Walkthrough 1 (Counter w/switches)

0. SETUP

Open ISE version 13.4 on the desktop homepage. Then select new project and use the following settings for the project settings:

a.



Name the project Counter\_Test and select HDL as the top-level source type.

b.



For Evaluation Development Board choose Spartan-6 SP605 Evaluation Platform and, for our purposes, choose the HDL Source Analysis to be VHDL-93. Then click next and choose finish

c. In the Design panel, right click the chip xc6slx45t-3fgg484 and select new source. In the new window, select the verilog module and name it counter\_program. The coding may now begin

1. Coding

a. In the module counter\_program, you must put this within the parentheses:

output [3:0] led,

input clk,

input switch1,

input switch2

These are representations of the physical hardware that will be used to program the counter. The first is the four GPIO LEDs, the second is the onboard 27 MHz clock, and the last two are switches that will be used later.

b. Next define these 4 registers:

reg [31:0] count = 32'b00000000000000000000000000000000;

reg [31:0] tempc = 32'b00000000000000000000000000000000;

reg [3:0] templ = 4'b0000;

reg [3:0] led = 4'b0000;

These are essentially the verilog equivalents to variables in a c program. Many of them are temporary variables to hold values, but the special one is the led, which is defined as an output. This allows the led to be initially instantiated in the program with a four bit value of 0.

c. Insert this:

always @ (posedge clk)

begin

end

This states that at every positive edge of the input (in this case, the onboard clock), the code within it will execute.

d. Insert this code between the begin and end within the always @ statement:

tempc = count + 1;

count = tempc;

This will allow the count to increment and its values later be used for other parts of the code.

c. Insert this after the previous statement (within the always clock block)

if (count == 32'b00000010000000000000000000000000)

begin

tempc = 32'b00000000000000000000000000000000;

count = 32'b00000000000000000000000000000000;

if (led > 4'b1111)

begin

templ = 4'b0000;

led = 4'b0000;

end

else

begin

templ = led + 1;

led = templ;

end

end

else

begin

tempc = tempc;

count = count;

end

When counter reaches its 29th bit , it will reset the value of both “tempc” and “count”, at this, the value of “led” is checked, if it is equal to 1111, it will set the value of “templ” and “led” to 0000, else, it will increment “templ” and make “led” equal to this value. If “counter” has not reached this value, then “tempc” and “count” remain at their current values.

This will allow a basic counter to operate on the board’s LEDs.

The next step is to modify the code to allow for switches to interact with the counter.

d. Add the following above the “tempc = count + 1; count = tempc;”

if (switch1 == 1)

begin

tempc = tempc;

count = count;

end

Next, encompass the rest of the code in an else statement corresponding to the if previous if statement. (Put “else begin” above the “tempc = count + 1” and an “end” before the closing “end” of the “always @” statement).

e. Add this statement to the end of the code (within the always clock block)

if (switch2 == 1)

begin

led = 4'b0000;

templ = 4'b0000;

end

This will allow the led to be reset.

2. Constraints

a. In the design menu, right click the verilog module (the V) and select new source. Select implementation constraints file, choose a name, and insert this code:

NET "led[0]" LOC = "D17";

NET "led[1]" LOC = "AB4";

NET "led[2]" LOC = "D21";

NET "led[3]" LOC = "W15";

NET "clk" LOC = "AB13";

NET "switch1" LOC = "C18";

NET "switch2" LOC = "Y6";

This will allow the inputs and outputs set in the code to be directed to the pins on the board that are needed for the code to function.

3. Processes to run code

Right click “Configure target device” in the design panel in the processes subsection, then click “Run” this will run all necessary processes in order for the program to function on the FPGA. Once a window pops up, press ok to proceed.

4. Impact

In this window, double-click boundary scan, then right click the new white screen and select “initialize chain” and close the pop-ups. Double click the right most chip and find your project on the computer. Once you find the project folder, select the BIT file to use. (It should have the name of the module in the code). Once complete, right click the right most chip, select “program”, and on the pop up, hit “apply” and ok. Let the new process complete and then test the program on the board.

Walkthrough 2 (Output Clock)

0. SETUP

Open ISE version 13.4 on the desktop homepage. Then select new project, choose a name, and use the following settings for the project settings:

a.



Name the project Clock\_Test and select HDL as the top-level source type.

b.



For Evaluation Development Board choose Spartan-6 SP605 Evaluation Platform and, for our purposes, choose the HDL Source Analysis to be VHDL-93. Then click next and choose finish

c. In the Design panel, right click the chip xc6slx45t-3fgg484 and select new source. In the new window, select the verilog module and name it Clock\_program. The coding may now begin

1. Coding

a. Insert this into the parentheses

input clk\_in, output clk\_out

These will represent your hardware for the program.

b. Insert this code after the semicolon:

assign clk\_out = clk\_in;

This will allow the output wire to have an input from a clock signal defined by the constraints.

2. Constraints

There are three ways to define the constraints:

The first orientation will be able to have an input clock from the user GPIO SMA and output to the “user clock in SMA” (this is just one of several available SMA ports):

Net "clk\_in" LOC = A3;

Net "clk\_out" LOC = M19;

The second orientation will have an input clock from the onboard clock and output from the “user clock in SMA” (this is just one of several available SMA ports):

Net "clk\_in" LOC = AB13;

Net "clk\_out" LOC = M19;

The third orientation will have an input clock from the reference clock sma and an output from the user GPIO SMA:

Net "clk\_in" LOC = M19;

Net "clk\_out" LOC = A3;

If you are looking at the board itself, you should see the following names:

M19 is the USER\_CLK\_N SMA

A3 us the USER\_GPIO\_N SMA

3. Processes to put the program on the board

Right click “Configure target device” in the design panel in the processes subsection, then click “Run” this will run all necessary processes in order for the program to function on the FPGA. Once a window pops up, press ok to proceed.

4. Board setup and IMPACT

a. To set up the board, put a Pasternack sma adapter on the USER\_GPIO\_N sma and on the USER\_CLK\_N sma. Then add the BNC cable to each adapter. If a configuration with an input clock that is not on the board is chosen, then put the chosen input clock into the AFG port and the output clock into the desired oscilloscope port.

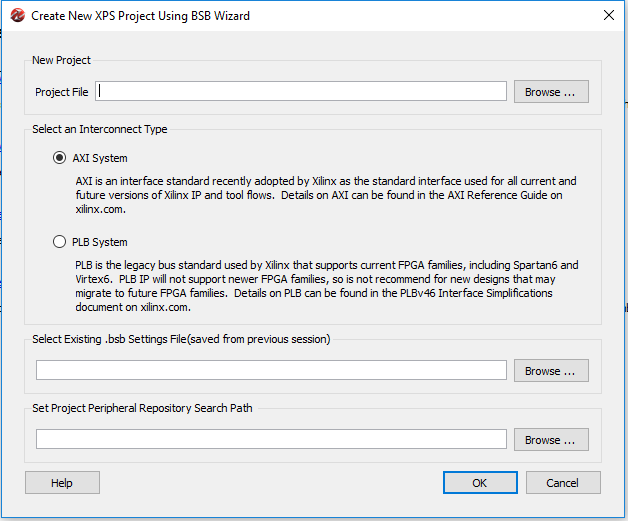
b. In this window, double-click boundary scan, then right click the new white screen and select “initialize chain” and close the pop-ups. Double click the right most chip and find your project on the computer. Once you find the project folder, select the BIT file to use. Once complete, right click the right most chip, select “program”, and on the pop up, hit “apply” and ok. Let the new process complete and then test the program on the board.

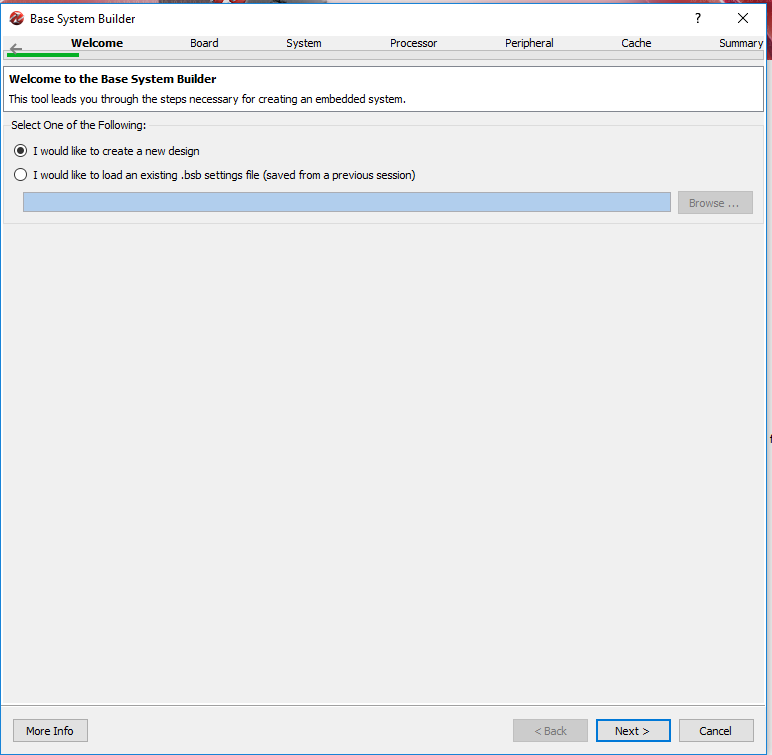
Walkthrough 3 XPS Counter

0. Setting up XPS

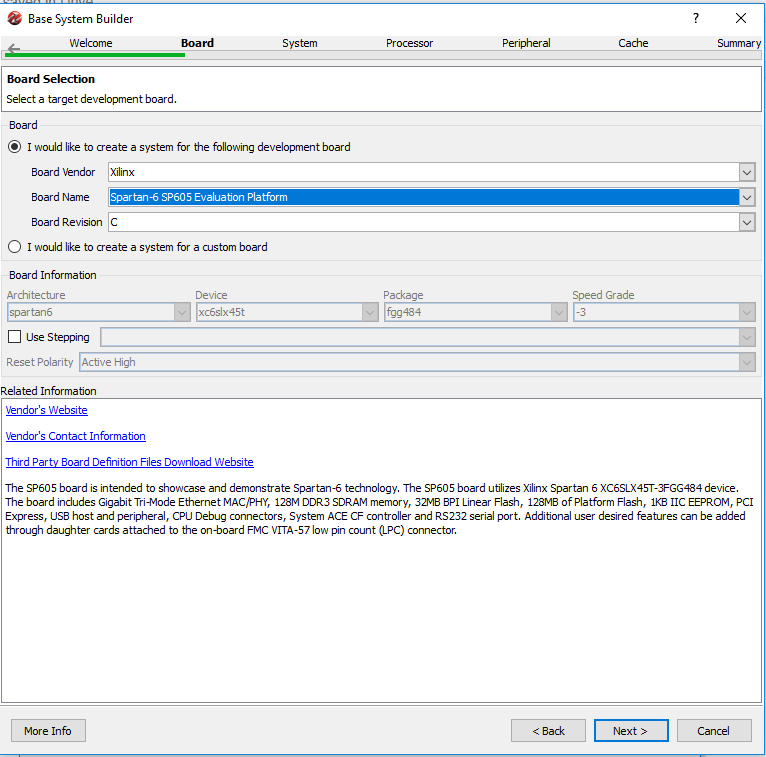
a. Open Xilinx Platform Studio 13.4 (when “Xilinx” typed in search bar below, it is the first “Xilinx Platform Studio” that pops up).

b. Select “Create New Project Using Base System Builder”

c.  when this window pops up choose a file path. When browsing, you can select any file path you wish and create a new folder once in the browsing menu, however the folder cannot contain spaces or special characters other than “\_”. Once a location is chosen, choose the interconnect type to be the PLB system and press ok.

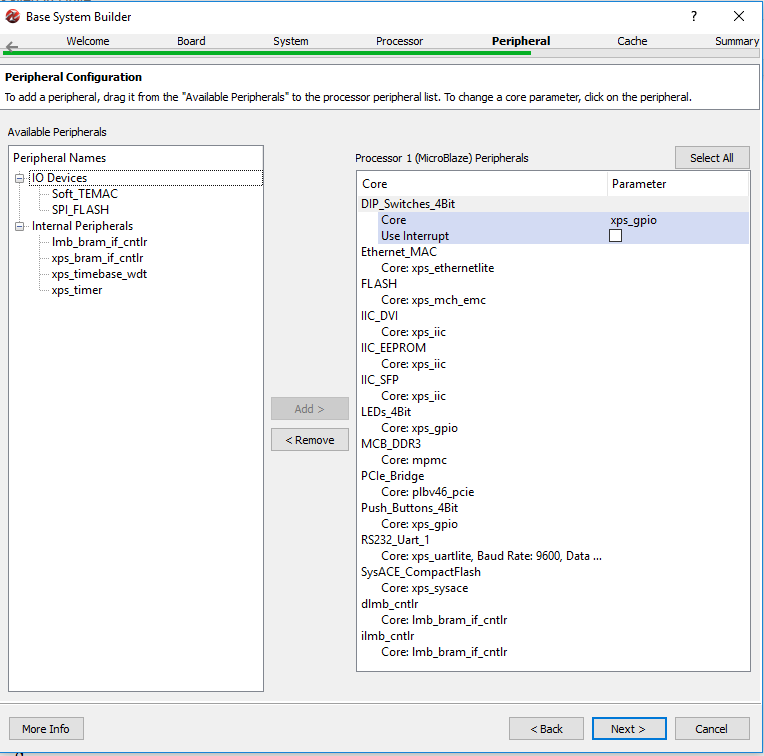
d. 

Choose create new design and click next.

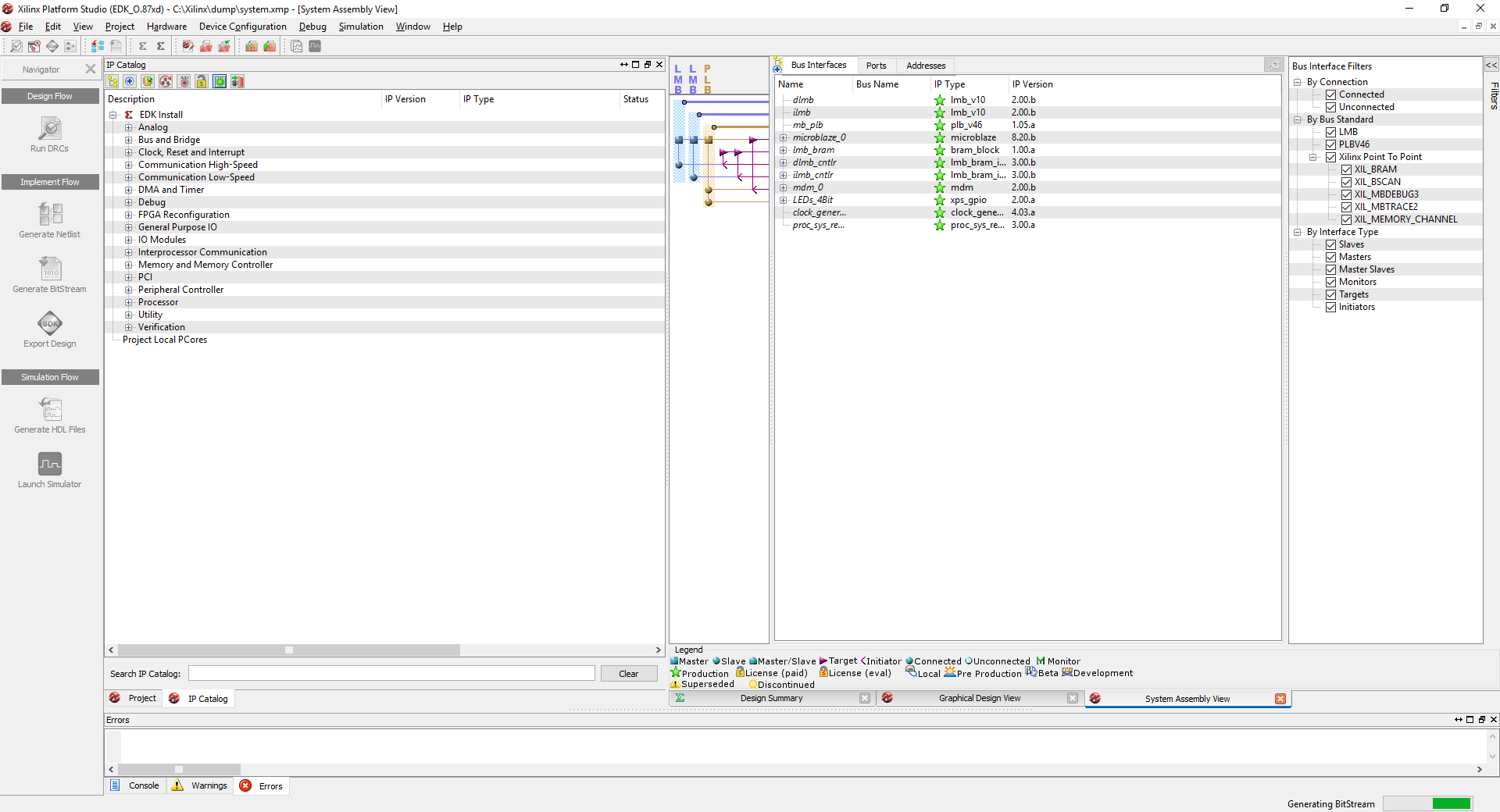
e. 

When choosing a system, choose the “Spartan-6 SP605 Evaluation Platform” and then click next.

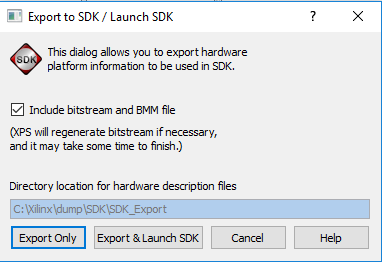
f. Click next two more times (single processor model, and standard processor configuration).

g. 

At this screen, remove Ethernet\_MAC, FLASH, IIC\_DVI, IIC\_EEPROM, IIC\_SFP, SysACE\_CompactFlash, PCIe\_Bridge, RS232\_Uart\_1, and DIP\_Switches\_4Bit, Push\_Buttons\_4Bit, and MCB\_DDR3. Then click next twice and then finish.

h. 

When this screen pops up, click “Generate Bitstream” and wait. Once complete click “Export Design.”

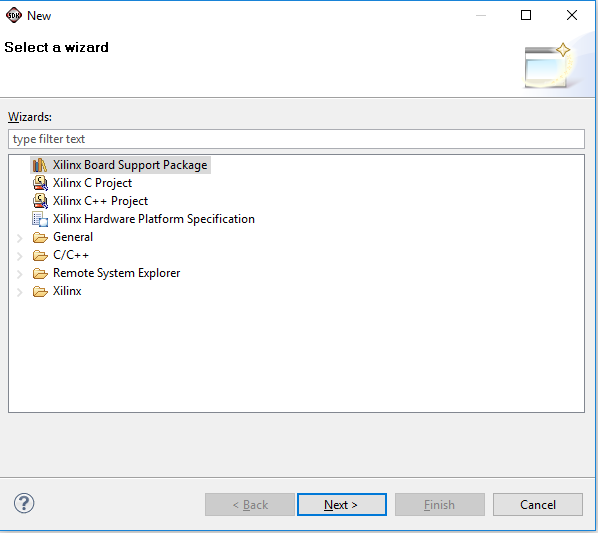


At this, click “Export & Launch SDK.”

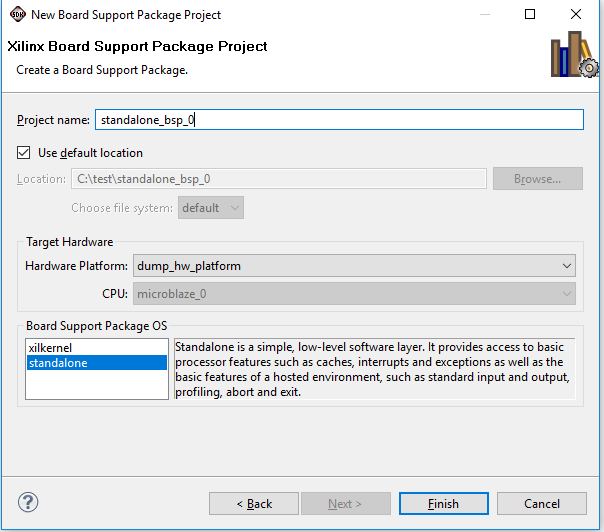
1. Setting up SDK

a. Again choose another folder to place the project in, you can create another folder as well, but do not put capital letters, special characters (except “\_”), and spaces.

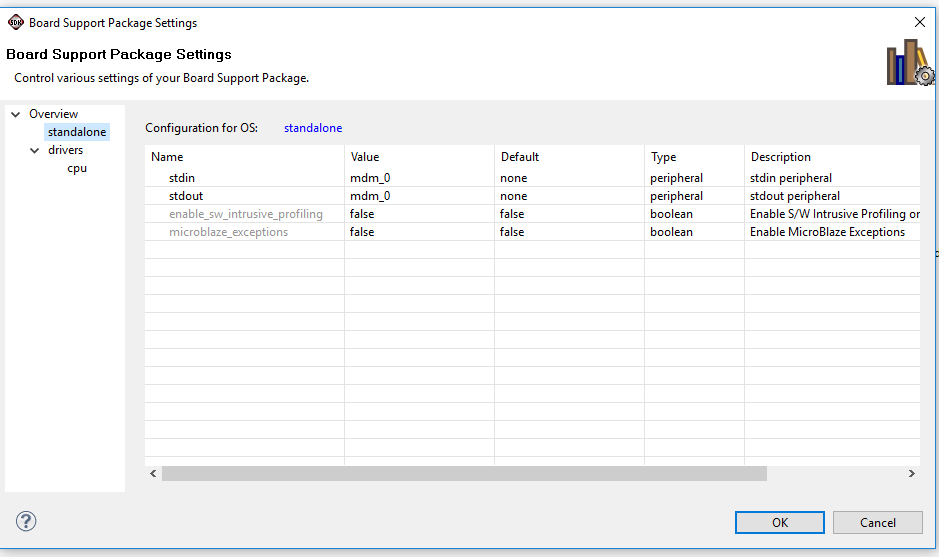
b. Next right click the “Project Explorer” (far left panel), hover over new, and select other.



Select “Xilinx Board Support Package” and press next.

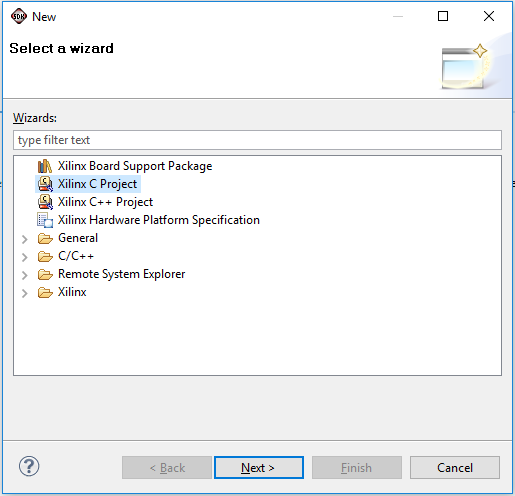


Leave as is and press finish.



Make sure “stdin” and “stdout” are mdm\_0, then select ok.

C.

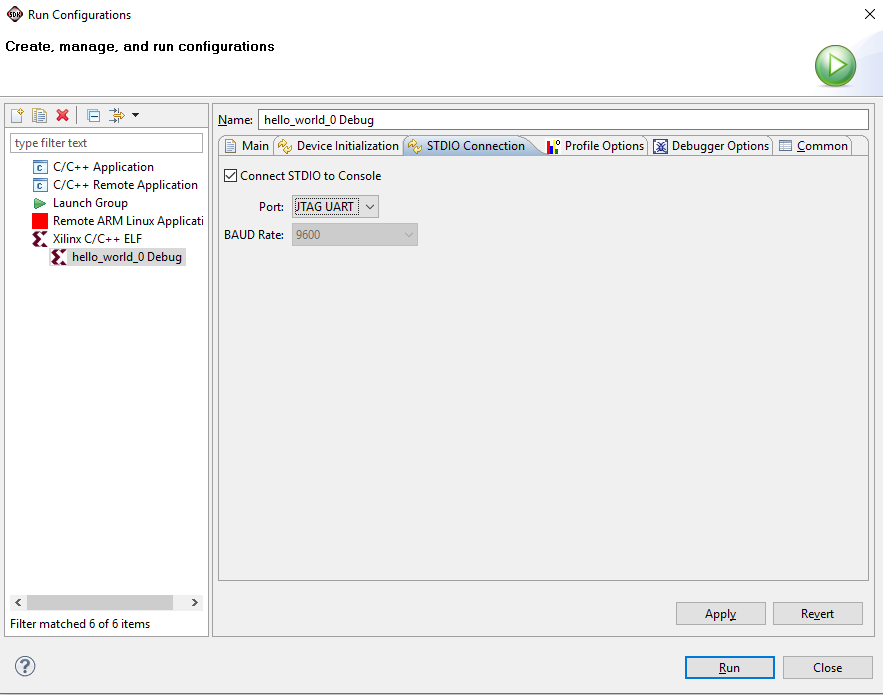


Once again right click the “Project Explorer,” then select “Xilinx C Project.” Click next on the next screen and then press finish.

2. Programming

a. Now select the dropdown for the compile/ run button (highlighted button) 

When it drops down select “Run Configurations”



When this window pops up, double click the “Xilinx C/C++ ELF” and select “STDIO Connection.” Then check the “Connect STDIO to Console” box and change the “Port” to “JTAG UART” and hit apply.

b. Now delete this code:

print("Hello World\n\r");

cleanup\_platform();

return 0;

c. Add this code:

#include "xparameters.h"

#include "xgpio.h"

#define LED\_DELAY 150000

This code will allow access the xparameters.h and the xgpio.h files, which have functions that will be integral to the program. The defined constant will also be used.

d. Add this code in the main function

XGpio led;

int dip\_check;

int Delay = 0;

XGpio\_Initialize(&led, XPAR\_LEDS\_4BIT\_DEVICE\_ID);

XGpio\_SetDataDirection(&led, 1, 0x00000000);

int Data = XGpio\_DiscreteRead(&led, 1);

As stated, these functions and variables are a part of the files that were included at the beginning of the files.

e. Copy this code after the previous code.

While (1)

{

Data = XGpio\_DiscreteRead(&led, 1) + 1;

XGpio\_DiscreteWrite(&led, 1, Data);

for (Delay = 0; Delay < LED\_DELAY; Delay++);

}

This unending loop will allow the counter to continuously count and reset.

f. Now press the program fpga button and press run.