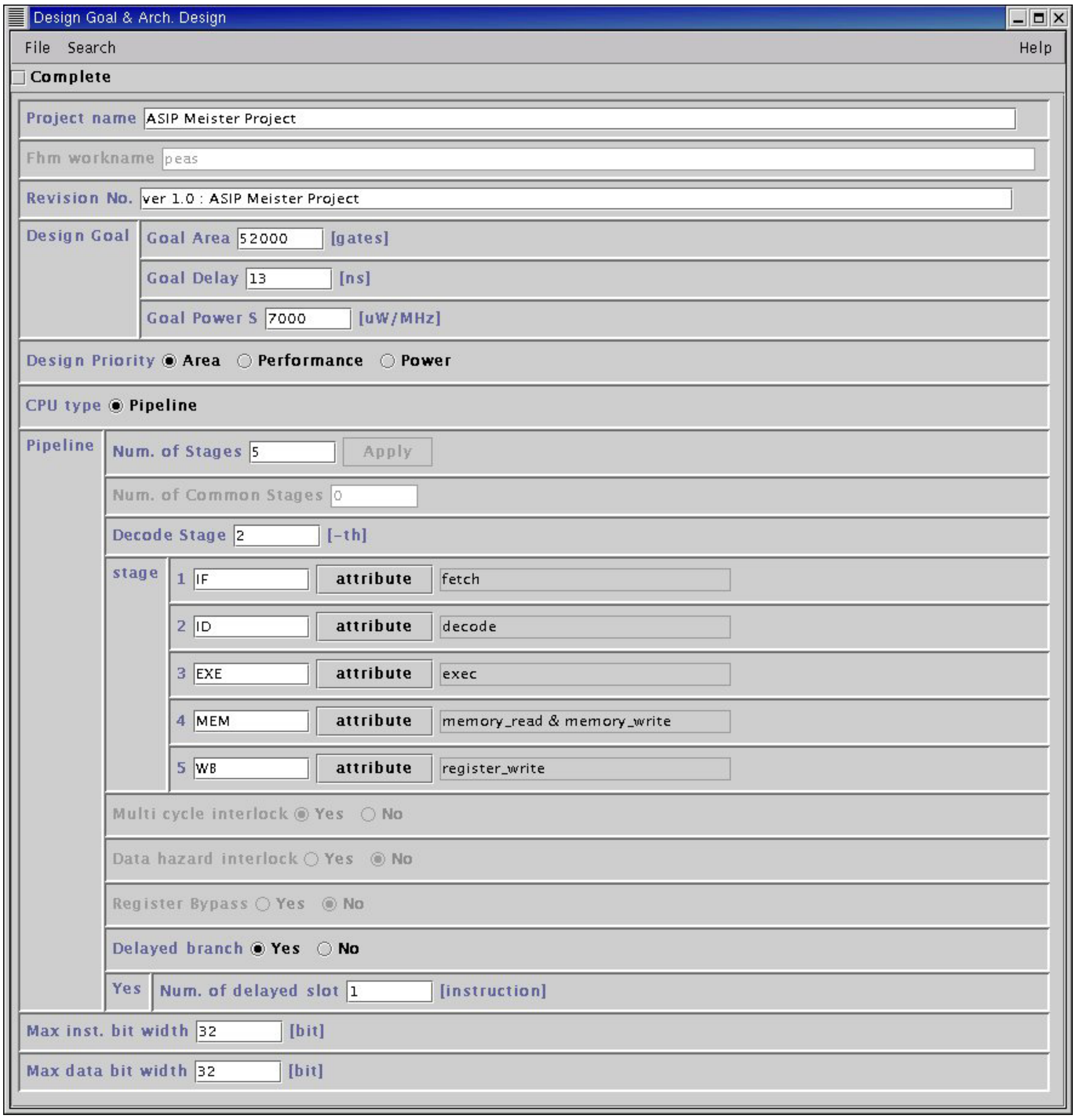
Yes: Num. of delayed slot 1 [instruction]

Max inst. bit width 32 [bit]

Max data bit width 32 [bit]

Figure 12: Architecture parameter about delayed branch and bit width

Now we have set all the necessary parameters for [Design Goal & Arch. Design] window as shown in Figure 13. If you have not set correctly, please refer to "Processor DLX Integer Specification" and check how to set the parameters.



Project name: ASIP Meister Project
Fhm workname: peas
Revision No.: ver 1.0 : ASIP Meister Project
Design Goal: Goal Area 52000 [gates]; Goal Delay 13 [ns]; Goal Power S 7000 [uW/MHz]
Design Priority: [✓ Area] Performance Power
CPU type: [✓ Pipeline]
Pipeline: Num. of Stages 5; Num. of Common Stages 0; Decode Stage 2 [-th]

stage		
1	IF	fetch
2	ID	decode
3	EXE	exec
4	MEM	memory_read & memory_write
5	WB	register_write

Multi cycle interlock: [✓ Yes] No
Data hazard interlock: Yes [✓ No]
Register Bypass: Yes [✓ No]
Delayed branch: [✓ Yes] No
Yes: Num. of delayed slot 1 [instruction]
Max inst. bit width 32 [bit]
Max data bit width 32 [bit]

Figure 13: [Design Goal & Arch. Design] window with DLX integer processor setting

Click [Complete] button on the left top of the window and click [File] > [Close] to return to the main window.

6.2. [Resource Declaration] window: declaring resource modes

ASIP Meister original database, FHM-DB (Flexible Hardware Model Database), is used to provide hardware models. Functions of each hardware resource are described as a function set, and it will be used in the later [Micro Op. description] window.

In [Resource Declaration] step, you can select resources to implement from FHM-DB, set parameters for the resources.

Click [Resource Declaration] in the main window (Figure 14) and [Resource Declaration] window will open (Figure 15).

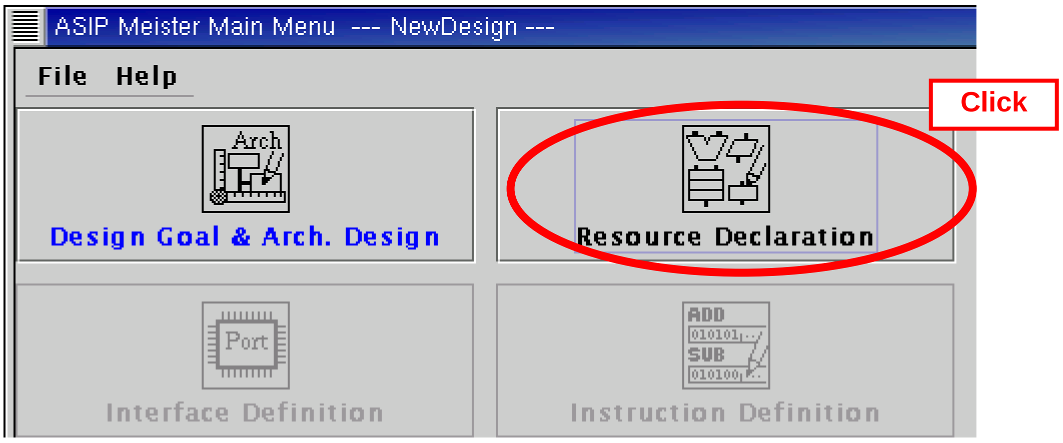


Figure 14: [Resource Declaration] in main menu

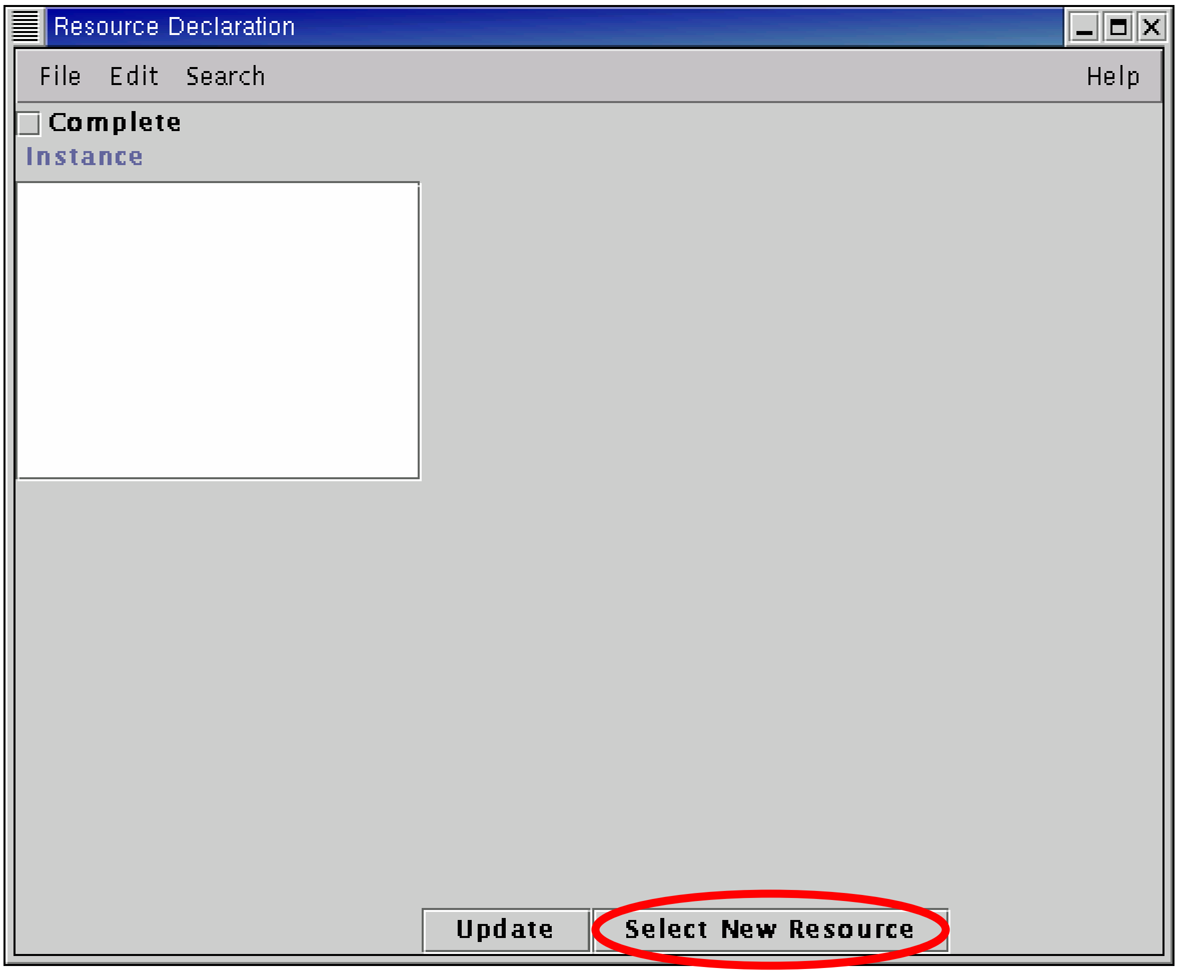


Figure 15: [Resource Declaration] window

To select a resource from FHM-DB to implement, click [Select New Resource] button to open [FHM Browser] window (Figure 16).

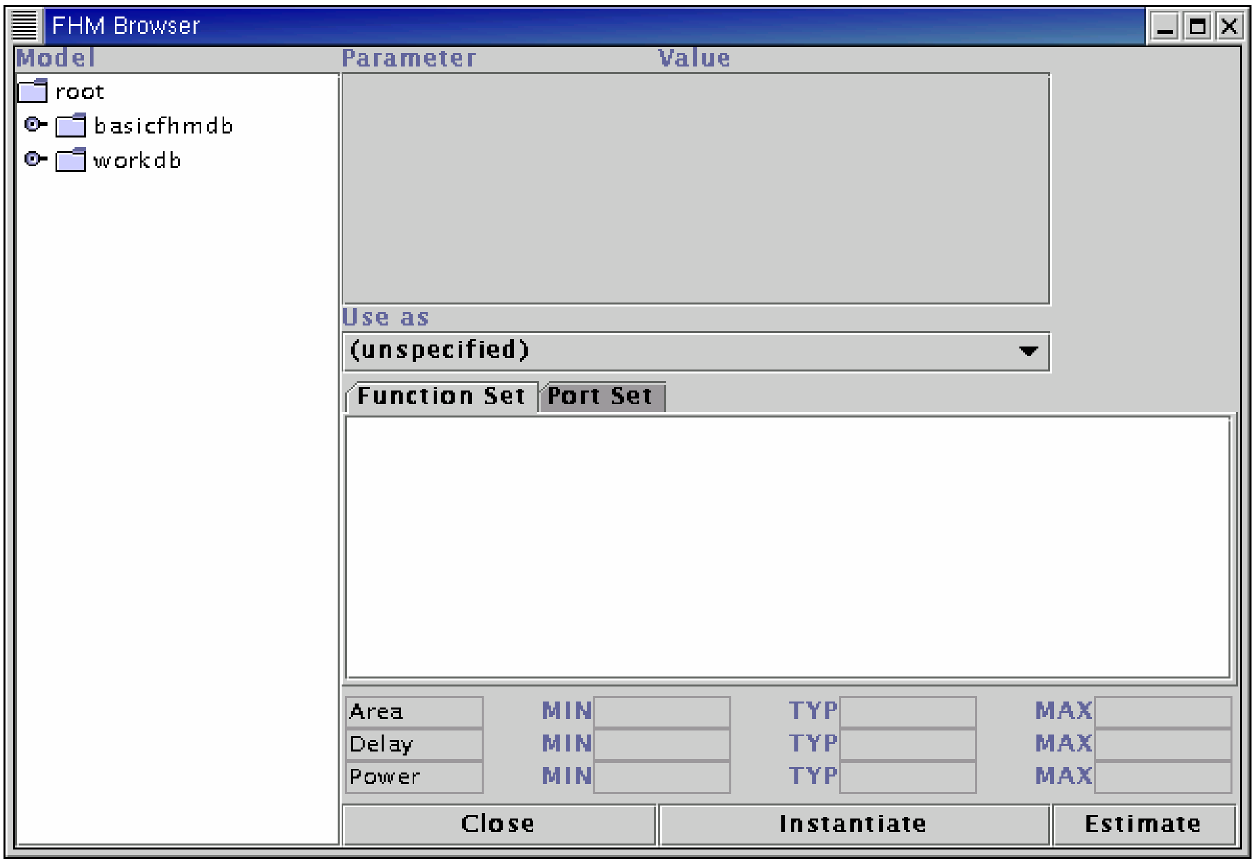


Figure 16: [FHM Browser] window

[Model] list in the left side of the window displays the list of FHMs (Figure 17).

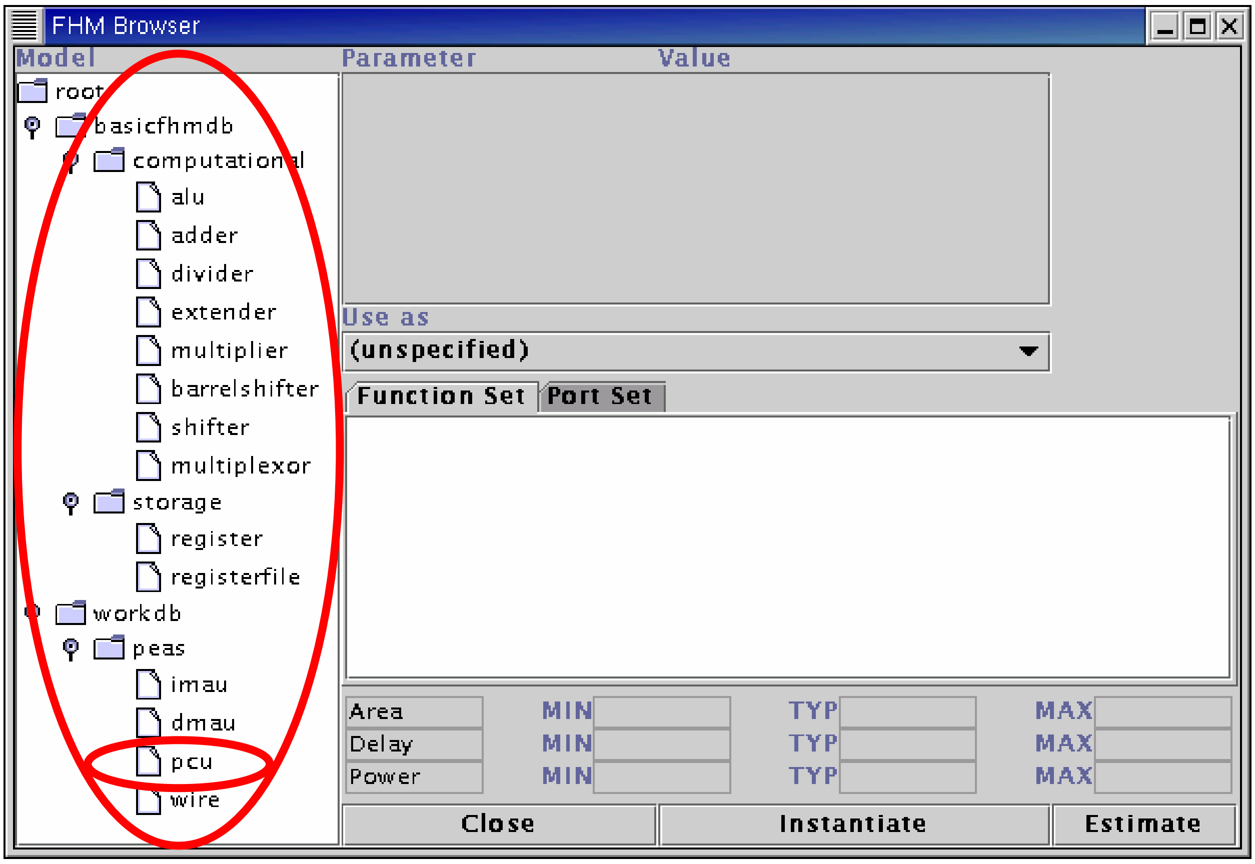
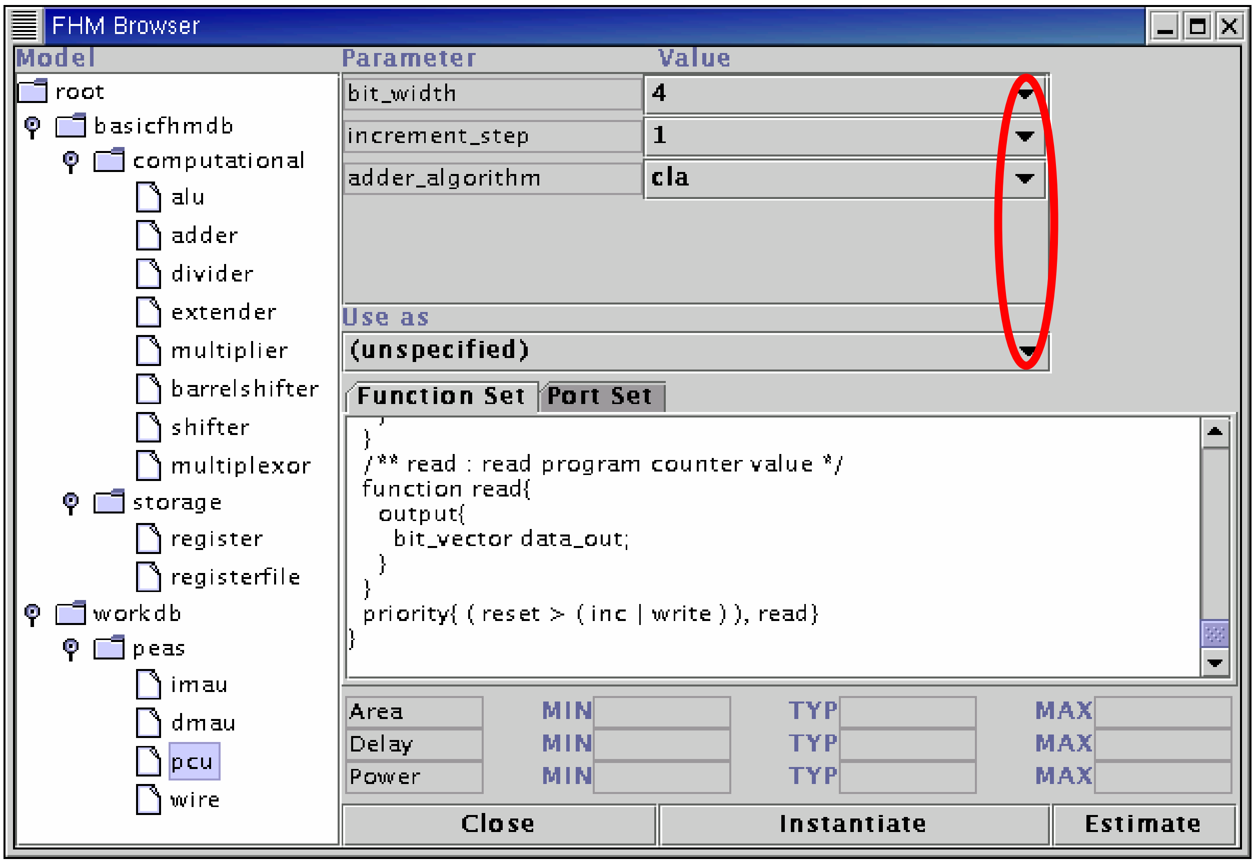


Figure 17: [FHM Browser] with resource tree

When you select a resource, the window will display the parameters of the resource. The required parameters depend on the resource.

For the details of the each resource in FHM-DB, please refer to each [Function Set].

In this tutorial, we use “pcu” model for the program counter. When you click [pcu] in [Model] list, [pcu] setting window opens as Figure 18.



Parameter	Value
bit_width	4
increment_step	1
adder_algorithm	cla

Use as: (unspecified)

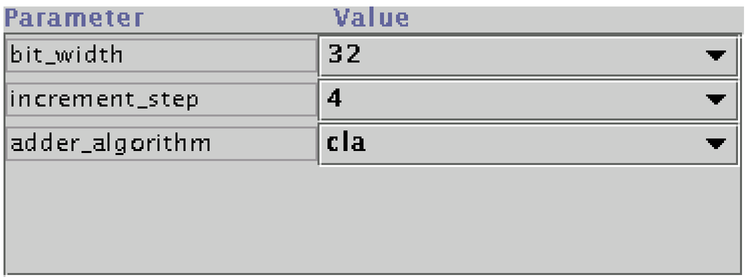
```
 }
 /** read : read program counter value */
 function read{
   output{
     bit_vector data_out;
   }
 }
 priority{ ( reset > ( inc | write ) ), read}
}
```

Figure 18: [FHM Browser] with “pcu” model information

You have to set the following parameters for the program counter (pcu).

- Bit width [bit_width]
- The number of increment steps [Increment_step]
- Algorithm of the increment [adder_algorithm]
- Role of resource [Use as]

To set the parameter, click the pull down menu (triangle mark) and select the value from the menus. Figure 19 shows the value of each parameters of “pcu”.



Parameter	Value
bit_width	32
increment_step	4
adder_algorithm	cla

Figure 19: [bit_width], [incremental_step] and [adder_algorithm] of “pcu”

Select a role from [Use as] pull down menu (Figure 20) when the resource has a specific role in the processor. If the values in the pull down menu do not match the role of the resource, always select [(unspecified)].

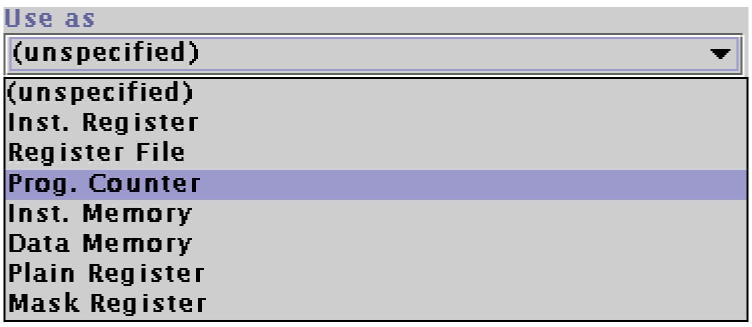


Figure 20: selection for usage of resource

After you have set the parameters for the resource, you can confirm the estimated values for the resource. Click [Estimate] button (Figure 21) and check the estimated values of [Area], [Delay] and [Power] (Figure 22).

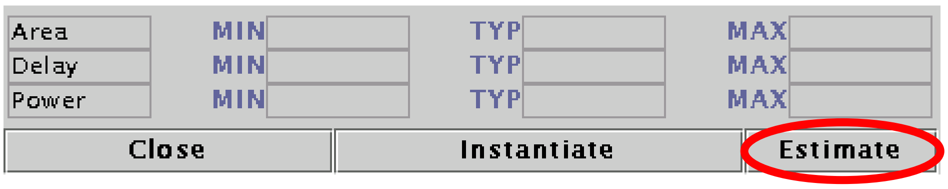
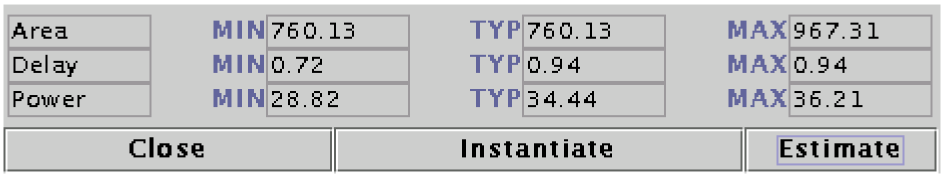


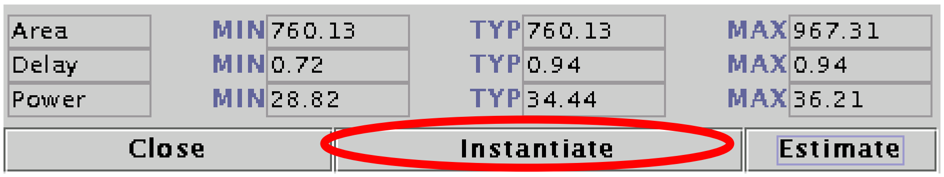
Figure 21: [Estimate] button



Area	MIN 760.13	TYP 760.13	MAX 967.31
Delay	MIN 0.72	TYP 0.94	MAX 0.94
Power	MIN 28.82	TYP 34.44	MAX 36.21

Figure 22: Resource quality estimation

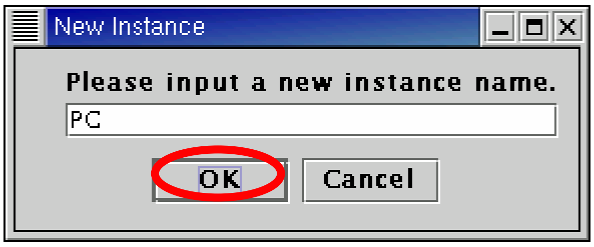
And then, click [Instantiate] button (Figure 23) to register the resource and [New Instance] window will open. In this window, we enter the resource name and register the resource.



Area	MIN 760.13	TYP 760.13	MAX 967.31
Delay	MIN 0.72	TYP 0.94	MAX 0.94
Power	MIN 28.82	TYP 34.44	MAX 36.21

Figure 23: [Instantiate] button

Here, enter “PC” as a name of “pcu” and click [OK] button (Figure 24).

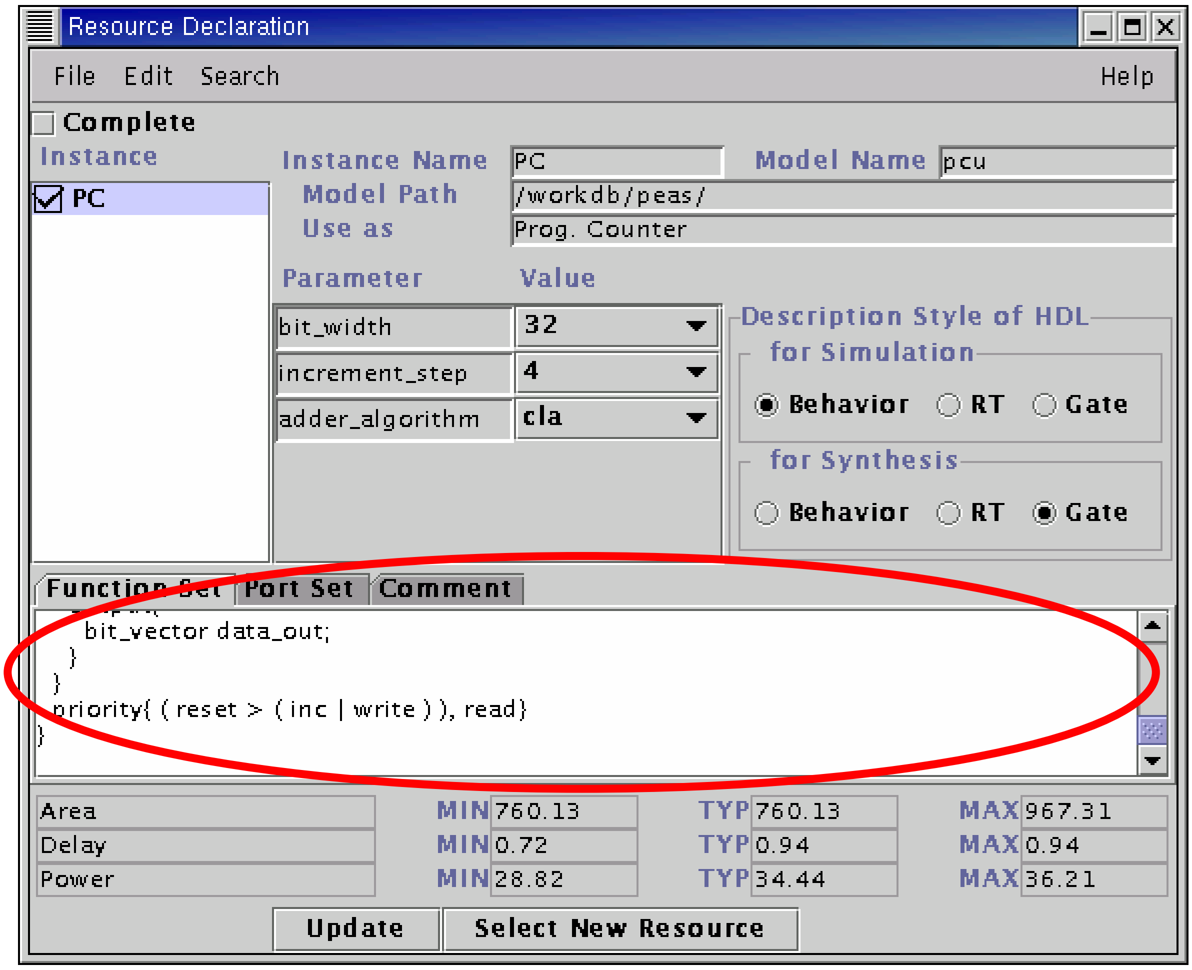


Please input a new instance name.

PC

Figure 24: Instance name input

Maving to [Resource Declaration] window, [Instance] frame in the left side of the window should show [PC] as a resource (Figure 25).



[ ] Complete

Instance: [✓ PC]

Instance Name	PC
Model Name	pcu
Model Path	/workdb/peas/
Use as	Prog. Counter

Parameter	Value
bit_width	32
increment_step	4
adder_algorithm	cla

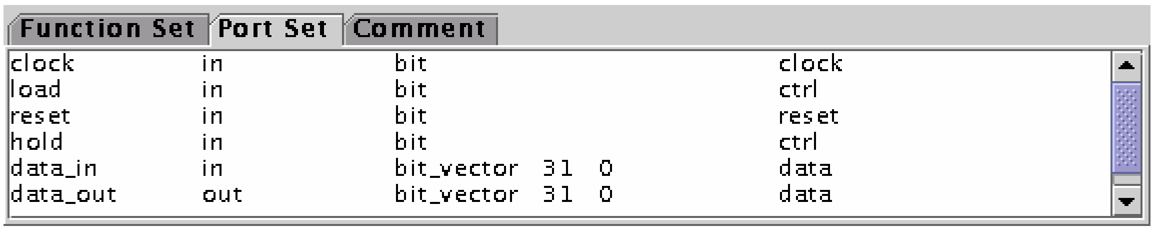
Description Style of HDL
- for Simulation: [✓ Behavior] RT Gate
- for Synthesis: Behavior RT [✓ Gate]

```
    bit_vector data_out;
  }
 }
priority{ ( reset > ( inc | write ) ), read}
}
```

Area	MIN	760.13	TYP	760.13	MAX	967.31
Delay	MIN	0.72	TYP	0.94	MAX	0.94
Power	MIN	28.82	TYP	34.44	MAX	36.21

Figure 25: Function set column

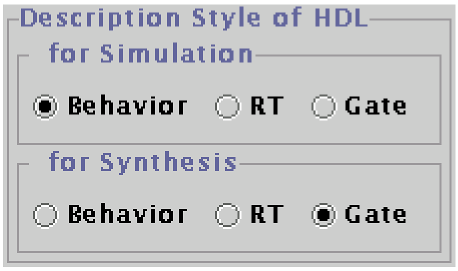
The description in [Function Set] tab will be used in [Micro.Op description] window. Confirm the content of [Function Set] sub window (Figure 25). The port of "PC" is defined as shown in Figure 26. Check port names, signal directions, data type and signal attributes.



clock	in	bit	clock
load	in	bit	ctrl
reset	in	bit	reset
hold	in	bit	ctrl
data_in	in	bit_vector 31 0	data
data_out	out	bit_vector 31 0	data

Figure 26: [Port set] column

You need to set the abstract level for the simulation and logic synthesis. To do this, click [Behavior] button in [for Simulation] group and [Gate] button in [for Synthesis] group. [Description style of HDL] shows the abstraction level of HDL descriptions for the simulation and logic synthesis (Figure 27). In this tutorial, do not change these values.


Description Style of HDL

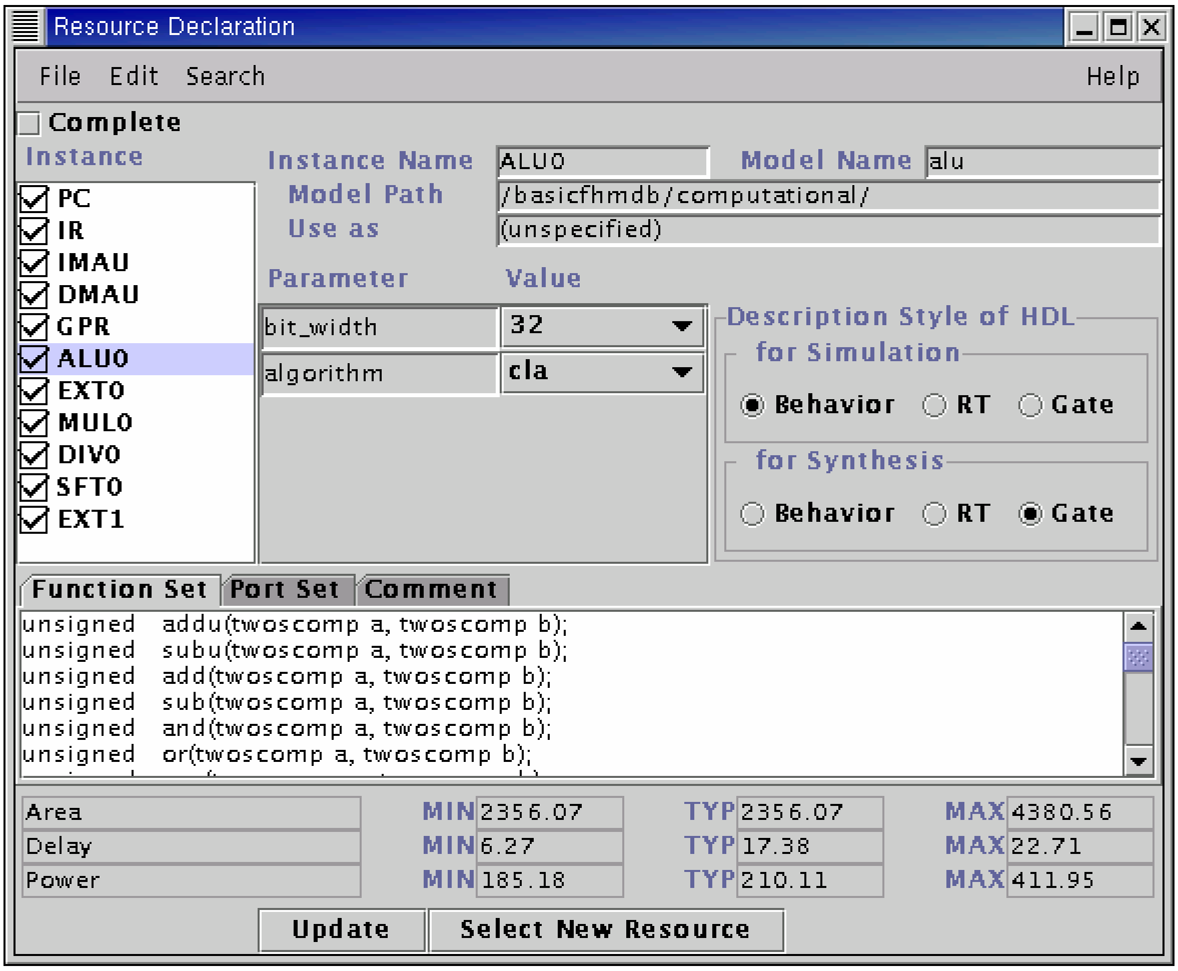
for Simulation: [✓ Behavior] RT Gate

for Synthesis: Behavior RT [✓ Gate]

Figure 27: Selection of HDL description style

Likewise refer to “Resource for use” chapter of “Processor DLX Integer Specification” and declare resources as shows in Figure 28.

You can select the declared resource to implement or not by checking or unchecking the checkbox in [Instance] frame in the left side of the window.



Complete

Instance: PC, IR, IMAU, DMAU, GPR, ALU0, EXT0, MUL0, DIV0, SFT0, EXT1

Instance Name: ALU0
Model Name: alu
Model Path: /basicfhmdb/computational/
Use as: (unspecified)

Parameter	Value
bit_width	32
algorithm	cla

Description Style of HDL
for Simulation: [✓ Behavior] RT Gate
for Synthesis: Behavior RT [✓ Gate]

```
unsigned   addu(twoscomp a, twoscomp b);
unsigned   subu(twoscomp a, twoscomp b);
unsigned   add(twoscomp a, twoscomp b);
unsigned   sub(twoscomp a, twoscomp b);
unsigned   and(twoscomp a, twoscomp b);
unsigned   or(twoscomp a, twoscomp b);
```

	MIN	TYP	MAX
Area	2356.07	2356.07	4380.56
Delay	6.27	17.38	22.71
Power	185.18	210.11	411.95

Figure 28: Resource declaration window with DLX integer processor setting

After declaring required resources, check [Complete] checkbox and click [File] > [Close] to return to the main window.

6.3. [Storage Specification] window: storage specification definition

This section explains how to describe storage specification. In this phase, designers define storages for the processor such as general purpose register, program counter, instruction register and so on. The information of storage is used for instruction definition and behavior definition.

Click [Storage Spec.] button in the main window (Figure 29) and [Storage Spec.] window opens.

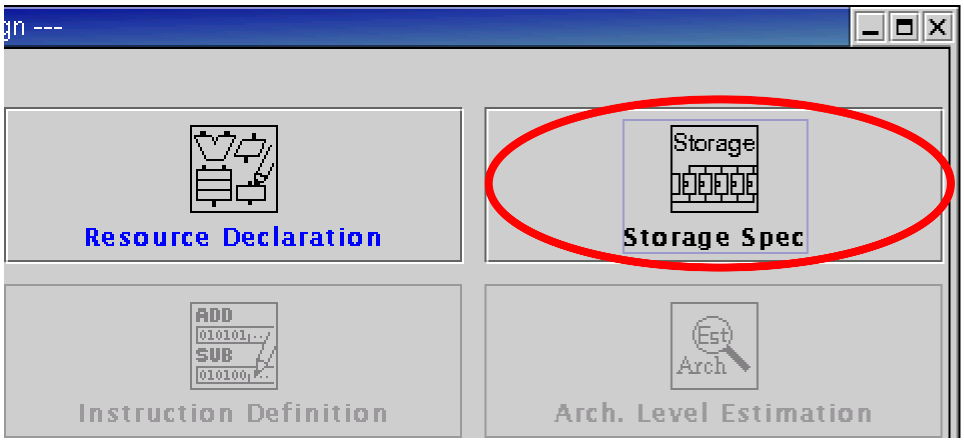
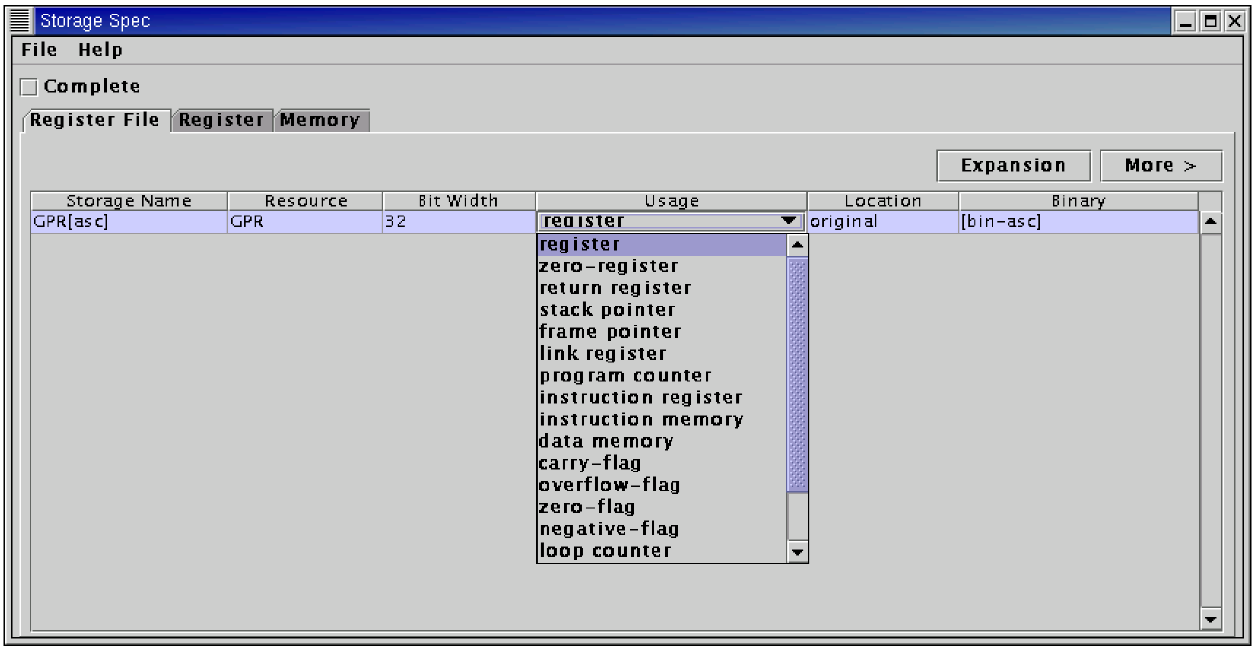


Figure 29: [Storage Spec] in main menu

Storage specification consists of three parts; register file specification, register specification and memory specification. The following sections explain each part.

6.3.1. Register file specification

In this part, register file specification is defined. You can find register file resource when you click storage specification button in the main window, because the resource of register file was already declared in the previous step: “Resource Declaration.” You specify storage name, bit width, usage, location, and binary representation in this part.



Storage Name	Resource	Bit Width	Usage	Location	Binary
GPR[asc]	GPR	32	register	original	[bin-asc]

Figure 30: Register file specification definition in [Storage Spec] window

- **Storage name**

In this field, enter storage name used in assembly code. You can use key words to specify the storage number of each register in a register file.

[asc]:	ascending order
[dsc]:	descending order

- **Resource**

In this field, enter resource instance name. The name must be chosen from the instance names declared in “Resource Declaration” step.

- **Bit width**

Enter the bit width of storage. Enter the same width defined at resource declaration step.

- **Usage**

Enter the usage of storage such as program counter, zero register, stack pointer, and so on.

- **Location**

Enter the location of register. This part is used when you would like to define overlapped register.

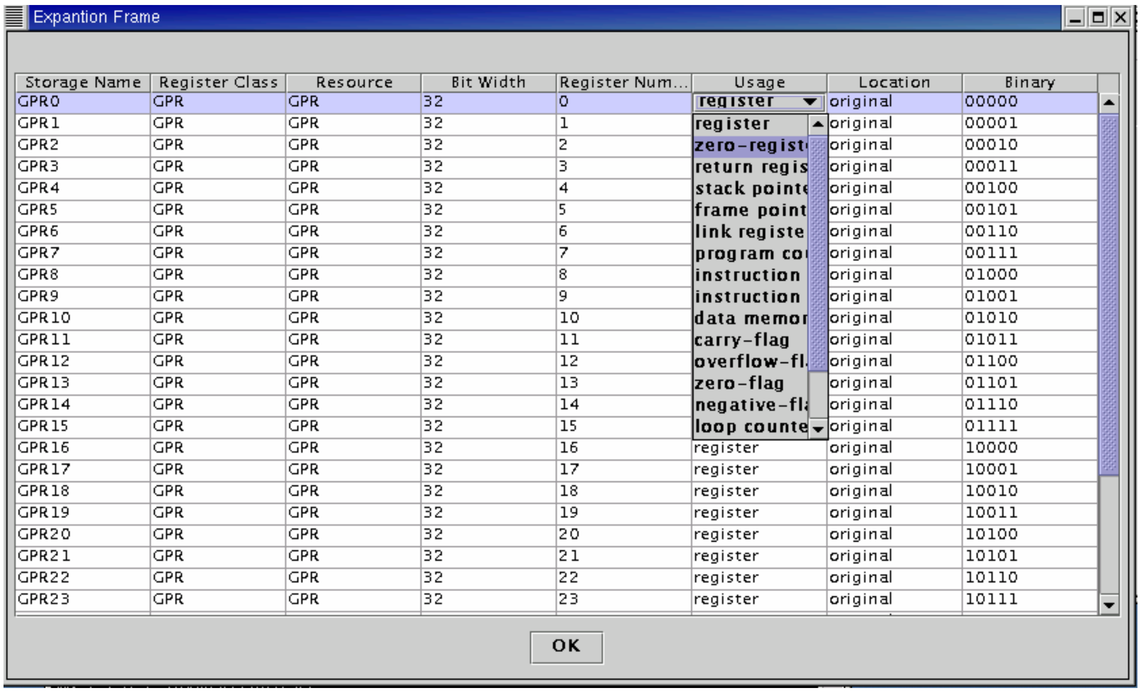
- **Binary**

Enter binary representation of each register in a register file. This information is used in assembler.

In this tutorial, you define a register file.

Storage name:	GPR[asc]
Resource:	GPR
Bit width:	32
Usage:	register
Location:	original
Binary:	[bin-asc]

When you finish describing the information, click [Expansion] button and [Expansion Frame] opens (Figure 31).


Expantion Frame

Storage Name	Register Class	Resource	Bit Width	Register Num...	Usage	Location	Binary
GPR0	GPR	GPR	32	0	register	original	00000
GPR1	GPR	GPR	32	1	register	original	00001
GPR2	GPR	GPR	32	2	zero-regist	original	00010
GPR3	GPR	GPR	32	3	return regis	original	00011
GPR4	GPR	GPR	32	4	stack pointe	original	00100
GPR5	GPR	GPR	32	5	frame point	original	00101
GPR6	GPR	GPR	32	6	link registe	original	00110
GPR7	GPR	GPR	32	7	program co	original	00111
GPR8	GPR	GPR	32	8	instruction	original	01000
GPR9	GPR	GPR	32	9	instruction	original	01001
GPR10	GPR	GPR	32	10	data memor	original	01010
GPR11	GPR	GPR	32	11	carry-flag	original	01011
GPR12	GPR	GPR	32	12	overflow-fl	original	01100
GPR13	GPR	GPR	32	13	zero-flag	original	01101
GPR14	GPR	GPR	32	14	negative-fl	original	01110
GPR15	GPR	GPR	32	15	loop counte	original	01111
GPR16	GPR	GPR	32	16	register	original	10000
GPR17	GPR	GPR	32	17	register	original	10001
GPR18	GPR	GPR	32	18	register	original	10010
GPR19	GPR	GPR	32	19	register	original	10011
GPR20	GPR	GPR	32	20	register	original	10100
GPR21	GPR	GPR	32	21	register	original	10101
GPR22	GPR	GPR	32	22	register	original	10110
GPR23	GPR	GPR	32	23	register	original	10111

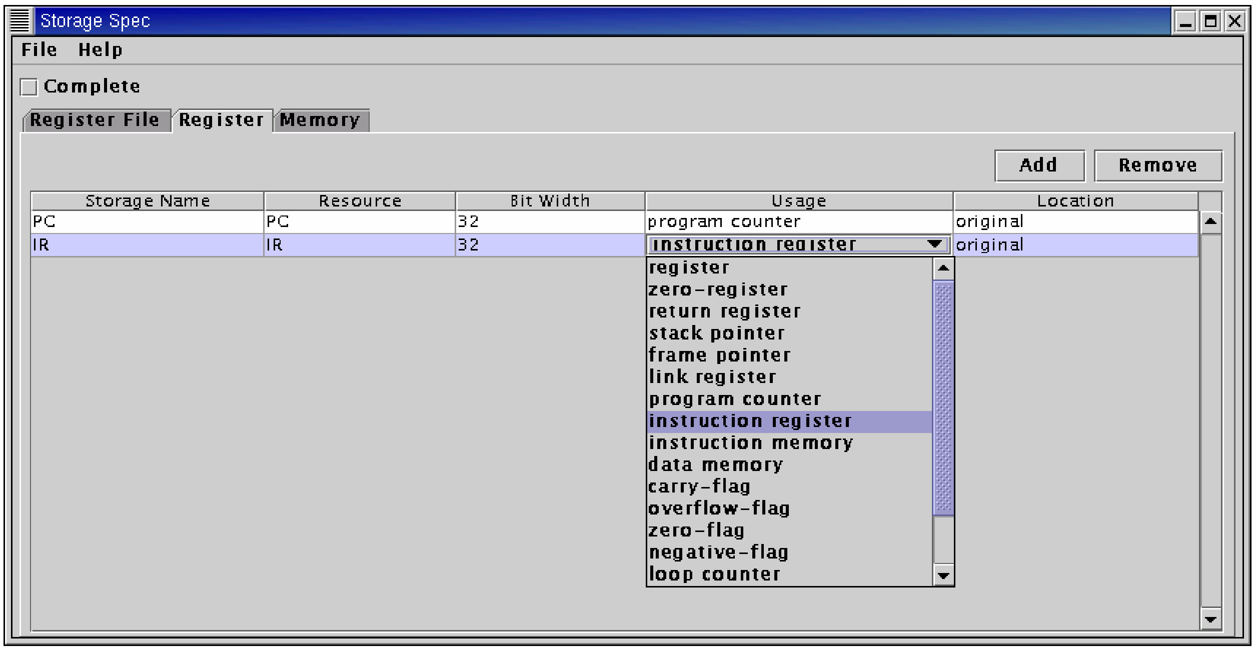
Figure 31: Detail information of register file specification

Each register can be seen in expansion frame window. You can specify usage of each register in this part. In this tutorial, usage of registers will be changed as follow;

● GPR0:	zero register
● GPR28:	stack pointer
● GPR29:	frame pointer
● GPR30:	link register
● GPR31:	return register

6.3.2. Register specification

When you click [Register] tab, you can find register definition window (Figure 32). In this part, you can specify the following information; storage name, resource, bit width, usage and location. This information is the same as register file definition.



Storage Name	Resource	Bit Width	Usage	Location
PC	PC	32	program counter	original
IR	IR	32	instruction register	original

Figure 32: Detail information of register specification

You will describe the information for program counter and instruction register in this tutorial. For the program counter, use the following specification;

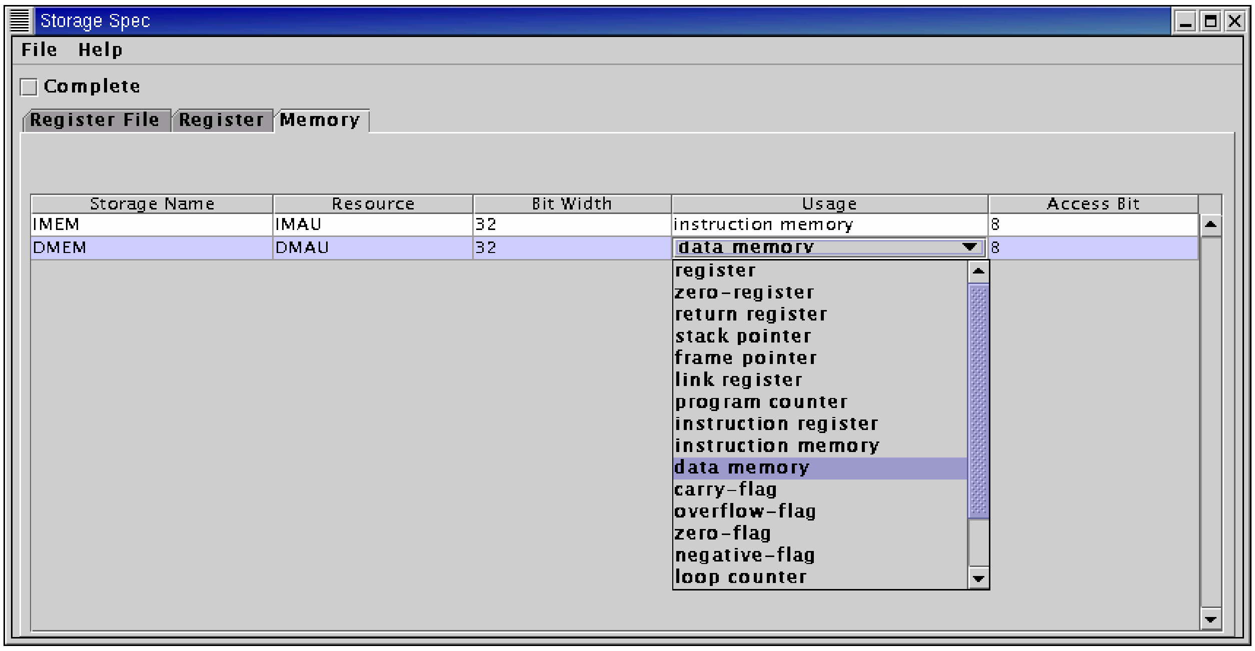
● Storage name:	PC
● Resource:	PC
● Bit width:	32
● Usage:	program counter
● Location:	original

For the instruction register, use the following specification;

● Storage name:	IR
● Resource:	IR
● Bit width:	32
● Usage:	instruction register
● Location:	original

6.3.3. Memory specification

When you click [Memory] tab, you can find memory specification part (Figure 33). In this part, you can specify the following information; storage name, resource, bit width, usage and access bit. Access bit indicates minimum bit width when a processor accesses to the memory.



Storage Name	Resource	Bit Width	Usage	Access Bit
IMEM	IMAU	32	instruction memory	8
DMEM	DMAU	32	data memory	8

Figure 33: Detail information of memory specification

In this tutorial, we define instruction memory and data memory.
For the instruction memory, use the following specification;

● Storage name:	IMEM
● Resource:	IMAU
● Bit width:	32
● Usage:	instruction memory
● Access bit:	32

For the data memory, use the following specification;

● Storage name:	DMEM
● Resource:	DMAU
● Bit width:	32
● Usage:	data memory
● Access bit:	8

When you finish describing the storage specification, click the complete button and close the [Storage Spec.] window.

6.4. [Interface Definition] window: defining I/O interfaces

This section explains how to set the entity name of the processor and I/O ports information.
In ASIP Meister, only IMAU (Instruction Memory Access Unit) and DMAU (Data Memory Access Unit) can access to external ports. And key words are used to specify attributes of the external ports that are accessed from the each unit. There are several types of ports you can declare;

1) External access ports for IMAU and DMAU (one unit available for IMAU and DMAU)
2) Clock and Reset ports
3) Ports for interrupts

Click [Interface Definition] (Figure 34) and [Interface Definition] window opens and default ports are displayed (Figure 35).

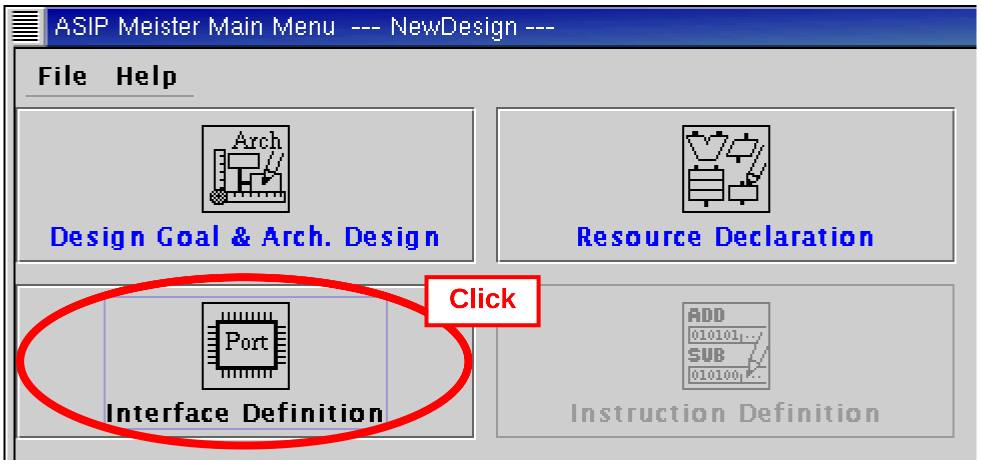
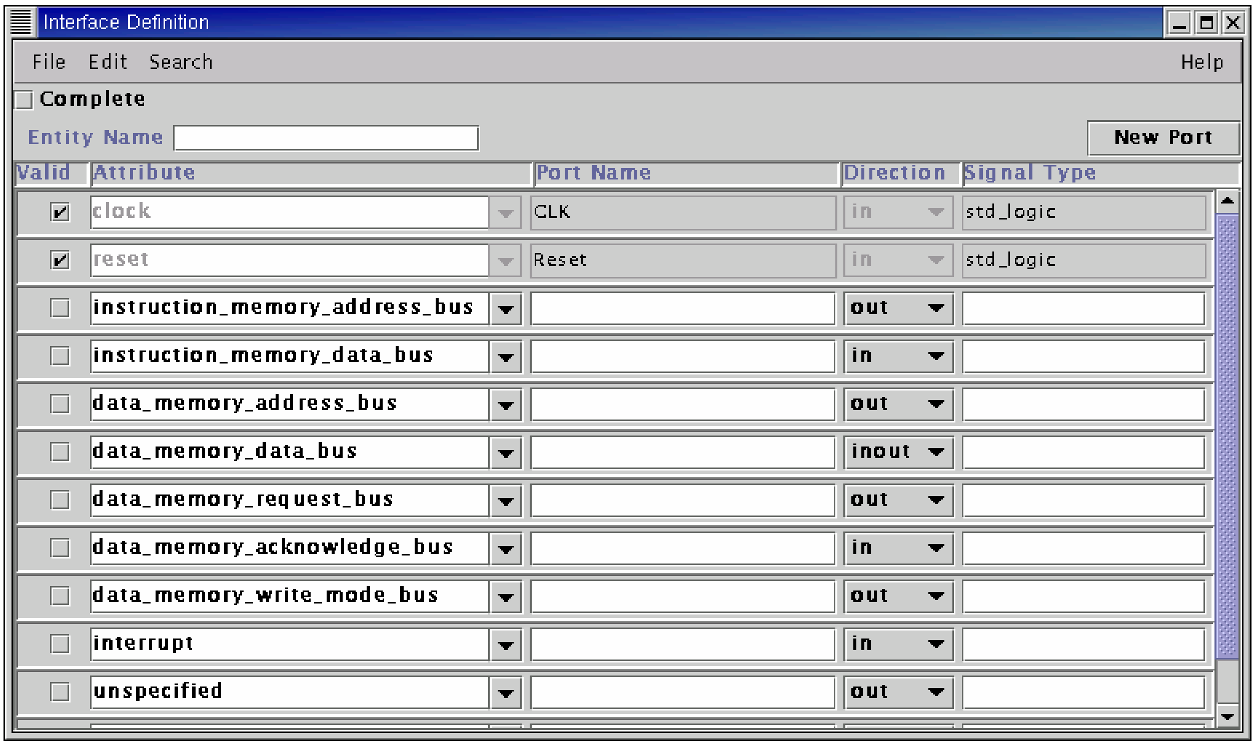


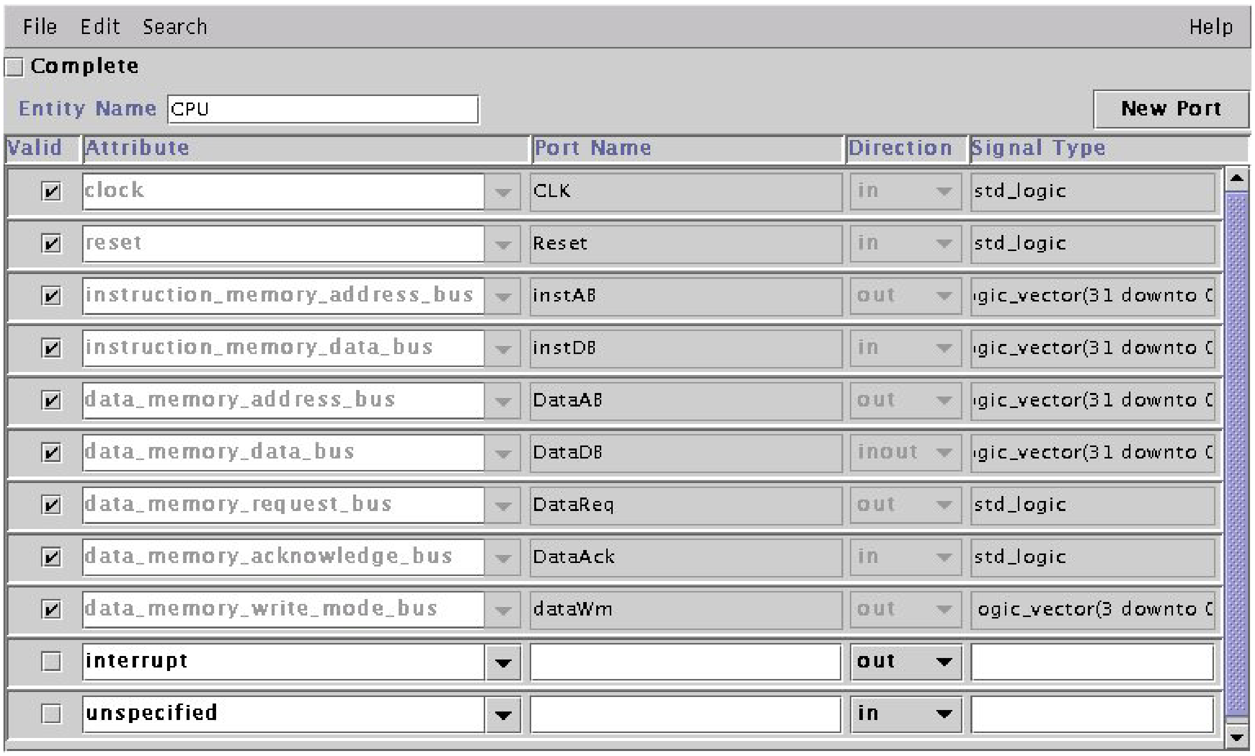
Figure 34: [Interface Definition] in main menu



Valid	Attribute	Port Name	Direction	Signal Type
✓	clock	CLK	in	std_logic
✓	reset	Reset	in	std_logic
	instruction_memory_address_bus		out	
	instruction_memory_data_bus		in	
	data_memory_address_bus		out	
	data_memory_data_bus		inout	
	data_memory_request_bus		out	
	data_memory_acknowledge_bus		in	
	data_memory_write_mode_bus		out	
	interrupt		in	
	unspecified		out	

Figure 35: [Interface Definition] window

To declare a new port, click [New Port] button. This adds a new row to the window as shown in Figure 35. In this tutorial, refer to “I/O port” chapter in “Processor DLX Integer Specification” and enter the data to the window as in the next figure.



Complete

Entity Name: CPU

Valid	Attribute	Port Name	Direction	Signal Type
✓	clock	CLK	in	std_logic
✓	reset	Reset	in	std_logic
✓	instruction_memory_address_bus	instAB	out	ıgic_vector(31 downto C
✓	instruction_memory_data_bus	instDB	in	ıgic_vector(31 downto C
✓	data_memory_address_bus	DataAB	out	ıgic_vector(31 downto C
✓	data_memory_data_bus	DataDB	inout	ıgic_vector(31 downto C
✓	data_memory_request_bus	DataReq	out	std_logic
✓	data_memory_acknowledge_bus	DataAck	in	std_logic
✓	data_memory_write_mode_bus	dataWm	out	ogic_vector(3 downto C
	interrupt		out	
	unspecified		in	

Figure 36: [Interface Definition] window with DLX integer processor setting

- **[Entity Name]: Entity name of the processor**

The name entered in this field is used as an entity name of the processor in the VHDL description. The file name of VHDL description of the processors becomes the name with the extension ".vhd". The name should conform to the VHDL naming convention.

- **[Port name]: Port name**

The name entered in this field is used as an external port name for processor in the VHDL description. The name should conform to the VHDL naming convention.

- **[Direction]: Direction of the signal**

Select [in] for input, [out] for output, and [inout] for both (Figure 37).

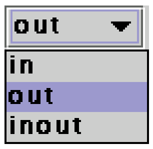


Figure 37: Port selection

- **[Signal Type]: Signal type of VHDL**

The type declared in "IEEE.std_logic_1164.all package" is applicable.

- **[Attribute]: Attribute keyword**

Table 1 describes the attribute keywords and their functions.

Table 1: Attribute keywords and functions

Attribute keyword	Function
clock	Connecting to Clock
reset	Connecting to Reset
instruction_memory_address_bus	Connecting to address bus of IMAU
instruction_memory_data_bus	Connecting to data bus of IMAU
data_memory_address_bus	Connecting to address bus of DMAU
data_memory_data_bus	Connecting to data bus of DMAU
data_memory_request_bus	Connecting to request bus of DMAU
data_memory_acknowledge_bus	Connecting to acknowledge bus of DMAU
data_memory_write_mode_bus	Connecting to write mode bus of DMAU

After setting [Interface Definition] window, check [Complete] checkbox and click [File] > [Close] to return to the main window.

6.5. [Instruction Definition] window: defining instruction type and instruction set

This section explains the declaration process of instruction types and instructions in ASIP Meister. First, you declare an instruction type and then specify how to separate fields. Next, declare an instruction that belongs to each instruction type and assign the unique value (opecode) to the each instruction.

Define instruction types "R_R", "R_I", "B", and "J" according to "Instruction Set" chapter in "Processor DLX Integer Specification".

Click [Instruction Definition] in the main window (Figure 38) and [Instruction Definition] window opens (Figure 39).

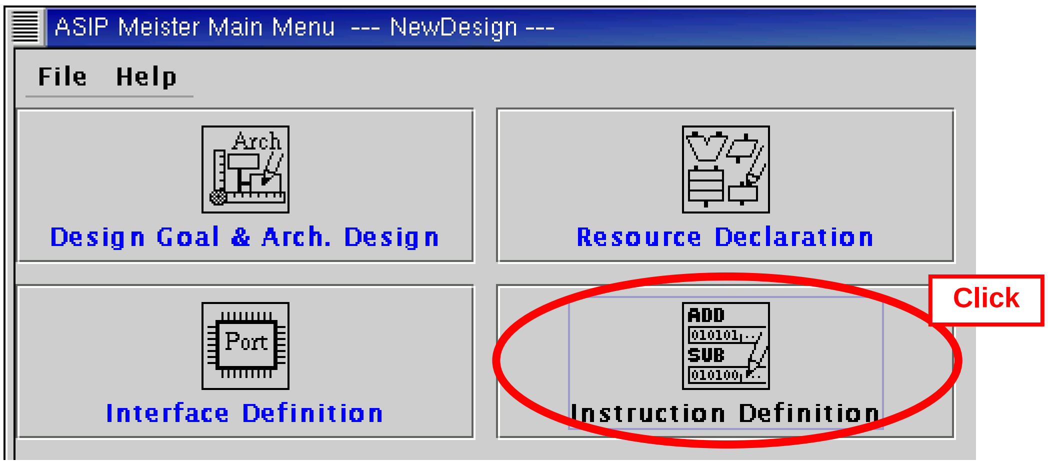


Figure 38: [Instruction Definition] in main menu

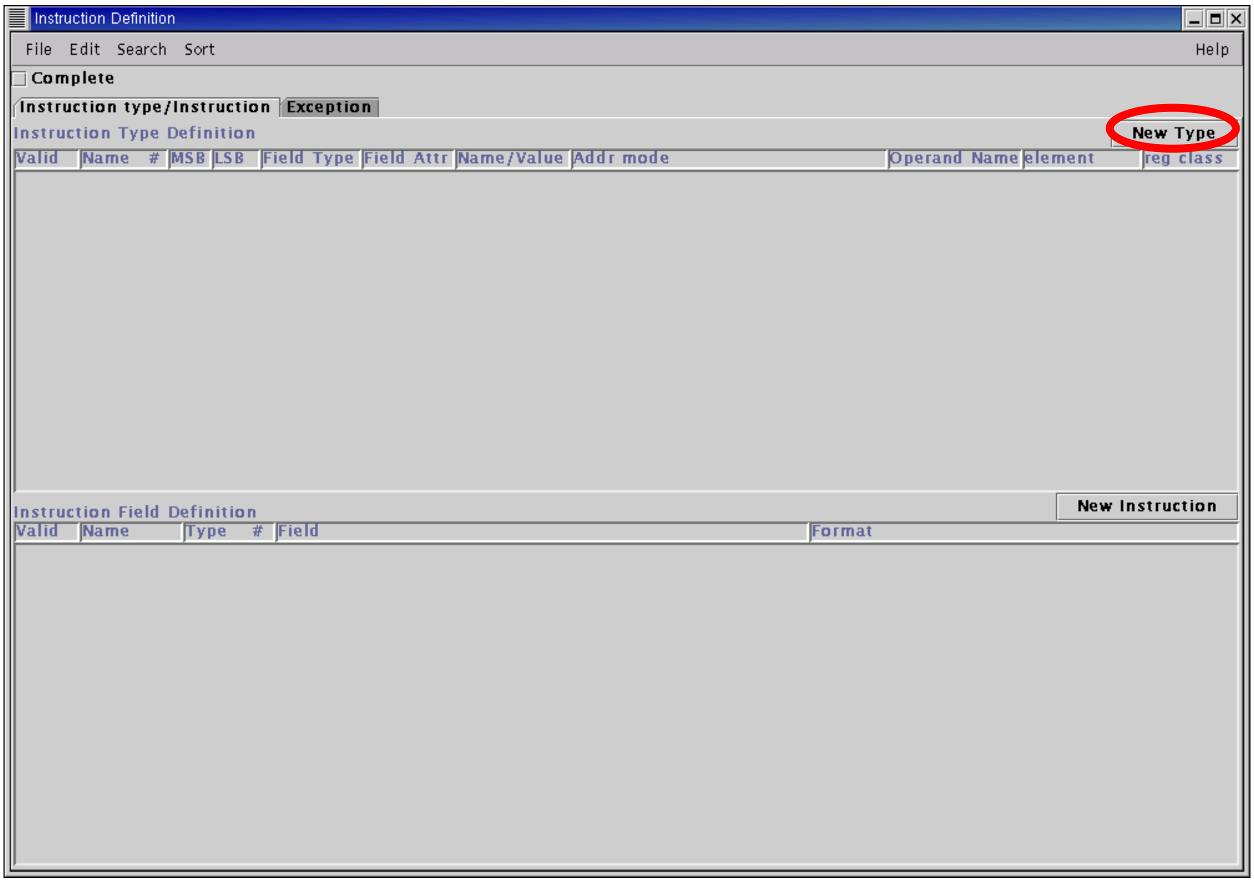
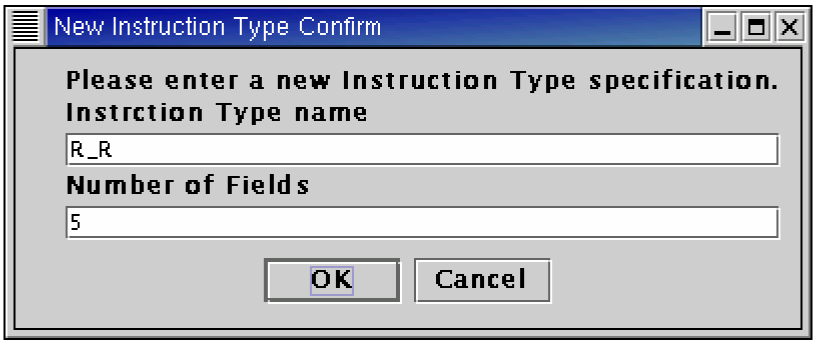

Instruction Definition
File Edit Search Sort Help
Complete
Instruction type/Instruction
Exception
Instruction Type Definition
New Type
Valid Name # MSB LSB Field Type Field Attr Name/Value Addr mode Operand Name element reg class
Instruction Field Definition
New Instruction
Valid Name Type # Field Format

Figure 39: [Instruction Definition] window

6.5.1. Define the instruction type

In this tutorial, we will try to declare "R_R" instruction type.

To declare a new instruction type, click [New Type] button to open [New Instruction Type Confirm] window (Figure 40). This window has two fields to enter the name of a new instruction type and number of the fields for it.



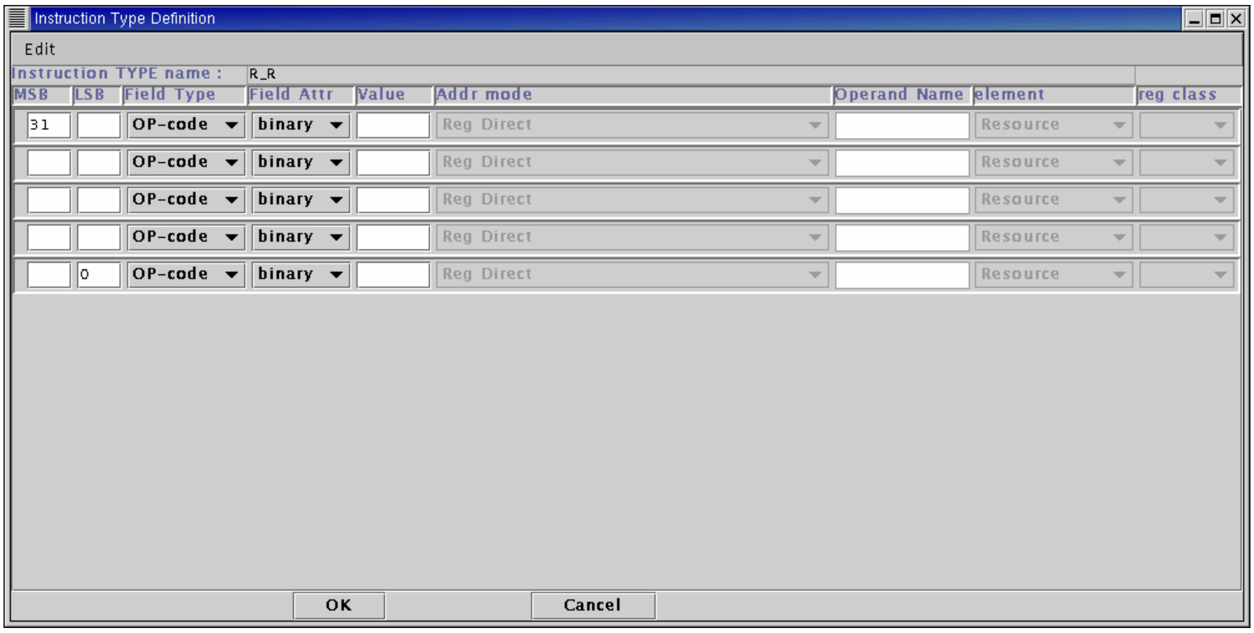
Please enter a new Instruction Type specification.

Instrction Type name: R_R

Number of Fields: 5

Figure 40: [New Instruction Type Confirm] window

Enter "R_R" and "5" to each field and click [OK] button. [Instruction Type Declaration Window] which has five rows opens as shown in Figure 41. With this window, you can declare bit fields of the instruction type and unique binaries for the instruction type as follows;


Instruction Type Definition

Edit

Instruction TYPE name : R_R

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31		OP-code	binary		Reg Direct		Resource	
		OP-code	binary		Reg Direct		Resource	
		OP-code	binary		Reg Direct		Resource	
		OP-code	binary		Reg Direct		Resource	
	0	OP-code	binary		Reg Direct		Resource	

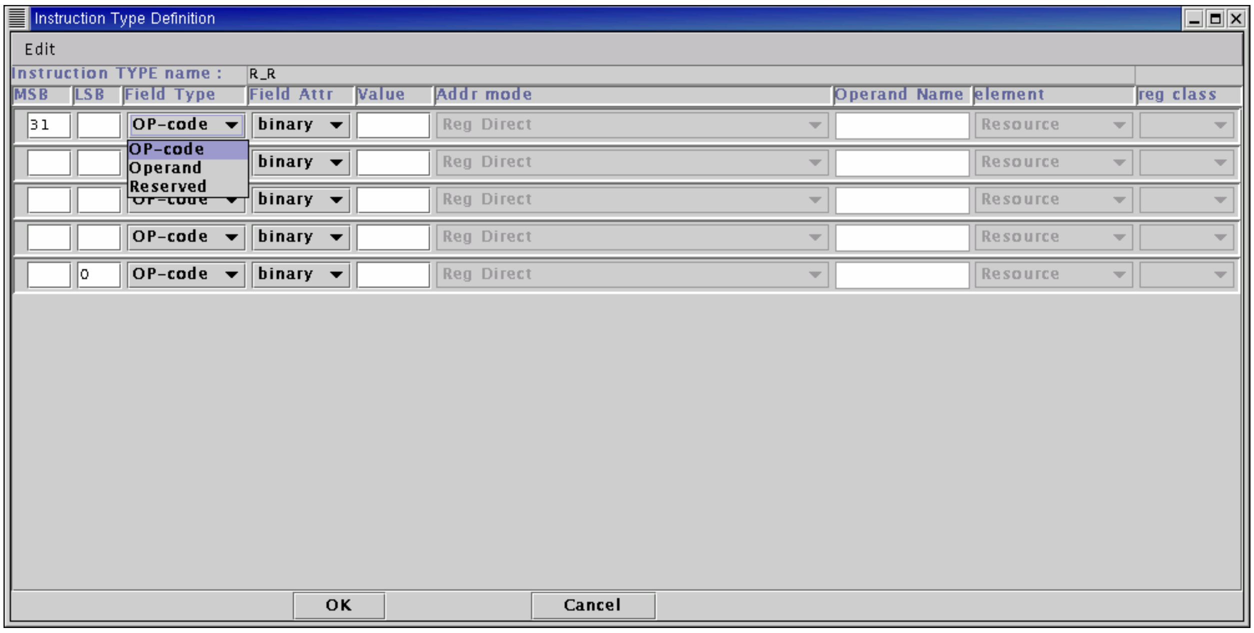
Figure 41: [Instruction Type Definition] window

- **MSB/LSB**

Enter the value of most significant bit ([MSB]) and least significant bit ([LSB]).

- **Field Type**

Click the triangle and select a value from the pull down menu (Figure 42).


Instruction Type Definition

Edit

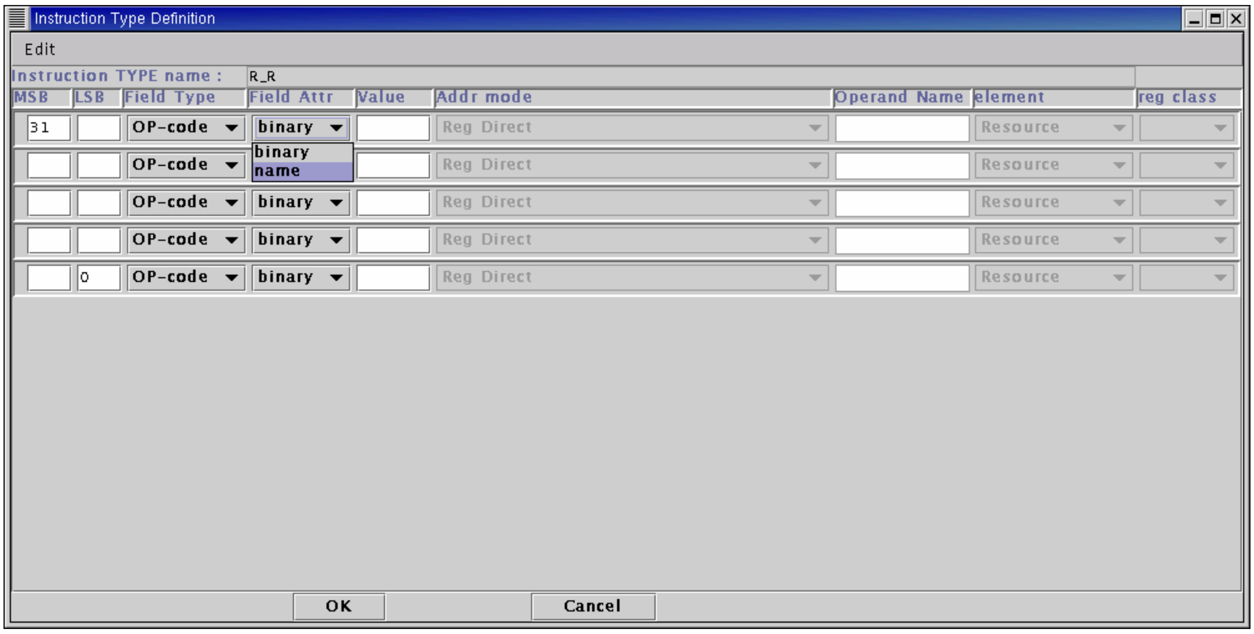
Instruction TYPE name : R_R

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31		OP-code	binary		Reg Direct		Resource	
			binary		Reg Direct		Resource	
			binary		Reg Direct		Resource	
		OP-code	binary		Reg Direct		Resource	
	0	OP-code	binary		Reg Direct		Resource	

Figure 42: [Field Type] pull down menu

- **Field Attr**

Click the triangle and select an attribute from the pull down menu (Figure 43). Depending on the option selected in [Field Type], pull down menu shows different attributes (Table 2).



Instruction TYPE name : R_R

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31		OP-code	binary		Reg Direct		Resource	
		OP-code	binary / name		Reg Direct		Resource	
		OP-code	binary		Reg Direct		Resource	
		OP-code	binary		Reg Direct		Resource	
	0	OP-code	binary		Reg Direct		Resource	

Figure 43: [Field Attr] pull down menu

Table 2: Field attribute for each field type

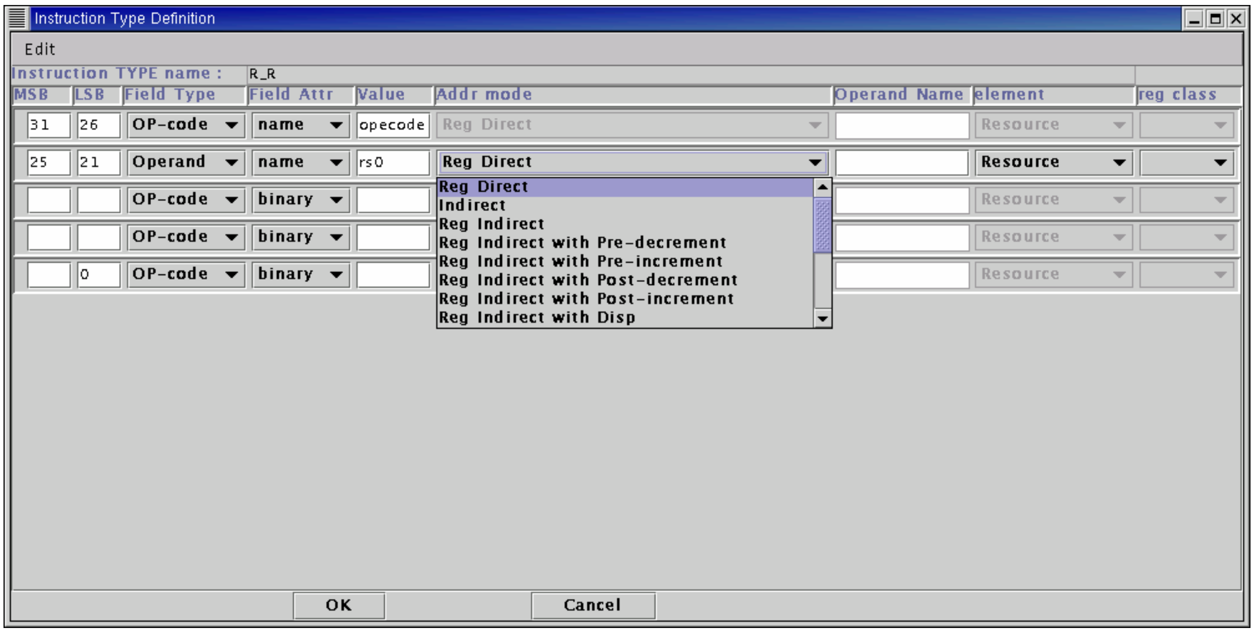
Field Type	Available attributes in [Field Attr]	Description
OP-code	binary	The value is fixed binary pattern for all instructions of the same instruction type.
	name	The value is unfixed and each instruction of the same instruction type must have a different value. (The exact value is defined later.)
Operand	name	The value is fixed name for all instructions of the same instruction type.

- **Value**

Enter binary value or name to the each field.

- **Addr mode**

[Addr mode] means addressing mode of each operand (Figure 44). ADD instruction has three operands: rs0, rs1 and rd. Addressing mode of each operand is register direct mode. In this part, you will select "REG Direct" for addressing mode from the pull down menu. Compiler generator and meta assembler use this information.



Instruction Type Definition

Edit

Instruction TYPE name : R_R

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	26	OP-code	name	opecode	Reg Direct		Resource	
25	21	Operand	name	rs0	Reg Direct		Resource	
		OP-code	binary				Resource	
		OP-code	binary				Resource	
	0	OP-code	binary				Resource	

Reg Direct
Indirect
Reg Indirect
Reg Indirect with Pre-decrement
Reg Indirect with Pre-increment
Reg Indirect with Post-decrement
Reg Indirect with Post-increment
Reg Indirect with Disp
OK
Cancel

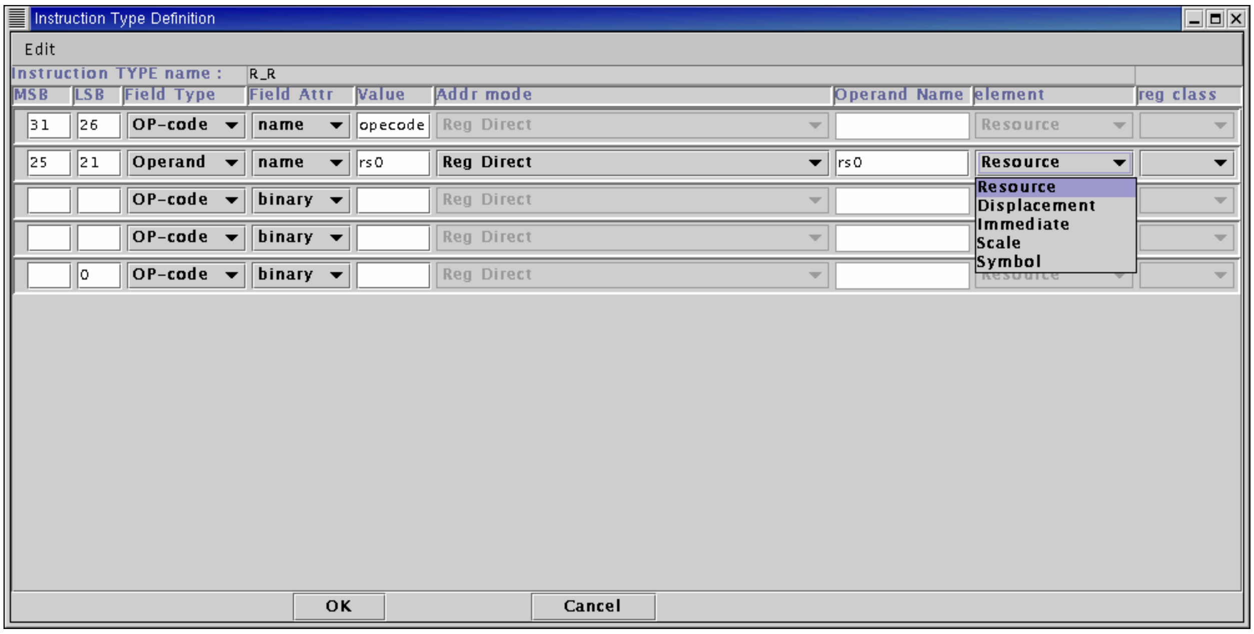
Figure 44: [Addr mode] pull down menu

- **Operand Name**

In this part, operand name for assembly language is specified. Operand name can be used at instruction format declaration. The format is used in assembler and compiler.

- **element**

Some addressing modes consist of several bit fields. In this part, you specify the meaning of each element. You can select from the pull-down menu (Figure 45). In "R_R" instruction type, you will select "Resource".



Instruction Type Definition

Edit

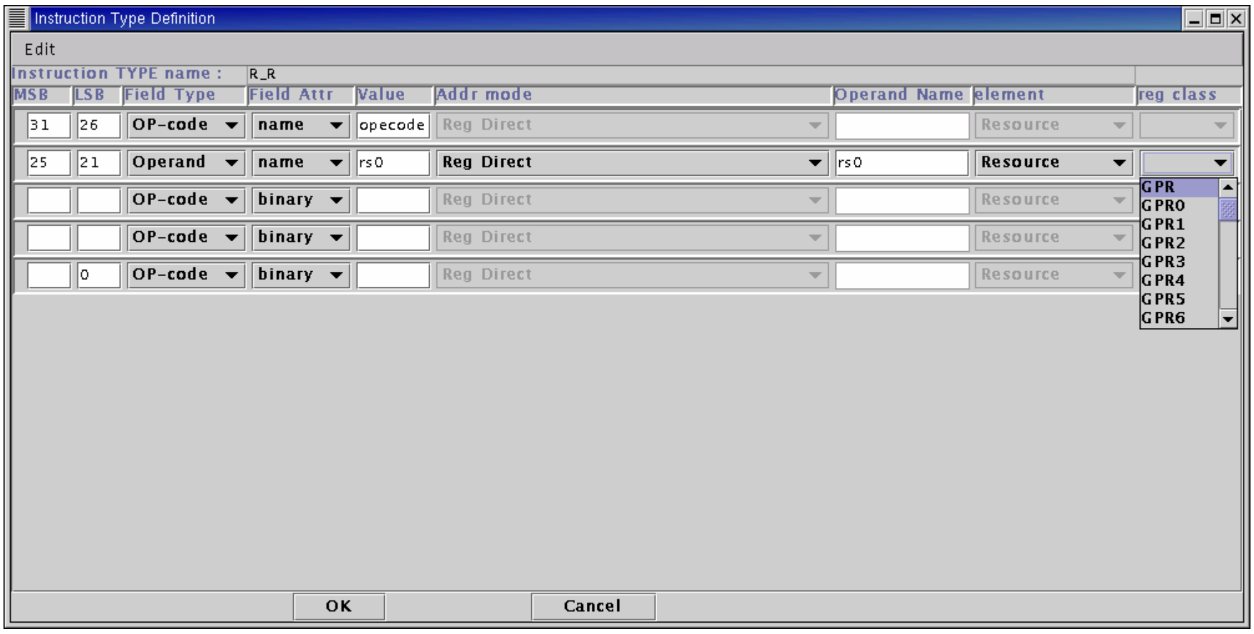
Instruction TYPE name : R_R

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	26	OP-code	name	opecode	Reg Direct		Resource	
25	21	Operand	name	rs0	Reg Direct	rs0	Resource	
		OP-code	binary		Reg Direct			
		OP-code	binary		Reg Direct			
	0	OP-code	binary		Reg Direct		Resource	

Figure 45: [element] pull down menu

- **reg class**

[reg class] means the register class. When the addressing mode of the operand uses registers, you have to specify the register class that the operand of the instruction can use (Figure 46). “rs0” of ADD can be specified “GPR” register class. Hence, you will specify GPR in this part. Register class is declared in the previous step.

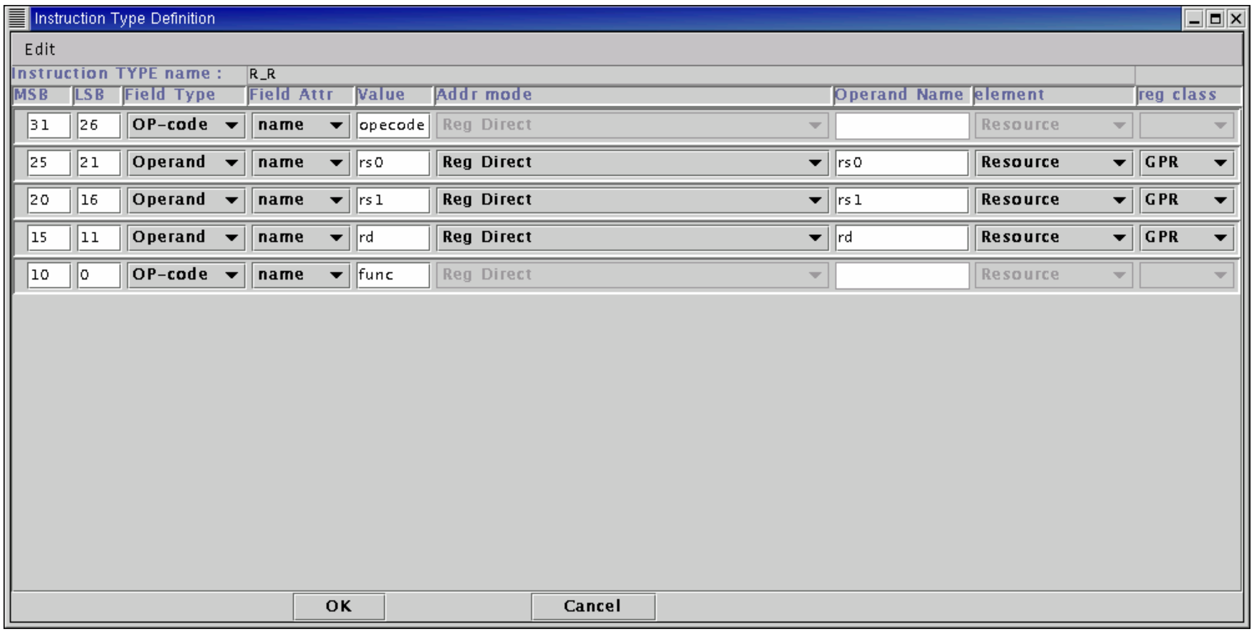


Instruction TYPE name : R_R

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	26	OP-code	name	opecode	Reg Direct		Resource	
25	21	Operand	name	rs0	Reg Direct	rs0	Resource	
		OP-code	binary		Reg Direct		Resource	
		OP-code	binary		Reg Direct		Resource	
	0	OP-code	binary		Reg Direct		Resource	

Figure 46: [reg class] pull down menu

After you have set all the options in the window, instruction type "R_R" should be defined as shown in Figure 47.

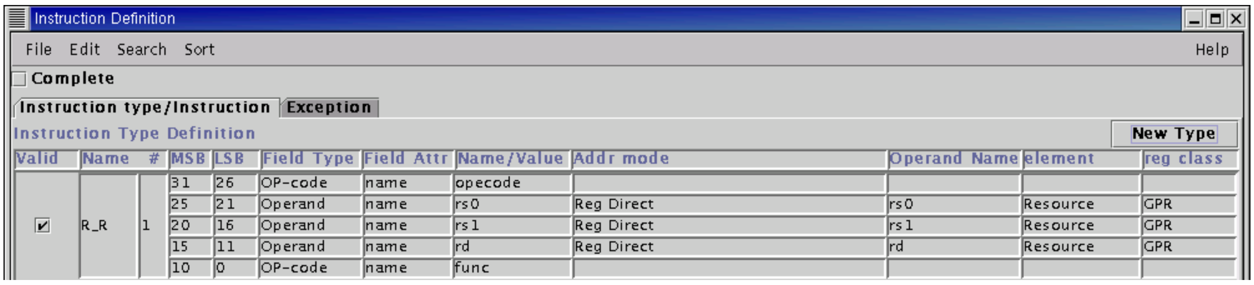


Instruction TYPE name : R_R

MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	26	OP-code	name	opecode	Reg Direct		Resource	
25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
20	16	Operand	name	rs1	Reg Direct	rs1	Resource	GPR
15	11	Operand	name	rd	Reg Direct	rd	Resource	GPR
10	0	OP-code	name	func	Reg Direct		Resource	

Figure 47: “R_R” type setting

Then click [OK] button and return to [Instruction Definition] window. "R_R" type is defined as shown in Figure 48.



Valid	Name	#	MSB	LSB	Field Type	Field Attr	Name/Value	Addr mode	Operand Name	element	reg class
✓	R_R	1	31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
			20	16	Operand	name	rs1	Reg Direct	rs1	Resource	GPR
			15	11	Operand	name	rd	Reg Direct	rd	Resource	GPR
			10	0	OP-code	name	func				

Figure 48: [Instruction Definition] with DLX integer processor setting

Likewise define the other instruction types.

6.5.2. Defining the instruction

The next step is to define the add instruction "ADD" by using "R_R" instruction type. Click [New Instruction] button in [Instruction Definition] window to open [Input] window (Figure 49).

Enter "ADD" to the field and click [OK] button and [Instruction Declaration Window] opens (Figure 50).

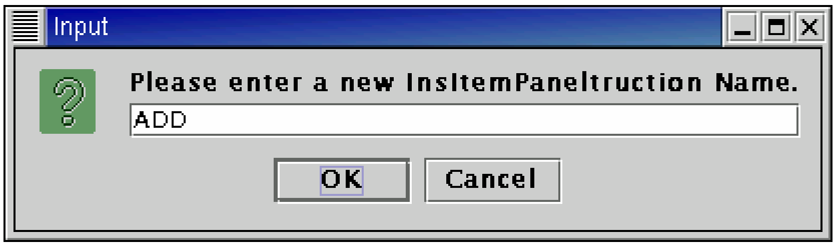
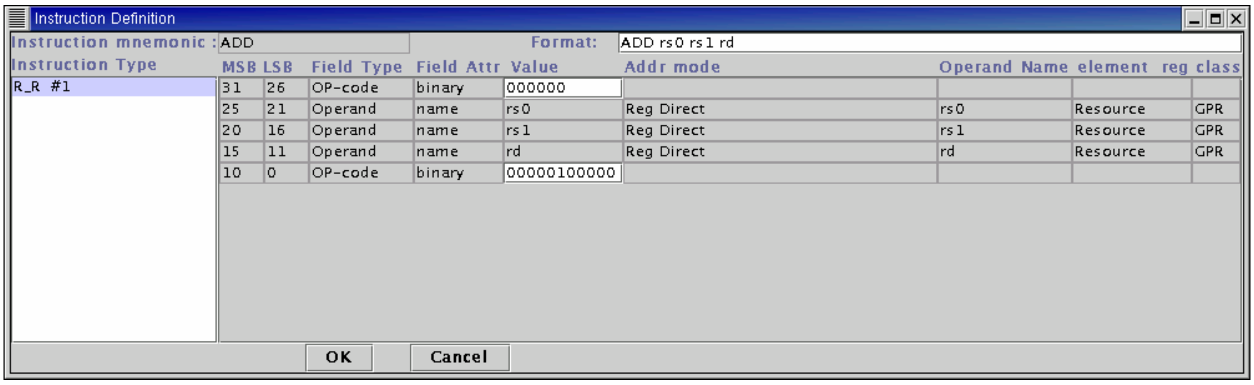


Figure 49: Registration of instruction name



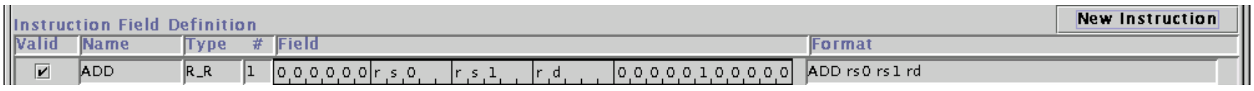
MSB	LSB	Field Type	Field Attr	Value	Addr mode	Operand Name	element	reg class
31	26	OP-code	binary	000000				
25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
20	16	Operand	name	rs1	Reg Direct	rs1	Resource	GPR
15	11	Operand	name	rd	Reg Direct	rd	Resource	GPR
10	0	OP-code	binary	00000100000				

Figure 50: [Instruction Definition] window

From [Instruction Type] list in the left side of the window, select the instruction type "R_R". This is used for the add instruction "ADD".

Only the first and fifth fields are enabled for entering the value. This is a unique field for instructions of "R_R" instruction type.

After entering the binary pattern to the fifth field, click [OK] button to end the "ADD" instruction definition and return to [Instruction Definition Window] window. The window shows "ADD" instruction definition as shown in Figure 51.



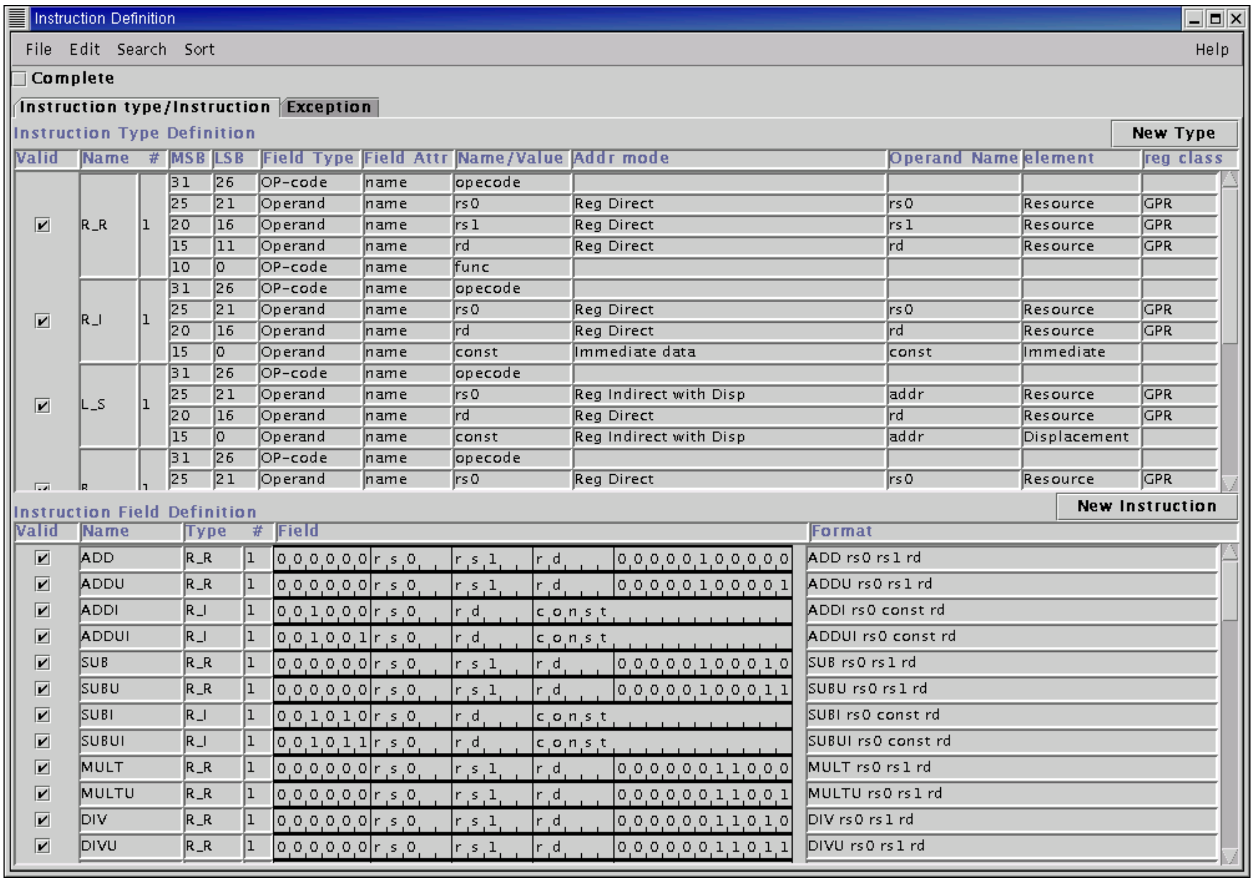
Instruction Field Definition

New Instruction

Valid	Name	Type	#	Field	Format
✓	ADD	R_R	1	000000 rs0 rs1 rd 00000100000	ADD rs0 rs1 rd

Figure 51: "ADD" instruction definition

Likewise define other instructions. Figure 52 shows DLX integer processor setting of instruction types and instructions.



Instruction Type Definition

New Type

Valid	Name	#	MSB	LSB	Field Type	Field Attr	Name/Value	Addr mode	Operand Name	element	reg class
✓	R_R	1	31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
			20	16	Operand	name	rs1	Reg Direct	rs1	Resource	GPR
			15	11	Operand	name	rd	Reg Direct	rd	Resource	GPR
			10	0	OP-code	name	func				
✓	R_I	1	31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR
			20	16	Operand	name	rd	Reg Direct	rd	Resource	GPR
			15	0	Operand	name	const	Immediate data	const	Immediate	
✓	L_S	1	31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Indirect with Disp	addr	Resource	GPR
			20	16	Operand	name	rd	Reg Direct	rd	Resource	GPR
			15	0	Operand	name	const	Reg Indirect with Disp	addr	Displacement	
			31	26	OP-code	name	opecode				
			25	21	Operand	name	rs0	Reg Direct	rs0	Resource	GPR

Instruction Field Definition

New Instruction

Valid	Name	Type	#	Field	Format
✓	ADD	R_R	1	000000 rs0 rs1 rd 00000100000	ADD rs0 rs1 rd
✓	ADDU	R_R	1	000000 rs0 rs1 rd 00000100001	ADDU rs0 rs1 rd
✓	ADDI	R_I	1	001000 rs0 rd const	ADDI rs0 const rd
✓	ADDUI	R_I	1	001001 rs0 rd const	ADDUI rs0 const rd
✓	SUB	R_R	1	000000 rs0 rs1 rd 00000100010	SUB rs0 rs1 rd
✓	SUBU	R_R	1	000000 rs0 rs1 rd 00000100011	SUBU rs0 rs1 rd
✓	SUBI	R_I	1	001010 rs0 rd const	SUBI rs0 const rd
✓	SUBUI	R_I	1	001011 rs0 rd const	SUBUI rs0 const rd
✓	MULT	R_R	1	000000 rs0 rs1 rd 00000011000	MULT rs0 rs1 rd
✓	MULTU	R_R	1	000000 rs0 rs1 rd 00000011001	MULTU rs0 rs1 rd
✓	DIV	R_R	1	000000 rs0 rs1 rd 00000011010	DIV rs0 rs1 rd
✓	DIVU	R_R	1	000000 rs0 rs1 rd 00000011011	DIVU rs0 rs1 rd

Figure 52: [Instruction Definition] with DLX integer processor setting

6.5.3. Setting the interrupt/exception

Now we will define the reset interrupt operation of "Processor DLX Integer Specification". If you click the [Exception] tab, [Internal/Exception Declaration] panel appears as shown in Figure 53.

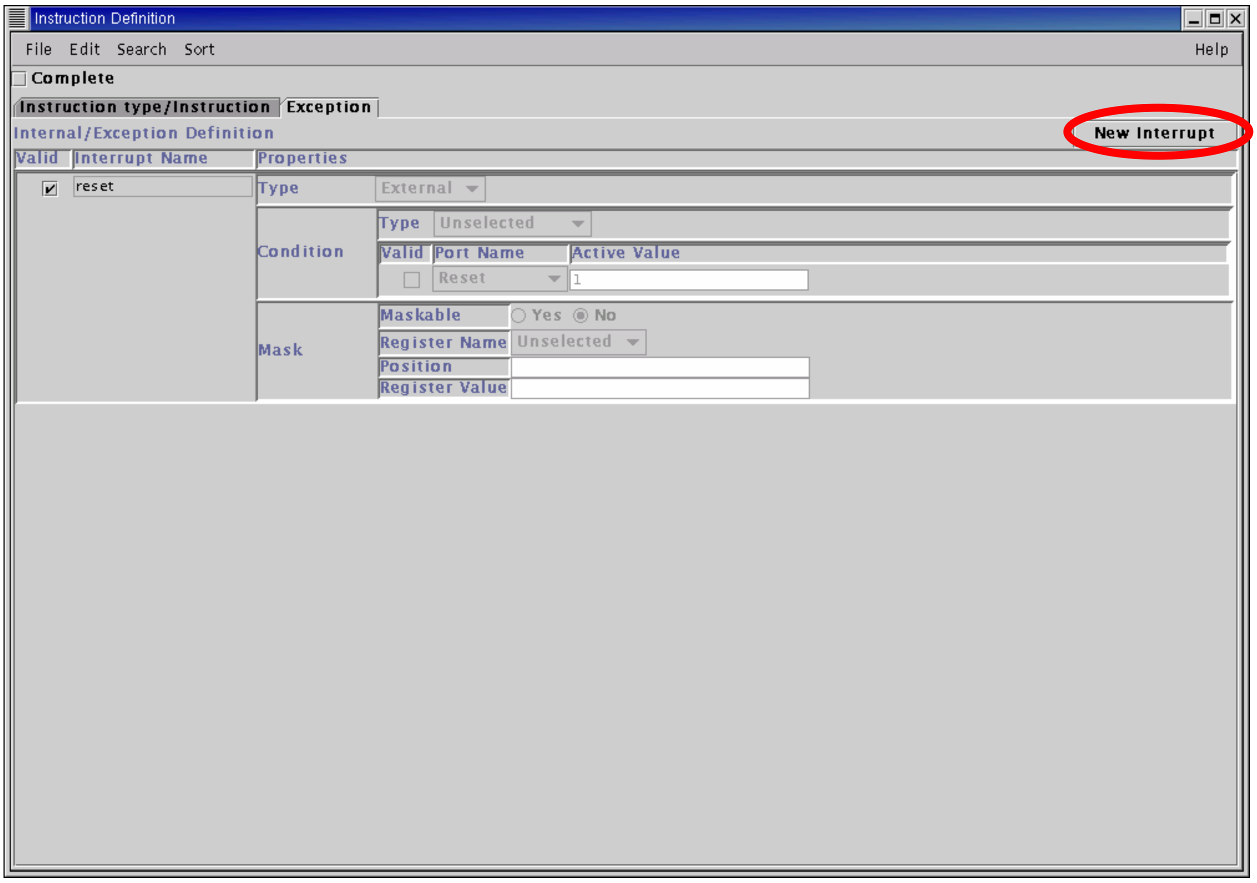


Figure 53: Interrupt/exception declaration window

To define a new interrupt/exception, click the [New Interrupt] button. This will create a new row for the new interrupt/exception. Table 3 describes the fields and their meanings.

Table 3: Descriptions of interrupt/exception

Field	Description
Valid	Click to enable/disable the interrupt/exception.
Interrupt Name	Name of interrupt/exception
Cycle	Number of the execution cycle for the execution stage.
Type	Interrupt/exception type. You can select [External] or [Internal]. External: External signal causes the condition. Internal: Internal signal causes the condition.
Condition	Logical expression to cause the interrupt/exception
Mask	Masking setting

After you have set the all fields of “reset” interrupt, click the [Complete] checkbox and go to [File] > [Close] to return to the main window.

6.6. [Arch. Level Estimation] window: estimating the design qualities

This section explains how to estimate the design qualities of the target processor. In this section, by using the currently set information, ASIP Meister estimates design qualities (area, delay and power consumption). Since this is the estimation before finalizing the details of the design, the estimated values are not exact values, but these values can help your following design steps. If the estimation result shows lower value than your expectation, you can change the settings and re-estimate the qualities.

Click [Arch. Level Estimation] in the main window (Figure 54) and [Estimation Confirm] window opens (Figure 55).

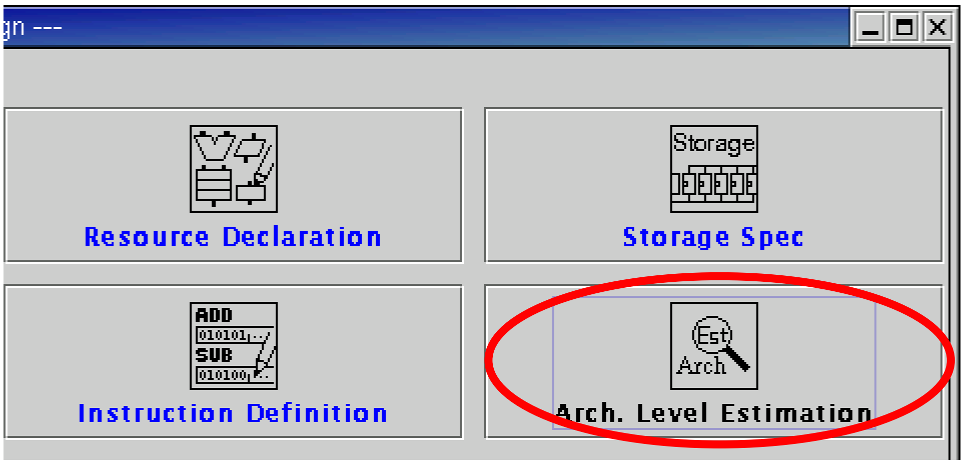
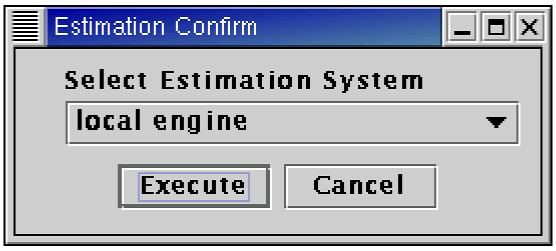


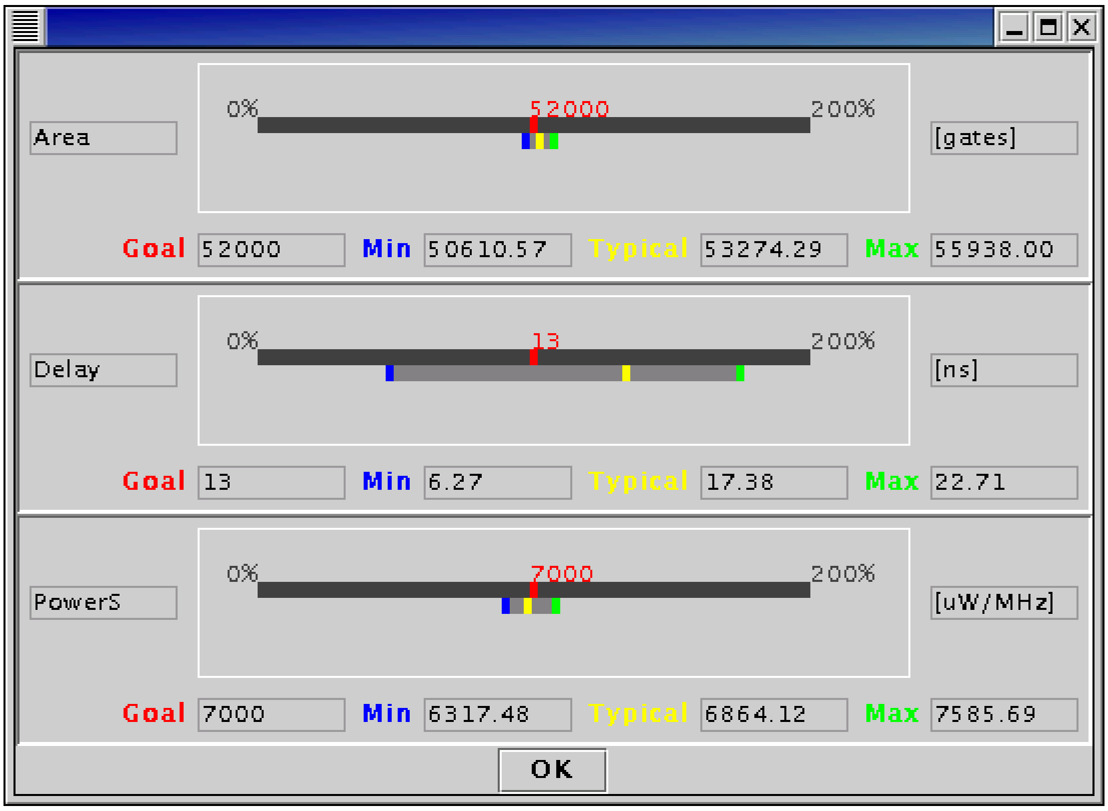
Figure 54: [Arch. Level Estimation] in main menu



Select Estimation System
local engine

Figure 55: [Estimation Confirm] window

The current version only supports [local engine] for [Select Estimation System] pull down menu. So click [Execute] button. The estimation engine will estimate design qualities and show the results of the estimation (Figure 56).



Area [gates]
Goal 52000 Min 50610.57 Typical 53274.29 Max 55938.00

Delay [ns]
Goal 13 Min 6.27 Typical 17.38 Max 22.71

PowerS [uW/MHz]
Goal 7000 Min 6317.48 Typical 6864.12 Max 7585.69

Figure 56: Estimation results

Confirm displayed results and click [OK] button and select [File]>[Close] to return to the main window.

6.7. [C Definition] window: defining C language specification

This section explains how to describe C language specification. C Definition phase includes as follows: data type definition, structure declaration and CKF (Compiler Known Function) declaration. In this tutorial, you don’t need to change default value of this phase.

Click [C Definition] button in the main window (Figure 57) and [C Definition] window opens (Figure 58).

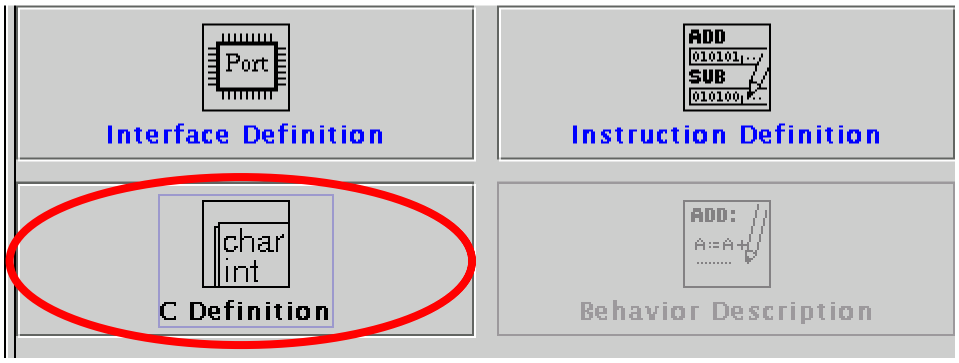
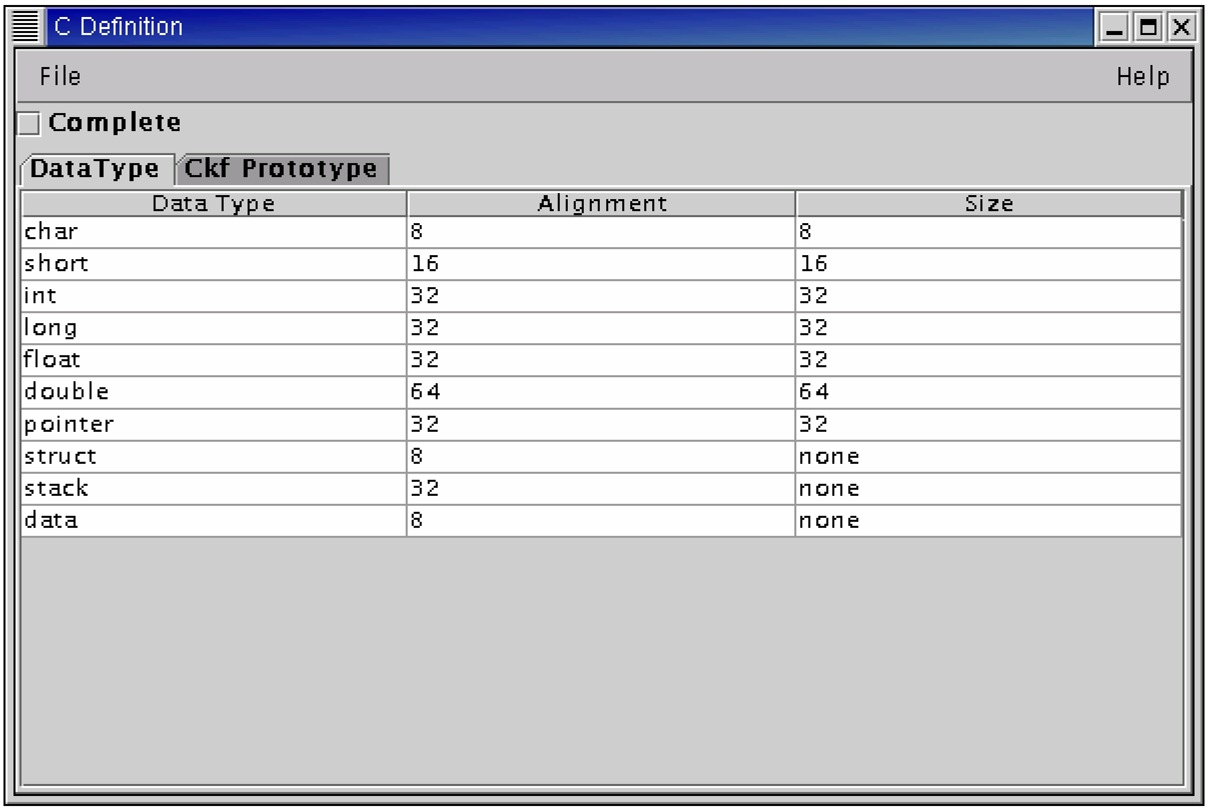


Figure 57: [C Definition] in main menu



Data Type	Alignment	Size
char	8	8
short	16	16
int	32	32
long	32	32
float	32	32
double	64	64
pointer	32	32
struct	8	none
stack	32	none
data	8	none

Figure 58: [C Definition] window

In this tutorial, please just check the complete button and select [File]>[Close] to return to the main window.

6.8. [Behavior Description] window: describing behavior of instructions

This section explains how to describe behavior of instructions. Each behavior is written in C like semantics. This information is used for instruction set simulator generation and compiler generation.

Click [Behavior Description] button in the main window (Figure 59) and [Behavior Description] window opens (Figure 60).

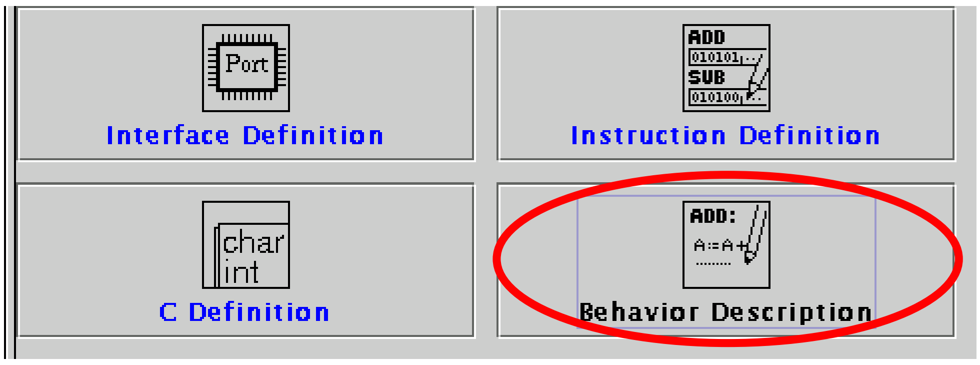
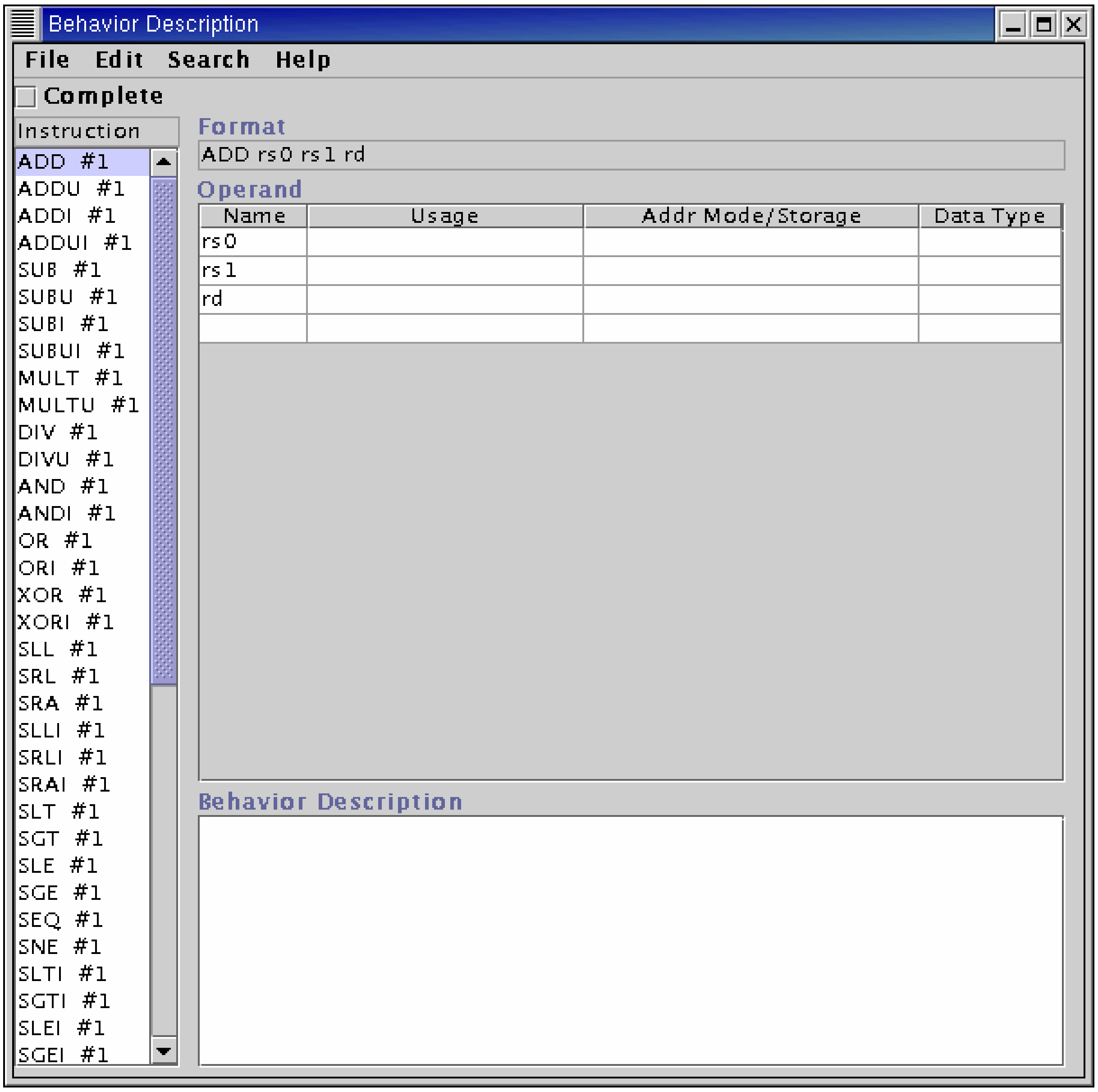


Figure 59: [Behavior Description] in main menu



ADD #1
ADDU #1
ADDI #1
ADDUI #1
SUB #1
SUBU #1
SUBI #1
SUBUI #1
MULT #1
MULTU #1
DIV #1
DIVU #1
AND #1
ANDI #1
OR #1
ORI #1
XOR #1
XORI #1
SLL #1
SRL #1
SRA #1
SLLI #1
SRLI #1
SRAI #1
SLT #1
SGT #1
SLE #1
SGE #1
SEQ #1
SNE #1
SLTI #1
SGTI #1
SLEI #1
SGEI #1

ADD rs0 rs1 rd

Name	Usage	Addr Mode/Storage	Data Type
rs0			
rs1			
rd			

Figure 60: [Behavior Description] window

The window consists of the following items;

● Instruction list:	All instructions can be seen in this part.
● Format:	Instruction format can be seen in this part.
● Operand:	Operands that are terms of expression can be seen in this part.
● Behavior Description:	Behavior is described in this part.

To define the behavior of instruction, select target instruction and fill in operands declaration and behavior description according to “Processor DLX Integer Specification”. If you finish input, push complete button and exit.

6.9. [Micro Op. Description] window: describing micro operation

This section explains how to describe micro operations of instructions. Refer to "How to write the micro operation description" for the method of describing the micro operations.

Click the [Micro Op. Description] in the main window (Figure 61) and open the window (Figure 62).

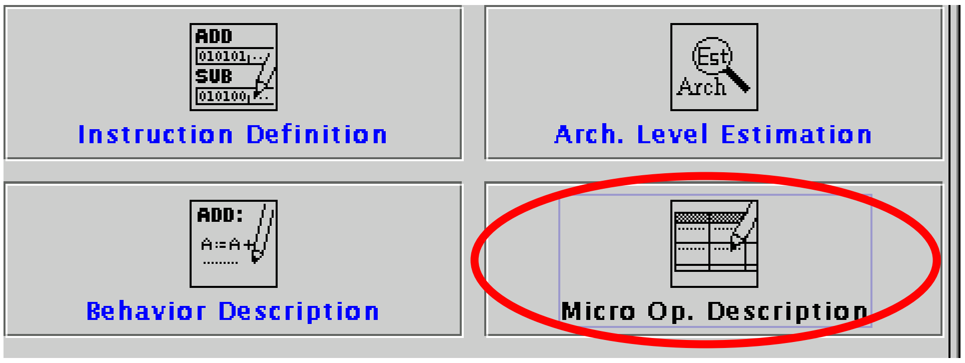


Figure 61: [Micro Op. Description] in main menu

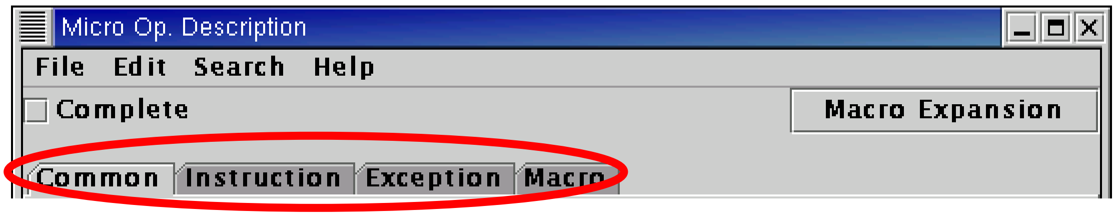


Figure 62: [Micro Op. Description] window

In the window, click the tabs to select the sub windows to be defined;

● Instruction:	Operation description for each instruction
● Exception:	Operation description for interrupt/exception)process
● Macro:	Macro description

In this tutorial, we will define in the order of [Macro], [Instruction] and [Exception].

6.9.1. Macro definition: [Macro]

First, we will define "FETCH" macro. Click [Macro] > [New Macro] to open [New Macro Confirm] window (Figure 63).

Enter "FETCH()" to [Macro name & args] field and “1” to [Num. of Stages] field. Click [OK] button and the macro is listed in [Micro Op Description] window.

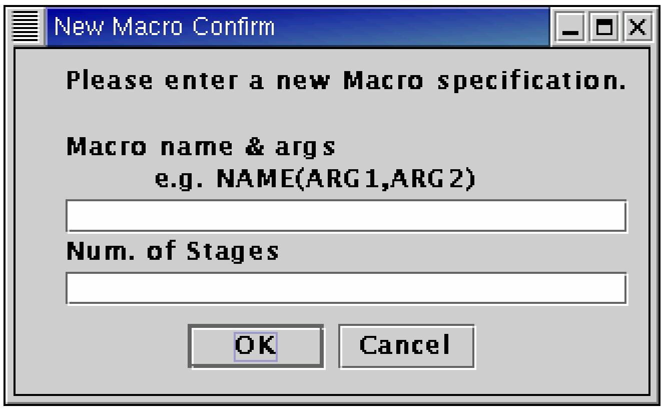
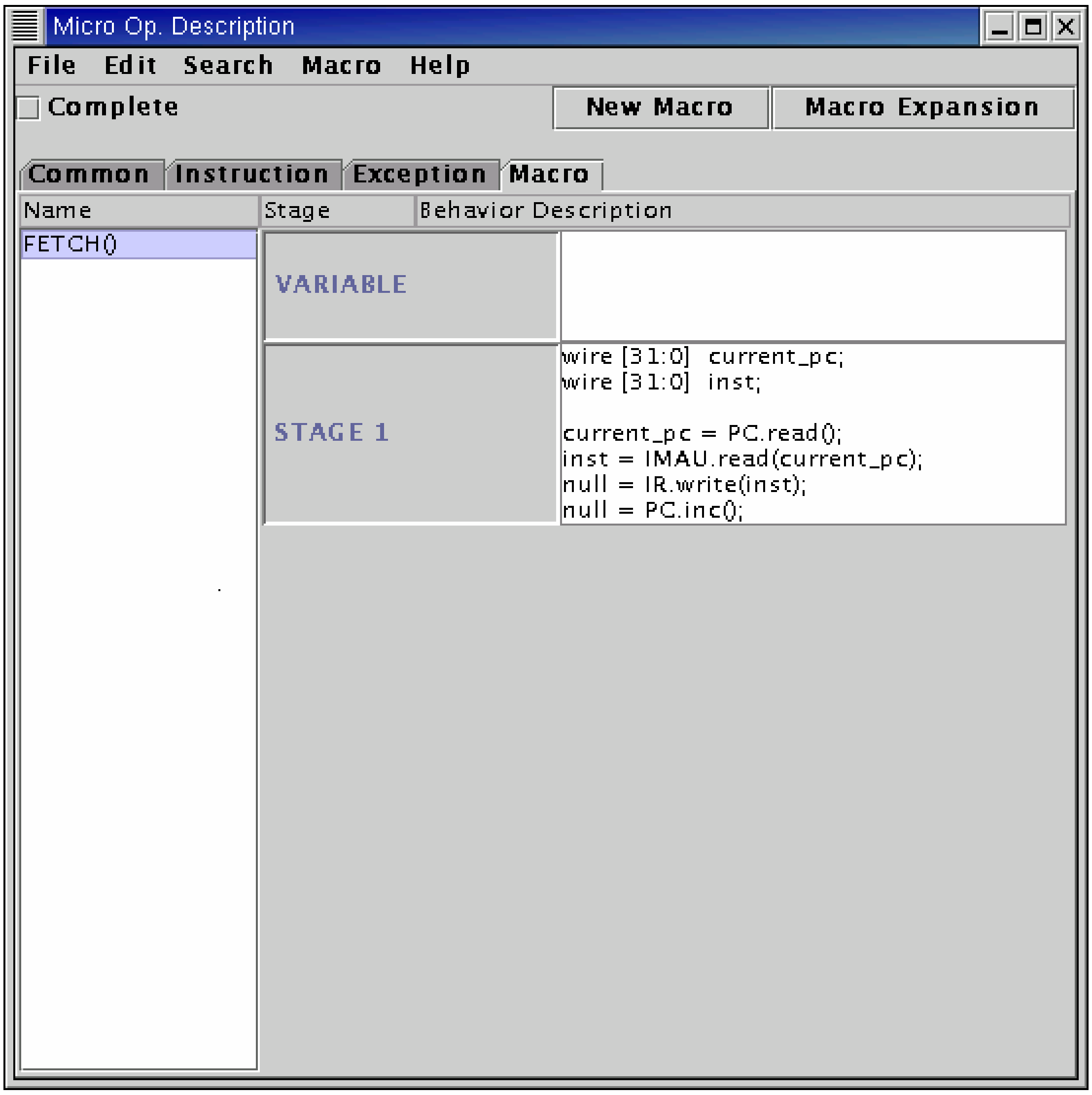


Figure 63: [New Macro Confirm] window

Next, we input macro operations. Figure 64 shows the description of “FETCH” macro.



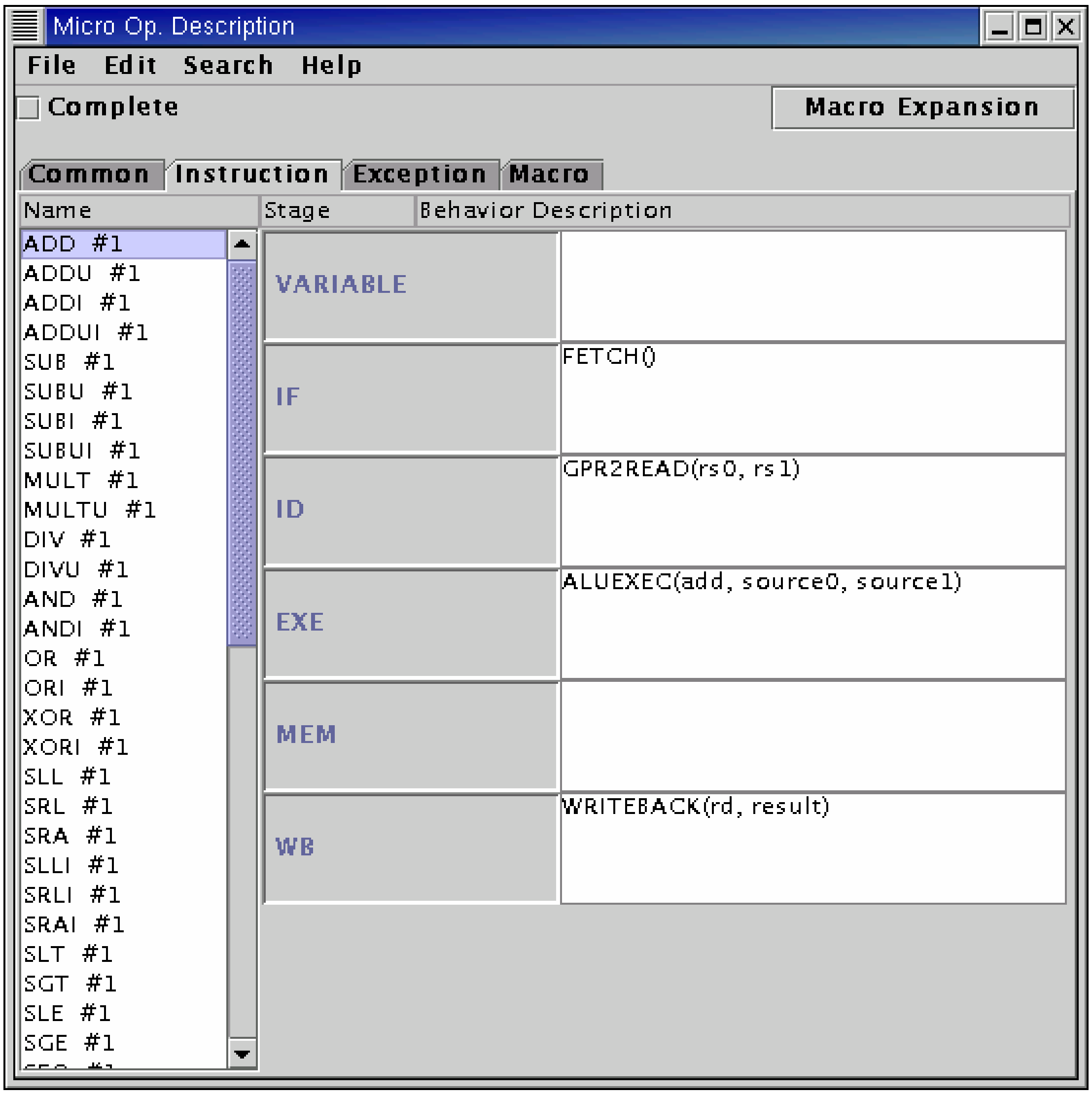
```
wire [31:0]  current_pc;
wire [31:0]  inst;

current_pc = PC.read();
inst = IMAU.read(current_pc);
null = IR.write(inst);
null = PC.inc();
```

Figure 64: “FETCH” macro description

Likewise, make other macros.

6.9.2. Operation description of instruction: [Instruction]

To describe operations of instruction, click [Instruction] tab. [Instruction] frame in the left side of the window lists instructions defined in [Instruction Definition] window. [Stage] column shows the variable declaration (VARIABLE) and the stage names defined in [Design Goal&Arch.] window. Write the description for each corresponding stage in [Behavior Description] field.
Figure 65 shows the description of instruction "ADD".

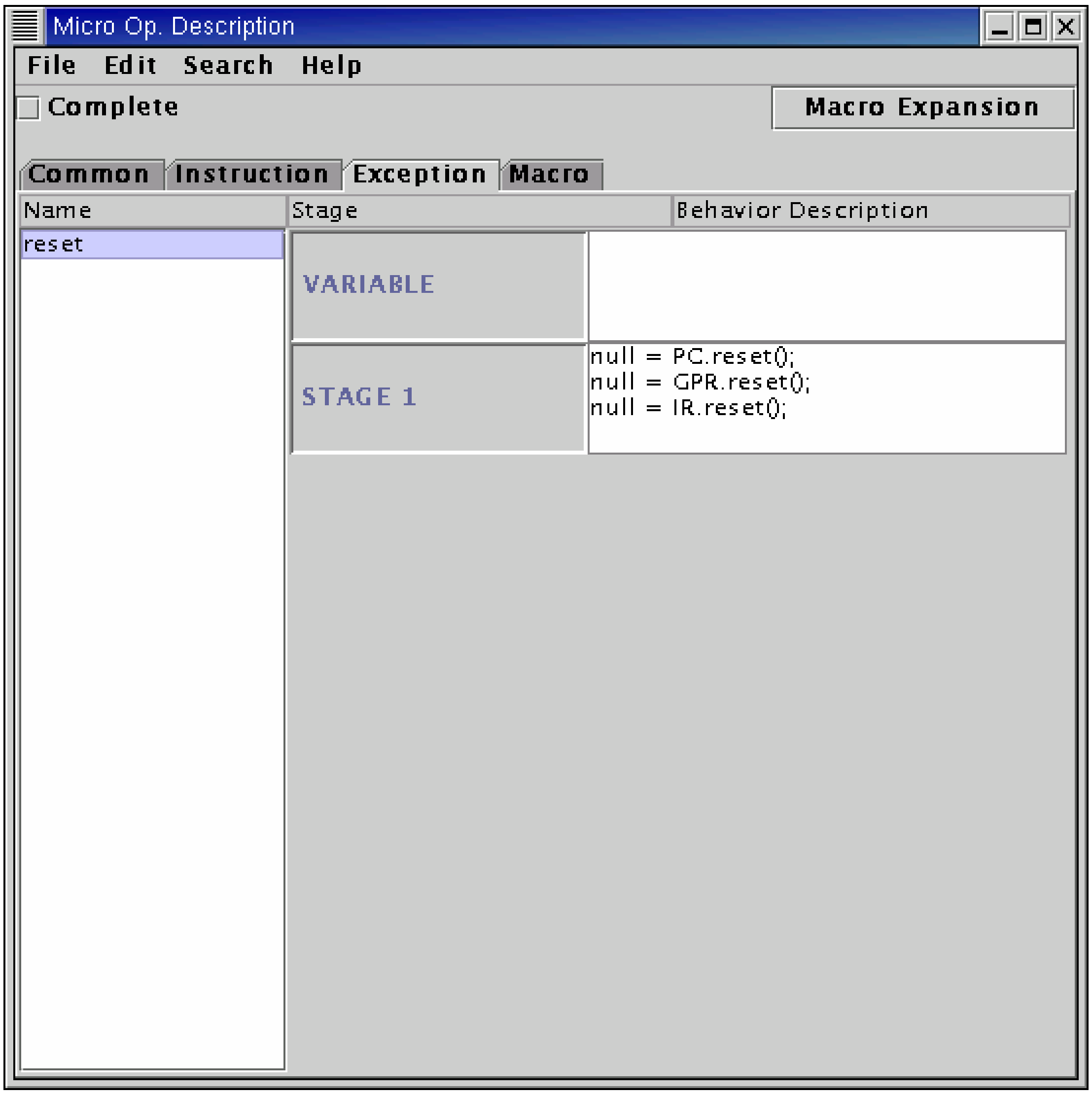


Name	Stage	Behavior Description
ADD #1	VARIABLE	
ADDU #1	IF	FETCH()
ADDI #1	ID	GPR2READ(rs0, rs1)
ADDUI #1	EXE	ALUEXEC(add, source0, source1)
SUB #1	MEM	
SUBU #1	WB	WRITEBACK(rd, result)
SUBI #1		
SUBUI #1		
MULT #1		
MULTU #1		
DIV #1		
DIVU #1		
AND #1		
ANDI #1		
OR #1		
ORI #1		
XOR #1		
XORI #1		
SLL #1		
SRL #1		
SRA #1		
SLLI #1		
SRLI #1		
SRAI #1		
SLT #1		
SGT #1		
SLE #1		
SGE #1		

Figure 65: "ADD" instruction description

6.9.3. Interrupt/exception description: [Exception]

Now we write the operation of "reset" exception defined in [Instruction Definition] window.
Click [Exception] tab to define interrupt/exception processes. Figure 66 shows the description of "reset" interrupt.



```
null = PC.reset();
null = GPR.reset();
null = IR.reset();
```

Figure 66: “reset” interrupt description

After you have written all the descriptions, check [Complete] checkbox and go to [File] > [Close] to return to the main window.

6.10. [HDL Generation] window: generating HDL

This section explains how to generate HDL descriptions for the simulation and validation and for the logical synthesis. ASIP Meister uses the abstraction level that was set earlier with the [Description Style of HDL] radio button of [Resource Declaration] window.

Click [HDL Generation] in the main window (Figure 67) and [Generation Confirm] window opens (Figure 68).

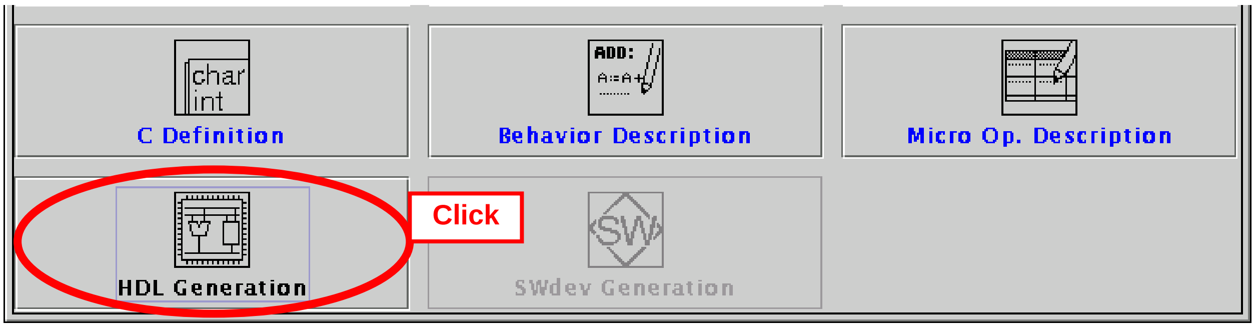
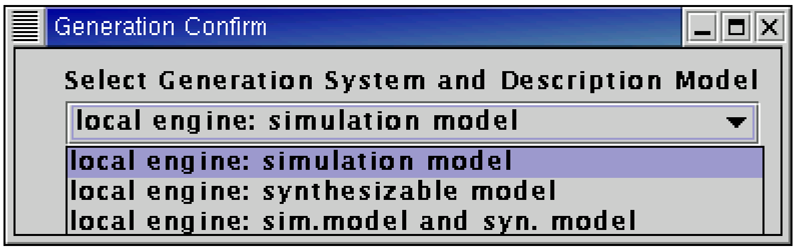


Figure 67: [HDL Generation] in main menu


Generation Confirm

Select Generation System and Description Model

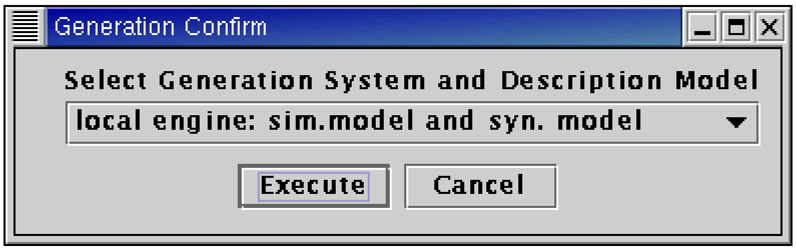
local engine: simulation model

local engine: simulation model
local engine: synthesizable model
local engine: sim.model and syn. model

Figure 68: [Generation Confirm] window

By selecting an option from the pull down menu, you can choose to generate the simulation model, the synthesizable model and both.

Select [sim. Model and syn. Model] to generate the both models and click [Execute] button to start the HDL generation engine.


Generation Confirm

Select Generation System and Description Model

local engine: sim.model and syn. model

Figure 69: Simulation model and synthesizable model generation

When the engine complete the generation, the message window like Figure 70 opens. Confirm that no error is displayed in the window and click [OK] button to return to the main widow.

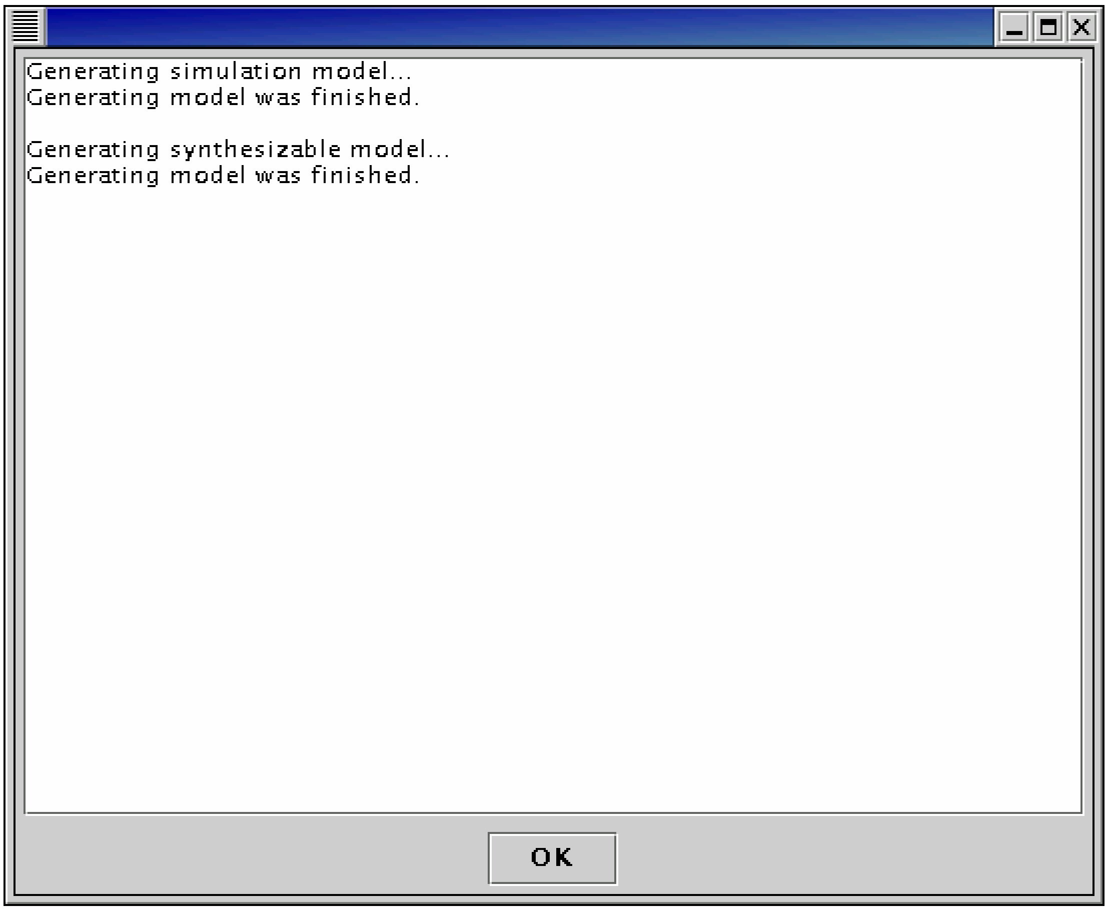

```
Generating simulation model...
Generating model was finished.

Generating synthesizable model...
Generating model was finished.
```

Figure 70: HDL description generation report

Now the current directory has a new directory “meister”. The “meister” directory has two new directories for the simulation model and the logical synthesizable model. Files listed below are saved under the each directory;

- dlx_integer.sim/xxx.vhd: VHDL descriptions of the simulation model (Figure 71) "CPU.vhd" is the top entity.
- dlx_integer.syn/xxx.vhd: VHDL descriptions for the logical synthesis (Figure 72) "CPU.vhd" is the top entity.
- dlx_integer.syn/xxx.scr: Script files for Design Compiler from Synopsys Co. (Figure 73)

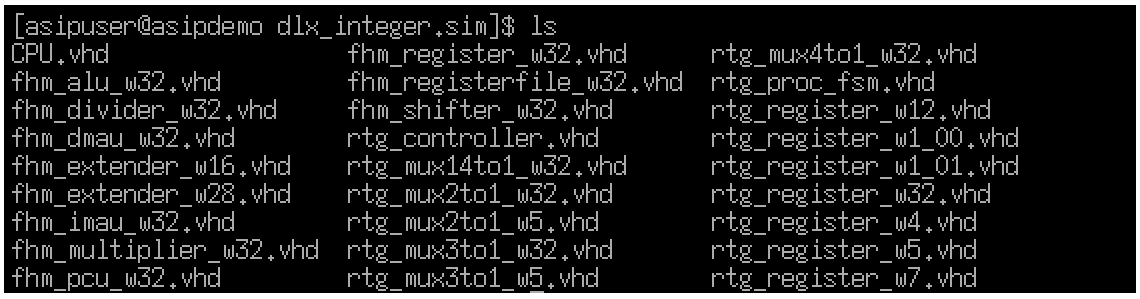
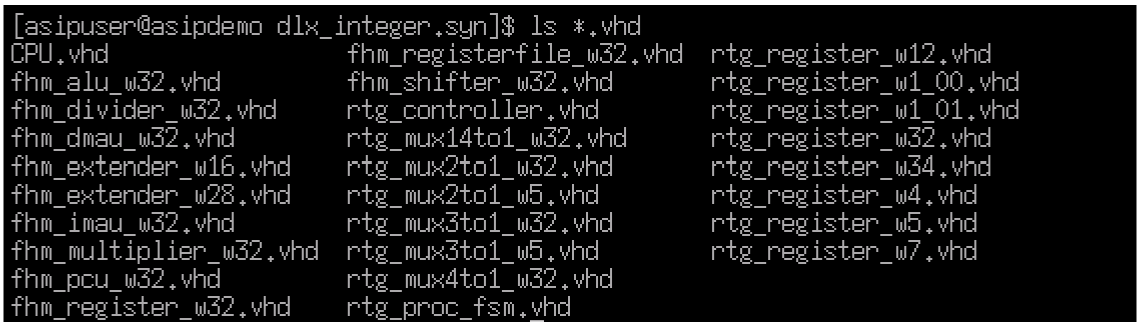
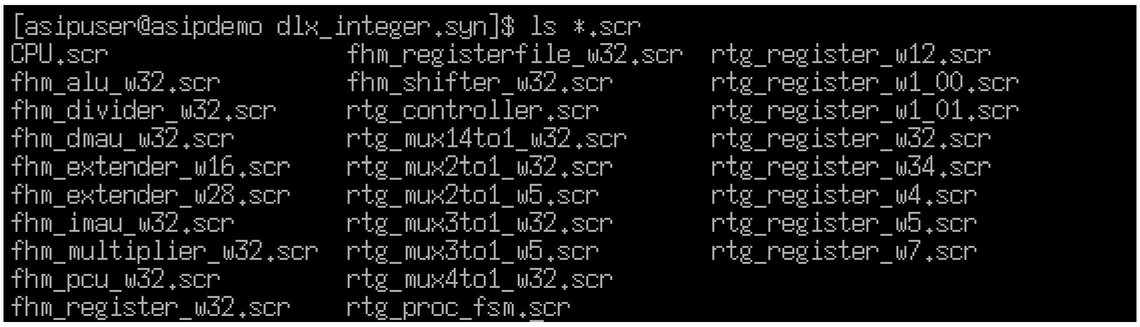

```
[asipuser@asipdemo dlx_integer.sim]$ ls
CPU.vhd                 fhm_register_w32.vhd      rtg_mux4to1_w32.vhd
fhm_alu_w32.vhd         fhm_registerfile_w32.vhd  rtg_proc_fsm.vhd
fhm_divider_w32.vhd     fhm_shifter_w32.vhd       rtg_register_w12.vhd
fhm_dmau_w32.vhd        rtg_controller.vhd        rtg_register_w1_00.vhd
fhm_extender_w16.vhd    rtg_mux14to1_w32.vhd      rtg_register_w1_01.vhd
fhm_extender_w28.vhd    rtg_mux2to1_w32.vhd       rtg_register_w32.vhd
fhm_imau_w32.vhd        rtg_mux2to1_w5.vhd        rtg_register_w4.vhd
fhm_multiplier_w32.vhd  rtg_mux3to1_w32.vhd       rtg_register_w5.vhd
fhm_pcu_w32.vhd         rtg_mux3to1_w5.vhd        rtg_register_w7.vhd
```

Figure 71: Example list of VHDL descriptions in dlx_integer.sim/ directory



```
[asipuser@asipdemo dlx_integer.syn]$ ls *.vhd
CPU.vhd                  fhm_registerfile_w32.vhd  rtg_register_w12.vhd
fhm_alu_w32.vhd          fhm_shifter_w32.vhd       rtg_register_w1_00.vhd
fhm_divider_w32.vhd      rtg_controller.vhd        rtg_register_w1_01.vhd
fhm_dmau_w32.vhd         rtg_mux14to1_w32.vhd      rtg_register_w32.vhd
fhm_extender_w16.vhd     rtg_mux2to1_w32.vhd       rtg_register_w34.vhd
fhm_extender_w28.vhd     rtg_mux2to1_w5.vhd        rtg_register_w4.vhd
fhm_imau_w32.vhd         rtg_mux3to1_w32.vhd       rtg_register_w5.vhd
fhm_multiplier_w32.vhd   rtg_mux3to1_w5.vhd        rtg_register_w7.vhd
fhm_pcu_w32.vhd          rtg_mux4to1_w32.vhd
fhm_register_w32.vhd     rtg_proc_fsm.vhd
```

Figure 72: Example list of VHDL descriptions in dlx_integer.syn/ directory



```
[asipuser@asipdemo dlx_integer.syn]$ ls *.scr
CPU.scr                  fhm_registerfile_w32.scr  rtg_register_w12.scr
fhm_alu_w32.scr          fhm_shifter_w32.scr       rtg_register_w1_00.scr
fhm_divider_w32.scr      rtg_controller.scr        rtg_register_w1_01.scr
fhm_dmau_w32.scr         rtg_mux14to1_w32.scr      rtg_register_w32.scr
fhm_extender_w16.scr     rtg_mux2to1_w32.scr       rtg_register_w34.scr
fhm_extender_w28.scr     rtg_mux2to1_w5.scr        rtg_register_w4.scr
fhm_imau_w32.scr         rtg_mux3to1_w32.scr       rtg_register_w5.scr
fhm_multiplier_w32.scr   rtg_mux3to1_w5.scr        rtg_register_w7.scr
fhm_pcu_w32.scr          rtg_mux4to1_w32.scr
fhm_register_w32.scr     rtg_proc_fsm.scr
```

Figure 73: Example list of script files for synthesis in dlx_integer.syn/ directory

6.11. [SWdev Generation] window: generating descriptions for software development tools

This section explains how to generate descriptions for software development tools such as compiler, assembler and simulator.
Click [SWdev Generation] in the main window (Figure 74) and [Generation Confirm] window opens (Figure 75).

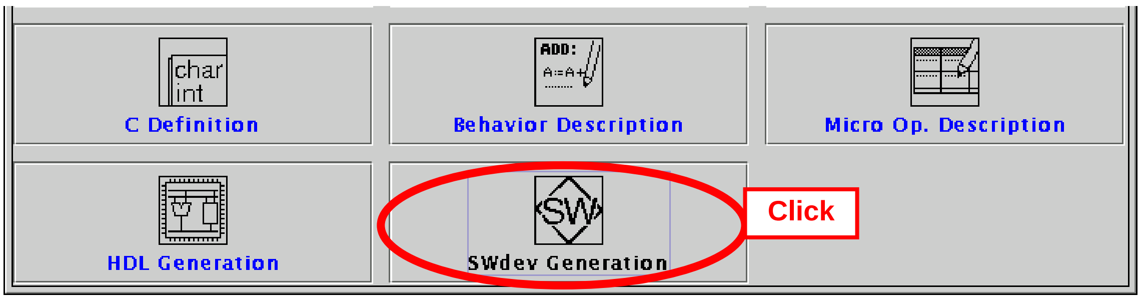


Figure 74: [SWdev Generation] in main menu

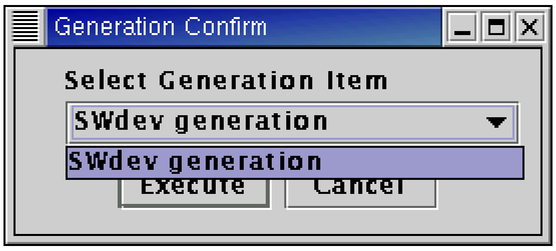


Figure 75: [Generation Confirm] window

Select [SWdev generation] to generate software development tools and click [Execute] button to start the software tools generation engine (Figure 76).

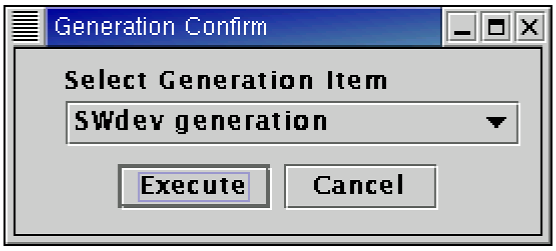
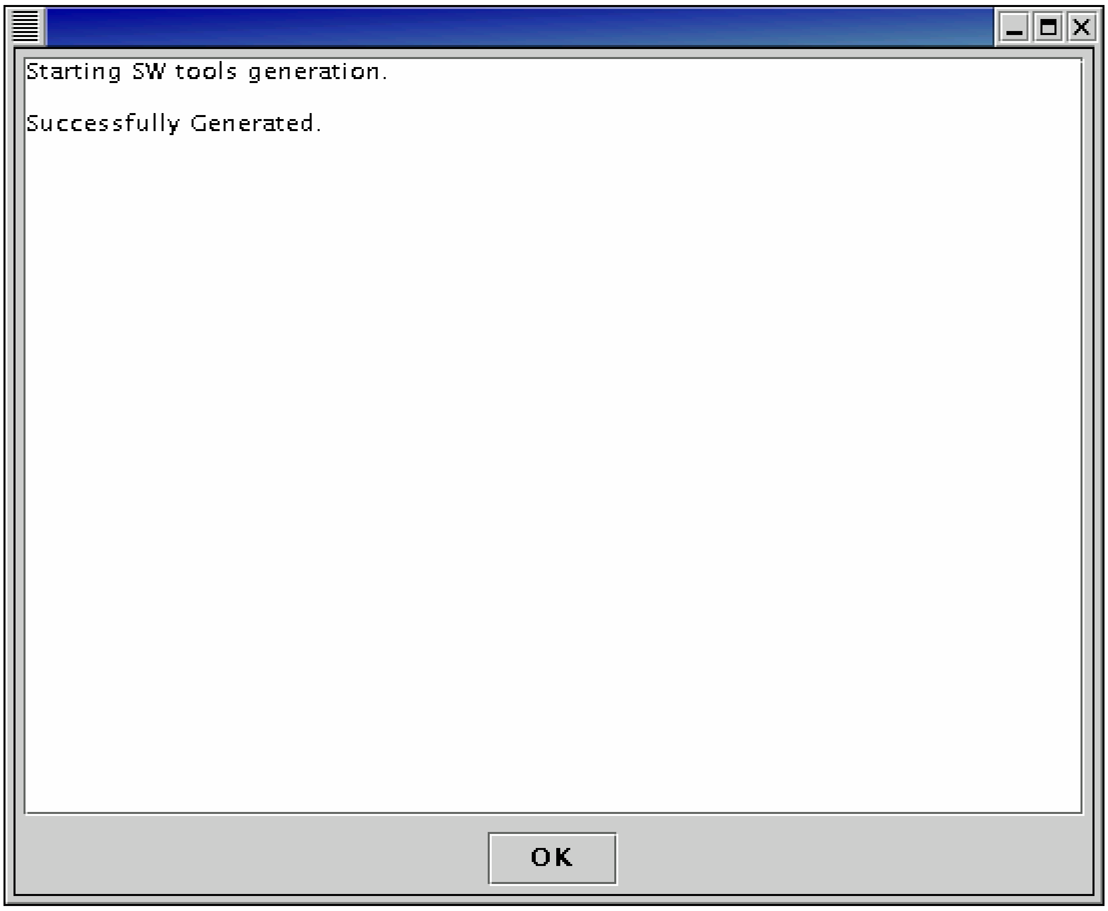


Figure 76: Software tools description generation

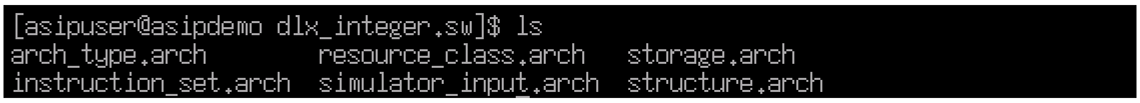
When the engine completes the generation, the message window like Figure 77 opens. Confirm that no error is displayed in the window and click [OK] button to return to the main widow.



```
Starting SW tools generation.

Successfully Generated.
```

Figure 77: Generation report of descriptions for software development tools

The current directory has a new file “meister/dlx_integer.des” for meta-assembler “/usr/local/ASIPmeister/bin/pas” and a new directory named “meister/dlx_integer.sw”. The files in “meister/dlx_integer.sw” directory are for the compiler generator (Figure 78).



```
[asipuser@asipdemo dlx_integer.sw]$ ls
arch_type.arch        resource_class.arch   storage.arch
instruction_set.arch  simulator_input.arch  structure.arch
```

Figure 78: Example list of descriptions for compiler generator and assembler in dlx_integer.sw/ directory

Now, the tutorial lesson is all finished!

7. Tutorial appendix

7.1. 1. How to view the ASIP Meister version information

In the main window, click [Help] > [Version].

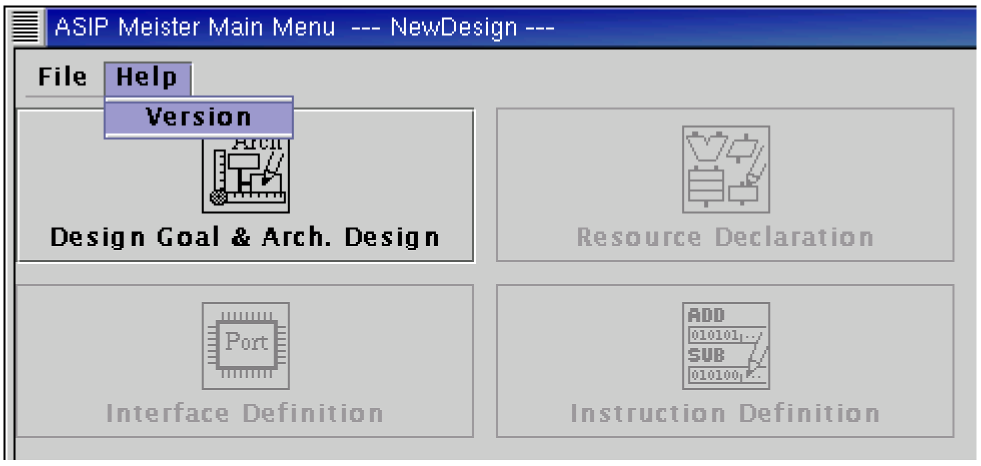
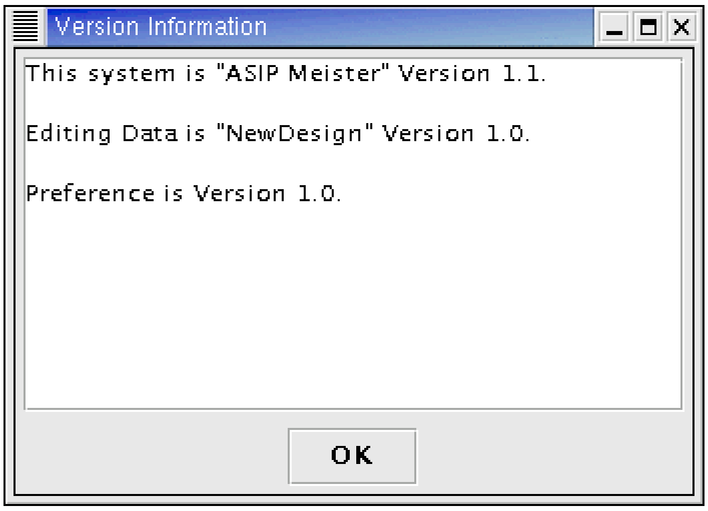


Figure 79: [Version] in main menu

[Version Information] window opens and the version of ASIP Meister is displayed.



```
This system is "ASIP Meister" Version 1.1.

Editing Data is "NewDesign" Version 1.0.

Preference is Version 1.0.
```

Figure 80: [Version Information] window

Index