ProNoC

Processing Tile Generator
First Tutorial

Summary This tutorial teaches how to develop a shared bus (Wishbone bus) based system on chip (SoC) and a simple software implementation using **ProNoC Processing Tile Generator**. The desired SoC will be generated by connecting open-source IP cores on Altera DE2-115 FPGA board.

System Requirements: You will need Altera DE2-115 development board and a computer system running Linux OS with:

1. Installed the ProNoC GUI software and its dependency packages.
2. Installed/Pre-built GNU toolchain of the aeMB soft-core processor.
3. Installed Quarts II (Web-edition or full) compiler.

For more information about the ProNoC and GNU toolchain installation please refer to the ProNoC system installation file located in `/DoC` folder.

Objectives:

1. To design a Wishbone bus based system on chip hardware architecture using ProNoC Electronic Design Automation (EDA) software.
2. To develop a simple software application running on generated SoC.
3. To interact with on-board memory units using JTAG to wishbone interface module.

Desired SoC

Schematic Figure 1 illustrates the desired hardware architecture in this tutorial. This architecture consists of:

1. Two seven-segments display connected each to one separate 7-bit general purpose output (GPO)[1].
2. A 2-bit external interrupt.
3. A 32-bit timer.
4. An aeMB processor.
5. An interrupt controller.
6. A single port RAM.
7. A JTAG to Wishbone interface (Jtag_wb).
8. A Wishbone Bus.

[1] Please note that you can achieve same functionality by using only single 14-bit GPO while both seven-segments are connected to. However, we deliberately select two GPOs to show that you can have multiple number of any IP cores that are included in ProNoC library.

9. A Clock source (not shown in Figure 1).

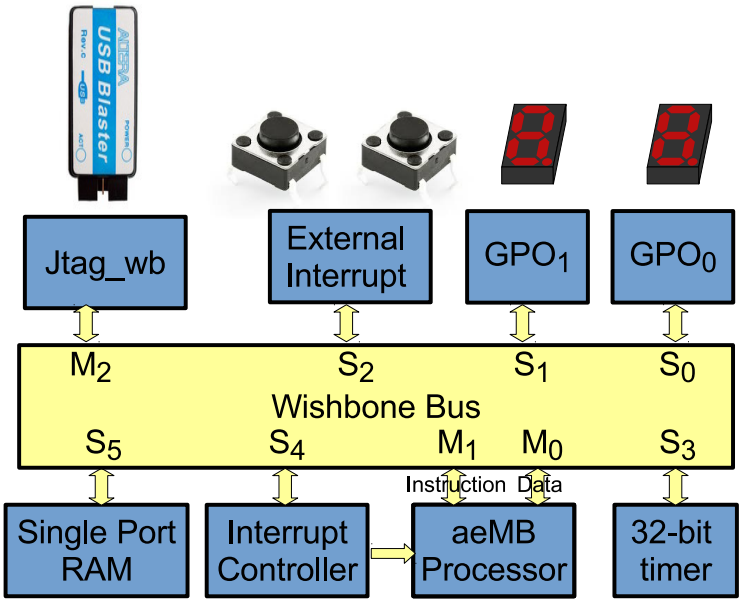


Figure 1: The schematic of desired SoC in this Tutorial.

Application Software

The aim of this tutorial is to design an up/down timer. The system starts counting form zero at reset time. The value of counter increases by one at each 500 ms. Pressing the first interrupt button freezes/unfreezes the counting display while the second interrupt signal works as count-down/count-up change mode.

Create New SoC Using ProNoC Processing Tile Generator

Open `mpsoc/perl_gui` in terminal and run ProNoC GUI application:

```
./ProNoC.pl
```

It should open The GUI interface as illustrated in Figure 2.

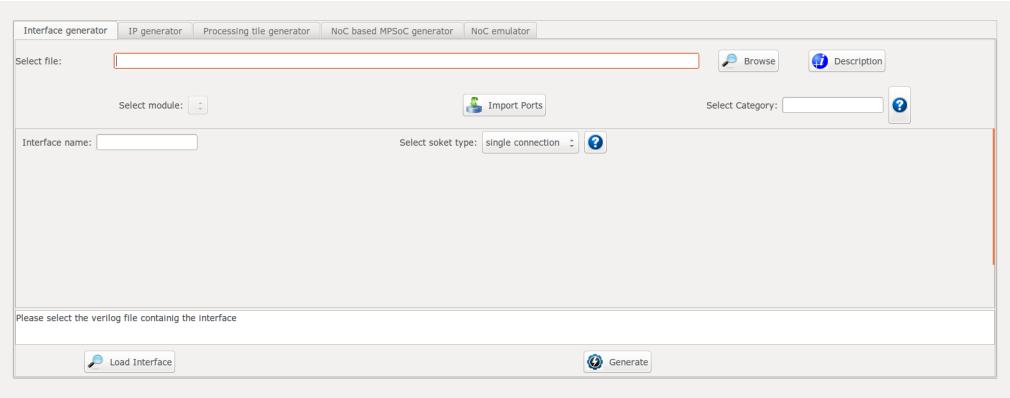


Figure 2: ProNoC GUI first page snapshot.

Then select the *Processing Tile Generator*:

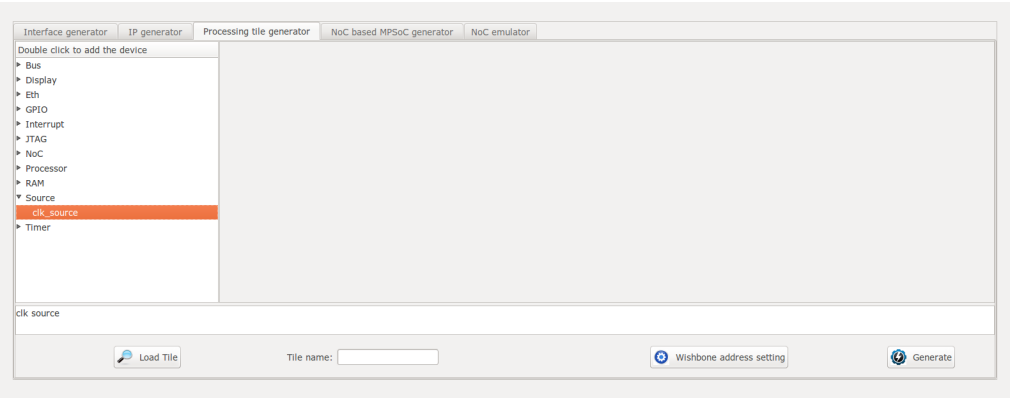


Figure 3: ProNoC New Processing Tile generator snapshot.

At the left tree view window you can see the list of available IP categories. Clicking on each category will expand its containing IP cores. Each IP core can be added to GUI by double clinking on its name. The added IP core has three setting columns:

(a) In first column you can shift IP core box position up/down in GUI interface, remove the IP core or set its parameters (if any).

(b) In the second column you can rename the IP core instance name.

(c) Third column shows all *(Plug)* interfaces of this module. here you can connect each plug to one appropriate *(socket)* interface. (Each interface is categorized into two types of plug and socket. See `/Doc/intfc_gen.pdf` for more information). You can also export the interface as SoC's input/output (IO) ports here.

Now let start calling required IPs. We start with `clk_source`:

Add clk source This module provides clk and reset interfaces for all other IPs. It also synchronizes the reset signal.

1. Click on `Source` category, then double click on `clk_source`.
2. Rename the `clk_source` instance name as source. leave the interfaces as `IO`.

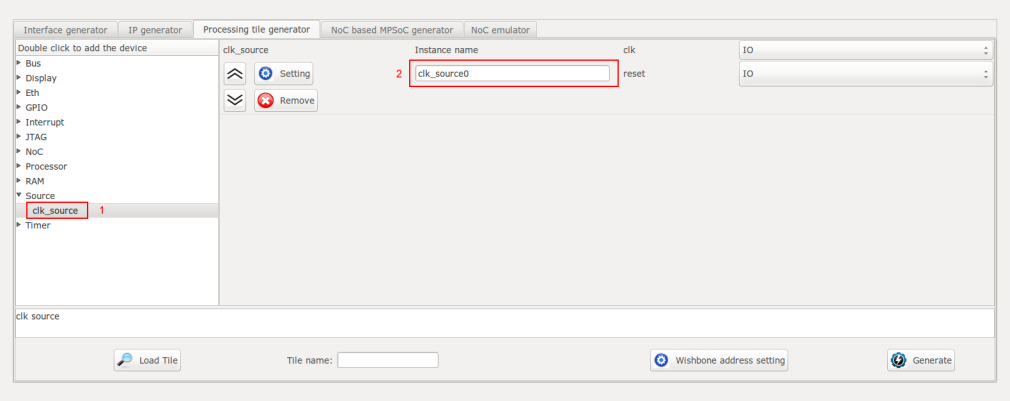


Figure 4: Adding clock source.

Add Wishbone Bus:

1. Click on `Bus` category and double click on `Wishbone_bus`.
2. In parameter setting set `M` (master interfaces number) as 3 and `S` (slave interfaces number) as 6. These values are obtained from Figure 1. You can changed them later if you want to add/remove any IPs.
3. Rename the instance name as `bus`.
4. Connect the clock and source interfaces to `clk_source` module.

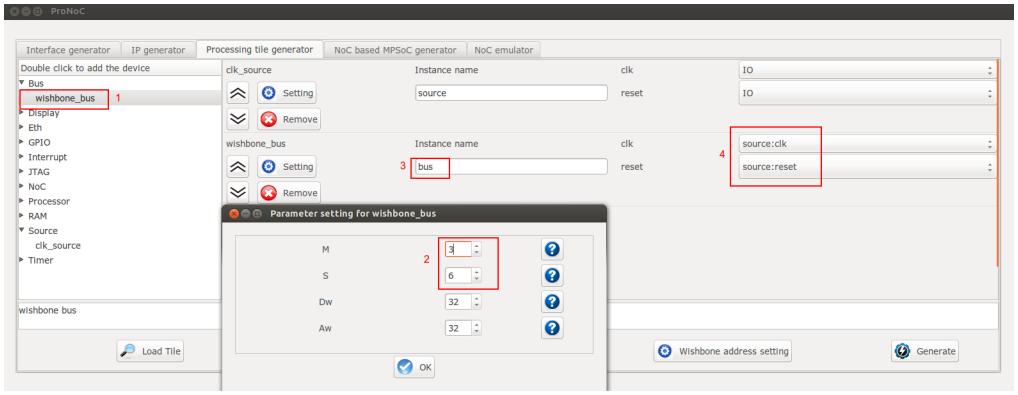


Figure 5: Adding Wishbone bus.

Add GPO$_0$:

1. Click on `GPIO` category and then double click on `gpo`.
2. In parameter setting set `PORT_WIDTH` as 7.
3. Rename the instance name as `hex0`.
4. Connect the clock and source interfaces to `clk_source` module.
5. In interface connection column connect `wb` (Wishbone bus) interface to `bus: wb_slave[0]`

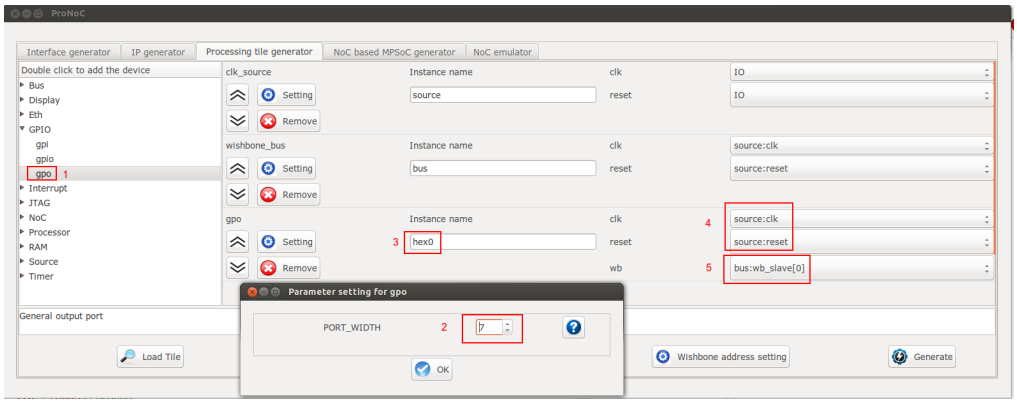


Figure 6: Adding GPIO$_0$.

The socket interface has the following format:

`connection-IP-instance-name : interface-name [interface number].`

hence, `bus:wb_slave[0]` means that GPO wb interface is connected to the bus wishbone slave port number zero. Note that the interface number is not important optionally you can connect it to any other 5 slave interfaces of the bus module.

Add GPO$_1$:

1. Click on `GPIO` category and then double click on `gpo`.
2. In parameter setting set `PORT_WIDTH` as 7.
3. Rename the instance name as `hex1`.
4. Connect the clock and source interfaces to `clk_source` module.
5. Connect `wb` (Wishbone bus) interface to `bus:wb_slave[1]`.

Add Interrupt controller:

1. Click on `Intrrupt` category and then double click on `int_ctrl`.
2. In parameter setting set `INT_NUM` (interrupt number) as 2.
3. Rename the instance name as `int_ctrl`.
4. Connect the clock and source interfaces to `clk_source` module.
5. Connect `wb` (Wishbone bus) interface to `bus:wb_slave[2]`.

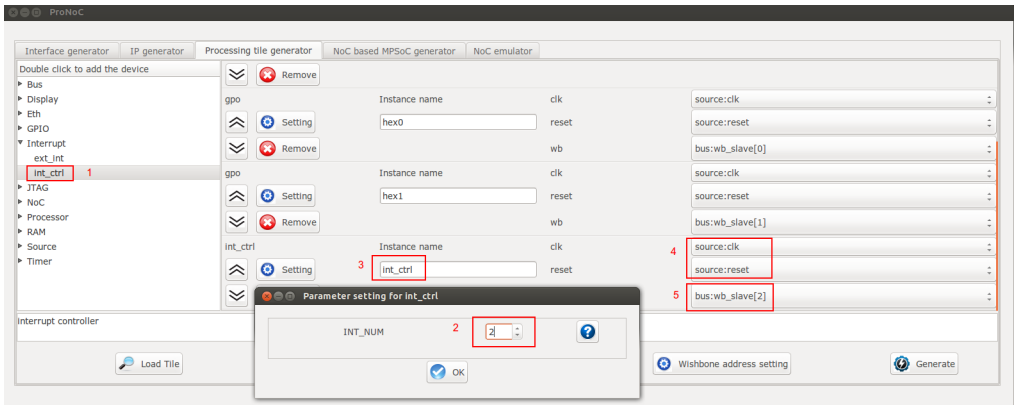


Figure 7: Adding Interrupt controller.

Add External Interrupt:

1. Click on `Intrrupt` category and then double click on `ext_int`.
2. In parameter setting set `EXT_INT_NUM` (external interrupt number) as 2.
3. Rename the instance name as `ext_int`.
4. Connect the clock and source interfaces to `clk_source` module.
5. Connect `interrupt` interface to `int_ctrl:int_periph[0]`.
6. Connect `wb` (Wishbone bus) interface to `bus:wb_slave[3]`.

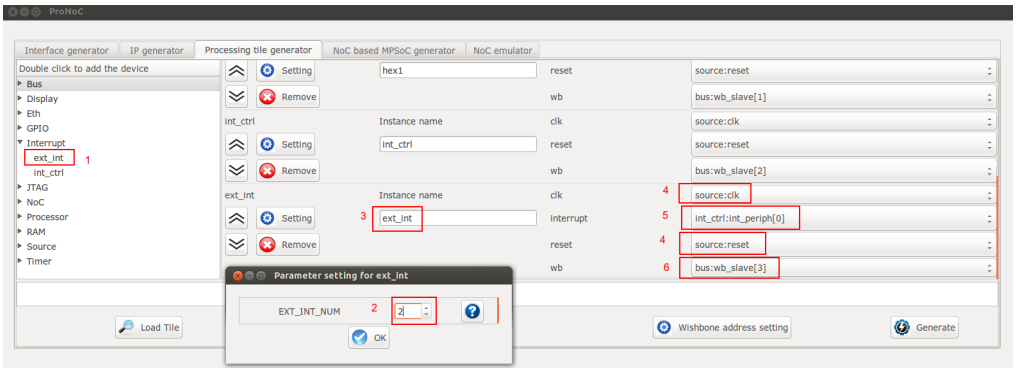


Figure 8: Adding External Interrupt

Add Timer:

1. Click on `Timer` category and then double click on `timer`.
2. Rename the instance name as `timer`.
3. Connect the clock and source interfaces to `clk_source` module.
4. Connect `interrupt` interface to `int_ctrl:int_periph[0]`.
5. Connect `wb` (Wishbone bus) interface to `bus:wb_slave[4]`.

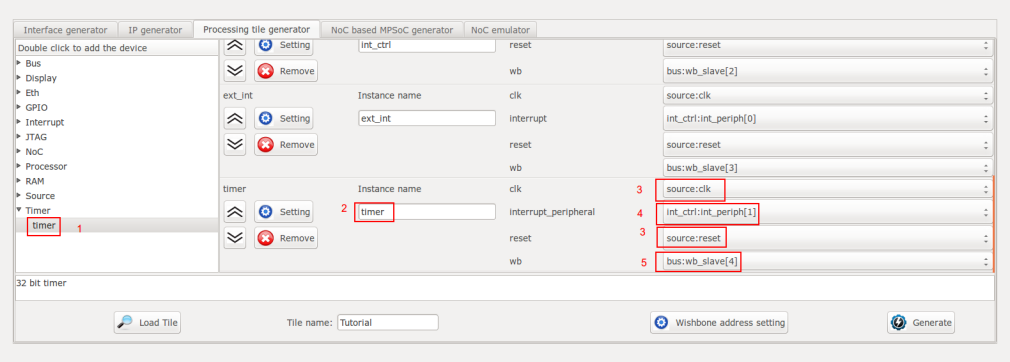


Figure 9: Adding Timer.

Add Single port RAM:

1. Click on `RAM` category and then double click on `single_port_ram`.
2. Leave parameter setting with their default values. This will genertaet a 16 KB memory. You can control the memory size using `Aw` (the memory address width) parameter.

3. Rename the instance name as `ram`.
4. Connect the clock and source interfaces to `clk_source` module.
5. Connect `wb` interface to `bus:wb_slave[5]`.

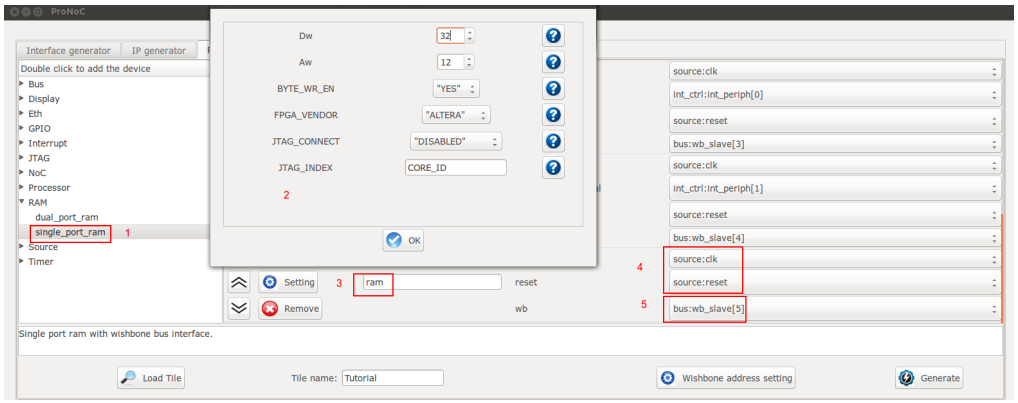


Figure 10: Adding Single port RAM.

Add JTAG to Wishbone Interface:

1. Click on `JTAG` category and then double click on `jtag_wb`.
2. Rename the instance name as `jtag_wb`.
3. Connect the clock and source interfaces to `clk_source` module.
4. Connect `wb` interface to `bus:wb_master[0]`.

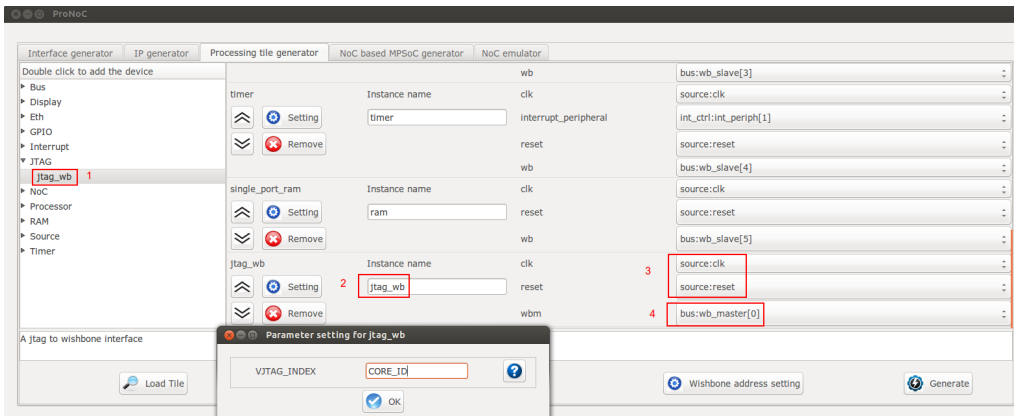


Figure 11: Adding JTAG to Wishbone Interface.

Add Processor:

1. Click on `Processor` category and then double click on `aeMB`.
2. Rename the instance name as `aeMB`.
3. Connect the clock and source interfaces to `clk_source` module.
4. Connect `enable` interface to `IO`
5. Connect `intrp` to `int_ctrl:int_cpu`
6. Connect `iwb` (instruction wishbone bus) interface to `bus:wb_master[1]`.
7. Connect `dwb` (data wishbone bus) interface to `bus:wb_master[2]`.

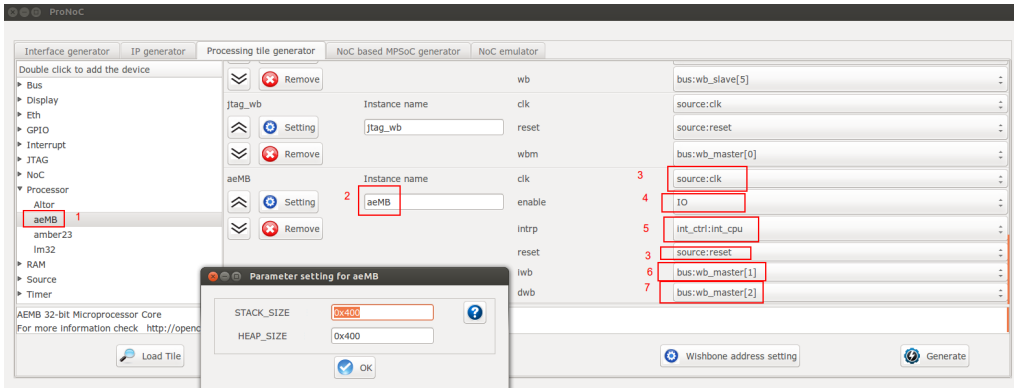


Figure 12: Add aeMB.

Check wishbone bus(es) addresses: After adding all required IP cores, now you can check the auto assigned wishbone addresses by clicking on `Wishbone address setting` button.

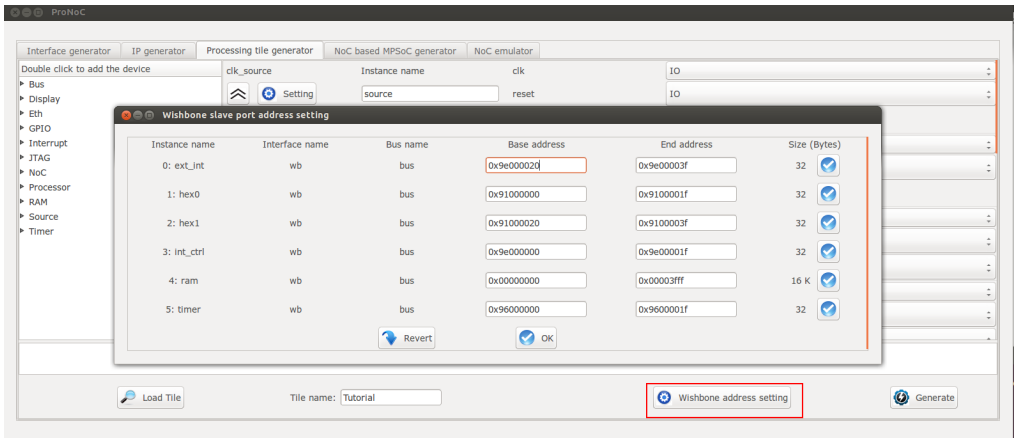


Figure 13: Wishbone bus addresses of Tutorial SoC.

These addresses are automatically set based on IP cores library setting, inserted parameters and numbers of repeating same IP core in the system. However, you are free to adjust them to new values as while as there is no conflict in inserted addresses.

Generate SoC RTL Code:

1. Set Tile name as `Tutorial`.
2. Press `Generate` button.

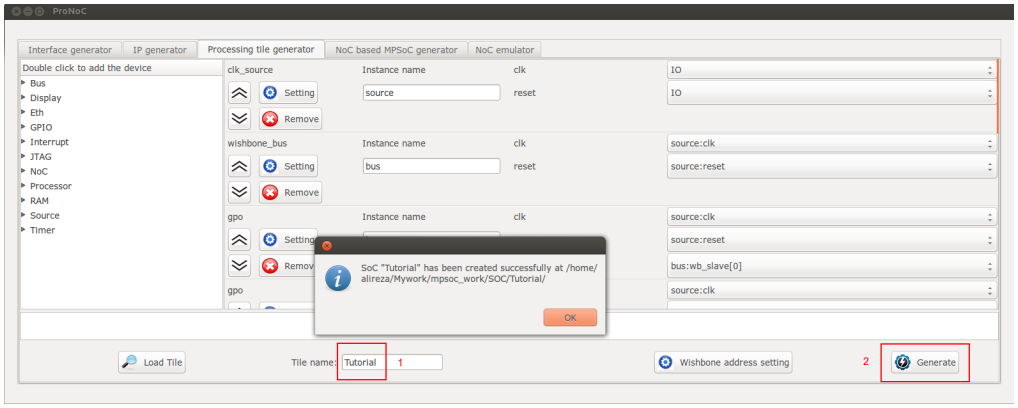


Figure 14: Generating the Tutorial SoC.

It must generate two folders in `mpsoc/soc/Tutorial` path:

- `sw`: This folder contain the required software files including the programming header files, in-system memory editing files and Makefile.
  - `Tutorial.h`: The SoC header file containing all peripheral devices' wishbone addresses and functions (some IPs may have additional header files).
  - `README`: This file contains SoC parameters, IP connection and wishbone bus addresses. This file also explain how to work with `Jtag_wb` IP core.
  - `program.sh`: A sample bash file that can be used for programing RAM.
- `src_verilog`: contains two Verilog files and a folder:
  - `Tutorial.v`: the generated SoC RTL code. This file contains all IPs instances and connections.
  - `Tutorial_top.v`: this file contains the Tutorial SoC module instance connected to a JTAG remote enable/reset controller which disable the SoC during programming time.
  - `lib`: This folder contains all IP cores RTL codes

Compile the generated RTL code using Quartus II software

1. Open Quartus II software, select **Create a new project**, set the following then click next.
   (a) Working directory: `[Path-to-mpsoc_work]/SOC/Tutorial`
   (b) Name of project: `top`
   (c) Name of Top Module : `top`

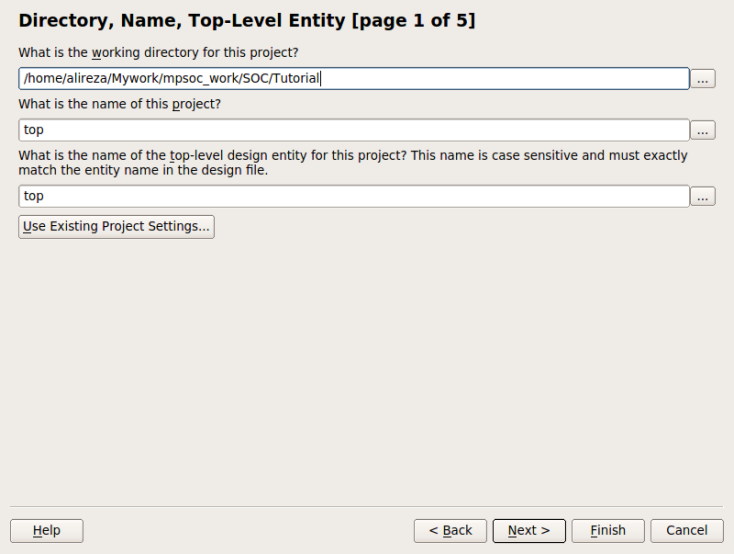


Figure 15: New Project setting.

2. In the next window add all Verilog files inside the project `src_verilog` folder and all of its subdirectories.

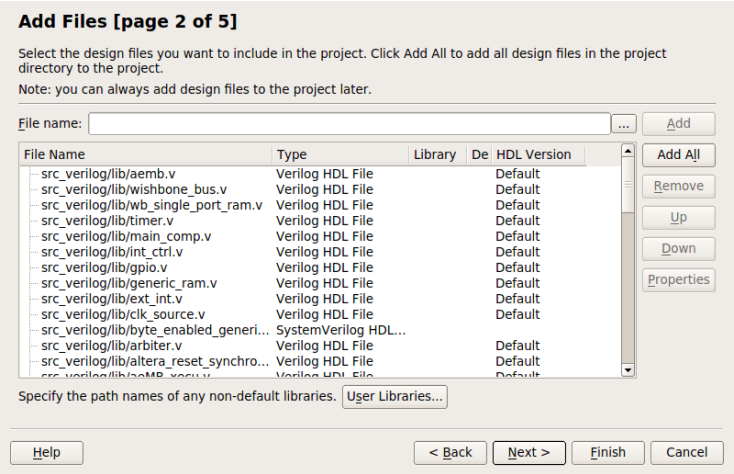


Figure 16: Add Tutorial RTL Codes to the Quartus.

3. At Family and Device setting, select ”Cyclone IV E” and ”EP4CE115F29C7” then click Finish.

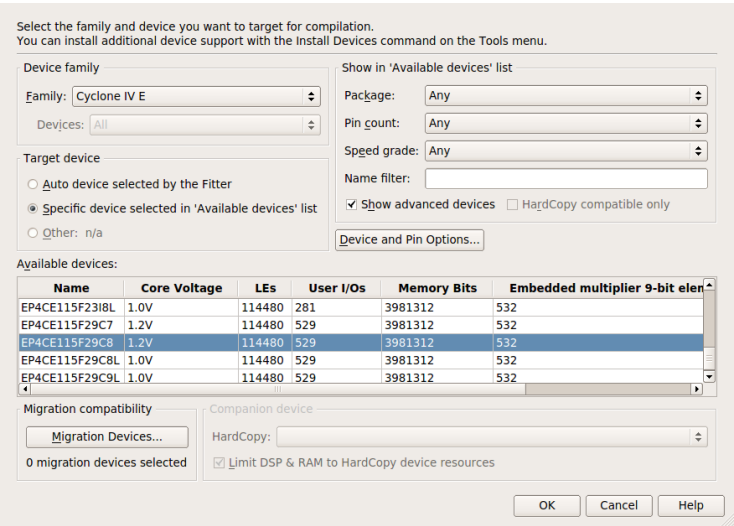


Figure 17: FPGA device selection.

4. Import Pin assigment for DE2-115 board by, Go to Toolbar → Assignment → Import assignment..., Browse for `DE2_115_pin_assignments.csv` file (located in `/mpsoc/perl_gui/examples` folder.

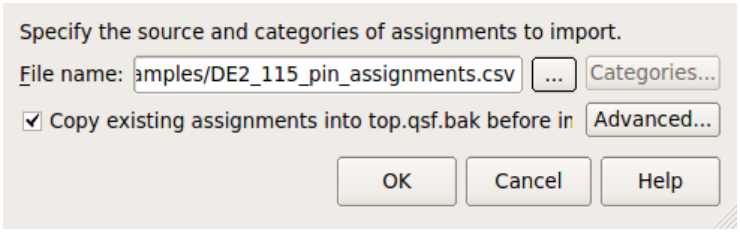


Figure 18: FPGA pin assignment.

5. Select File toolbar, select New.. , select Verilog HDL file

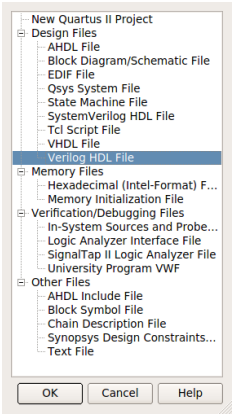
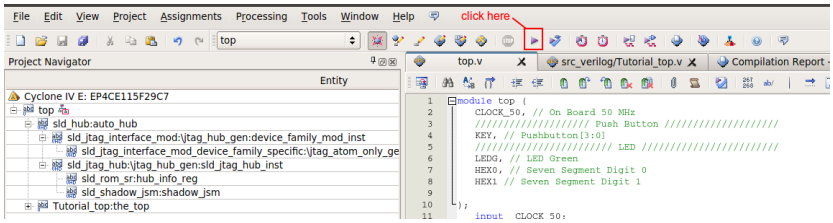


Figure 19: Add new file.

6. Copy bellow code to the new Verilog HDL file created and save it as top.v.

```
module top (
    CLOCK_50, // On Board 50 MHz
    ////////////////////// Push Button ////////////////////
    KEY, // Pushbutton[3:0]
    ////////////////////////// LED //////////////////////////
    LEDG, // LED Green
    HEX0, // Seven Segment Digit 0
    HEX1 // Seven Segment Digit 1

);
    input  CLOCK_50;
    output [0:0]LEDG;
    input  [2:0]KEY;
    output [6:0]HEX0, HEX1;

    wire reset;
    assign  reset=  ~KEY[0]; // use KEY[0] as reset button.
    assign  LEDG[0]     =   reset;

    Tutorial_top the_top(
        .aeMB_sys_ena_i(1'b1),
        .source_clk_in(CLOCK_50),
        .source_reset_in(reset),
        .ext_int_ext_int_i(KEY[2:1]),  //use KEY[1] &[2] as external
            intrrupt
        .hex0_port_o(HEX0),
        .hex1_port_o(HEX1)
    );

endmodule
```

7. Start compilation.



Software Development

1. Go to `mpsoc_work/SOC/Tutorial/sw` folder and open `main.c` file. Remove the file contents and replace with following C code:

```
#include "Tutorial.h"

// external intrrupt flag definition
```

```
#define ext_int_0   (1<<0)
#define ext_int_1   (1<<1)

#define UP  0
#define DOWN    1

#define RUN 0
#define FREAZE  1

const unsigned int seven_seg_tab [16] = {0x3F,0x06,0x5B,0x4F,0x66,0
    x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E, 0x79,0x71};

void timer_ISR_function( void );
void ext_int_ISR_function( void );
void display_on_seg(unsigned int);

unsigned int i;

void myISR( void ) __attribute__ ((interrupt_handler));
void myISR( void )
{
    if( int_ctrl_IPR & timer_INT )  timer_ISR_function();
    if( int_ctrl_IPR & ext_int_INT) ext_int_ISR_function();
    // Acknowledge Interrupts
    int_ctrl_IAR = int_ctrl_IPR;
}


unsigned int counter_mode= UP;
unsigned int counter_dsply= RUN;
void timer_ISR_function( void )
{
    i = (counter_mode == UP)? i+1 : i-1;
    if(i==100) i=0;
    if(i==-1) i=99;
    if (counter_dsply== RUN) display_on_seg(i);
    // Acknogledge Interrupt In Timer (Clear pending bit)
    timer_TCSR0 = timer_TCSR0;
}

inline void ext_int0_ISR_function(){
    counter_mode= (counter_mode==UP)? DOWN : UP;

}

inline void ext_int1_ISR_function(){
    counter_dsply= (counter_dsply== RUN) ? FREAZE : RUN;
}

void ext_int_ISR_function( void )
{
    if(ext_int_ISR  & ext_int_0)    ext_int0_ISR_function();
    if(ext_int_ISR  & ext_int_1)    ext_int1_ISR_function();
    // Clear any pending button interrupts
    ext_int_ISR     = ext_int_ISR;
```

```
}

void display_on_seg(unsigned int num){
    unsigned int seg1=num%10;
    unsigned int seg2=(num/10)%10;
    hex0_WRITE(~seven_seg_tab[seg1]);
    hex1_WRITE(~seven_seg_tab[seg2]);

}

int main(){

    i=0;
    hex0_WRITE(~seven_seg_tab[0]);
    hex1_WRITE(~seven_seg_tab[0]);

    // intrrupt setting
    ext_int_IER_RISE=ext_int_0 | ext_int_1;
    ext_int_GER =   0x3;

    timer_TCMP0 =   25000000;
    timer_TCSR0   = ( timer_EN | timer_INT_EN | timer_RST_ON_CMP);

    int_ctrl_IER=   ext_int_INT | timer_INT;
    int_ctrl_MER=   0x3;

    aembEnableInterrupts();
    while(1)
    {

    }//while
     return 0;
}
```

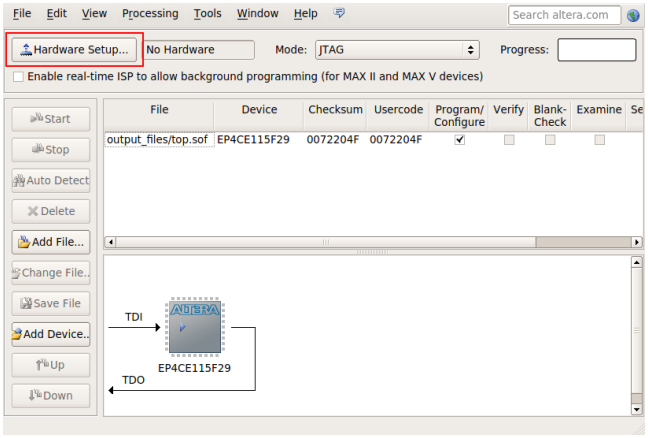
2. Open terminal in `mpsoc_work/SOC/Tutorial/sw` and run:

```
make
```

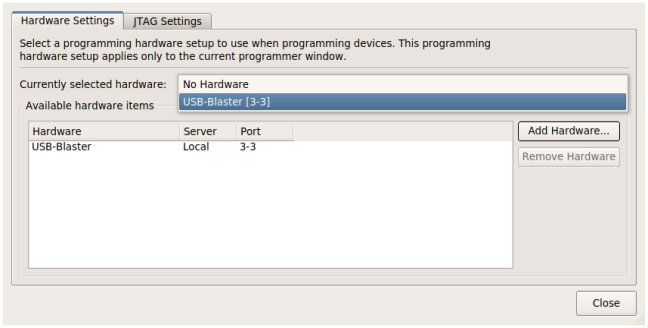
This will compile the C code using aeMB GNU toolchain. If every thing run successfully you must have `ram0.bin` in your `sw` directory.

Download Hardware file to the Board

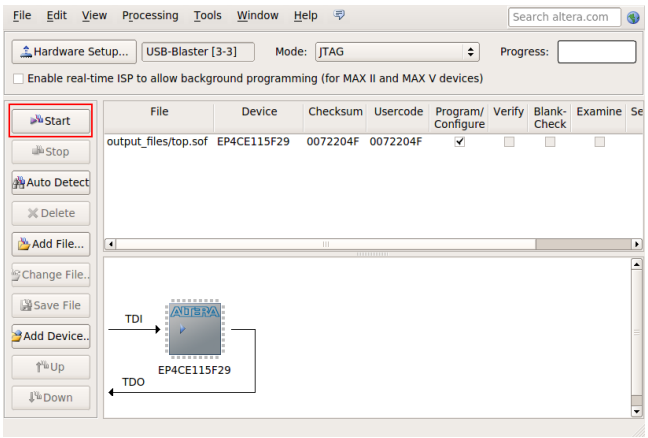
1. Power ON, and connect DE2 Board with PC.
2. Open Quartus Project earlier, at quartus toolbar, select Tools, then select Programmer.



3. On the Quartus Programmer click on Hardware Setup... then select USB-Blaster then Close.



4. At Quartus Programmer click Start to start download.



Download Software file to the Board

1. Open `sw` folder in terminal and run:

   ```
   sh program.sh
   ```

   This will upload the memory binary code to the board. Note if you have changed the RAM default parameter values during system development stage. You may need to modify following variables inside the `program.sh` file before running:

   ```
   OFSSET="0x00000000"
   BOUNDRY="0x00003fff"
   BINFILE="ram0.bin"
   VJTAG_INDEX="0"
   ```

   Check `sw/README` file for new values.

2. Reset the board using KEY[0]. The timer program should start working now. Check interrupts pins functionality.

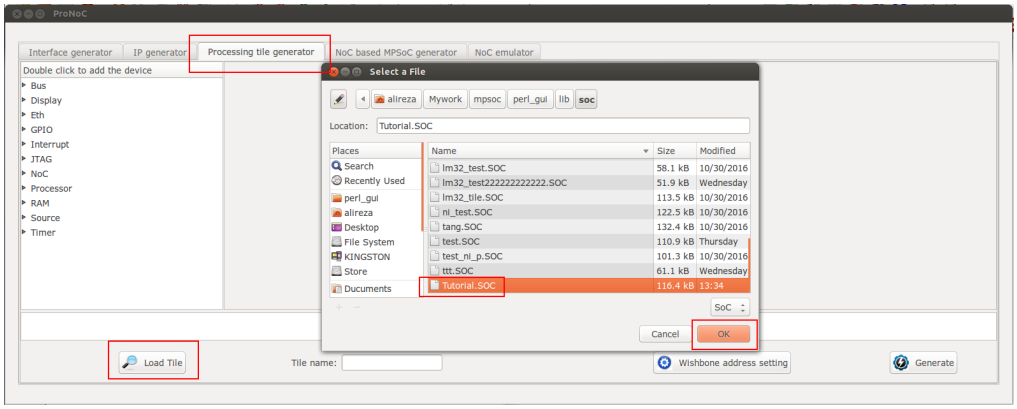
Modifying Tutorial SoC

Now let do some modifications to Tutorial SoC as follows:

1. Replace single port RAM with dual port RAM.
2. Connect first port of the memory to aeMB data port[2].
3. Connect second port of the memory to aeMB instruction port.

Load Tutorial SoC

Open ProNoC GUI in terminal click on Processing Tile Generator then click on `Load Tile`.

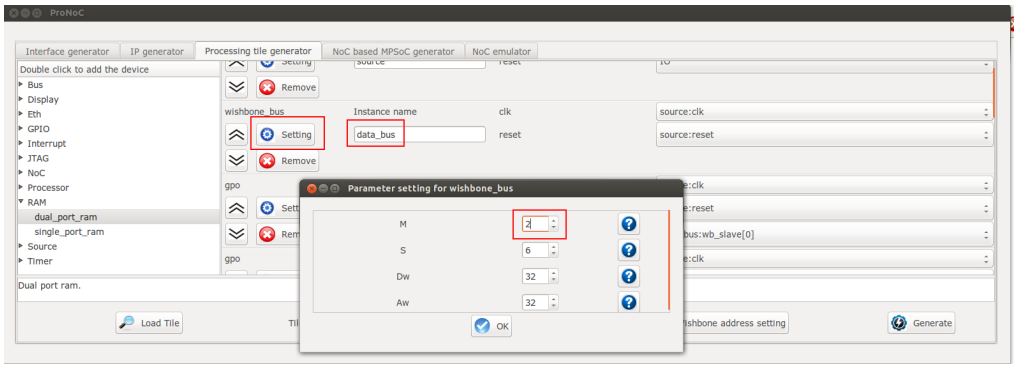


[2]Note that the byteenable is required for data port where it is only supported on the first port of Memory

Modify bus

In Tutorial SoC IPs box search for `wishbone_bus` and import following modifications:

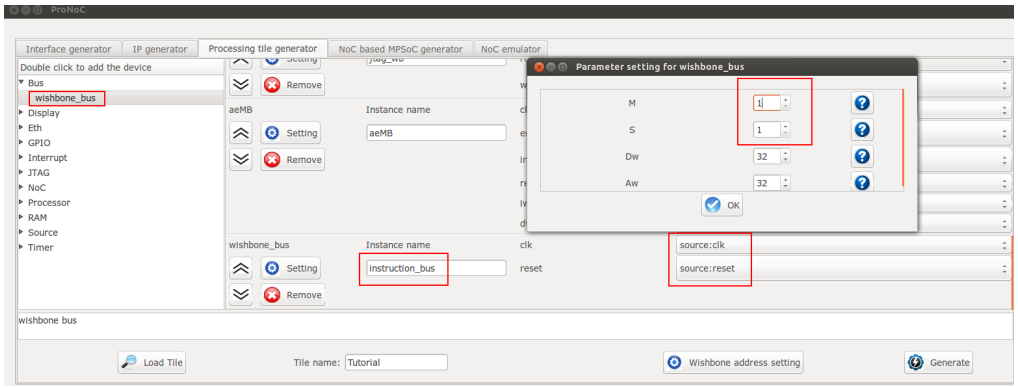
1. Press `setting` button and change `M` (Master interface number) to 2.
2. Rename this IP as `data_bus`.



Add instruction bus

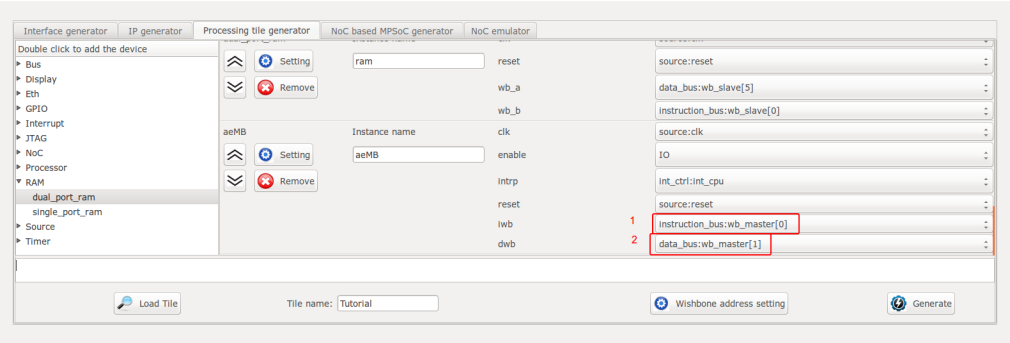
Add instruction bus for connecting aeMB instruction interface to dual port memory.

1. Click on `BUS` category and then double click on `wishbone_bus`.
2. Set both `M` master and `S` slave interface number as one.
3. Rename the instance name as `instruction_bus`.
4. Connect the clock and source interfaces to `clk_source` module.



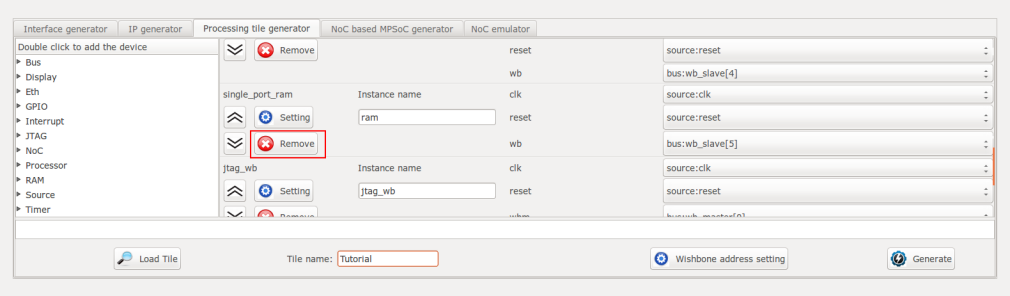
Modify aeMB's interfaces connection

1. Connect `iwb` to `instruction_bus:wb_master[0]`
2. Connect `dwb` to `data_bus:wb_master[1]`



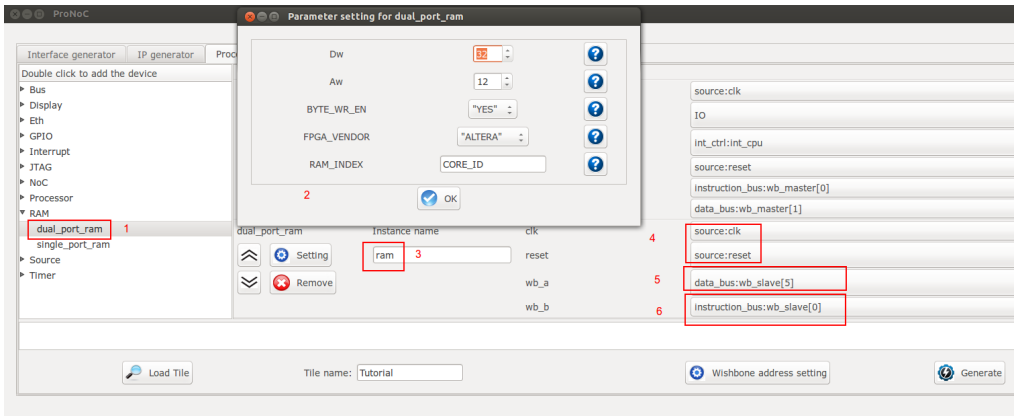
Remove single port RAM

Remove Single port ram by clicking on Remove button.



Add double port RAM

1. Click on `RAM` category and then double click on `dual_port_ram`.
2. Leave default parameter setting.
3. Rename the instance name as `instruction_bus`.
4. Connect the clock and source interfaces to `clk_source` module.
5. Connect `wb_a` (first Wishbone bus interface) to `data_bus:wb_slave[5]`.
6. Connect `wb_b` (second Wishbone bus interface) to `instruction_bus:wb_slave[0]`.



Generate the Modified SoC

Rename the Tile name as `Tutorial_1` then generate the code. Continue this tutorial with steps start from Section Compile the generated RTL code using Quartus II software.