ProNoC

Processing Tile Generator
Tutorial
Add Custom IP

Summary This tutorial teaches how to add custom intellectual property (IP) cores to **ProNoC Processing Tile Generator** using **IP Generator**. This tutorial uses a custom Verilog module for calculating the greatest common divisor (GCD) as an example hardware accelerator to be added to ProNoC IP library. This tutorial includes the software and hardware development of the desired Wishbone bus based system on chip (SoC) that is enhanced with GCD accelerator. This 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 and education 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 develop a custom Hardware Accelerator (HA) wishbone bus based IP core.
2. To extend ProNoC IP core library with a new IP core and software header file .

Greatest Common Divisor (GCD) Algorithm The Greatest Common Divisor (GCD) of two integers p and q, is the largest integer that divides both p and q. GCD can be obtained using Euclidean algorithm as follow:

Data: (p, q): A pair of 8-bit binary positive numbers.
Result: gcd: greatest common divisor
INITIALIZE;
while $p \neq q$ **do**
  if $p > q$ **then**
    $p = p - q$;
  end
  else if $p < q$ **then**
    $q = q - p$;
  end
  else
    $gcd = p$;
  end
end

Algorithm 1: Greatest Common Divisor algorithm.

The GCD flow chart:

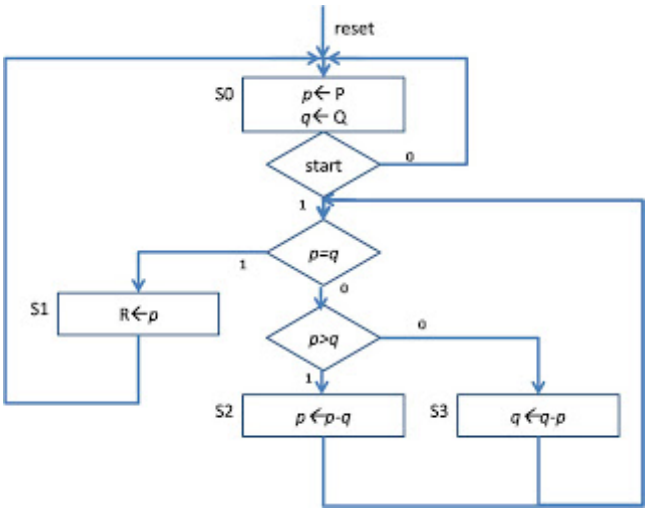


Figure 1: GCD flow chart.

Design GCD RTL

The GCD Verilog RTL code is as follows:

Listing 1: gcd.v

```
/************************
*   GCD
************************/

module gcd #(
parameter GCDw=32

)( clk, reset, enable, in1, in2, done, gcd);
    input clk, reset;
    input [GCDw-1 : 0] in1, in2;
    output [GCDw-1 : 0] gcd;
    input enable;
    output done;
    wire ldG, ldP, ldQ, selP0, selQ0, selP, selQ;
    wire AeqB, AltB;

    gcd_cu CU(
        .clk (clk),
        .reset (reset),
        .AeqB (AeqB),
        .AltB (AltB),
        .enable (enable),
        .ldG (ldG),
        .ldP (ldP),
        .ldQ (ldQ),
        .selP0 (selP0),
        .selQ0 (selQ0),
        .selP (selP),
```

```
        .selQ (selQ),
        .done (done)
    );


    gcd_dpu #(
        .GCDw(GCDw)
    )DPU(
        .clk (clk),
        .reset (reset),
        .in1 (in1),
        .in2 (in2),
        .gcd (gcd),
        .AeqB (AeqB),
        .AltB (AltB),
        .ldG  (ldG),
        .ldP (ldP),
        .ldQ (ldQ),
        .selP0 (selP0),
        .selQ0 (selQ0),
        .selP (selP),
        .selQ (selQ)
        );


endmodule



/************************
*   gcd_cu
************************/

module gcd_cu (clk, reset, ldG, ldP, ldQ, selP0, selQ0, selP, selQ, AeqB,
     AltB, done, enable);
    input clk, reset;
    input AeqB, AltB, enable;
    output ldG, ldP, ldQ, selP0, selQ0, selP, selQ, done;
    reg ldG, ldP, ldQ, selP0, selQ0, selP, selQ, done;


    //State encoding
    parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;
    reg [1:0] y;
    always @ (posedge reset or posedge clk) begin
        if (reset == 1) y <= S0;
        else begin
            case (y)
                S0: begin if (enable == 1) y <= S1;
                    else y <= S0;
                end
                S1: begin if (AeqB == 1) y <= S2;
                    else y <= S1;
                end
                S2: begin if (enable == 0) y <= S0;
```

```
                    else y <= S2;
                end
                default: y <= S0;
            endcase
        end
    end


    always @ (y or enable or AeqB or AltB) begin
        ldG = 1'b0; ldP = 1'b0; ldQ = 1'b0;
        selP0 = 1'b0;
        selQ0 = 1'b0;
        selP = 1'b0;
        selQ = 1'b0;
        done = 1'b0;
        case (y)
        S0: begin
            done = 1'b1;
            if (enable == 1)begin
                selP0 = 1; ldP = 1; selQ0 = 1; ldQ = 1; done = 0;
            end
        end

        S1: begin
            if (AeqB == 1) begin
                ldG = 1;
                done = 1;
            end
            else if (AltB == 1) begin
                ldQ = 1;
            end
            else begin
                ldP = 1; selP = 1; selQ = 1;
            end
        end
        S2: begin
            ldG = 1;
            done = 1;
        end
        default: ;
        endcase
        end
    endmodule



/************************
*   gcd_dpu
************************/

module gcd_dpu #(
    parameter GCDw=32

)( clk, reset, in1, in2, gcd, ldG, ldP, ldQ, selP0, selQ0, selP, selQ,
    AeqB, AltB);
    input clk, reset;
```

```
    input [GCDw-1:0] in1, in2;
    output [GCDw-1:0]  gcd;
    input ldG, ldP, ldQ, selP0, selQ0, selP, selQ;
    output AeqB, AltB;
    reg [GCDw-1:0]  reg_P, reg_Q;
    wire [GCDw-1:0]  wire_ALU;
    reg [GCDw-1:0]  gcd;
    wire AeqB, AltB;
    //RegP with Multiplex 2:1
    always @ (posedge clk or posedge reset)begin
        if (reset == 1) reg_P <= 0;
        else begin
            if (ldP == 1)begin
                if (selP0==1) reg_P <= in1;
                else reg_P <= wire_ALU;
            end
        end
    end

        //RegQ with Multiplex 2:1
    always @ (posedge clk or posedge reset) begin
        if (reset == 1) reg_Q <= 0;
        else begin
            if (ldQ == 1)begin
                if (selQ0==1) reg_Q <= in2;
                else reg_Q <= wire_ALU;
            end
        end
    end

    //RegG with enable signal
    always @ (posedge clk or posedge reset)begin
        if (reset == 1) gcd <= {GCDw{1'b0}};
        else begin
            if (ldG == 1) gcd <= reg_P;
        end
    end

    //Comparator
    assign AeqB = (reg_P == reg_Q)? 1'b1 : 1'b0;
    assign AltB = (reg_P < reg_Q) ? 1'b1 : 1'b0;

    //Subtractor
    assign wire_ALU = ((selP == 1) & (selQ == 1)) ? (reg_P - reg_Q) : (
        reg_Q - reg_P);
endmodule
```

Create `mpsoc/src_peripheral/GCD` directory and then copy the above `gcd.v` file inside it.

GCD Simulation

In order to verify GCD hardware module, we use Verilator simulator. Optionally you can use Modelsim as well.

1. If you have not yet installed Verilator simulator on your system run the following command in terminal

```
sudo apt-get install verilator
```

2. Open terminal in the folder which you have created `gcd.V` file and run:

```
verilator --cc  gcd.v
```

   If your code is successfully verilated, you will have an `obj_dir` directory that includes all generated GCD object files.

3. Open `obj_dir` folder and create `testbench.cpp` inside it:

Listing 2: testbench.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <verilated.h>
#include "Vgcd.h"    // From Verilating "gcd.v"

unsigned int input1[10] ={136, 25, 33220, 3627, 3450, 9375,  199317,
     157620, 5694235, 199307 };
unsigned int input2[10] ={248, 50, 2200,  4581, 6540, 61575, 103443,
     238844, 239871,  903443};
unsigned int expt_gcd[10] ={8,   25, 220,   9,    30,   75,    2523,
       284,    2161,    1};

Vgcd *gcd // Instantiation of module

unsigned int main_time = 0; // Current simulation time
int run;
unsigned int i=0,passed=1;

int main(int argc, char** argv) {
    Verilated::commandArgs(argc, argv);   // Remember args
    gcd = new Vgcd;
    /*********************
    *   initialize input
    *********************/
    gcd->reset=1;
    gcd->enable=0;
        gcd->in1=0;
        gcd->in2=0;
    main_time=0;
    run=0;

    while (!Verilated::gotFinish() && i<10) {

        if (main_time & 0x1) {
            gcd-> clk = 0;
            if(gcd-> done==1  && run>6){
                printf("%u : GCD(%u,%u)= %d\t",main_time,gcd->in1,
                    gcd->in2, gcd->gcd);
                if(gcd->gcd ==  expt_gcd[i]) printf(" Matched\n");
                else {passed=0; printf(" Error:Miss-matched\n");}
                i++;
                run=0;
```

```
            }
            if(gcd-> enable == 1 && run==5){
                gcd-> enable = 0;
            }
            if(run==4 && gcd->reset==0){
                gcd-> enable = 1;
                gcd-> in1 =  input1[i];
                gcd-> in2 = input2[i];


            }
            if (main_time >= 10 ) {
                gcd->reset=0;
                run++;
            }


        }//if
        else {
            gcd-> clk = 1; // Toggle clock

        }//else

        gcd->eval();
        main_time++;


    }
    if(passed) printf(  " ********** GCD Testing passed
        ************\n") ;
    else printf(  " ********** GCD Testing failed ************\n");
    gcd->final();

}

double sc_time_stamp () {       // Called by $time in Verilog
    return main_time;
}
```

4. Now create a Makefile inside `obj_dir`:

Listing 3: Makefile

```
# -*- Makefile -*-


default: sim

MUDUL = Vgcd


include Vgcd.mk

lib:
    $(MAKE) -f $(MUDUL).mk



###################################
# Compile flags

CPPFLAGS += -DVL_DEBUG=1
ifeq ($(CFG_WITH_CCWARN),yes)   # Local... Else don't burden users
CPPFLAGS += -DVL_THREADED=1
CPPFLAGS += -W -Werror -Wall
endif
#############################
# Linking final exe -- presumes have a sim_main.cpp


sim:    testbench.o $(VK_GLOBAL_OBJS) $(MUDUL)__ALL.a
    $(LINK) $(LDFLAGS) -g $^ $(LOADLIBES) $(LDLIBS) -o testbench $(LIBS)
        -Wall -O3 2>&1 | c++filt

testbench.o: testbench.cpp $(MUDUL).h

clean:
    rm *.o *.a main
```

5. Now to compile the testbench code open terminal in `obj_dir` directory and run:

```
make
```

Sample output:

```
g++  -I. -MMD -I/usr/local/share/verilator/include -I/usr/local/
    share/verilator/include/vltstd -DVL_PRINTF=printf -DVM_TRACE=0
     -DVM_COVERAGE=0       -DVL_DEBUG=1  -c -o testbench.o
    testbench.cpp
g++    -g testbench.o verilated.o Vgcd__ALL.a    -o testbench -lm
    -lstdc++ -Wall -O3 2>&1 | c++filt
```

This must generate a binary executable file inside `obj_dir` named as `testbench`.

6. To run the simulation run:

```
./testbench
```

Expected output:

```
37 : GCD(136,248)= 8     Matched
51 : GCD(25,50)= 25      Matched
109 : GCD(33220,2200)= 220       Matched
177 : GCD(3627,4581)= 9  Matched
217 : GCD(3450,6540)= 30         Matched
263 : GCD(9375,61575)= 75        Matched
305 : GCD(199317,103443)= 2523   Matched
365 : GCD(157620,238844)= 284    Matched
445 : GCD(5694235,239871)= 2161  Matched
557 : GCD(199307,903443)= 1      Matched
 ********** GCD Testing passed ************
```

Add Wishbone bus interface to GCD

After the GCD core is functionality verified, next is to add Wishbone bus interface to GCD hardware. This interface module provides memory-mapped access of GCD module's input/output ports for the processor. The memory-mapped addresses are illustrated in Table 1:

Table 1: GCD_IP internal register addresses.

Offset Address	Name	Description	Mode
0	DONE	Holds the value of done output port	Read-only
1	IN1	Write on GCD's module first input variable	Write-only
2	IN2	Write on GCD's module second input variable. Writing on this register will trigger the GCD's enable port	Write-only
3	GCD	Holds the generated GCD value	Read-only

Create following file alongside with `obj_dir`

Listing 4: gcd_ip.v

```
module gcd_ip#(
    parameter GCDw=32,
    parameter Dw =GCDw,
    parameter Aw =5,
    parameter TAGw =3,
    parameter SELw =4
)
(
    clk,
    reset,
    //wishbone bus interface
    s_dat_i,
    s_sel_i,
    s_addr_i,
```

```
    s_tag_i,
    s_stb_i,
    s_cyc_i,
    s_we_i,
    s_dat_o,
    s_ack_o,
    s_err_o,
    s_rty_o

);
    input                   clk;
    input                   reset;

    //wishbone bus interface
    input       [Dw-1       :   0]      s_dat_i;
    input       [SELw-1     :   0]      s_sel_i;
    input       [Aw-1       :   0]      s_addr_i;
    input       [TAGw-1     :   0]      s_tag_i;
    input                               s_stb_i;
    input                               s_cyc_i;
    input                               s_we_i;

    output      [Dw-1       :   0]      s_dat_o;
    output   reg                        s_ack_o;
    output                              s_err_o;
    output                              s_rty_o;

 //Wishbone bus registers address
     localparam DONE_REG_ADDR=0;
     localparam IN_1_REG_ADDR=1;
     localparam IN_2_REG_ADDR=2;
     localparam GCD_REG_ADDR=3;

    assign s_err_o        =   1'b0;
    assign s_rty_o        =   1'b0;

    wire[GCDw-1 :0] gcd;
    reg [GCDw-1 :0] readdata,in1,in2;
    wire done;

    assign s_dat_o =readdata;

    always @ (posedge clk or posedge reset) begin
        if(reset) begin
            s_ack_o   <=  1'b0;
        end else begin
            s_ack_o   <=   (s_stb_i & ~s_ack_o);
        end //reset
    end//always

    always @ (posedge clk or posedge reset) begin
        if(reset) begin
            readdata <= 0;
            in1 <= 0;
            in2 <= 0;
        end else begin
```

```
            if(s_stb_i && s_we_i) begin  //write regiters
                if(s_addr_i==IN_1_REG_ADDR[Aw-1: 0]) in1 <= s_dat_i;
                else if(s_addr_i==IN_2_REG_ADDR[Aw-1: 0]) in2 <= s_dat_i;
            end //sa_stb_i && sa_we_i
            else begin //read registers
                if (s_addr_i==DONE_REG_ADDR) readdata<={{GCDw{1'b0}},done
                    };
                if (s_addr_i==GCD_REG_ADDR)  readdata<=gcd;
            end
        end //reset
    end//always

    // start gcd calculation by writiing on in2 register
    wire start=(s_stb_i && s_we_i && (s_addr_i==IN_2_REG_ADDR[Aw-1: 0]));
    reg ps,ns;
    reg gcd_reset,gcd_reset_next;
    reg gcd_en,gcd_en_next;

    always @ (posedge clk or posedge reset) begin
        if(reset) begin
            ps<=1'b0;
            gcd_reset<=1'b1;
            gcd_en<=1'b0;
        end else begin
            ps<=ns;
            gcd_en<=gcd_en_next;
            gcd_reset<=gcd_reset_next;
        end
    end

    always @(*)begin
        gcd_reset_next=1'b0;
        gcd_en_next=1'b0;
        ns=ps;
        case(ps)
            1'b0:begin
                if(start) begin
                    ns=1'b1;
                    gcd_reset_next=1'b1;
                end
            end
            1'b1:begin
                gcd_en_next=1'b1;
                ns=1'b0;
            end
        endcase
    end


    gcd #(
        .GCDw(GCDw)
    ) the_gcd
    (
        .clk (clk),
        .reset (gcd_reset),
        .enable (gcd_en),
```

```
        .in1 (in1),
        .in2 (in2),
        .done (done),
        .gcd (gcd)
    );

endmodule
```

Add custom wishbone-based IP core to ProNoC Library

In this section, we show how to add previously generated GCD IP core to ProNoC library. However, this can be applied to any other wishbone based IP core.

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

   ```
   ./ProNoC.pl
   ```

   It should open The GUI interface as follows:

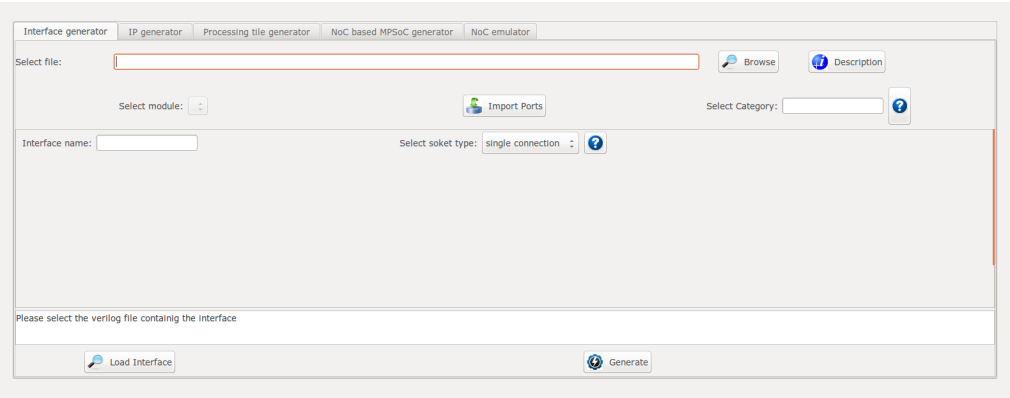


Figure 2: ProNoC GUI first page snapshot.

2. Then select the `IP Generator`:

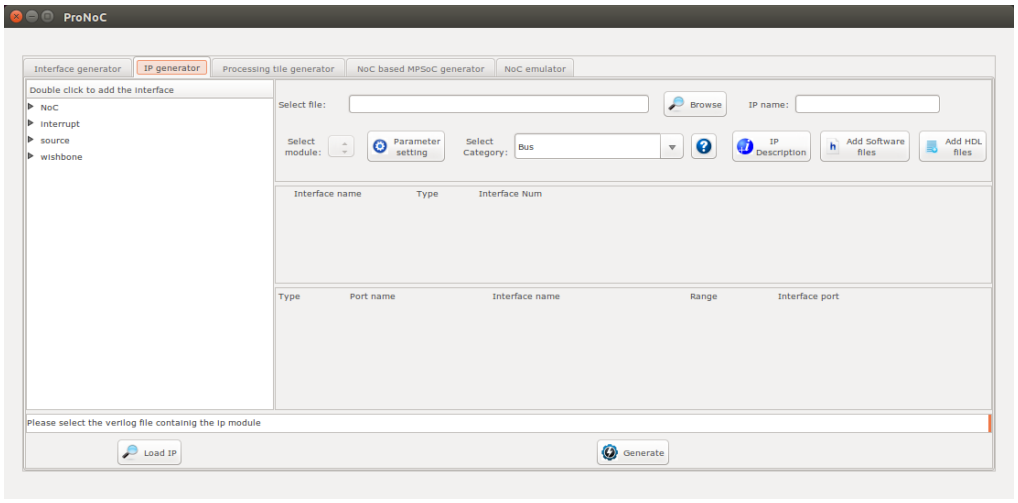


Figure 3: ProNoC New IP Generator snapshot.

3. Click on `Browse` and select `gcd_ip.v` file.
4. Enter `GCD` as category name.
5. Enter `gcd` as IP name.

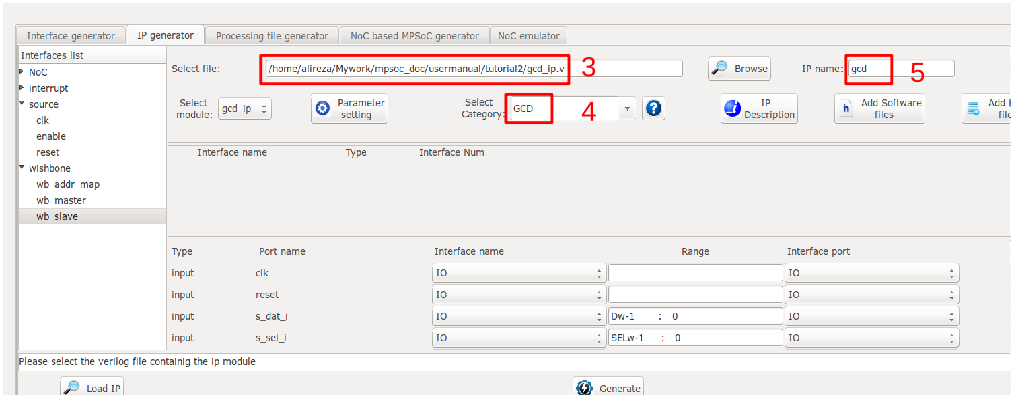


Figure 4: Select gcd.v file.

6. The `gcd_ip.v` file has one parameter named as `GCDw` which we want to be redefined by the end user during IP call time. To define the appropriate GUI interface for this parameter click on `parameter setting`.
7. In the newly open window, select `Combo-box` as widget type.

8. Enter `8,16,32` as widget content. It will allow the user to select one of these three values for this parameter during Processing tile generation.

9. In the next Combo-box select `Localparam`.

10. Click on button to add parameter information.

11. Enter parameter information as `GCD's Input/output width in bits` then press `ok`.

12. In parameter setting window press `ok` to add your setting.

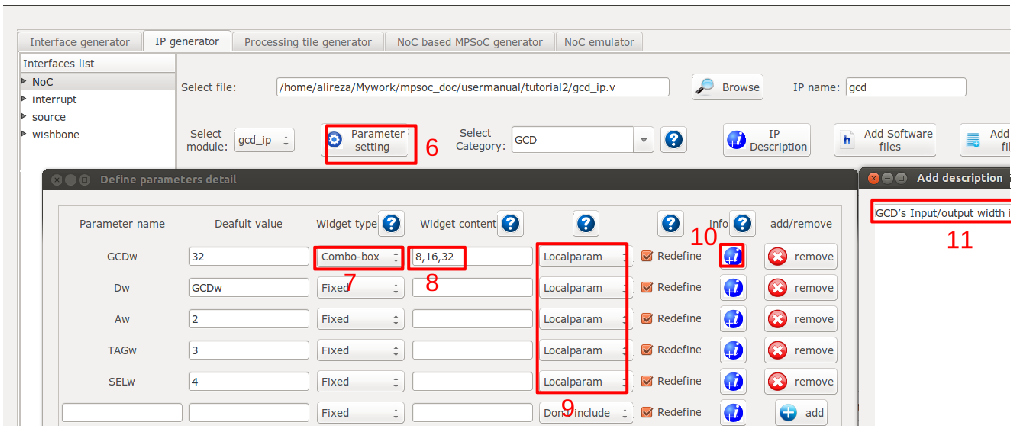


Figure 5: Select gcd.v file.

13. In `Interface list` window expand `source` and `wishbone` categories. Then double click on `clk`, `reset` and `wishbone` to add them to the GCD IP library.

14. In wishbone bus interface row click on button.

15. Select `custom devices` for wishbone address range.

16. Set block address range as 5. This results in allocating 32 Bytes for each instances of this module. The memory size must be selected equal or greater than the actual IP's internal register size. (GCD has four 32-bit internal register which is equal to 16 Bytes).

17. Press `ok`.

    Now we need to map each module individual port to its appropriate interface port. By selecting the interface name, the application automatically selects the nearest port which match with module port name. For this example it can match all ports correctly. However in general you may also needed to adjust the port name as well.

18. Select `plug:clk` for `clk` port.

19. Select `plug:reset` for `reset` port.

20. Connect all other ports to `plug:wb_slave`. port.

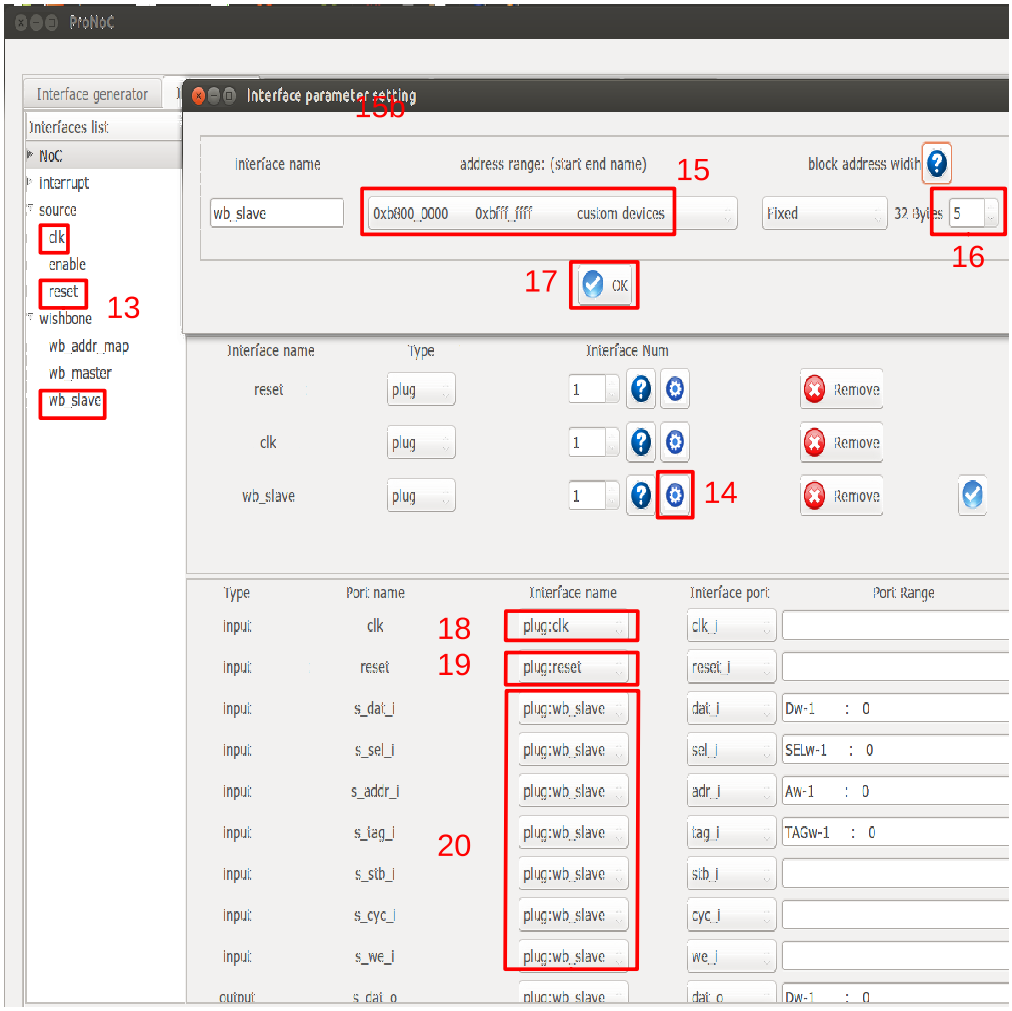


Figure 6: Select gcd.v file.

21. Click on `Add HDL Files` button.

22. In front of `Select file(s)` click on `Browse`.

23. Select `gcd.v` and `gcd_ip.v` files and press `ok`.

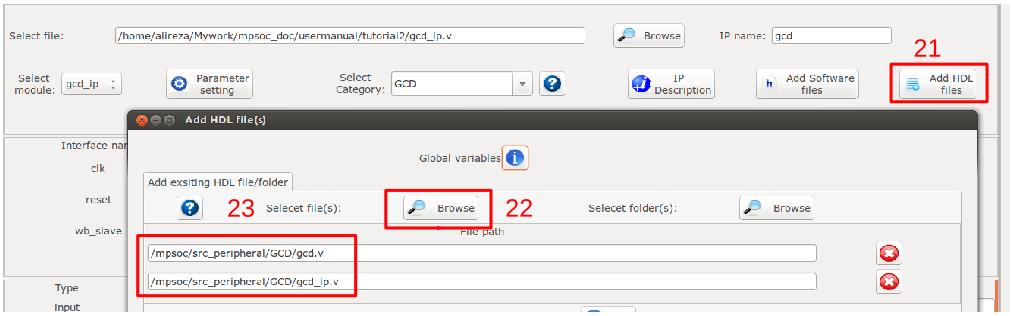


Figure 7: Select gcd.v file.

24. Click on Add `software files` button. In the newly opened window, you can add IP core's software library/header files. The listed files/folder here will be copied in generated SoC project folder inside `sw` directory.

25. Click on `Add to system.h` tab.

26. Copy following text on the new tab, then click on `Save` button.

```
#define ${IP}_DONE_ADDR (*((volatile unsigned int *) ($BASE)))
#define ${IP}_IN_1_ADDR (*((volatile unsigned int *) ($BASE+4)
    ))
#define ${IP}_IN_2_ADDR (*((volatile unsigned int *) ($BASE+8)
    ))
#define ${IP}_GCD_ADDR (*((volatile unsigned int *) ($BASE+12)
    ))

#define ${IP}_IN1_WRITE(value)  ${IP}_IN_1_ADDR=value
#define ${IP}_IN2_WRITE(value)  ${IP}_IN_2_ADDR=value

#define   ${IP}_DONE_READ()     ${IP}_DONE_ADDR
#define   ${IP}_GCD_READ()       ${IP}_GCD_ADDR

unsigned int gcd_hardware ( unsigned int p, unsigned int q ){
        GCD_IN1_WRITE(p) ;
        GCD_IN2_WRITE(q) ;
        while (GCD_DONE_READ()!=1);
        return GCD_GCD_READ();
}
```

The entered text here will be added to the `[SoC_name].h` file. This file contains all IP cores' wishbone bus addresses, functions and header files. You can use some global variables with `$[variable_name]` format here such as all IP core parameters and IP core Verilog instance name. These variables will be replaced with their exact values during SoC generation time). In this example, we used variable `${IP}` which is the IP core's instance name. Hence, in case this IP core

is called more than once in any SoC, each instance has distinguished addresses and functions.

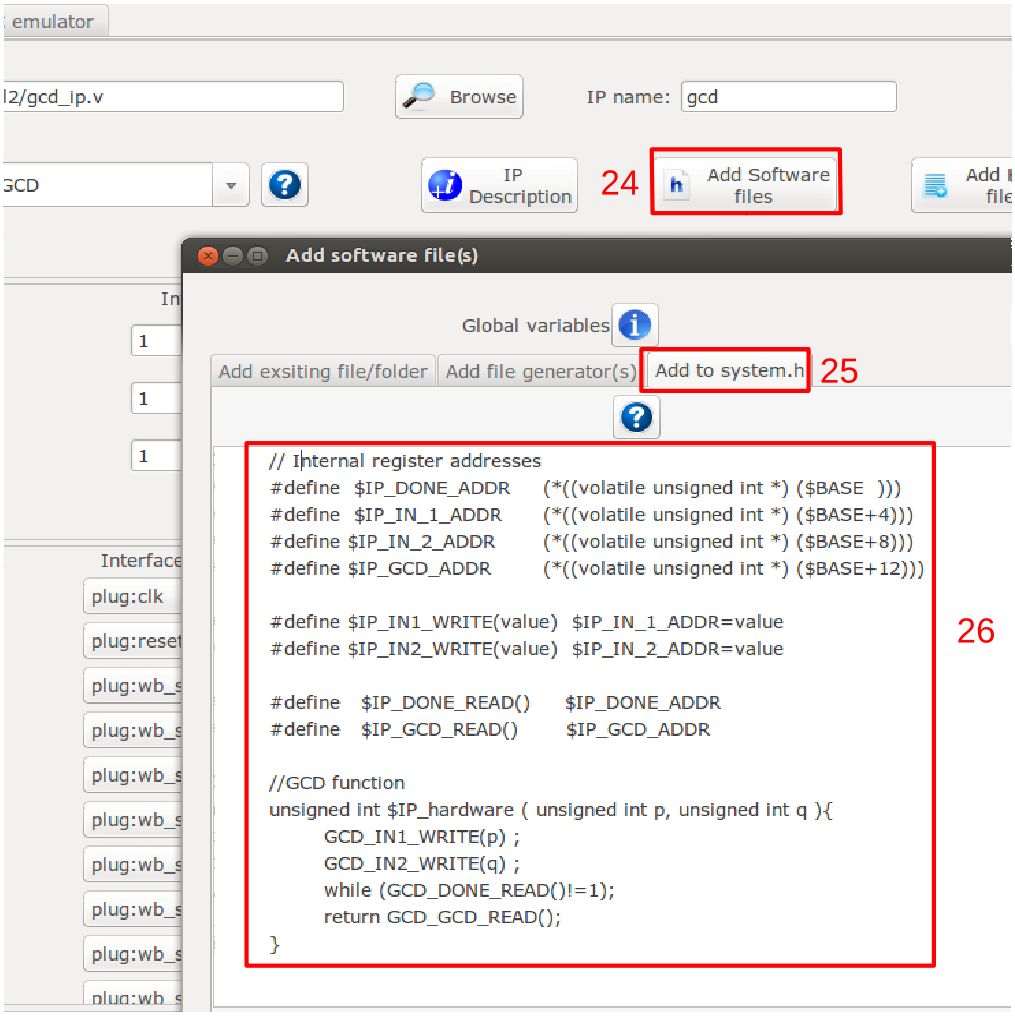


Figure 8: Add GCD software files.

27. Click on `Generate` to add the GCD IP core to the library.

Figure 9: Generate GCD IP.

Generate a new SoC enhanced with new IP core (GCD)

In this section, we aim to generate an embedded SoC enhanced using generated GCD IP core. The desired SoC schematic is shown in Figure 10.

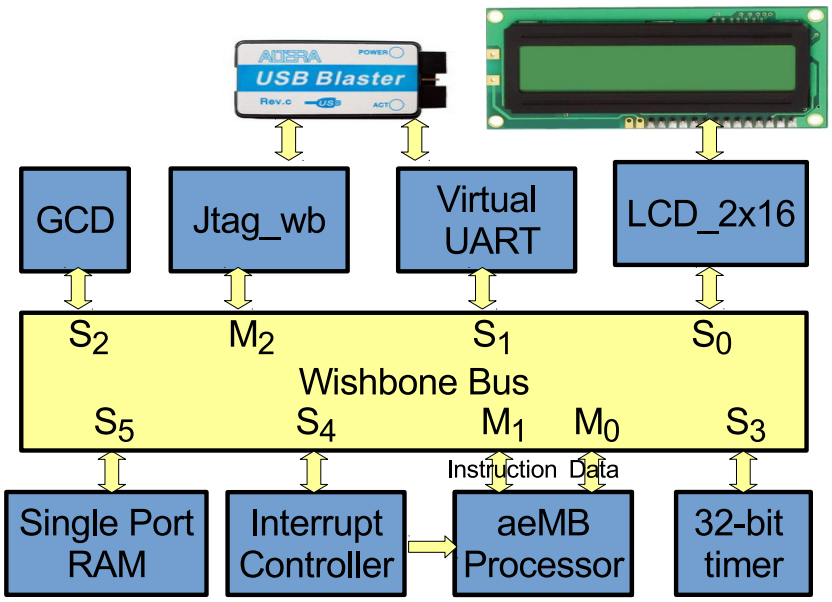


Figure 10: desired SoC with GCD IP core.

1. In ProNoC GUI Click on `Processing Tile Generator`. This tools facilitate the generation of a custom SoC using a list of available IP cores. Add all required IP cores according to the following stages:

   (a) Click on IP core category name to see the list of its containing IP cores.

   (b) Double click on each IP core name to add the IP core to the SoC. Add all IP cores listed in Table 2 first then continue with the next step.

   (c) Click on `Setting` button to open the IP core parameter setting window.

   (d) Adjust IP core parameters according to Table 2.

   (e) Rename the IP core instance name according to Table 2.

   (f) Connect IP cores interfaces as listed in Table 2.

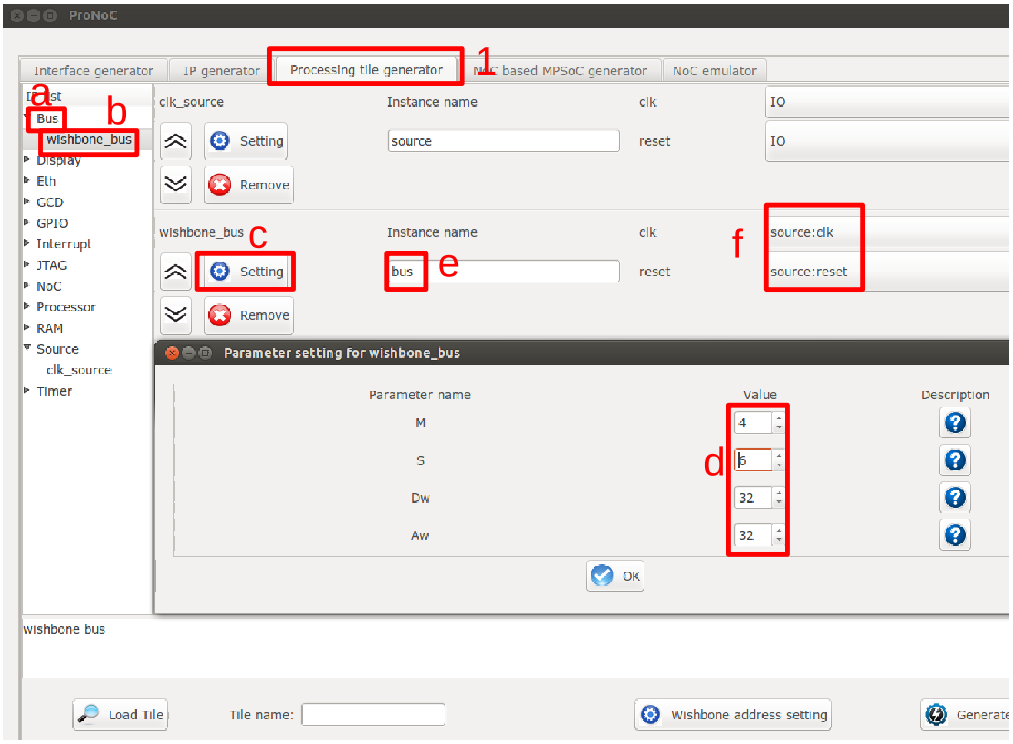

ProNoC
Interface generator
IP generator
Processing tile generator
NoC based MPSoC generator
NoC emulator
1
a
b
Bus
wishbone_bus
Display
Eth
GCD
GPIO
Interrupt
JTAG
NoC
Processor
RAM
Source
clk_source
Timer
clk_source
Setting
Remove
Instance name
source
clk
IO
reset
IO
wishbone_bus
c
Setting
Remove
Instance name
bus
e
clk
source:clk
f
reset
source:reset
Parameter setting for wishbone_bus
Parameter name
Value
Description
M
4
S
d
5
Dw
32
Aw
32
OK
wishbone bus
Load file
file name:
Wishbone address setting
Generate

Figure 11

Table 2: GCD SoC IP core list and setting.

Category	IP name	Parameter	Instance name	Interface connection
Source	clk_source	-	source	clk → IO reset → IO
Bus	wishbone_bus	M → 3 S → 6 Dw → 32 Aw → 32	bus	clk → source:clk reset → source:reset
RAM	single_port_ram	Dw → 32 Aw → 12 BYTE_WR_EN → "YES" FPGA_VENDOR → "ALTERA" JTAG_CONNECT → "DISABLED" JTAG_INDEX → CORE_ID	ram	clk → source:clk reset → source:reset wb → bus:wb_slave[0]
Display	lcd_2x16	CLK_MHZ → 50	lcd	clk → source:clk reset → source:reset wb → bus:wb_slave[1]
Interrupt	int_ctrl	INT_NUM → 1	int_ctrl	clk → source:clk reset → source:reset wb → bus:wb_slave[2]
JTAG	jtag_wb	VJTAG_INDEX → CORE_ID	jtag_wb	clk → source:clk reset → source:reset wbm → bus:wb_master[3]
Processor	aeMB	STACK_SIZE → 0X400 HEAP_SIZE → 0x400	aeMB	clk → source:clk reset → source:reset iwb → bus:wb_master[0] dwb → bus:wb_master[1] enable → IO intrp → int_ctrl:int_cpu
Timer	timer	-	timer	clk → source:clk reset → source:reset wb → bus:wb_slave[3] intrp → in_ctrl:int_periph[0]
JTAG	altera_jtag_uart	-	uart	clk → source:clk reset → source:reset intrpt → NC wb → bus:wb_slave[4]

2. Add the new GCD IP to SoC.

Table 3: GCD SoC IP core list and setting.

Category	IP name	Parameter	Instance name	Interface connection
GCD	gcd	GCDw → 32	gcd	clk → source:clk reset → source:reset wb → bus:wb_slave[5]

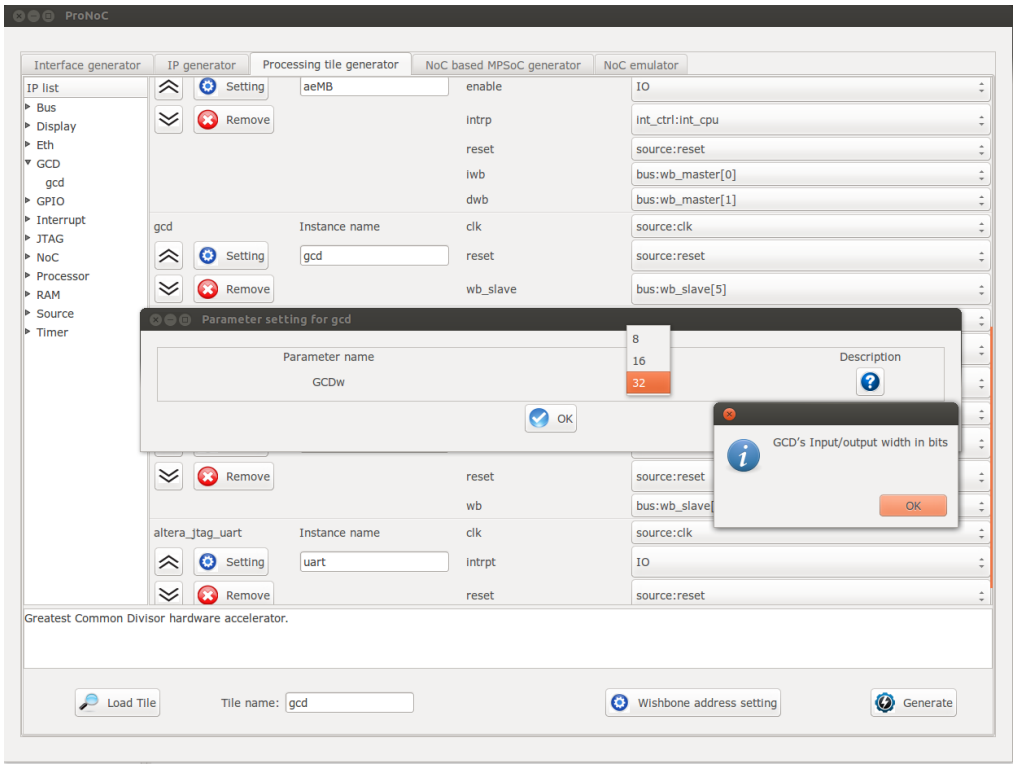


Figure 12

3. Set the tile name as `gcd_soc`.

4. Press generate button. This must generate a folder in `mpsoc_work/SOC/gcd_soc`.

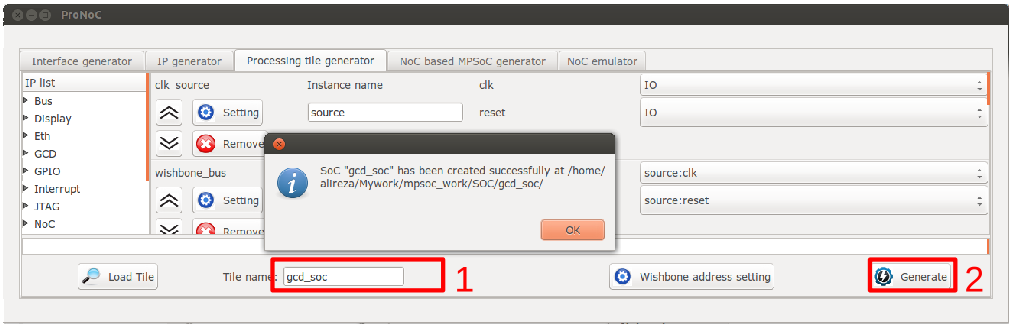


Figure 13

Compile the generated RTL code using Quartus II software

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

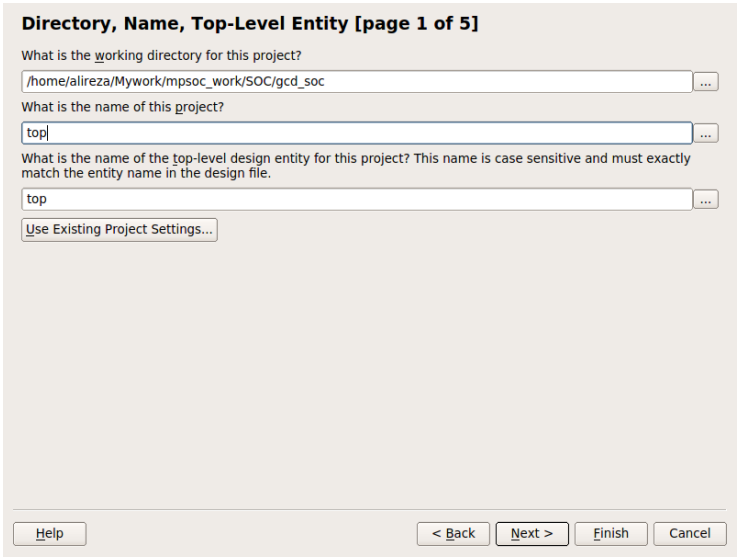


Figure 14: New Project setting.

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

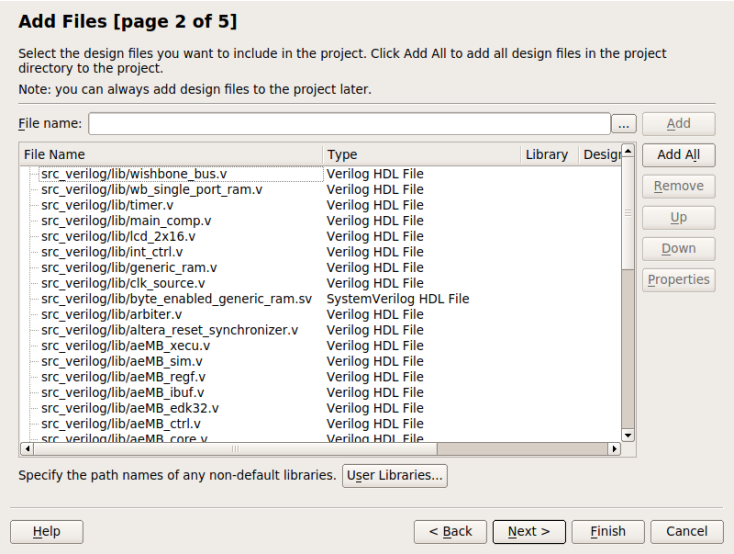


Figure 15: Add Tutorial RTL Codes to the Quartus.

(e) At Family and Device setting, select "Cyclone IV E" and "EP4CE115F29C7" then click Finish.

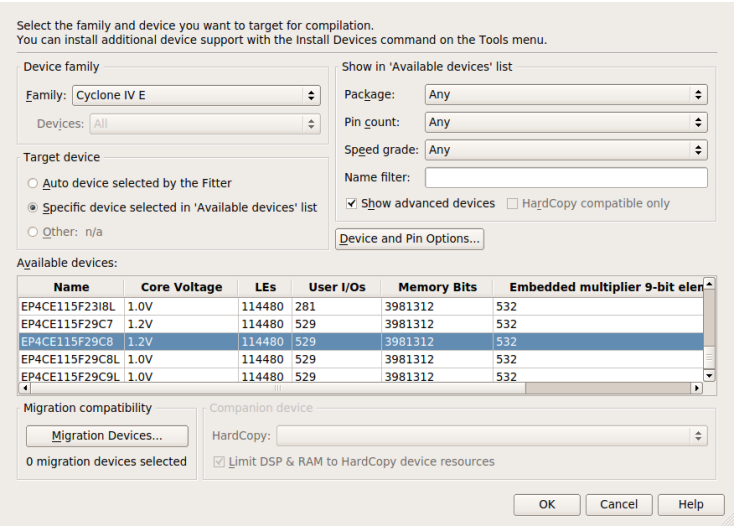


Figure 16: FPGA device selection.

(f) 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.

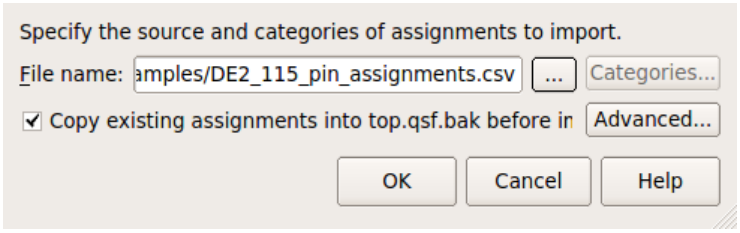


Figure 17: FPGA pin assignment.

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

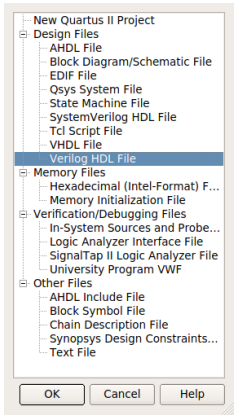


Figure 18: Add new file.

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

Listing 5: gcd_ip.v

```
module top (
    CLOCK_50, // On Board 50 MHz
    ////////////////////// Push Button ////////////////////
    KEY, // Pushbutton[3:0]
    ////////////////////////// LED //////////////////////////
    LEDG, // LED Green
    // LCD
    LCD_BLON,
    LCD_DATA,
    LCD_EN,
    LCD_ON,
    LCD_RS,
    LCD_RW

);
    input  CLOCK_50;
    output [0:0]LEDG;
    input [0:0]KEY;
    output                      LCD_BLON;
    inout              [7:0]        LCD_DATA;
    output                      LCD_EN;
    output                      LCD_ON;
    output                      LCD_RS;
    output                      LCD_RW;


    ///////////////////////////////////////////
    // LCD config
    assign LCD_BLON = 1'b0; // not supported
    assign LCD_ON = 1'b1; // alwasy on

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



    gcd_soc_top the_top(
        .aeMB_sys_ena_i(1'b1),
        .source_clk_in(CLOCK_50),
        .source_reset_in(reset),
        .lcd_lcd_data(LCD_DATA),
        .lcd_lcd_en(LCD_EN),
        .lcd_lcd_rs(LCD_RS),
        .lcd_lcd_rw(LCD_RW)
    );




endmodule
```

4. 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 "gcd_soc.h"


#define timer_start()    timer_TCSR0|=timer_EN
#define timer_stop()     timer_TCSR0&=~timer_EN
#define timer_reset()    timer_TLR0=0
#define timer_read()     timer_TLR0

unsigned int gcd_software ( unsigned int p, unsigned int q ){
    while (p != q) {
        if (p > q) p=p-q;
        else if (p < q) q=q-p;
    }
    return p;
}


int itoa(int dataIn, char* bffr, int radix){
    int temp_dataIn;
    temp_dataIn = dataIn;
    int stringLen=1;
    int len;
    while ((int)temp_dataIn/radix != 0){
        temp_dataIn = (int)temp_dataIn/radix;
        stringLen++;
    }
    len=stringLen;
    //printf("stringLen = %d\n", stringLen);
    temp_dataIn = dataIn;
    do{
        *(bffr+stringLen-1) = (temp_dataIn%radix)+'0';
        temp_dataIn = (int) temp_dataIn / radix;
    }while(stringLen--);
    return len;
}

void show_on_lcd (int A,int B, int C){
    char buff[32];
    int len;
    lcd_clr_dsply();
    lcd_goto_line(1);
    len=itoa(A, buff,10);
    lcd_show_text(buff,len);
```

```
    lcd_show_character(',');
    len=itoa(B, buff,10);
    lcd_show_text(buff,len);
    lcd_goto_line(2);
    len=itoa(C, buff,10);
    lcd_show_text(buff,len);
}


int main(){
    int A,B,C,D;

    unsigned int t_hw,t_sw;
    unsigned int speed;
    printf ("GCD test application\n");
    timer_TCSR0|=timer_EN;
    lcd_init();
    while(1){
        printf ("Enter number #1:\n");
        jtag_scanint(&A);
        printf ("Enter number #2:\n");
        jtag_scanint(&B);
        timer_reset();
        timer_start();
        C=gcd_hardware ( A, B);
        timer_stop();
        t_hw=timer_read();

        timer_reset();
        timer_start();
        D=gcd_software ( A, B);
        timer_stop();
        t_sw=timer_read();
        speed=(t_sw*10)/(t_hw);
        printf ("GCD_hardware (%d,%d) = %d\t clock_num=%d\n",A,B,C,
            t_hw);
        printf ("GCD_software (%d,%d) = %d\t clock_num=%d\n",A,B,D,
            t_sw);
        printf ("spead up=%d.%d times\n",speed/10,speed%10);

        show_on_lcd (A, B, C);



    }

return 0;
}
```

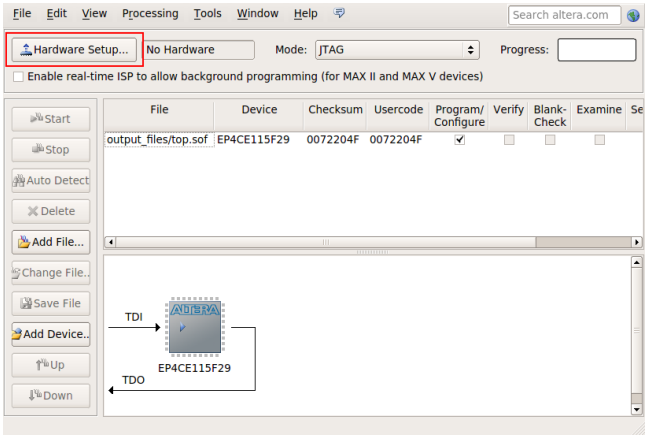
2. Open terminal in `mpsoc_work/SOC/Tutorial/sw` and run:

```
make
```

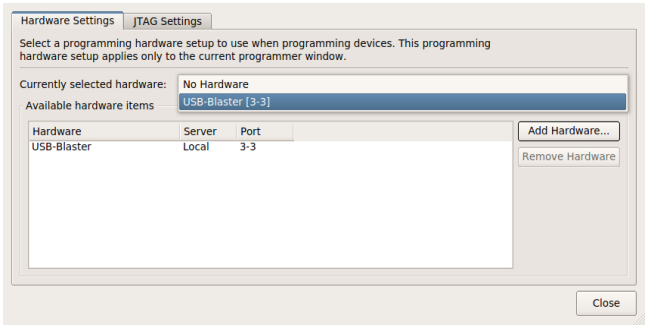
This will compile the C code using aeMB GNU toolchain. If every thing run successfully you must have `ram0.bin` in your `sw` directory.

Download Hardware file to the Board

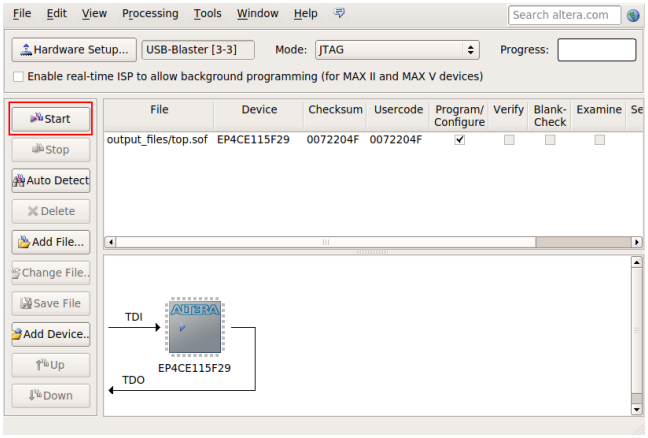
1. Power ON, and connect DE2 Board with PC.
2. Open Quartus Project earlier, at quartus toolbar, select Tools, then select Programmer.



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]. Then Open terminal and run `nios2-terminal`. Test the program by entering different values.

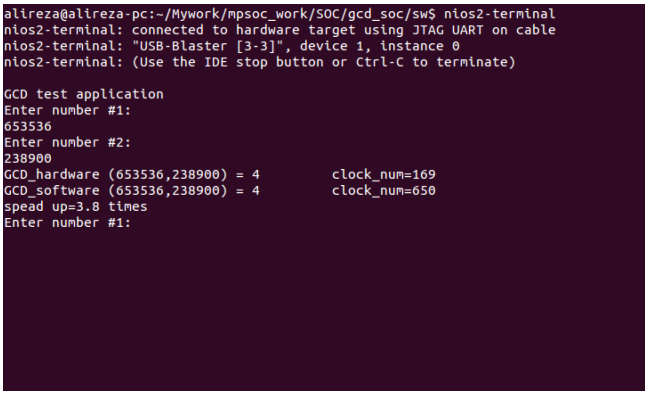


Figure 19