차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

System Integration II

Host-FPGA Interface, User-defined IP

2022.03.03 (Thu)

Road map

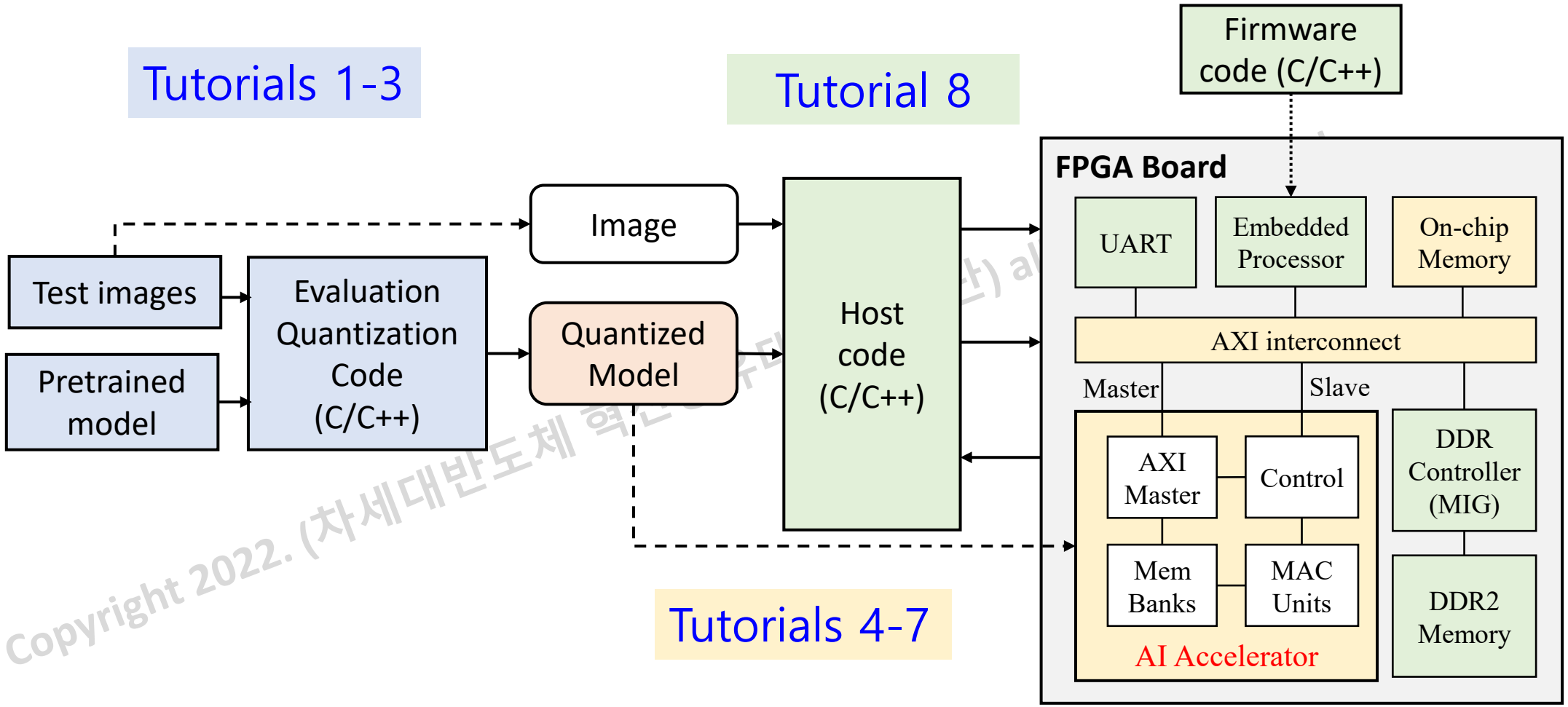
Review

FPGA board

PC-FPGA

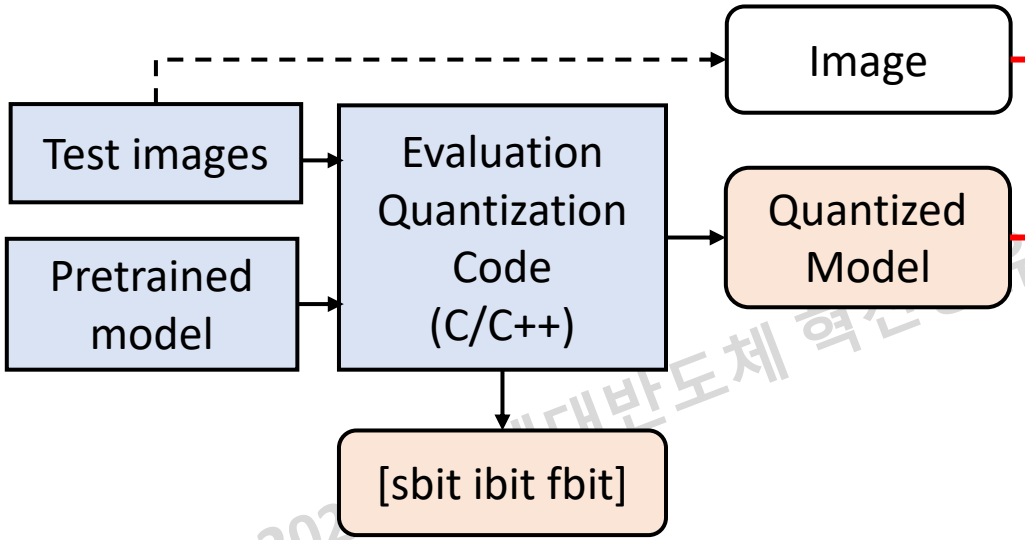
Custom IP

Top structure and tutorials



Data preparation

- Input image
- Model/Weight quantization
  - Batch normalization folding
- Activation quantization



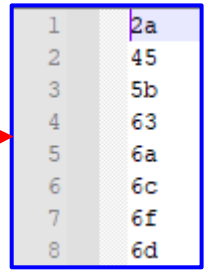
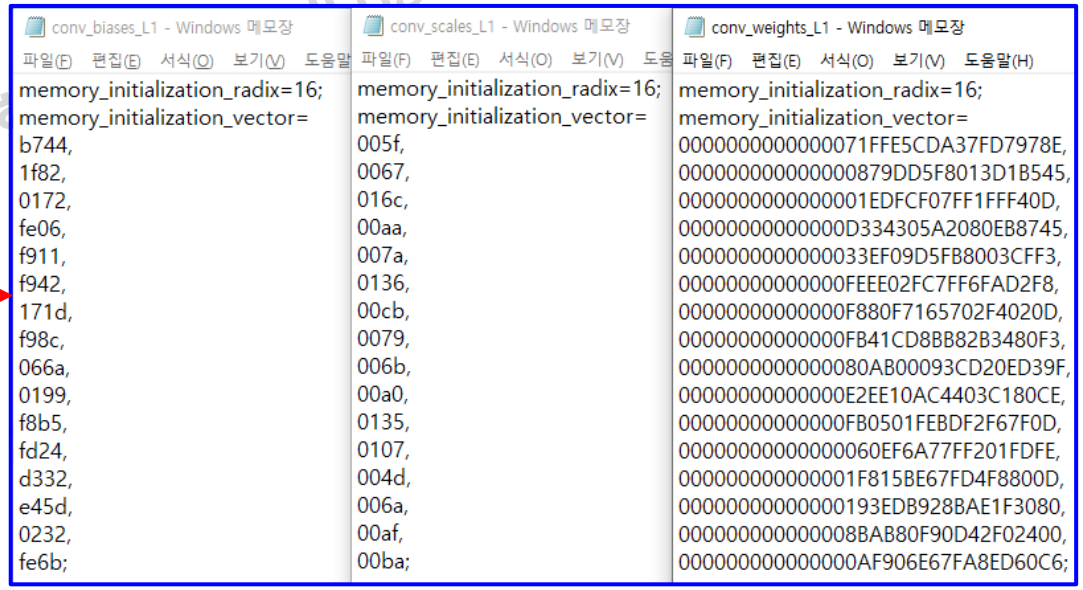


Image file

Parameter files



Data preparation

- Weight/activation quantization: How many bits are used for sign, integer, and fractional parts.
- Eight-bit representation (Weight, Activation): [sign ibit fbit]
  - Q(0,8):

[1, 0, 7] → {-128, -127, …, 0, 1, …, 127}/2^7

  - Q(4,4)

[1, 3, 4] → {-128, -127, …, 0, 1, …, 127}/2^4

- Naïve example:
  - Truncate LSB bits in the accumulation result

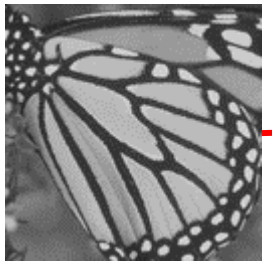
```
wire [ACT_BITS*To-1:0] all_acc_o = {
    acc_o[3][19:12], acc_o[2][19:12], acc_o[1][19:12], acc_o[0][19:12]
};
```

⇒ Must specify for data format for all layers

CNN Accelerator

- We assume that all files are generated, and then build H/W modules to do inference
  - MACs, on-chip buffers, controller

Image file



```
1  2a
2  45
3  5b
4  63
5  6a
6  6c
7  6f
8  6d
```

Parameter files

conv_biases_L1 - Windows 메모장

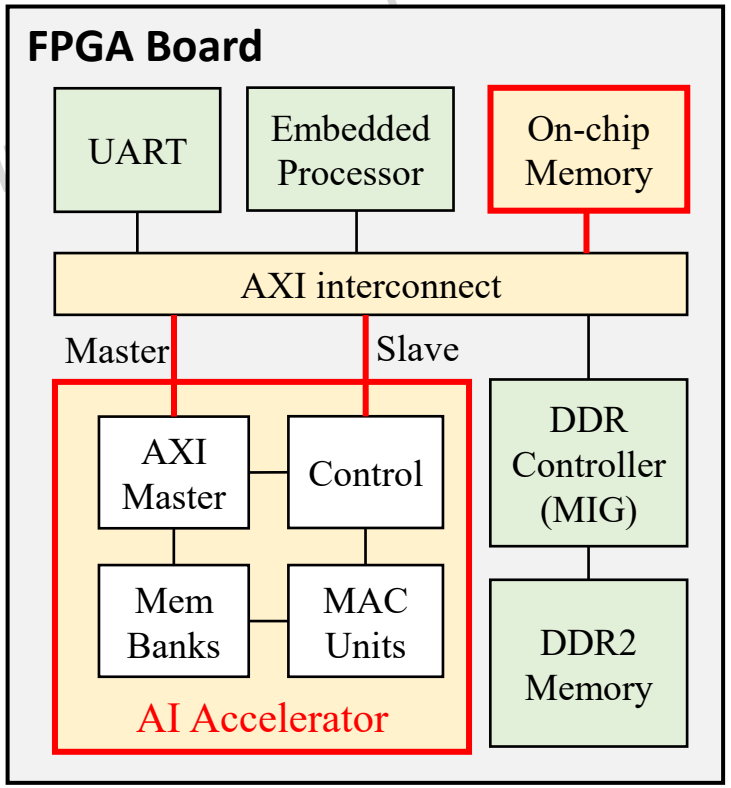
```
memory_initialization_radix=16;
memory_initialization_vector=
b744,
1f82,
0172,
fe06,
f911,
f942,
171d,
f98c,
066a,
0199,
f8b5,
fd24,
d332,
e45d,
0232,
fe6b;
```

conv_scales_L1 - Windows 메모장

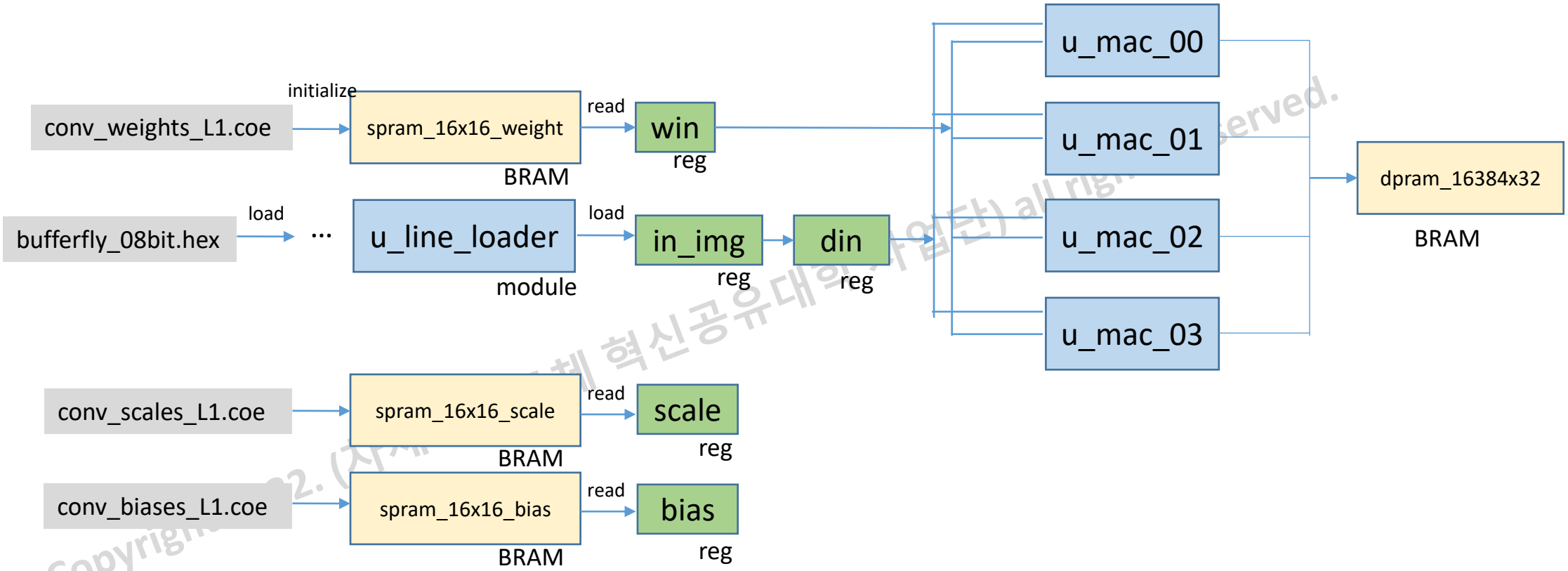
```
memory_initialization_radix=16;
memory_initialization_vector=
005f,
0067,
016c,
00aa,
007a,
0130,
00cb,
0079,
006b,
00a0,
0135,
0107,
004d,
006a,
00af,
00ba;
```

conv_weights_L1 - Windows 메모장

```
memory_initialization_radix=16;
memory_initialization_vector=
0000000000000071FFE5CDA37FD7978E,
0000000000000000879DD5F8013D1B545,
0000000000000001EDFCF07FF1FFF40D,
00000000000000D334305A2080EB8745,
0000000000000033EF09D5FB8003CFF3,
00000000000000FEEE02FC7FF6FAD2F8,
00000000000000F880F7165702F4020D,
00000000000000FB41CD8BB82B3480F3,
0000000000000080AB00093CD20ED39F,
00000000000000E2EE10AC4403C180CE,
00000000000000FB0501FEBDF2F67F0D,
0000000000000060EF6A77FF201FDFE,
000000000000001F815BE67FD4F8800D,
0000000000000193EDB928BAE1F3080,
000000000000008BAB80F90D42F02400,
000000000000000AF906E67FA8ED60C6;
```

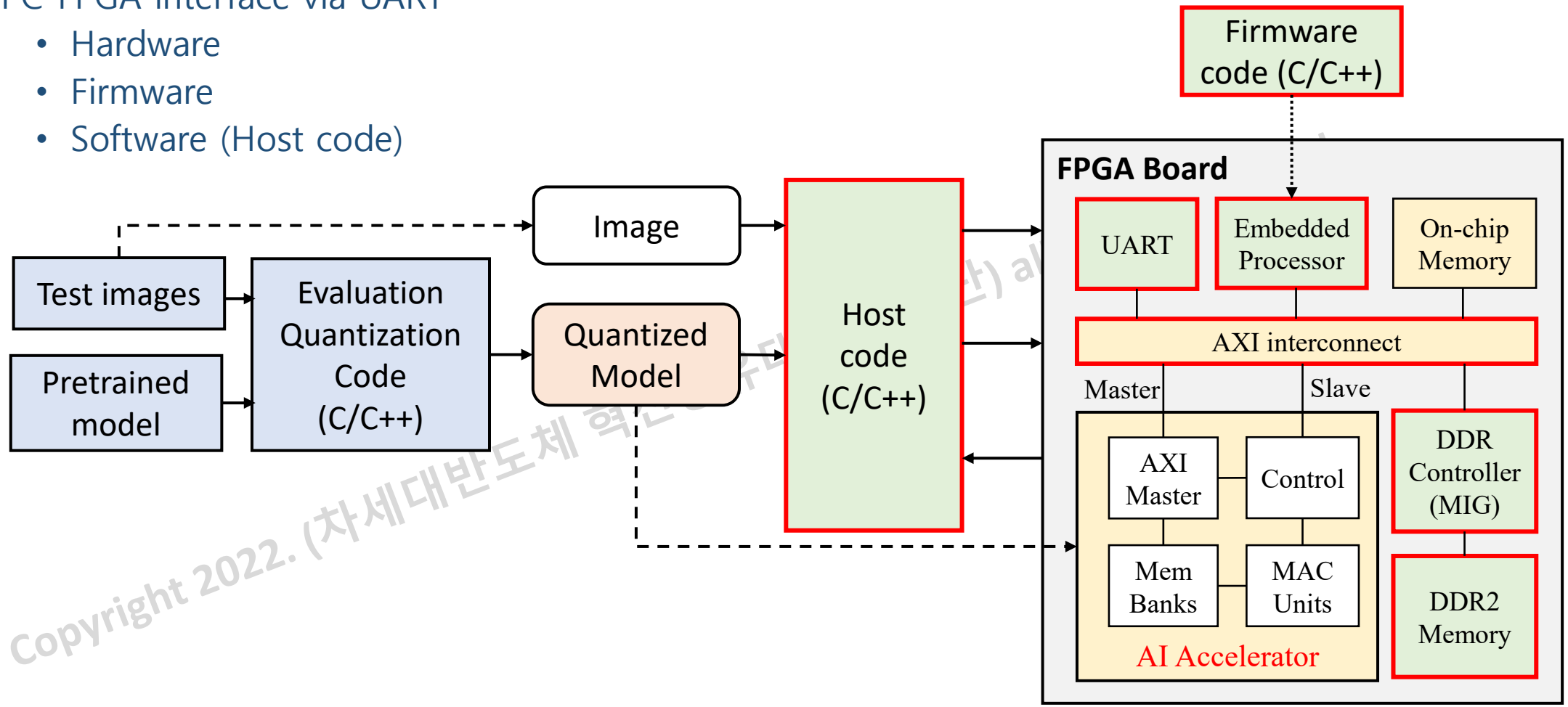


Overall data flow



Objective

- PC-FPGA interface via UART
  - Hardware
  - Firmware
  - Software (Host code)



Road map

Review

FPGA board

PC-FPGA

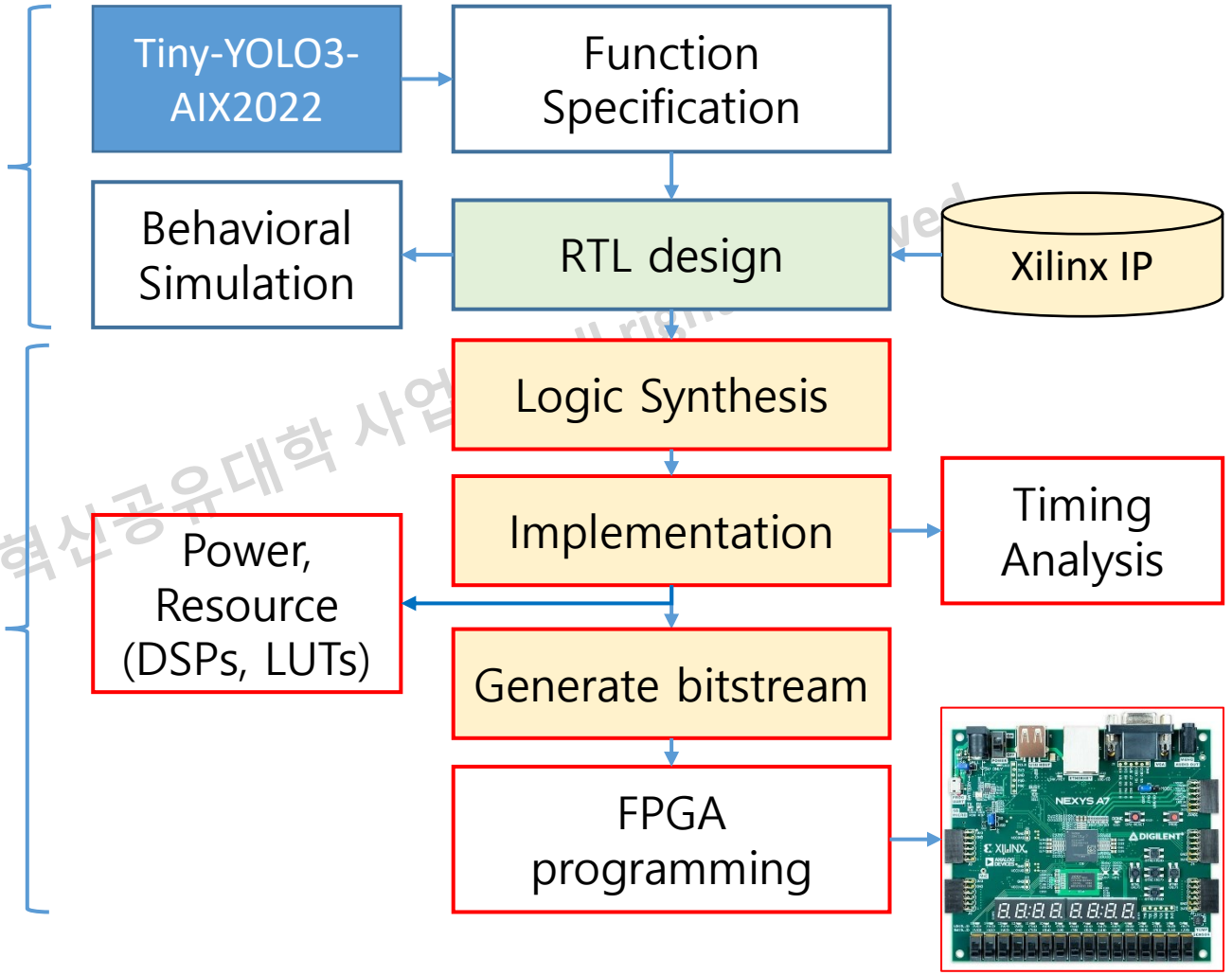
Custom IP

FPGA design flow

Function simulation
Text editor: *vim, notepad++, VS code*
RTL simulator: *ModelSim, ISE, Vivado*

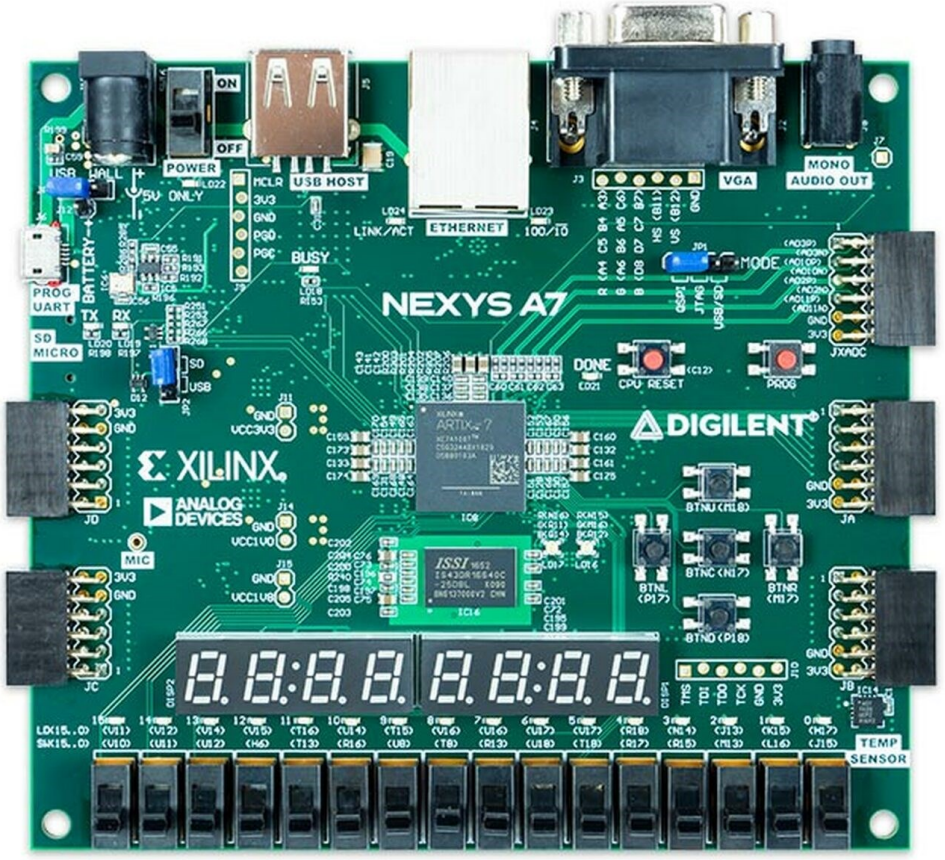
FPGA Implementation (*ISE, Vivado*)
- Synthesize the design
- Implementation: mapping, placement and routing

Optimization
- Analyze timing, power, resources

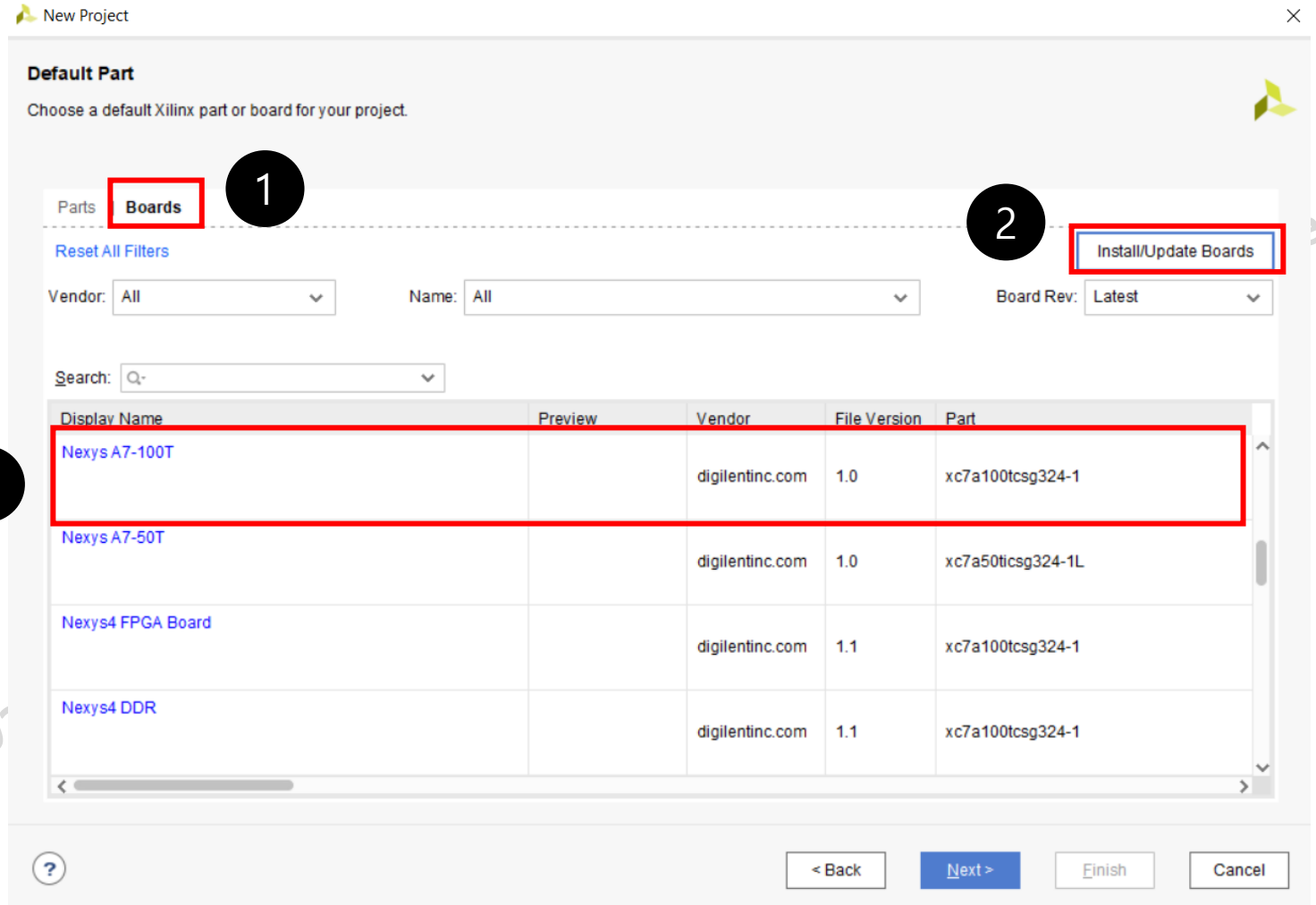


Nexys A7 FPGA board

- Xilinx Artix-7 FPGA XC7A100T-1CSG324C
- 15,850 logic slices
  - Each with four 6-input LUTs and 8 FFs
- 4,860 Kbits of fast block RAM
- 240 DSP slices
  - Dedicated to multiplication and accumulation (MAC)
- Internal clock speeds exceeding 450 MHz
- 128 MB DDR2 Memory
- USB-JTAG port for FPGA programming and communication

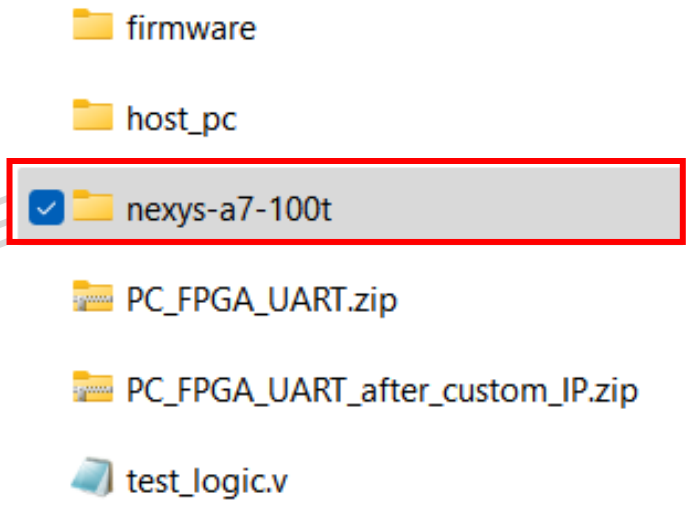


Add Nexys A7-100T



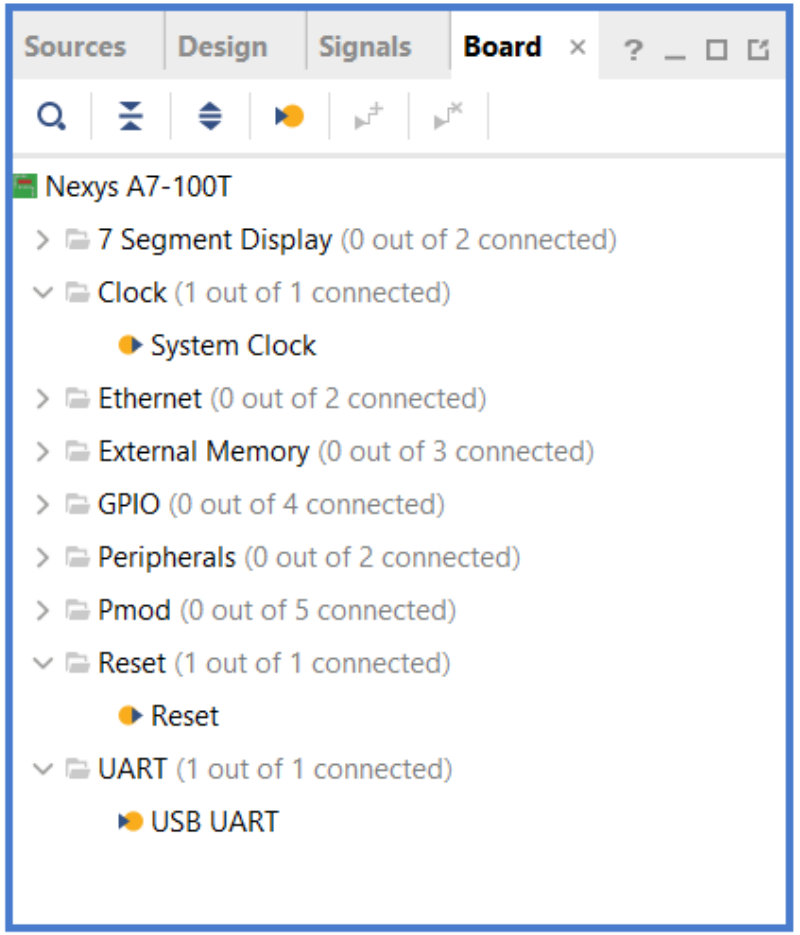
Add Nexys A7-100T

- Copy the folder"nexys-a7-100t" into your directories
  - "(your Vivado installation directory)₩2021.1₩data₩boards₩board_files"
  - The default path is: C:₩Xilinx₩Vivado₩2021.1₩data₩boards₩board_files
  - If folder board_files does not exist, create it and paste nexys-a7-100t inside it.



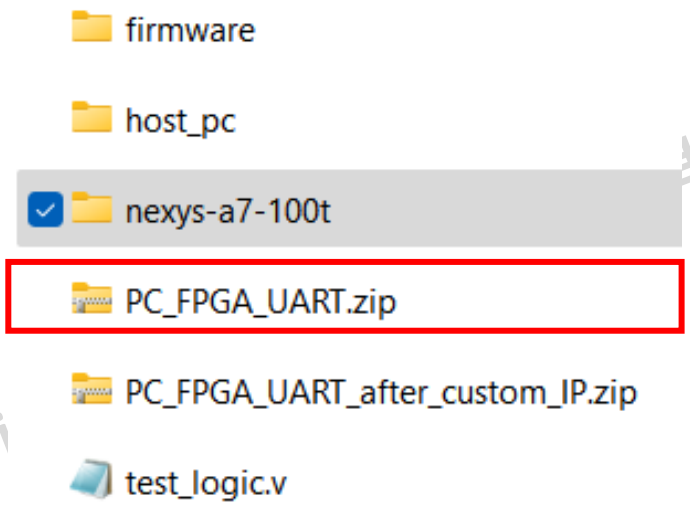
Nexys A7-100T

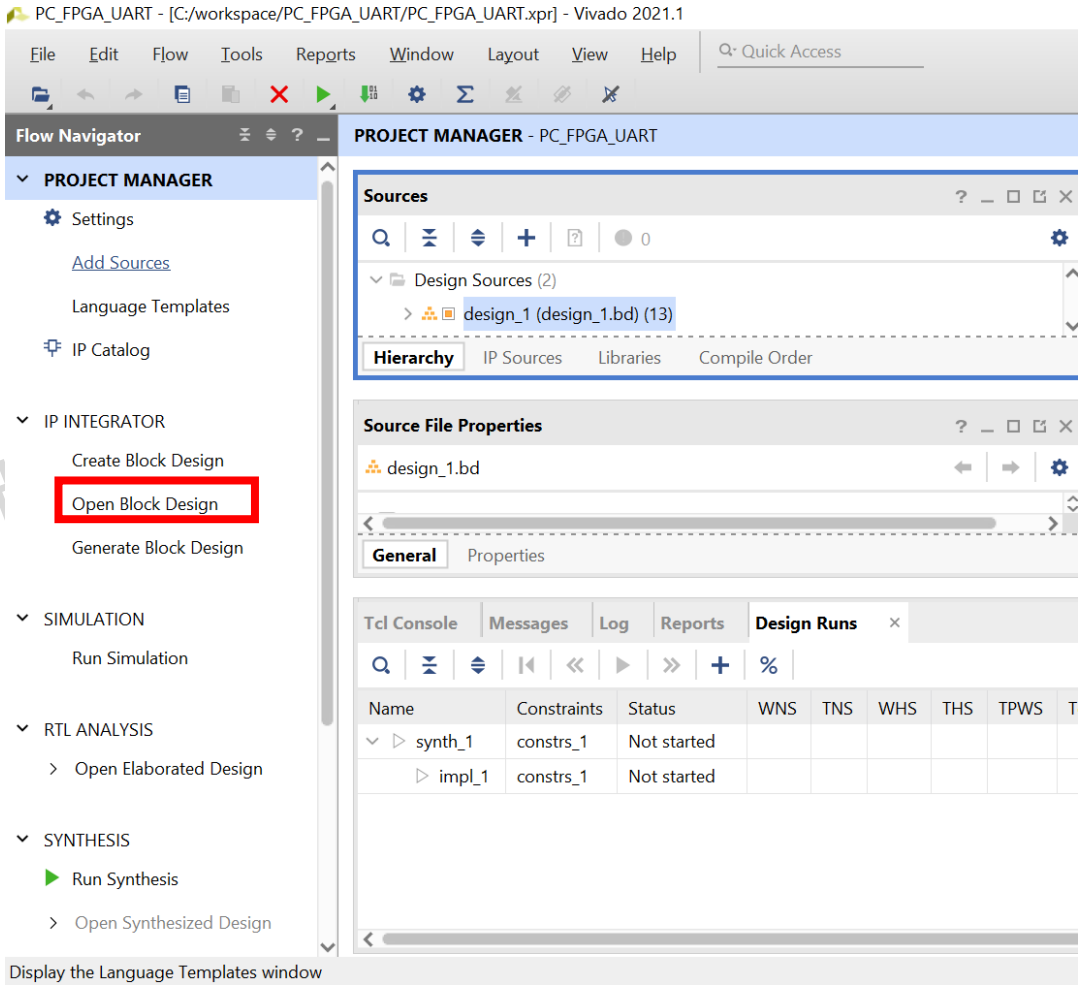
- After adding Nexys A7-100T, you can configure its components like:
  - Clock, Reset
  - IO and peripherals
  - PMOD
  - USB UART



PC-FPGA

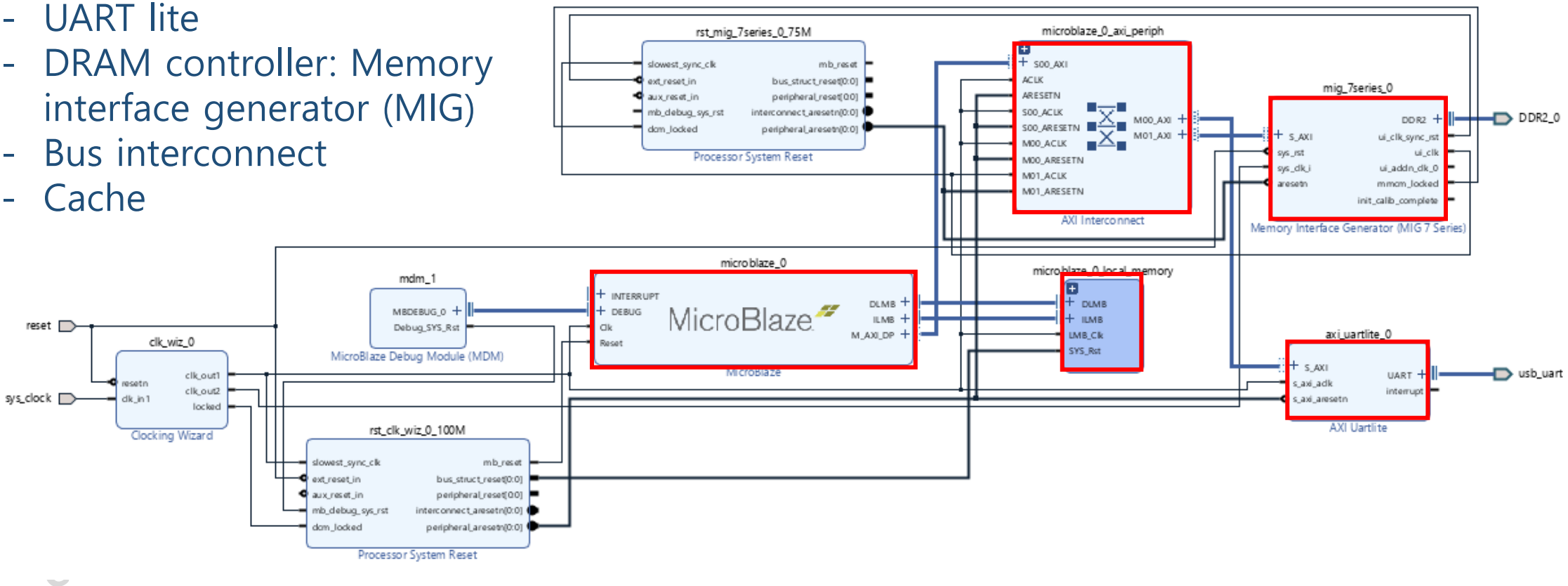
- Extract PC_FPGA_UART.zip
- Open “PC_FPGA_UART.xpr"
- Click “Open Block Design”.
  - Here, we can view and edit the block diagram of our hardware.



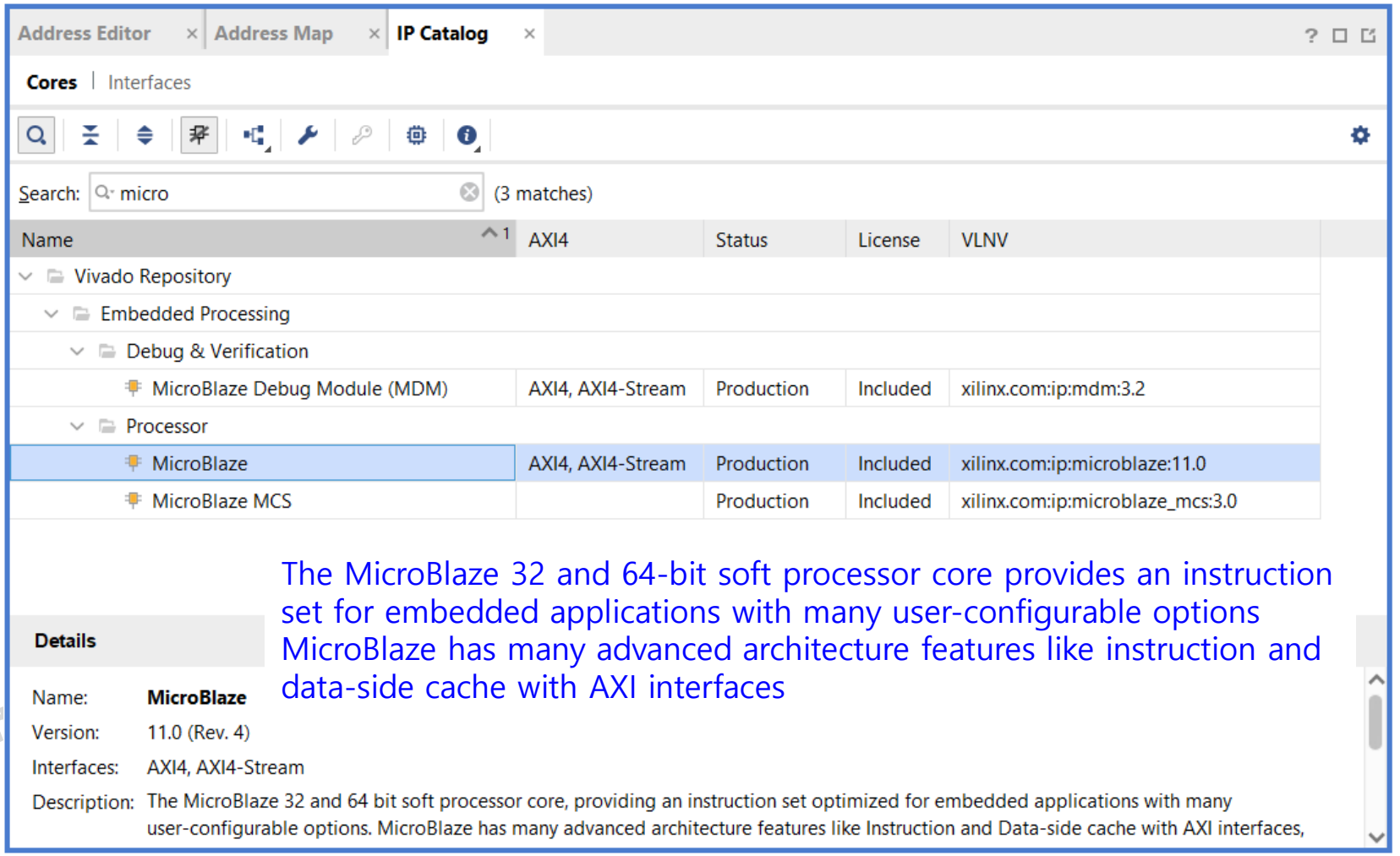


Block diagram

- MicroBlaze
- UART lite
- DRAM controller: Memory interface generator (MIG)
- Bus interconnect
- Cache

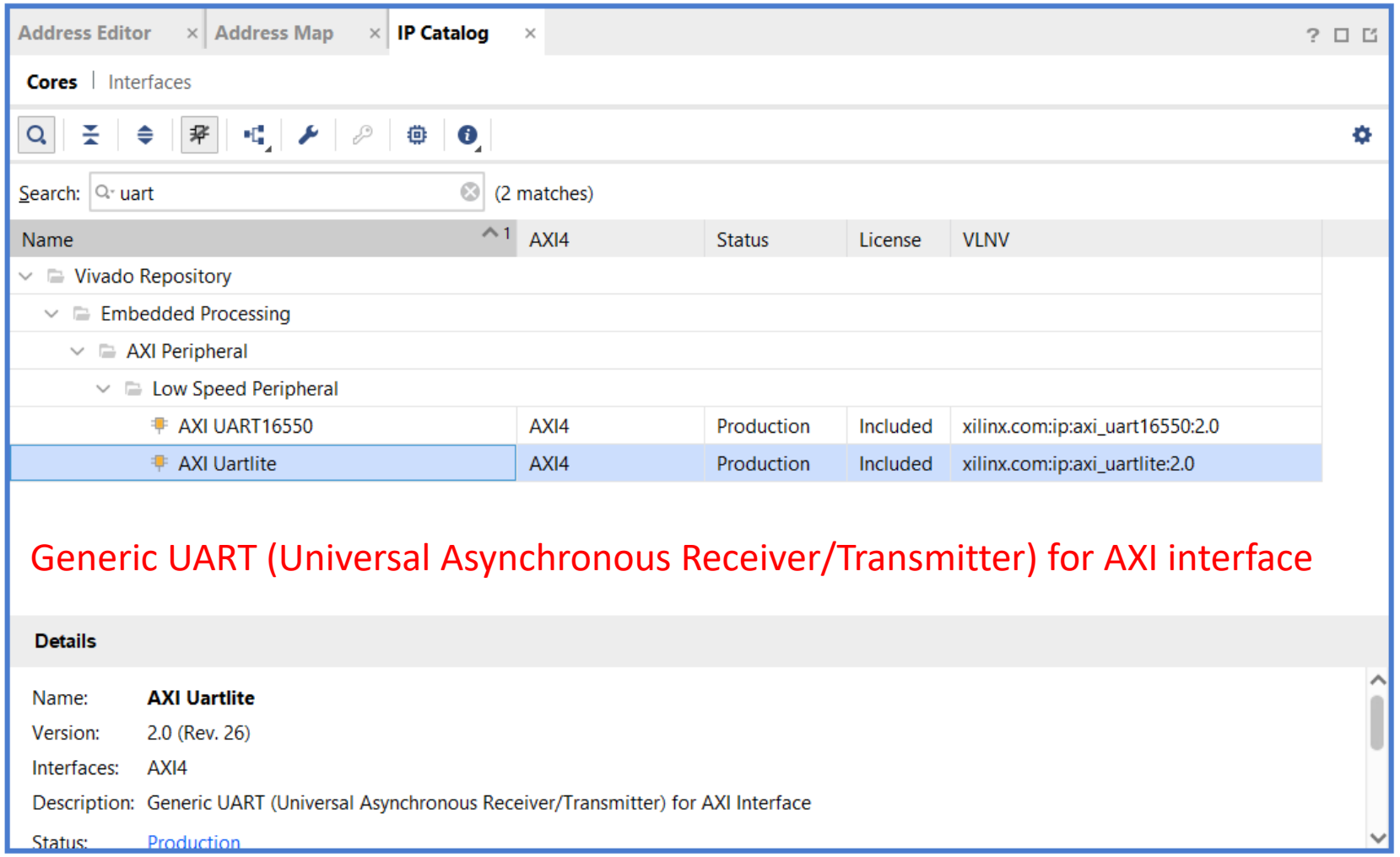


Soft processor core (Microblaze)

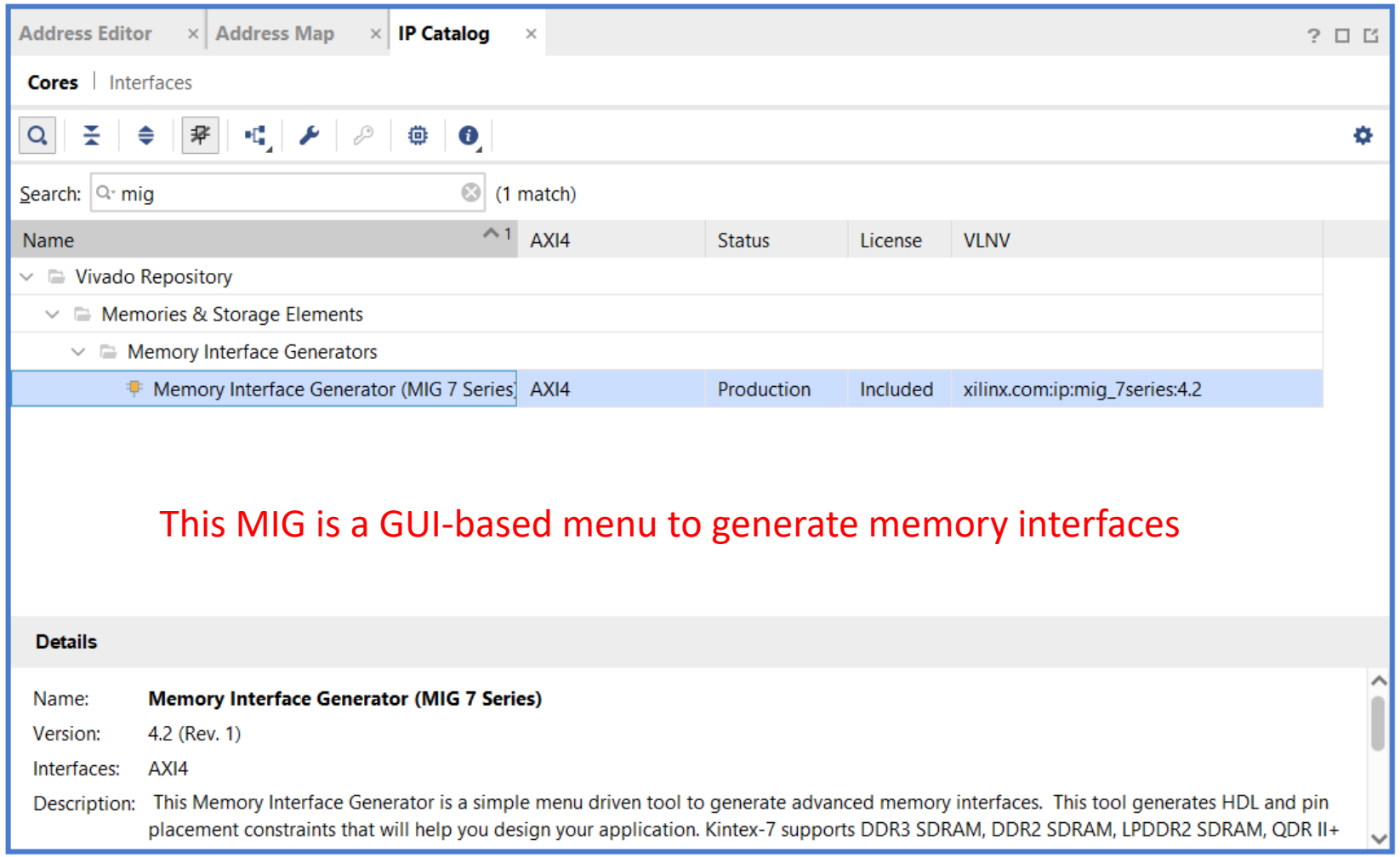


The MicroBlaze 32 and 64-bit soft processor core provides an instruction set for embedded applications with many user-configurable options MicroBlaze has many advanced architecture features like instruction and data-side cache with AXI interfaces

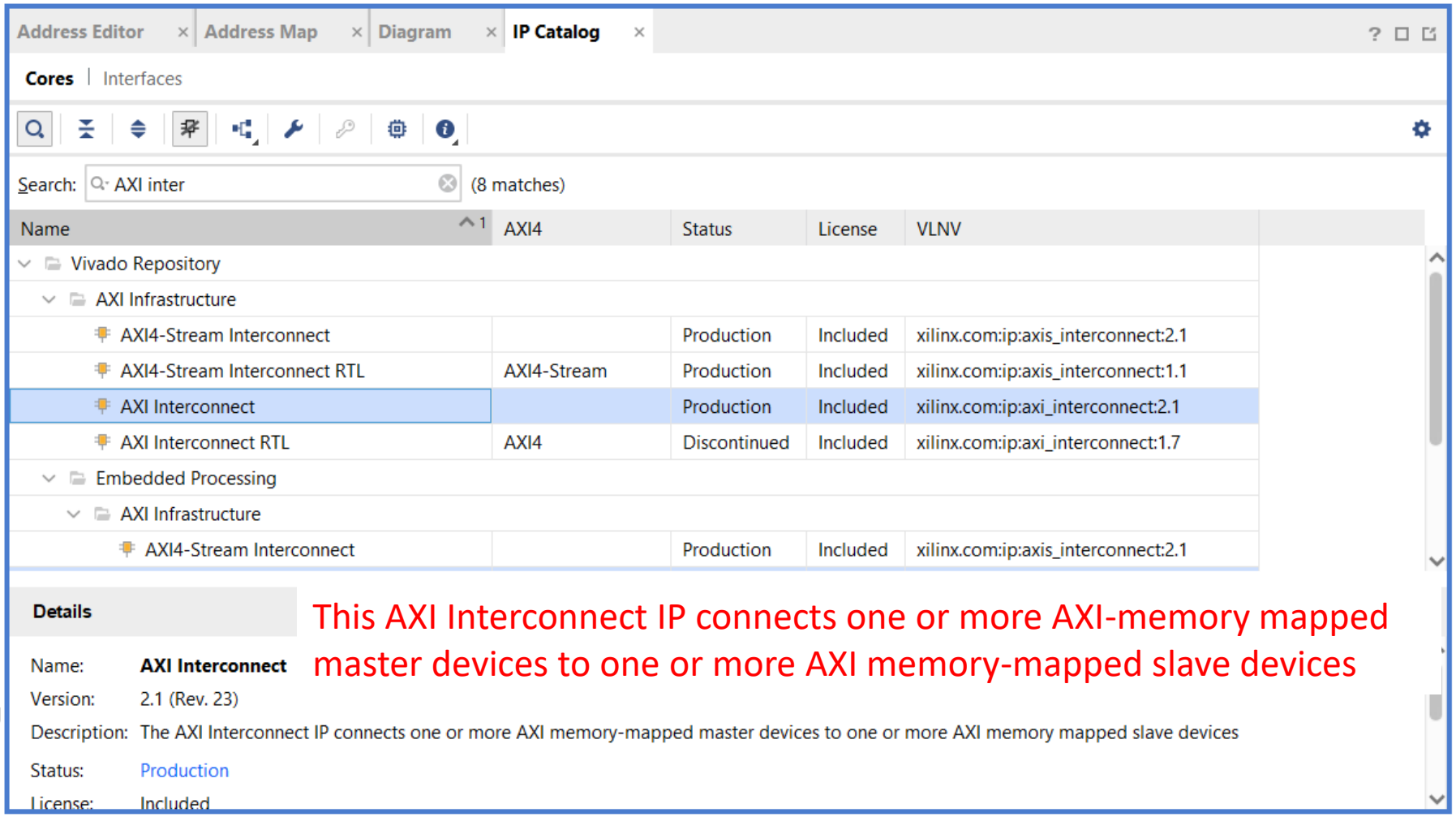
UART Lite



Memory interface generator (MIG)



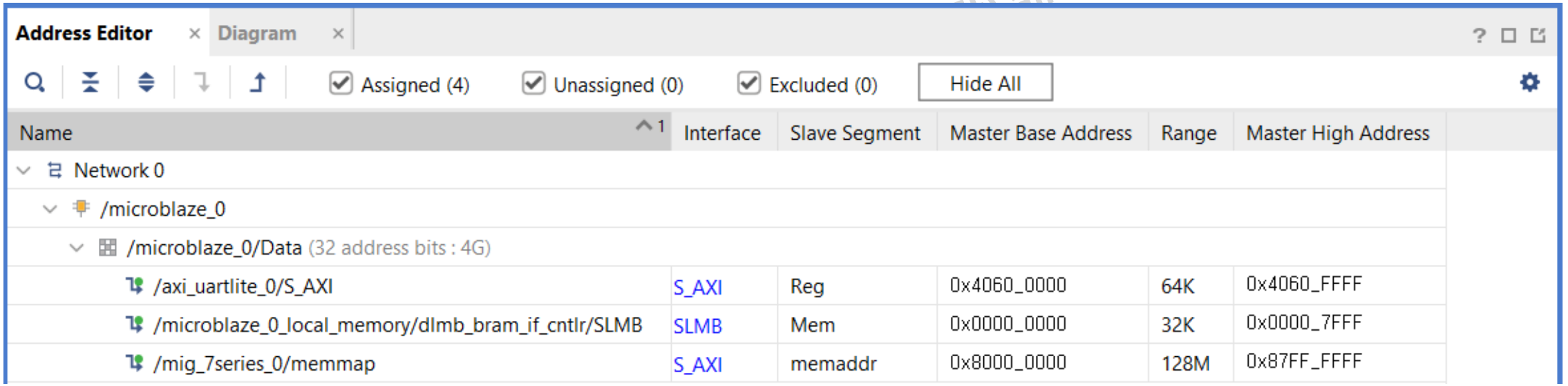
AXI interconnect



This AXI Interconnect IP connects one or more AXI-memory mapped master devices to one or more AXI memory-mapped slave devices

Address mapping

- Mapping addresses for masters and slaves
  - Base addresses and ranges
  - Example
    - UART: 0x4060_0000~0x4060_FFFF (64K)
    - MIG: 0x8000_0000~0x87FF_FFFF (128M)



Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/microblaze_0					
/microblaze_0/Data (32 address bits : 4G)					
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
/mig_7series_0/memmap	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF

Road map

Review

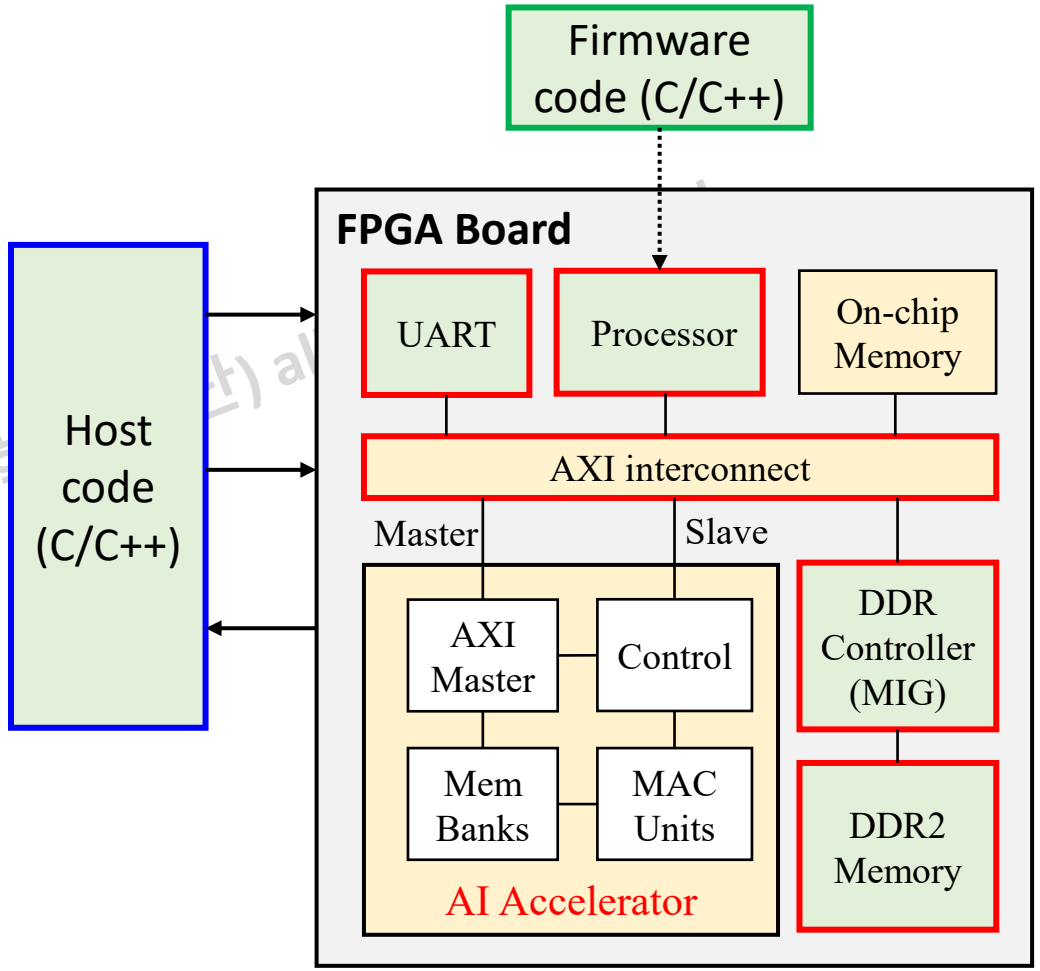
FPGA board

PC-FPGA

Custom IP

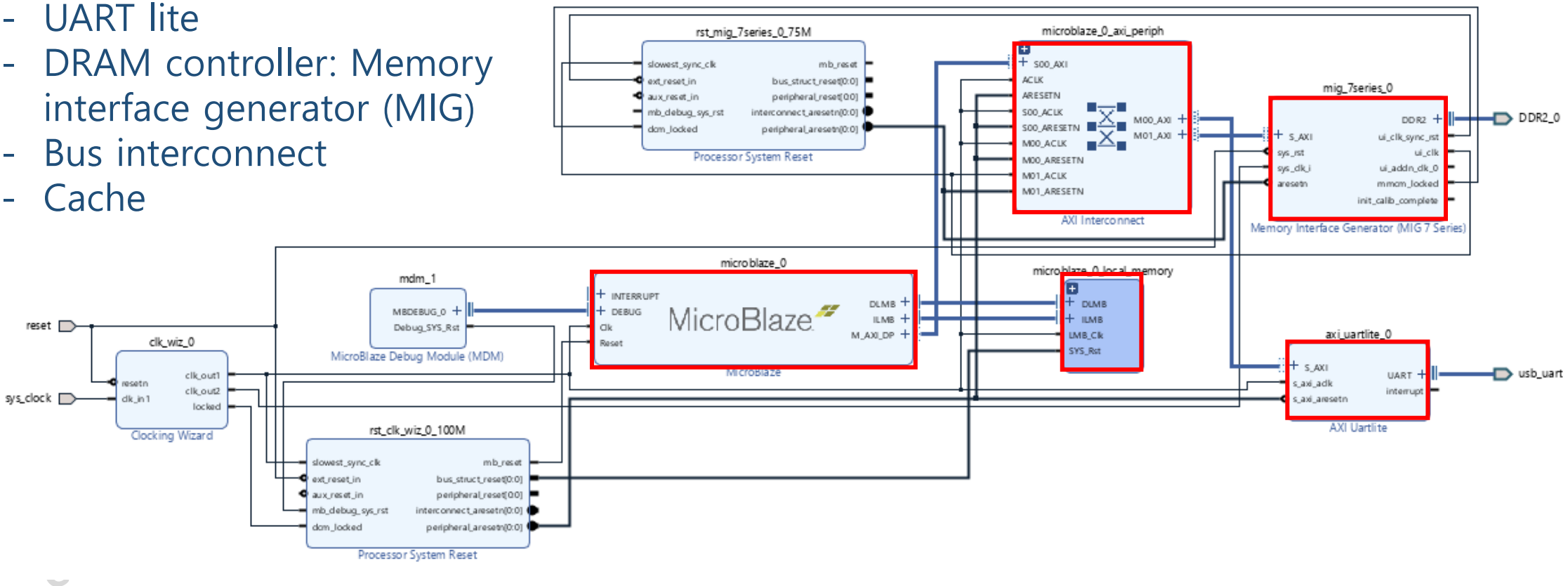
Host PC and FPGA interface

- Hardware (IP catalog, Vivado)
  - Processor (MicroBlaze)
  - Memory interface/controller
  - AXI Interconnect
  - UART-lite
- Firmware (Vitis)
  - To build an application on the Microblaze
  - Receive/parse a command from the Host PC
- Host PC
  - Read an input image or parameters
  - Send a command to the FPGA board



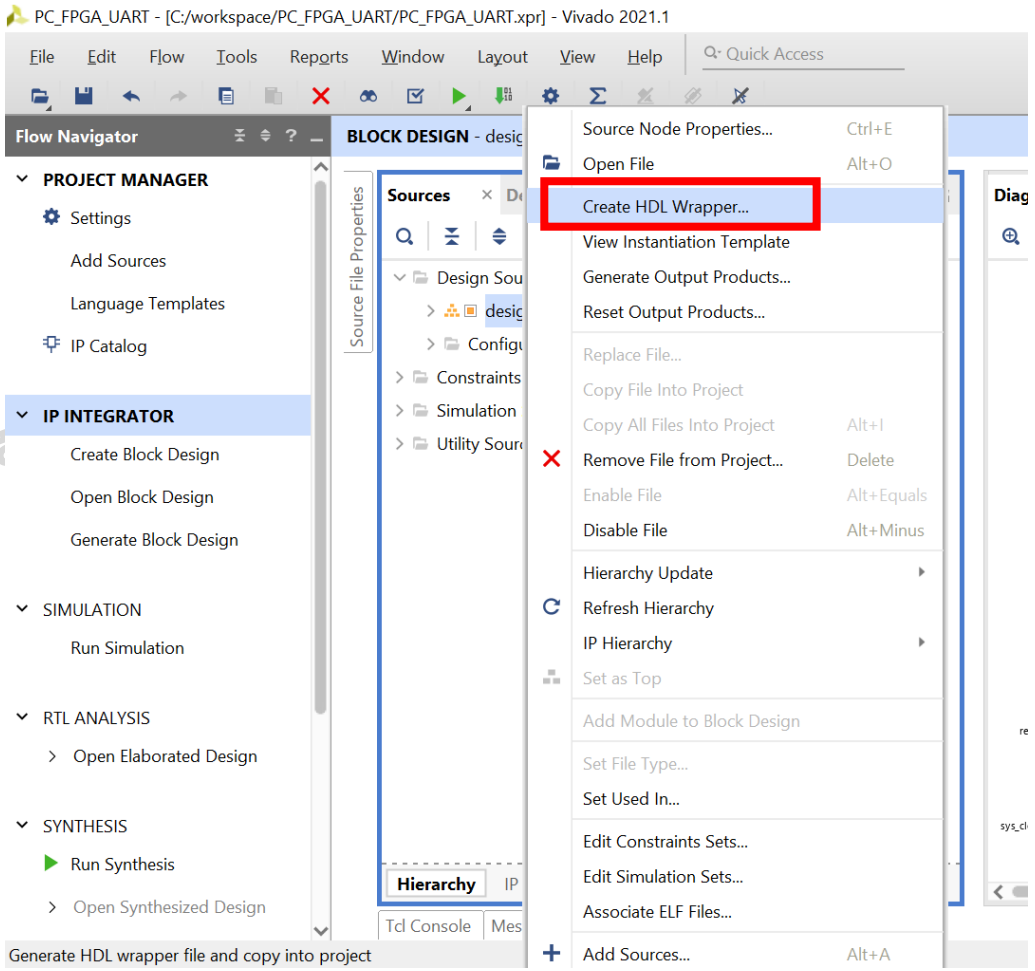
Block diagram

- MicroBlaze
- UART lite
- DRAM controller: Memory interface generator (MIG)
- Bus interconnect
- Cache



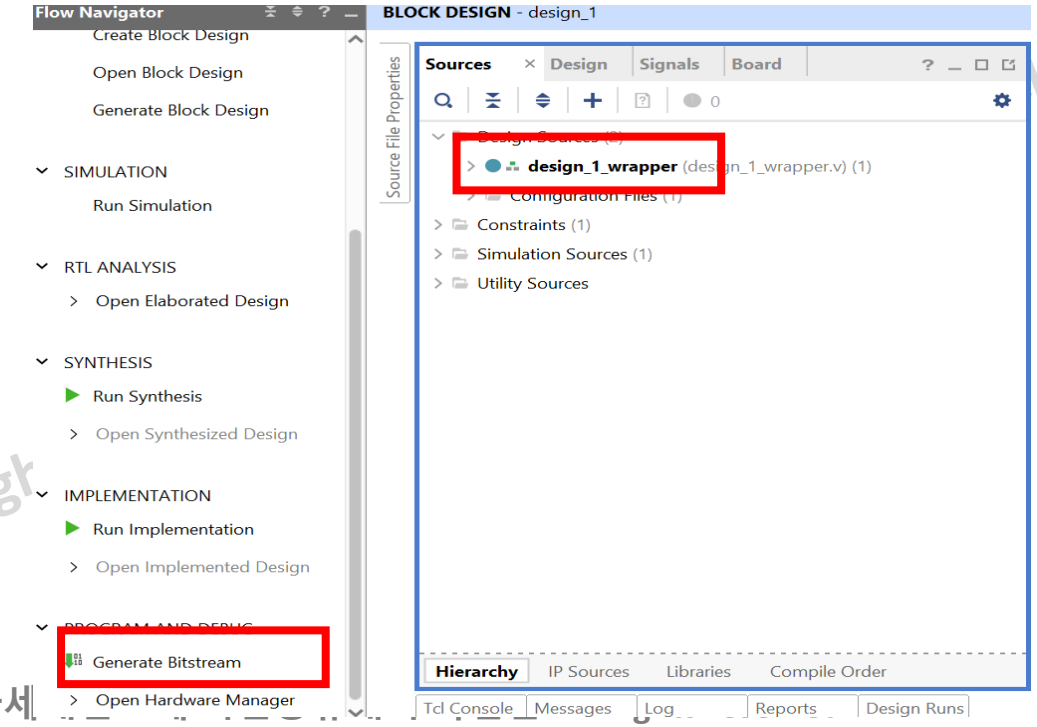
Generating bitstream and exporting hardware

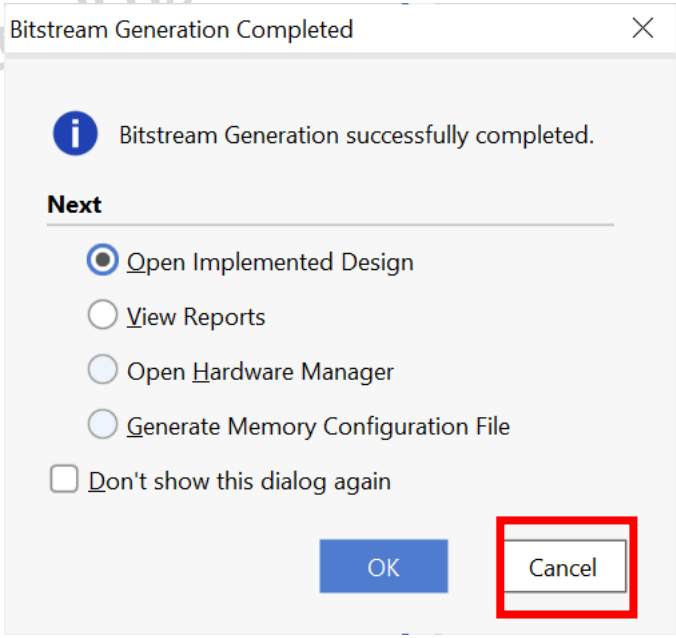
- Now, we will generate bitstream so that we can program the FPGA board with our design.
- In the Sources tab, right-click on design_1 and select create HDL Wrapper.
- Then select “Let Vivado…” and press OK.



Generating bitstream and exporting hardware

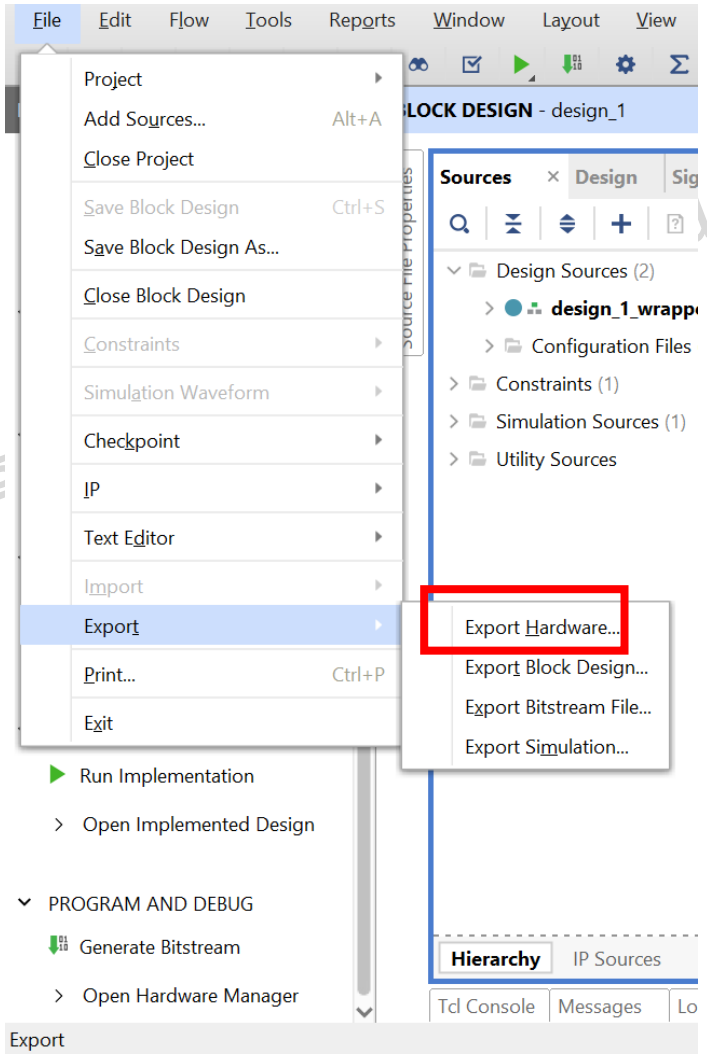
- After the previous step, design_1_wrapper.v is created.
- Now, click on Generate Bitstream
- Click next/OK at the pop-up window, then Vivado will start generating bitstream (it can take a while).
- As it finishes, a new window will open. Just press Cancel.





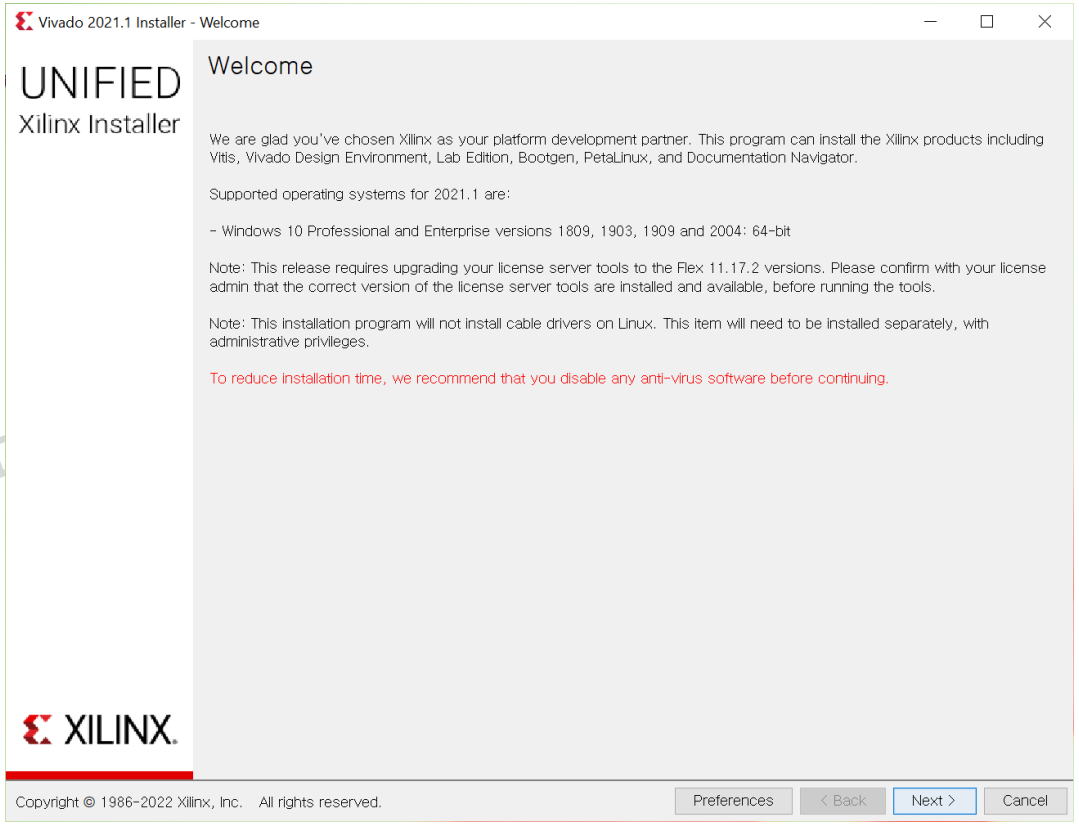
Generating bitstream and exporting hardware

- Now, we will export our hardware.
- Go to File -> Export -> Export Hardware.
- Select include bitstream
- Name the XSA file and select finish



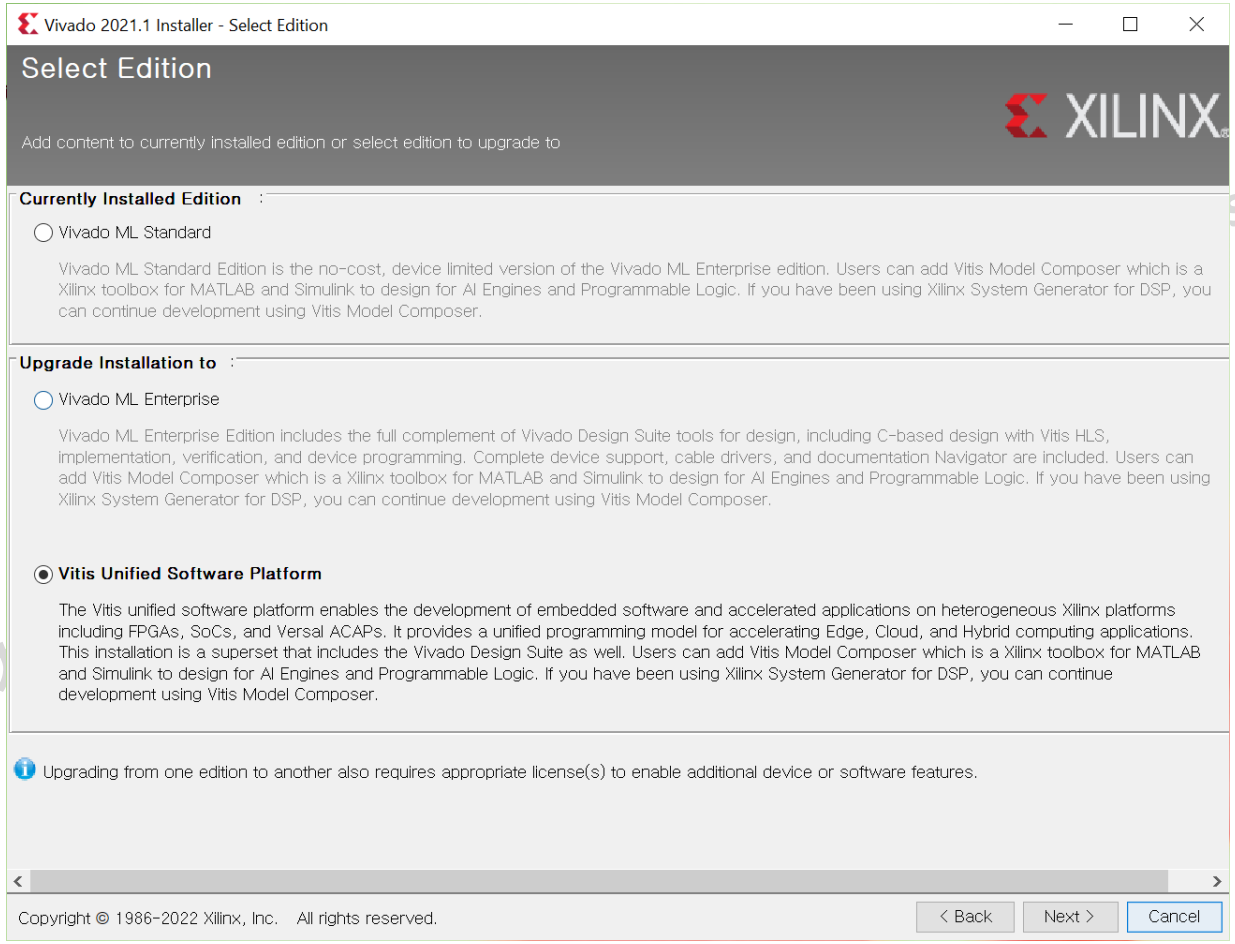
Firmware: Install Vitis

- We need to install Vitis, a tool we will use to write the firmware and run it on board.
- Search and run "Xilinx information center" on your Windows search (if you already installed Vivado)
- Click the "Manage Installs" tab, and select "Xilinx Design Tools..." -> "Add Tools/Devices"
- Then this window will open. Click next and enter your Xilinx account and password.



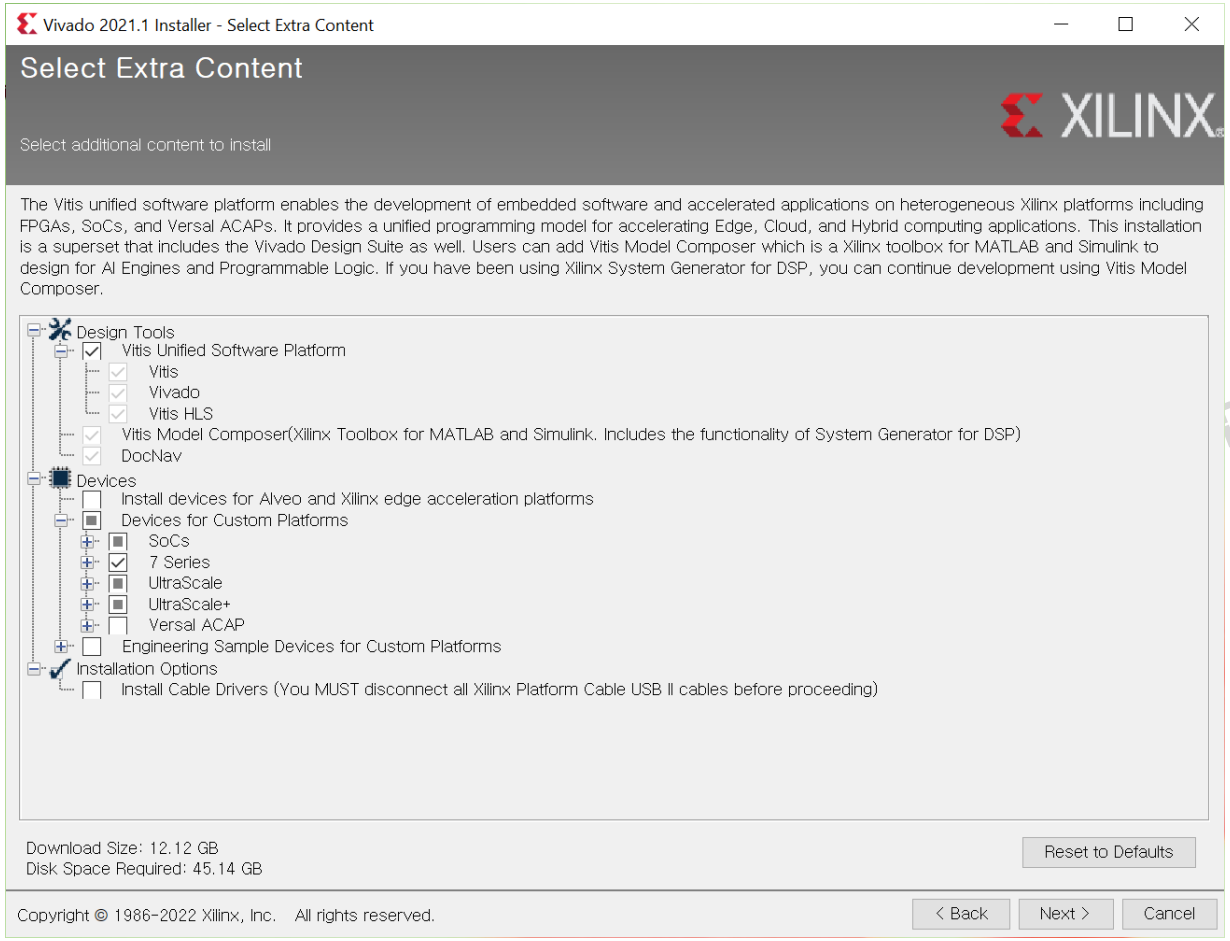
Install Vitis

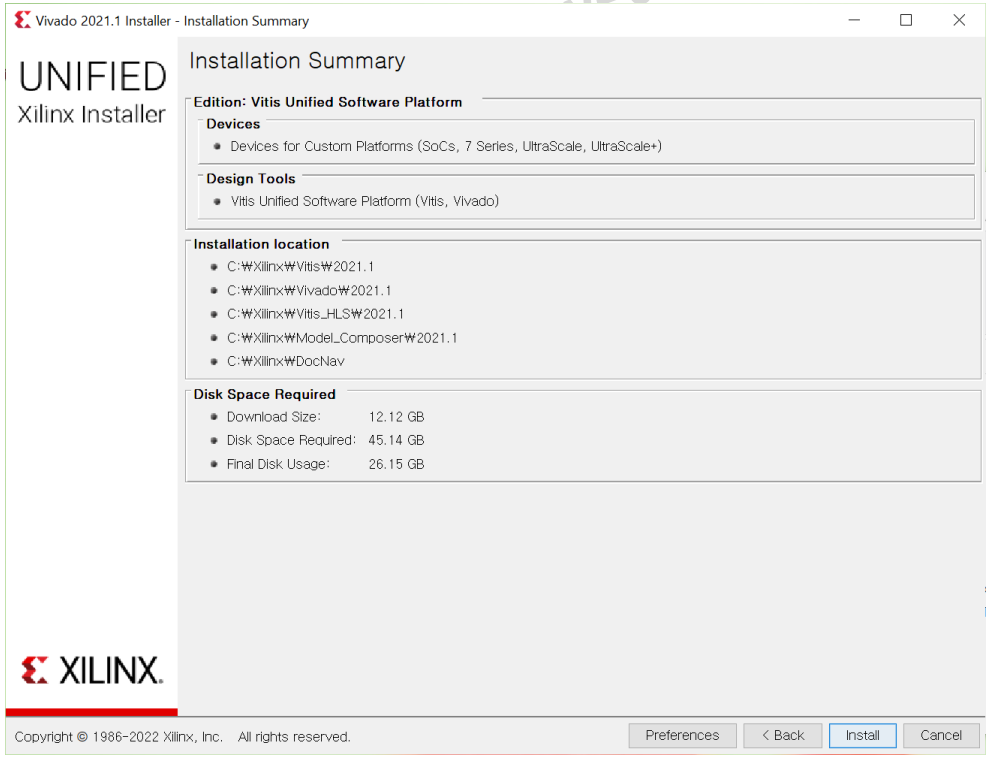
- At this page, select Vitis Unified Software Platform, and click next.



Install Vitis

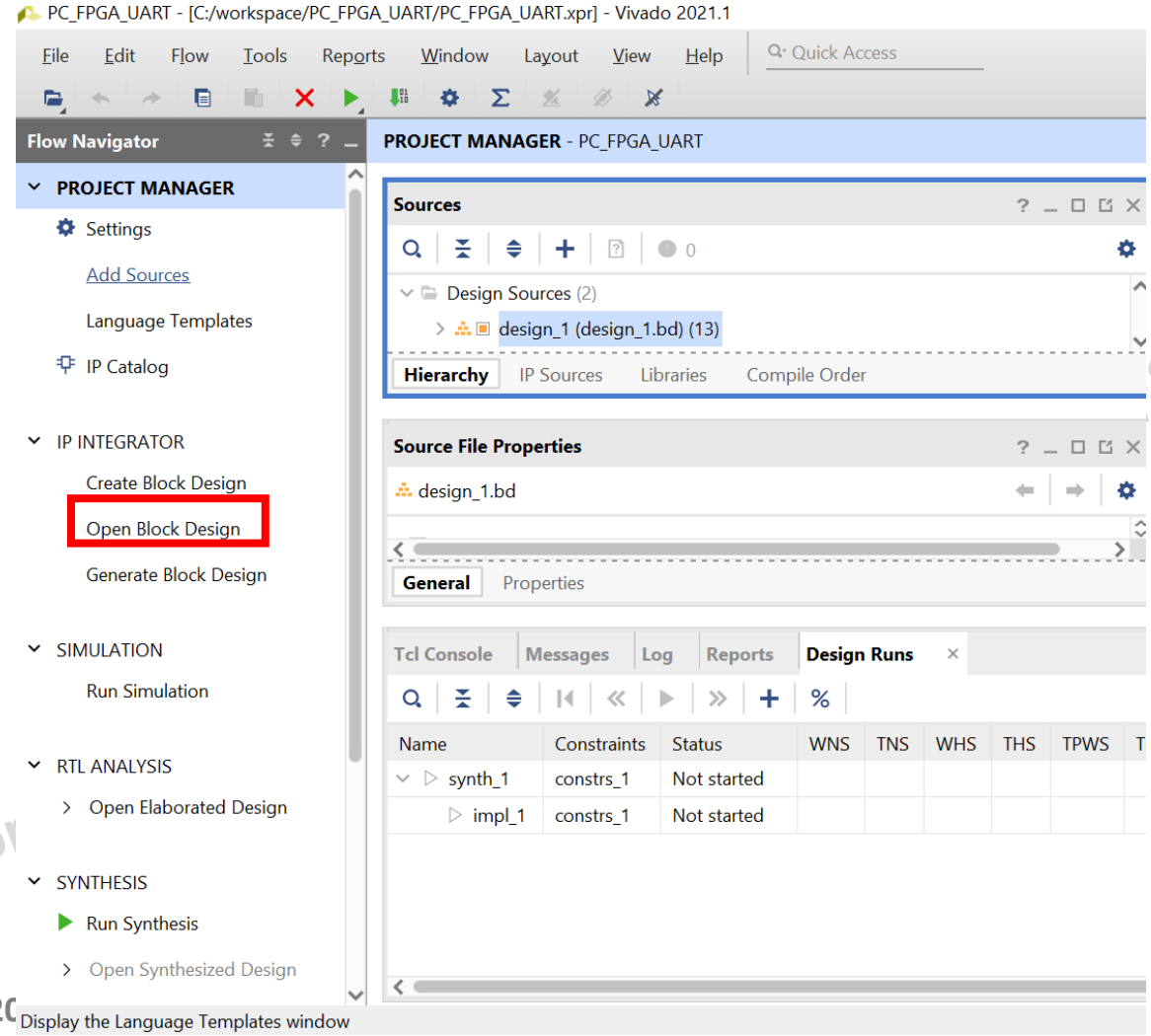
- Then, select as the picture and press next, and install(it takes a while).





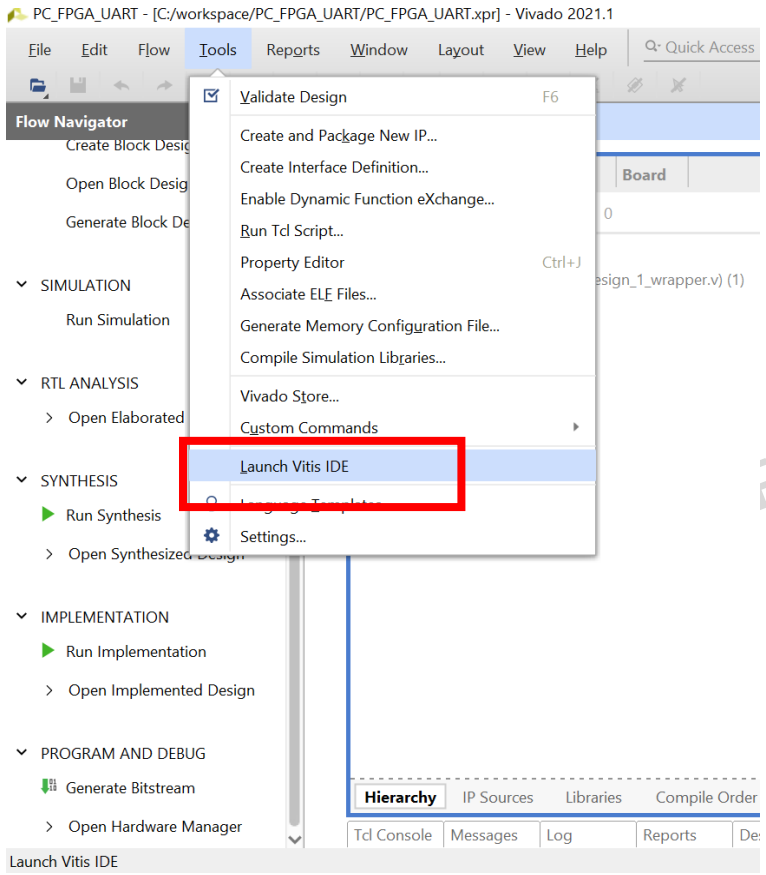
Creating a custom IP

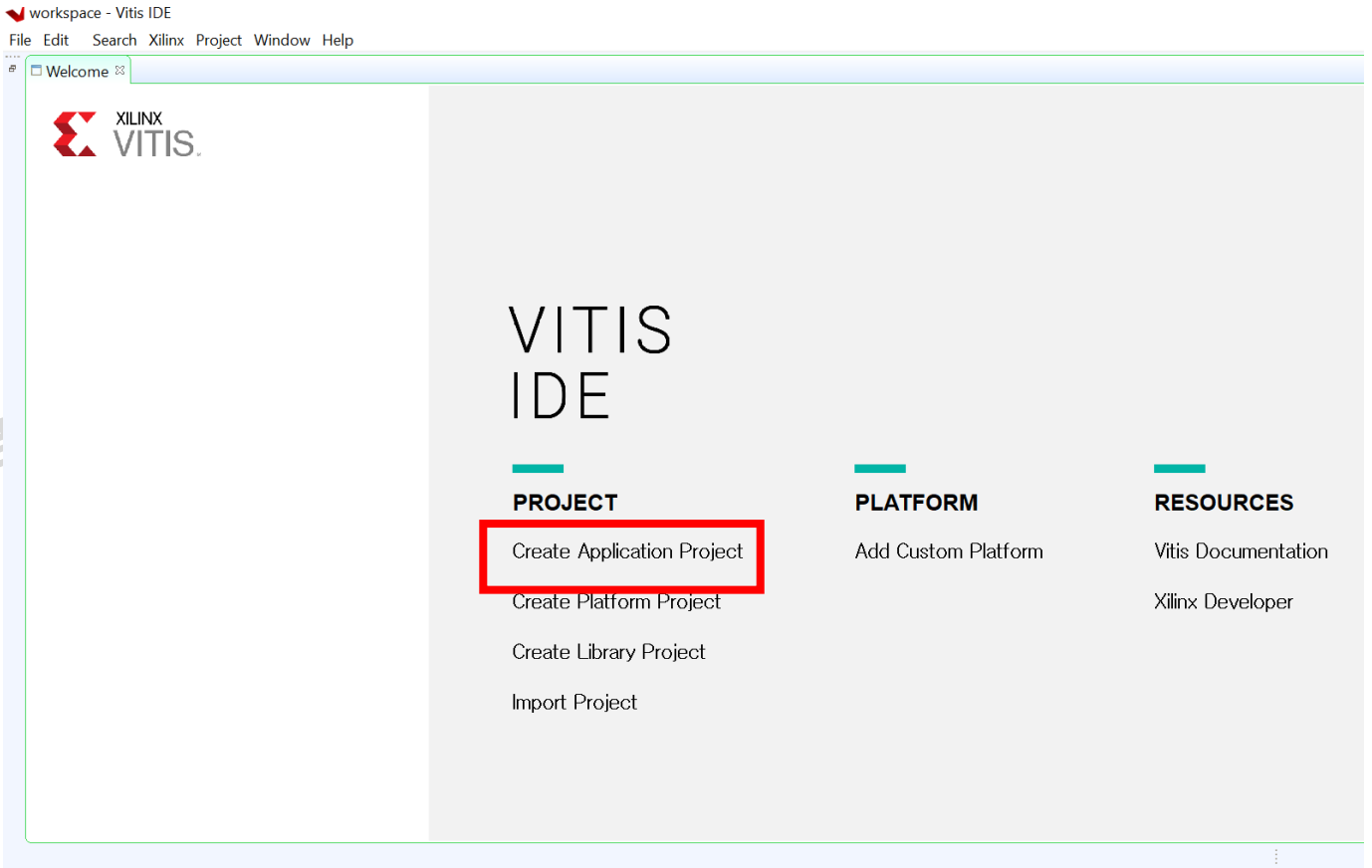
- Click "Open Block Design". Here, we can view and edit the block diagram of our hardware.



Creating a Vitis project

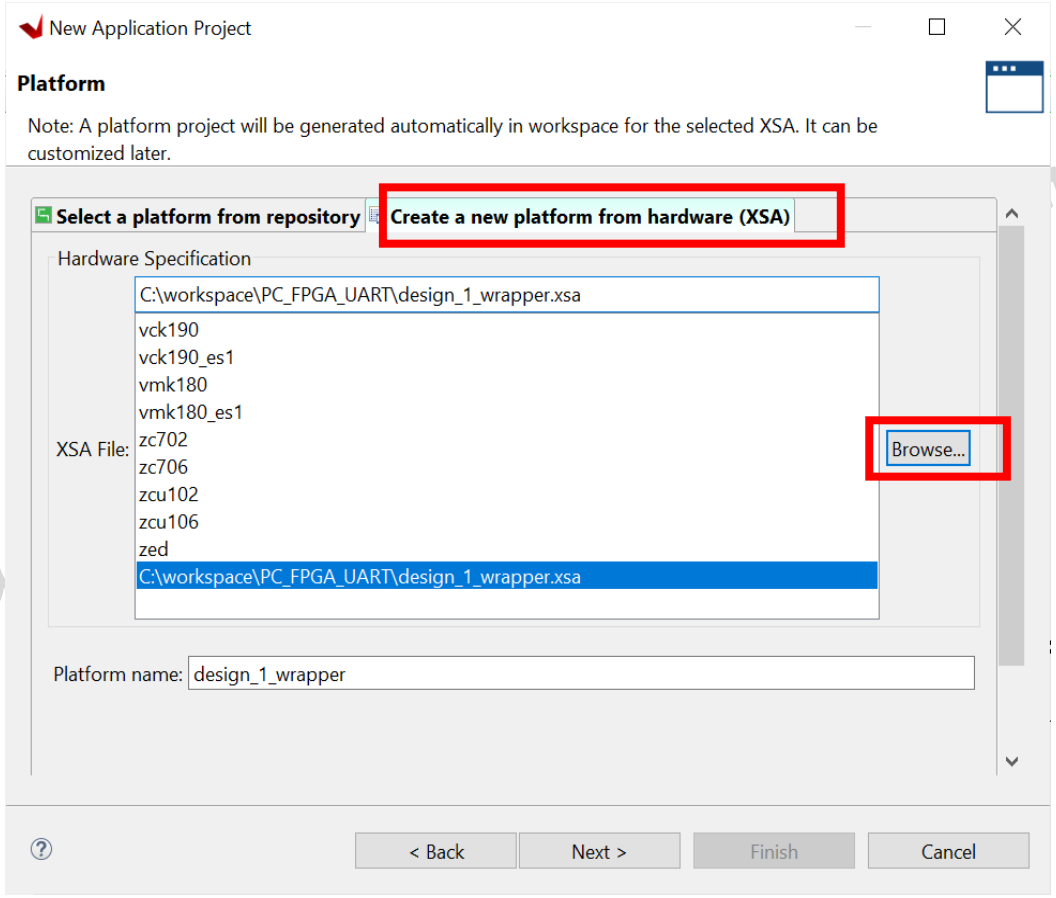
- With our .xsa file, we can create a Vitis project.
- First, go to Tools -> Launch Vitis IDE. Then select Create Application Project.





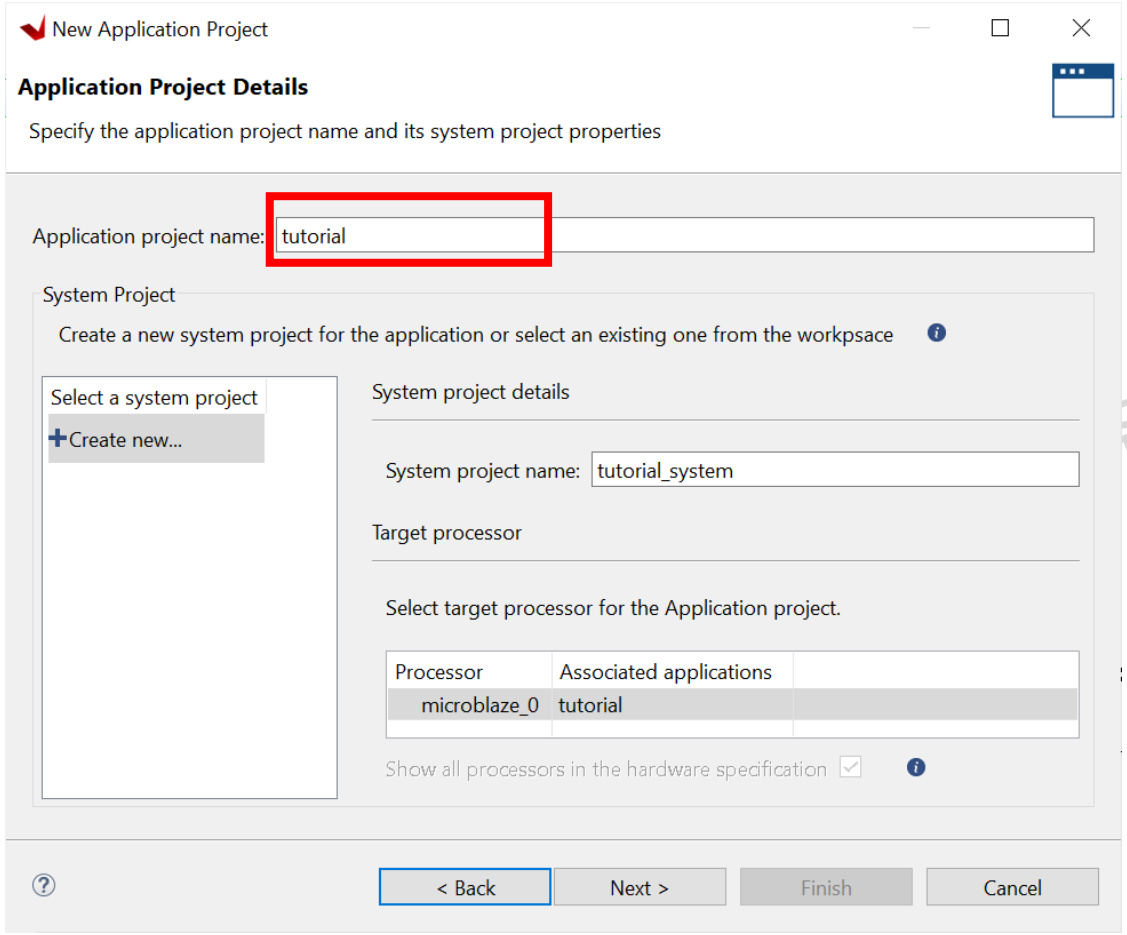
Creating a Vitis project

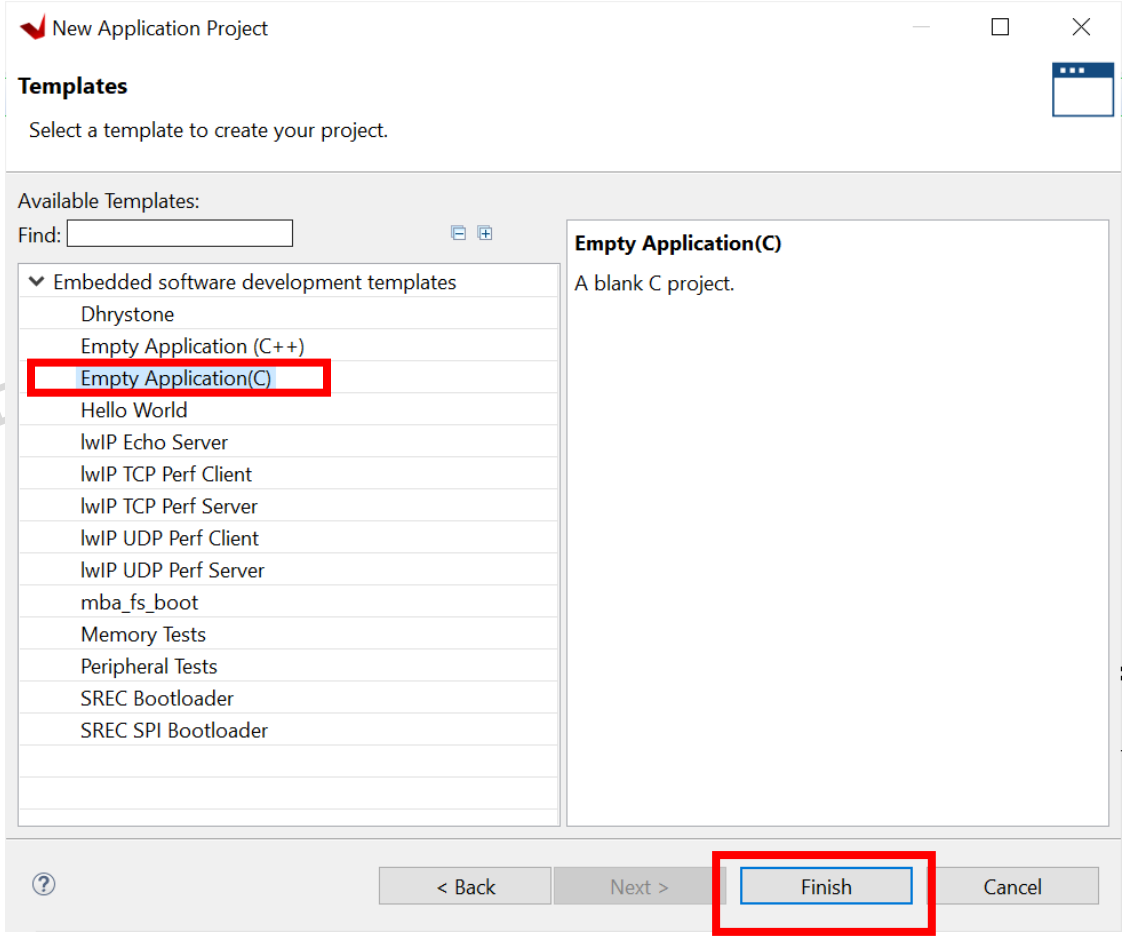
- Now, select the "Create a new platform from hardware(XSA) tab.
- Select Browse and select the .xsa file that has been created at the previous step.



Creating a Vitis project

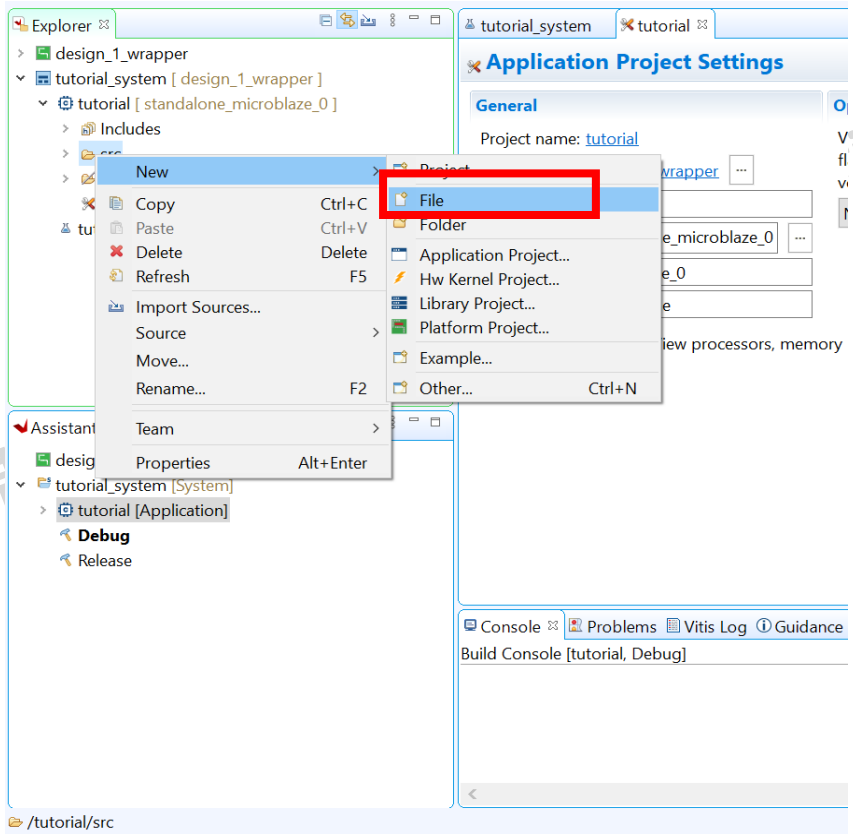
- At the following, type in the application project name and select "Empty Application(C)"

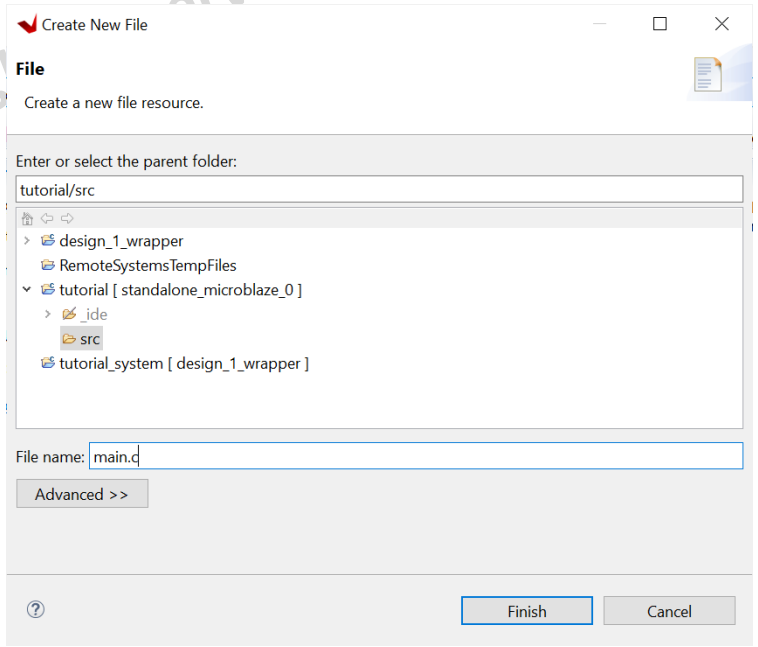




Creating a Vitis project

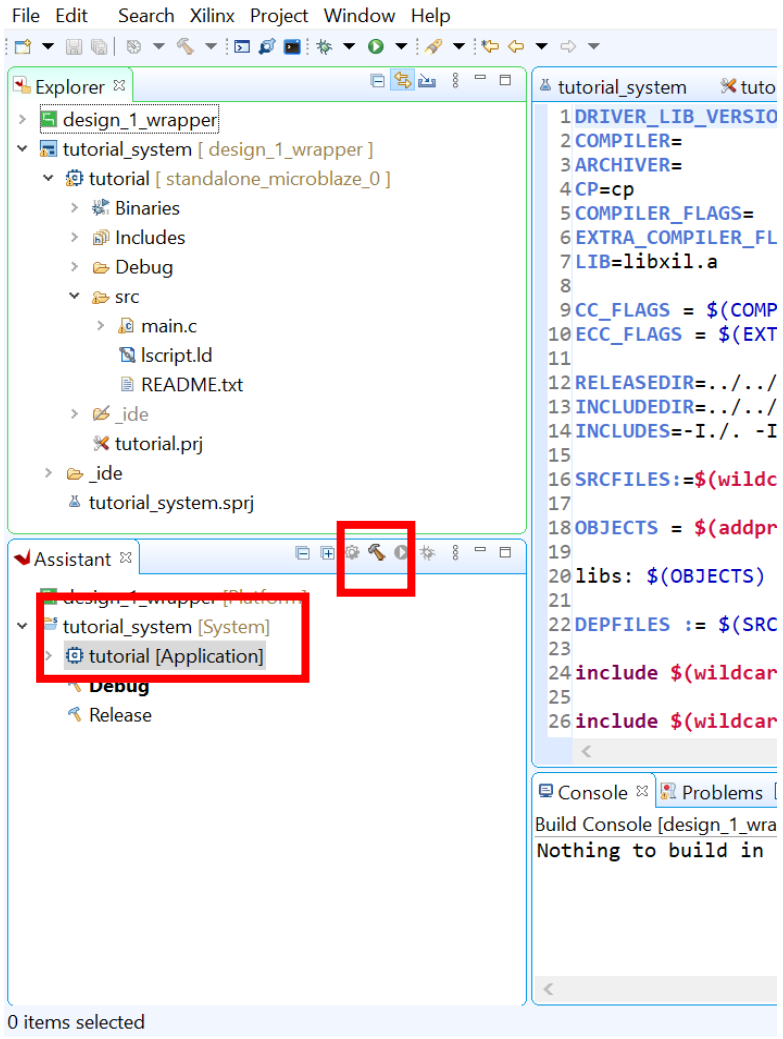
- Now an application project is made.
- Let's add our code in the project.
- Right-click tutorial_system -> tutorial -> src
- Then, select new -> file
- Create main.c
- Copy firmware₩main.c
  Into the created file





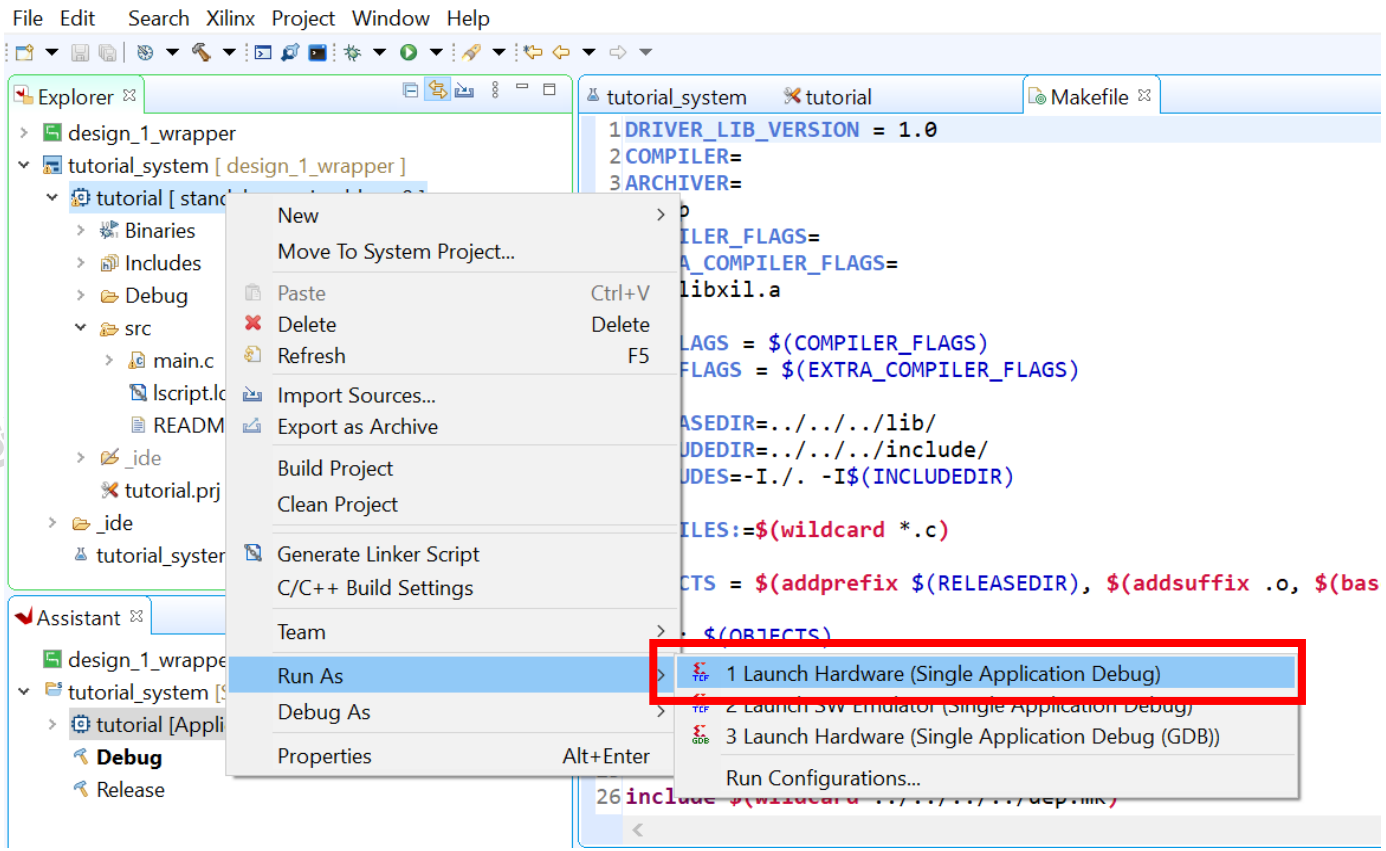
Creating a Vitis project

- Now, we are ready to build.
- In the Assistant tab, select tutorial_system and press the build button



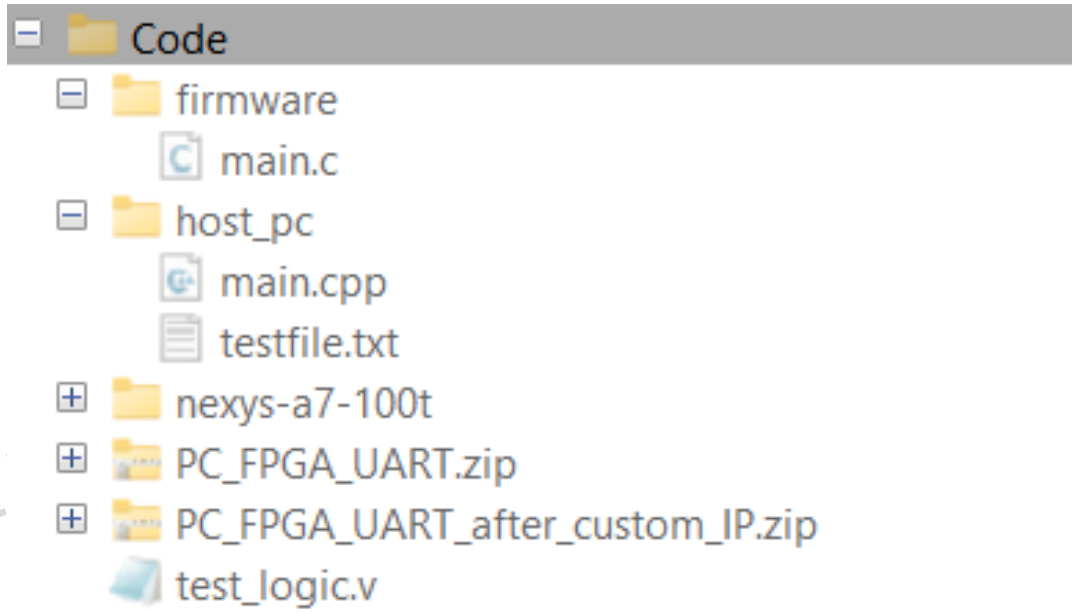
Creating a Vitis project

- Now, in the Explorer tab, select tutorial_system -> tutorial and right click.
- Select Run As -> 1 Launch Hardware
- main.c will run on the board now!



Code explanation

- Host PC code
  - main.cpp
  - testfile.txt
- Firmware code
  - main.c



host_pc₩main.cpp

Function Declarations

Main

```
16  void* open_port(const char* port_name);
17  void close_port(HANDLE comport_handle);
18  void test_hello(void* comport_handle);
19  void test_echo(void* comport_handle);
20  void store_file(void* comport_handle, const char* file_name, uint32_t base_addr, uint32_t words_to_send);
21  void verify_file(void* comport_handle, const char* file_name, uint32_t base_addr, uint32_t words_to_recv);
22  void store_testip(void* comport_handle, uint32_t offset, uint32_t data);
23  void load_testip(void* comport_handle, uint32_t offset);
24
25  int main() {
26      HANDLE comport_handle = open_port("COM8");
27      test_hello(comport_handle);
28      //test_echo(comport_handle);
29      //store_file(comport_handle, "testfile.txt", 0x00002400, 2304);
30      //verify_file(comport_handle, "testfile.txt", 0x00002400, 2304);
31      //store_testip(comport_handle, 0x00, 0xffeeddcc);
32      //store_testip(comport_handle, 0x04, 0xaabbccdd);
33      //load_testip(comport_handle, 0x08);
34      //load_testip(comport_handle, 0x0c);
35      //load_testip(comport_handle, 0x10);
36      //load_testip(comport_handle, 0x14);
37      //load_testip(comport_handle, 0x18);
38      //load_testip(comport_handle, 0x1c);
39      close_port(comport_handle);
40      return 0;
41  }
```

1. Open a USB UART port (e.g., COM8)
2. Execute a function

3. Close the COM port

host_pc₩main.cpp

Function descriptions

- void* open_port(const char* port_name);
  - Establishes connection with FPGA and returns HANDLE variable that is used in other functions
  - port_name must be checked
    - Windows: Check it in Device Manager
- void close_port(HANDLE comport_handle);
  - Closes connection with FPGA
- void test_hello(void* comport_handle);
  - Sends 1-byte message to FPGA, the firmware will send back "Hello, World!"
- void test_echo(void* comport_handle);
  - Sends 16-byte message to FPGA, the firmware will send the message back to PC.

host_pc₩main.cpp

Function descriptions

- void store_file(void* comport_handle, const char* file_name, uint32_t base_addr, uint32_t words_to_send);
  - Stores a file to the FPGA's DRAM.
- void verify_file(void* comport_handle, const char* file_name, uint32_t base_addr, uint32_t words_to_recv);
  - Loads a part of the FPGA's DRAM, and check if it matches the file on PC.
- void store_testip(void* comport_handle, uint32_t offset, uint32_t data);
  - Stores a 32-bit value to one of registers in the test_IP(custom IP)
- void load_testip(void* comport_handle, uint32_t offset);
  - Loads a 32-bit register in the test_IP to PC.

store_file (...) (1/3)

```
151  void store_file(void* comport_handle, const char* file_name, uint32_t base_addr, uint32_t words_to_send) {
152      //file name -> name of file to send. assuming txt file
153      //words to send -> number of words in file
154      //open file
155      FILE* f_send = fopen(file_name, "r");
156
157      //allocate buffer and read file to buffer
158      uint32_t* buffer = (uint32_t*)calloc(words_to_send, sizeof(uint32_t));
159
160      if (f_send != NULL) {
161          for (int i = 0; i < words_to_send; ++i) fscanf_s(f_send, "%X", &buffer[i]);
162      }
163
164      fclose(f_send);
165
166      //send 1-byte command to board
167      DWORD dw_byte_written = 0;
168      DWORD dw_byte_read = 0;
169      uint8_t mode_signal = MODE_STORE_RAM;
170
171      if (WriteFile(comport_handle, &mode_signal, 1, &dw_byte_written, NULL)) {
172          fprintf(stderr, "[  OK  ] MODE 0x%02X\n", mode_signal);
173      }
174      else {
175          fprintf(stderr, "[FAILED] MODE 0x%02X\n", mode_signal);
176          return;
177      }
```

Read a file and store its content in a buffer

Store to DRAM

Send a command

store_file (…) (2/3)

```
193     //send file metadata(DRAM base addr / file size)
194     uint32_t encoded_data[2];
195     encoded_data[0] = base_addr;
196     encoded_data[1] = words_to_send;
197
198     if (WriteFile(comport_handle, &encoded_data, 8, &dw_byte_written, NULL)) {
199         fprintf(stderr, "[  OK  ] SEND CMD\n");
200     }
201     else {
202         fprintf(stderr, "[FAILED] SEND CMD\n");
203         return;
204     }
205
206     if (ReadFile(comport_handle, &data_read, 16, &dw_byte_read, NULL)) { //expected response is "CMD receive"
207         fprintf(stderr, "[  OK  ] Response: ");
208         for (int i = 0; i < 16; i++) {
209             fprintf(stderr, "%c", data_read[i]);
210         }
211         fprintf(stderr, "\n");
212     }
213     else {
214         fprintf(stderr, "[FAILED] Resonse:\n");
215         return;
216     }
```

Send metadata
- Base address
- # 32-bit words

Check Response

store_file (...) (3/3)

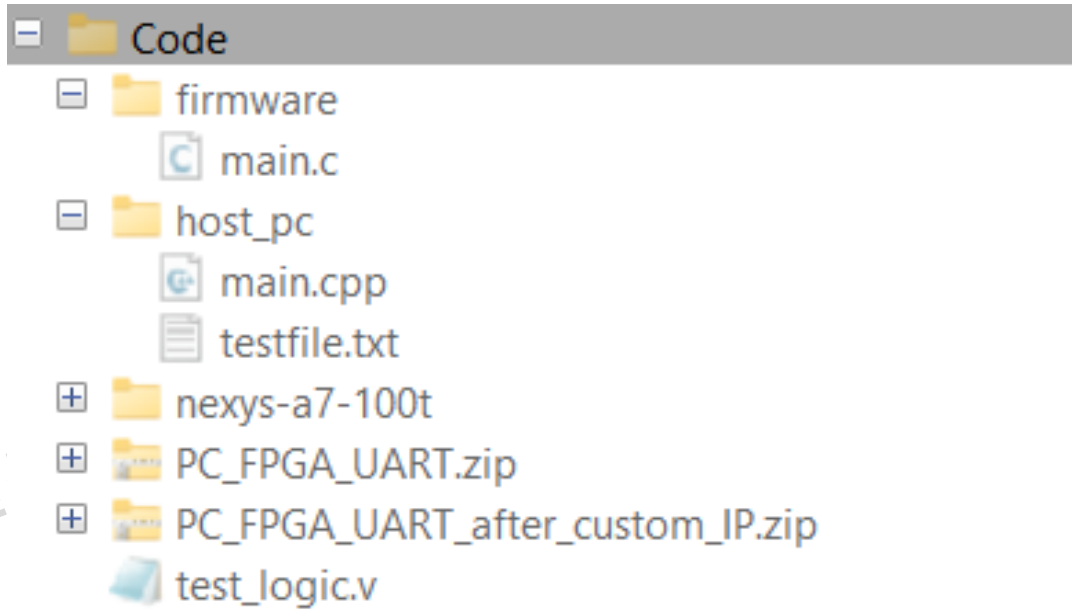
```
219     //send file
220     if (WriteFile(comport_handle, buffer, words_to_send * sizeof(uint32_t), &dw_byte_written, NULL)) {
221         fprintf(stderr, "[  OK  ] SEND %s\n", file_name);
222     }
223     else {
224         fprintf(stderr, "[FAILED] SEND %s\n", file_name);
225         return;
226     }
227
228
229     //get response
230     if (ReadFile(comport_handle, &data_read, 16, &dw_byte_read, NULL)) { //expected response is "Store complete"
231         fprintf(stderr, "[  OK  ] Response: ");
232         for (int i = 0; i < 16; i++) {
233             fprintf(stderr, "%c", data_read[i]);
234         }
235         fprintf(stderr, "\n");
236     }
237     else {
238         fprintf(stderr, "[FAILED] Resonse:\n");
239         return;
240     }
241
242     free(buffer);
```

Send data that are stored in "buffer"

Check Response

Code explanation

- Host PC code
  - main.cpp
  - testfile.txt
- Firmware code
  - main.c



firmware₩main.c

- Function descriptions
- int init_uart(u16 DeviceID);
  - Initializes UART
- void uart_recv(u8 num_bytes);
  - Receives num_bytes bytes from PC via UART, to global variable RecvBuffer[100]
- void uart_send(u8* data, u32 num_bytes);
  - Sends num_bytes bytes from FPGA to PC via UART. 'data' is the data to be sent.
- void write_addr(u32 addr, u8* data, u32 num_bytes);
  - Writes 'num_bytes' bytes to 'addr'. 'data' is the data to be sent.
- void read_addr(u32 addr, u8* readbuffer, u32 num_bytes);
  - Reads 'num_bytes' bytes from 'addr' to readbuffer.

firmware₩main.c

- In the main function
- The firmware waits for a message from the host PC.
- Depending on the message the host PC has sent, the firmware acts differently.
  - 0x01: send "Hello, World!" back to PC.
  - 0x02: receive 16-byte string from PC and send it back to PC.
  - 0x03: Receives data from the PC, and stores it in the DRAM, at a specified address offset.
  - 0x04: Load data from DRAM at a specified address offset, and sends it to PC.
  - 0x05: Receives 32-bit data from PC and stores it in the custom-IP(test_IP) at the specified offset.
  - 0x06: Loads a 32-bit register value at the specified offset of custom-IP(test_IP), and sends it to the PC.

Firmware: Main function

```
12  XUartLite UartLite;     /* Instance of the UartLite Device */
13
14  /*
15   * The following buffers are used in this example to send and receive data
16   * with the UartLite.
17   */
18  uint8_t RecvBuffer[100];    /* Buffer for Receiving Data */
19
20  uint8_t cHello[16]  =   { ' ', ' ', 'H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!', ' '};
21  uint8_t cWaiting[16]    =   { ' ', ' ', 'W', 'a', 'i', 't', 'i', 'n', 'g', '_', 'C', 'M', 'D', ' ', ' ', ' '};
22  uint8_t cCmd_recv[16]   =   { ' ', ' ', 'C', 'M', 'D', '_', 'R', 'e', 'c', 'e', 'i', 'v', 'e', ' ', ' ', ' '};
23  uint8_t cComplete[16]   =   { ' ', 'S', 't', 'o', 'r', 'e', ' ', 'c', 'o', 'm', 'p', 'l', 'e', 't', 'e', ' '};
24
25  int init_uart(u16 DeviceID);
26  void uart_recv(u8 num_bytes);
27  void uart_send(u8* data, u32 num_bytes);
28  void write_addr(u32 addr, u8* data, u32 num_bytes);
29  void read_addr(u32 addr, u8* readbuffer, u32 num_bytes);
30
31  int main(){
32
33      init_uart(UARTLITE_DEVICE_ID);
34
35  >   while (1) {···
114     }
115     return 0;
116 }
```

Function declarations

1. Initialize UART
2. Infinitive loop

Firmware: Loop

```
35      while (1) {
36          uart_recv(1);
37          if (RecvBuffer[0] == 0x01) {
38              uart_send(cHello, 16);
39          }
40          else if(RecvBuffer[0] == 0x02) {
41              uart_recv(16);
42              uart_send(RecvBuffer, 16);
43          }
44          else if(RecvBuffer[0] == 0x09) {
45              uart_recv(8);
46
47              uart_send(RecvBuffer, 8);
48          }
49          else if(RecvBuffer[0] == 0x03){ //store to DRAM
50              //send response
51              uart_send(cWaiting, 16);
52
53              //receive base addr and data size
54              uart_recv(8);
55              uint32_t addr = (RecvBuffer[3] << 24) + (RecvBuffer[2] << 16) + (RecvBuffer[1] << 8) + RecvBuffer[0];
56              uint32_t data_size = (RecvBuffer[7] << 24) + (RecvBuffer[6] << 16) + (RecvBuffer[5] << 8) + RecvBuffer[4];
57
58              //send response
59              uart_send(cCmd_recv,16);
60
61              //store to DRAM
62              for(int i = 0; i < data_size; i++){
63                  uart_recv(4);
64                  write_addr(XPAR_MIG_7SERIES_0_BASEADDR + addr + i * 4, RecvBuffer, 4);
65              }
66              uart_send(cComplete, 16);
67          }
```

Firmware: Loop

```
61          //store to DRAM
62          for(int i = 0; i < data_size; i++){
63              uart_recv(4);
64              write_addr(XPAR_MIG_7SERIES_0_BASEADDR + addr + i * 4, RecvBuffer, 4);
65          }
66          uart_send(cComplete, 16);
67      }
68      else if(RecvBuffer[0] == 0x04){ //load from DRAM
69          //send response
70
71
72
73
74
75          uint32_t data_size = (RecvBuffer[7] << 24) + (RecvBuffer[6] << 16) + (RecvBuffer[5] << 8) + RecvBuffer[4];
76
77          uart_send(cCmd_recv,16);
78
79          //load from DRAM and send
80          uint8_t temp[4];
81          for(int i = 0; i < data_size; i++){
82              read_addr(XPAR_MIG_7SERIES_0_BASEADDR + addr + i * 4, temp, 4);
83              uart_send(temp, 4);
84          }
85      }
```

Store to DRAM

Recall address mapping in Page. 21

/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
/mig_7series_0/memmap	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF

Load from DRAM

Road map

Review

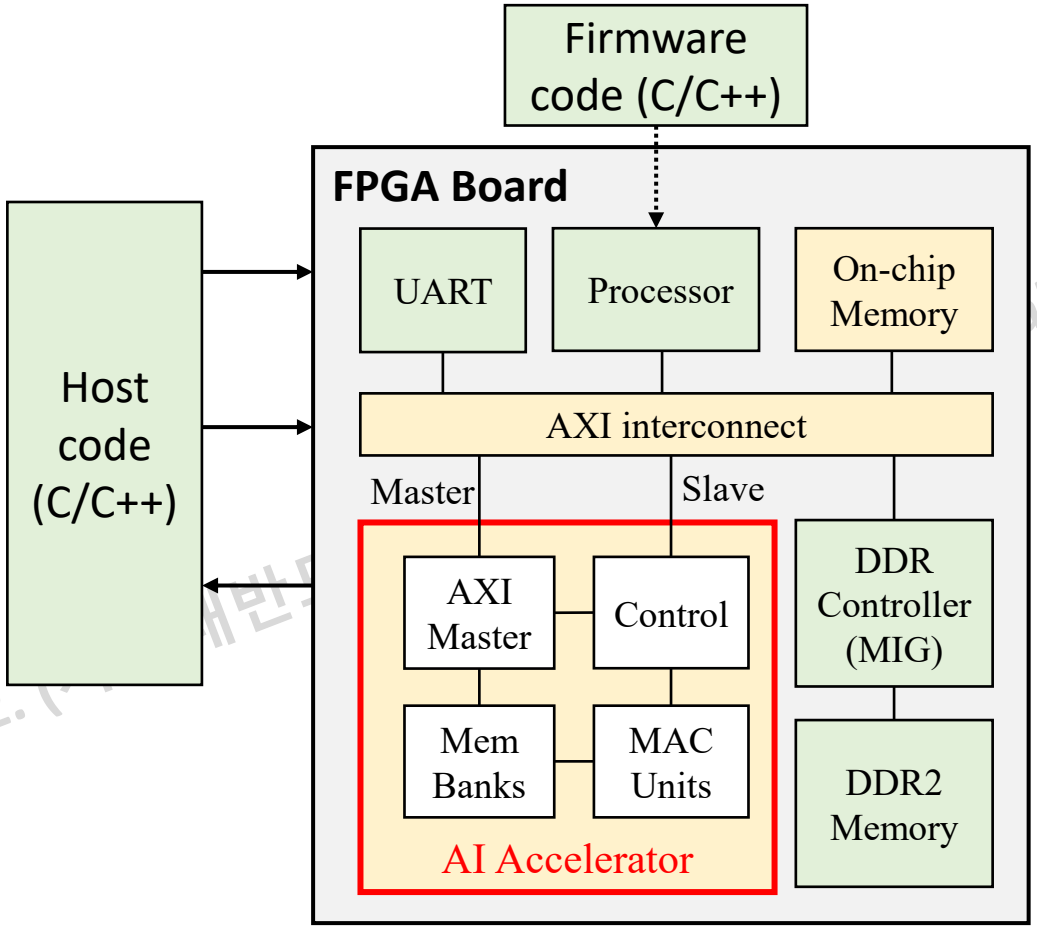
FPGA board

PC-FPGA

Custom IP

Motivation

- How to make a user-defined IP (e.g., AIX2022) and add it into the system



A simple IP

- Create a custom IP and add it to our design.
- It will be a simple logic calculator (test_logic.v)
  - Inputs
    - Two 32-bit inputs (a and b)
  - Outputs
    - Six 32-bit numbers (sum, difference, and, or, xor, xnor)
- We will map the ports in the memory as:

```
module test_logic(
input [31:0] a,
input [31:0] b,
output [31:0] sum,
output [31:0] difference,
output [31:0] bitwiseAnd,
output [31:0] bitwiseOr,
output [31:0] bitwiseXor,
output [31:0] bitwiseXNor
);
assign sum = a+b;
assign difference = a-b;
assign bitwiseAnd = a & b;
assign bitwiseOr = a | b;
assign bitwiseXor = a ^ b;
assign bitwiseXNor = a ~^ b;
endmodule
```

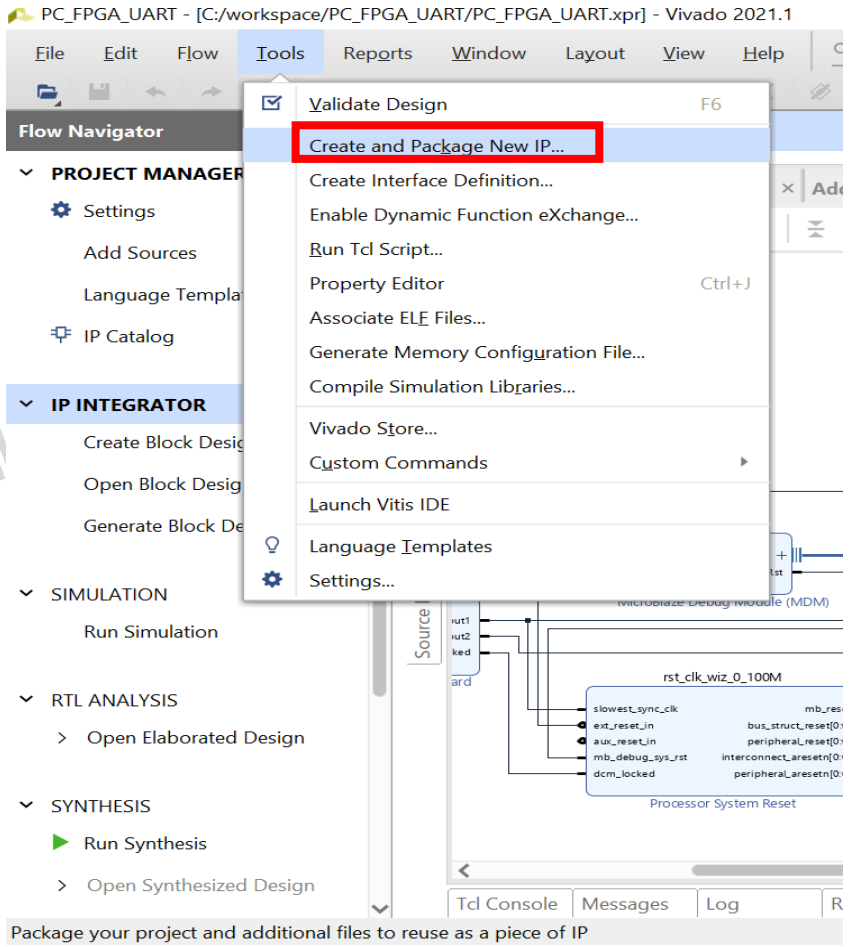
test_logic.v

Port	Offset
a	0x00
b	0x04
sum	0x08
difference	0x0c
bitwiseAnd	0x10
bitwiseOr	0x14
bitwiseXor	0x18
bitwiseXNor	0x1c

Address mapping

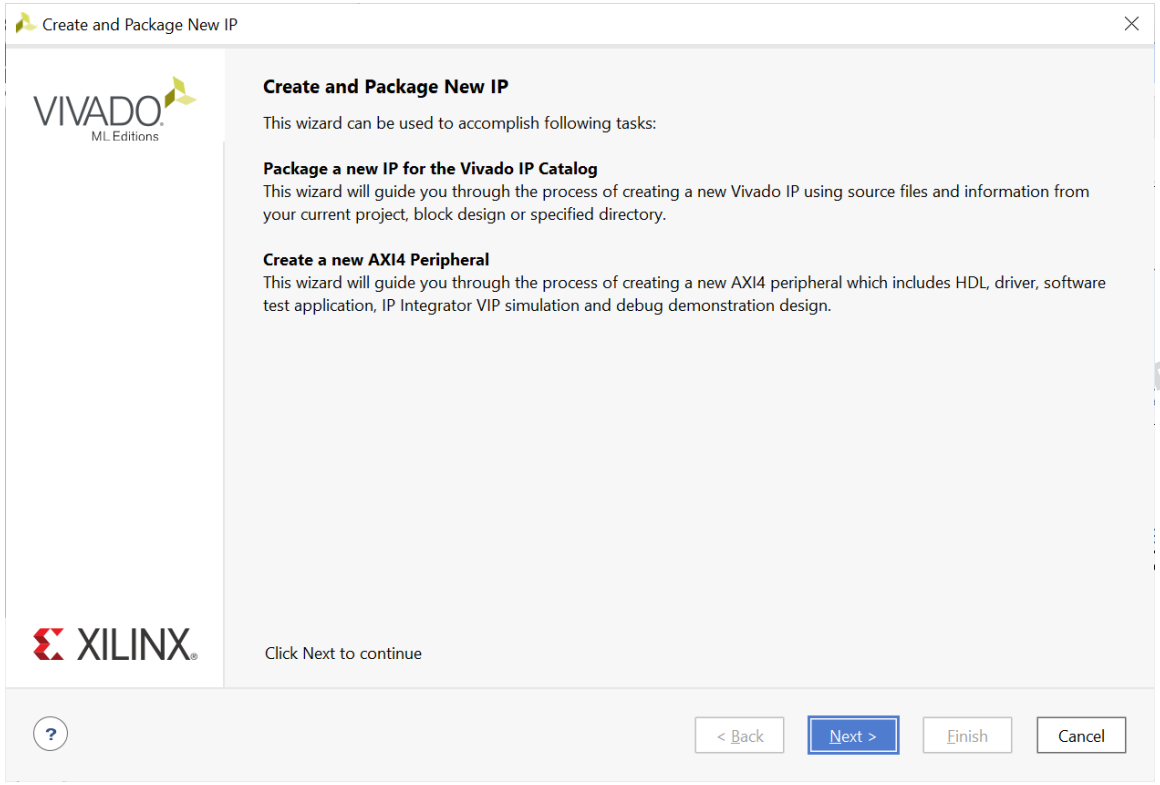
Creating a custom IP

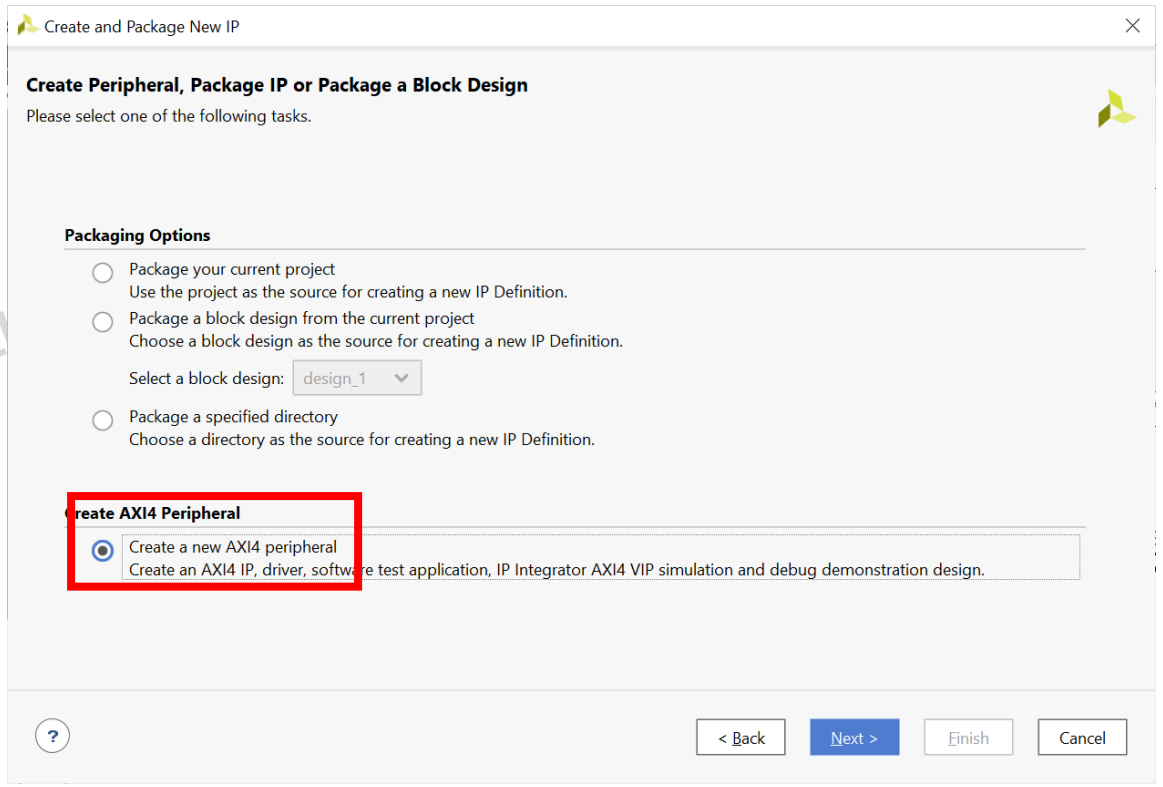
- Go to Tools -> Create and Package new IP.



Creating a custom IP

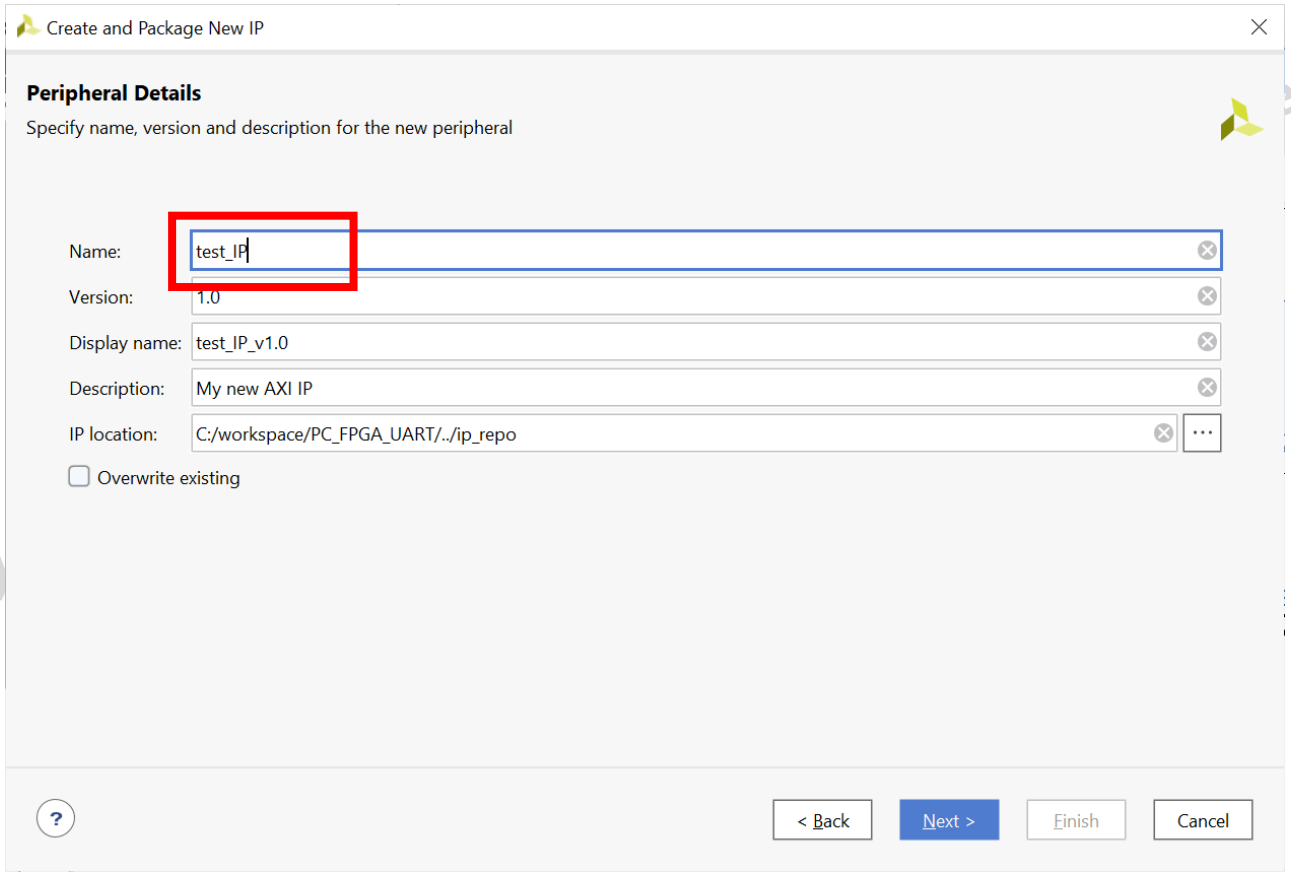
- Then, this window will appear. Select next.
- Then, select Create a new AXI4 peripheral, and select next.





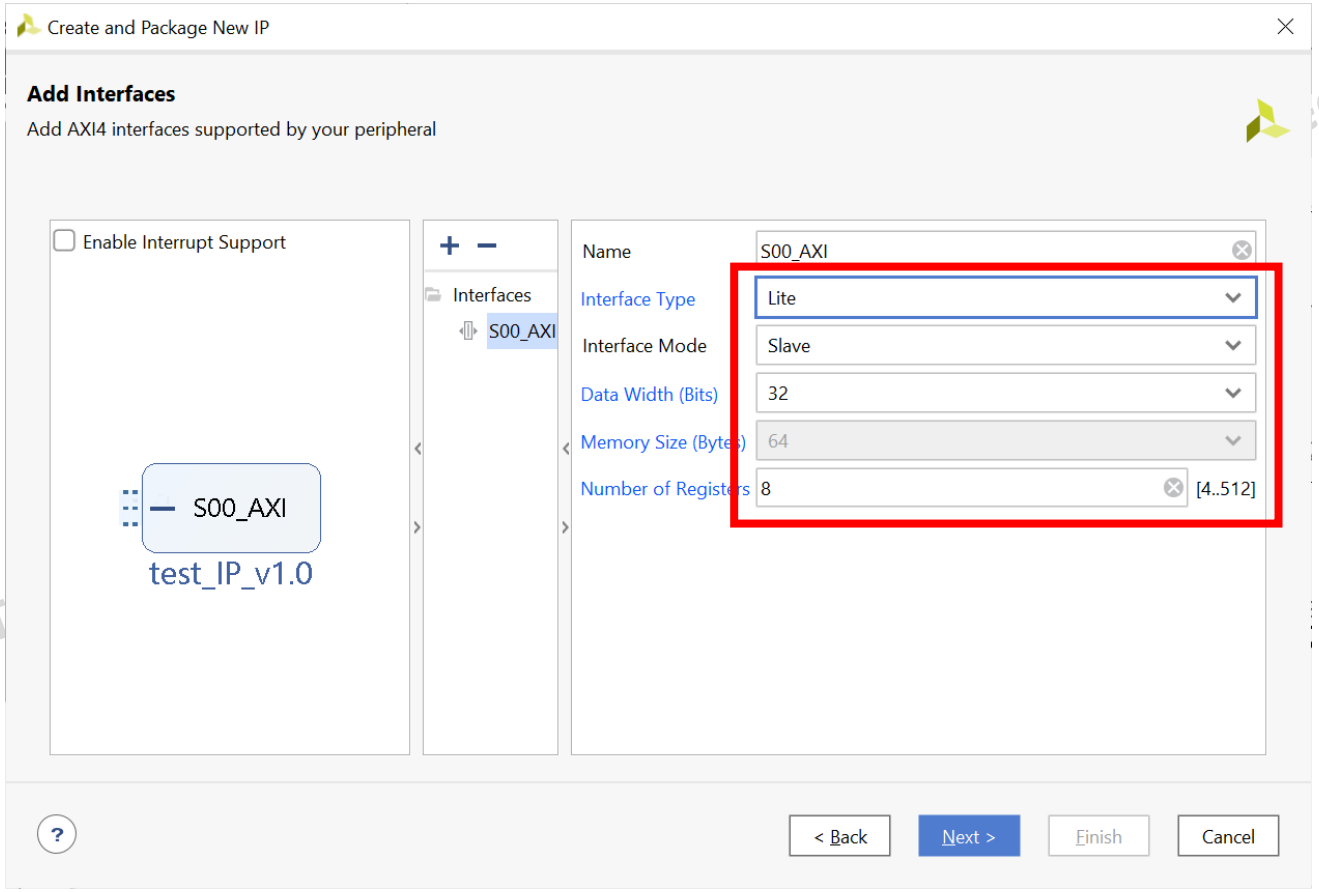
Creating a custom IP

- At the next step, select a name for the new IP.
- In this tutorial, we will name it test_IP.



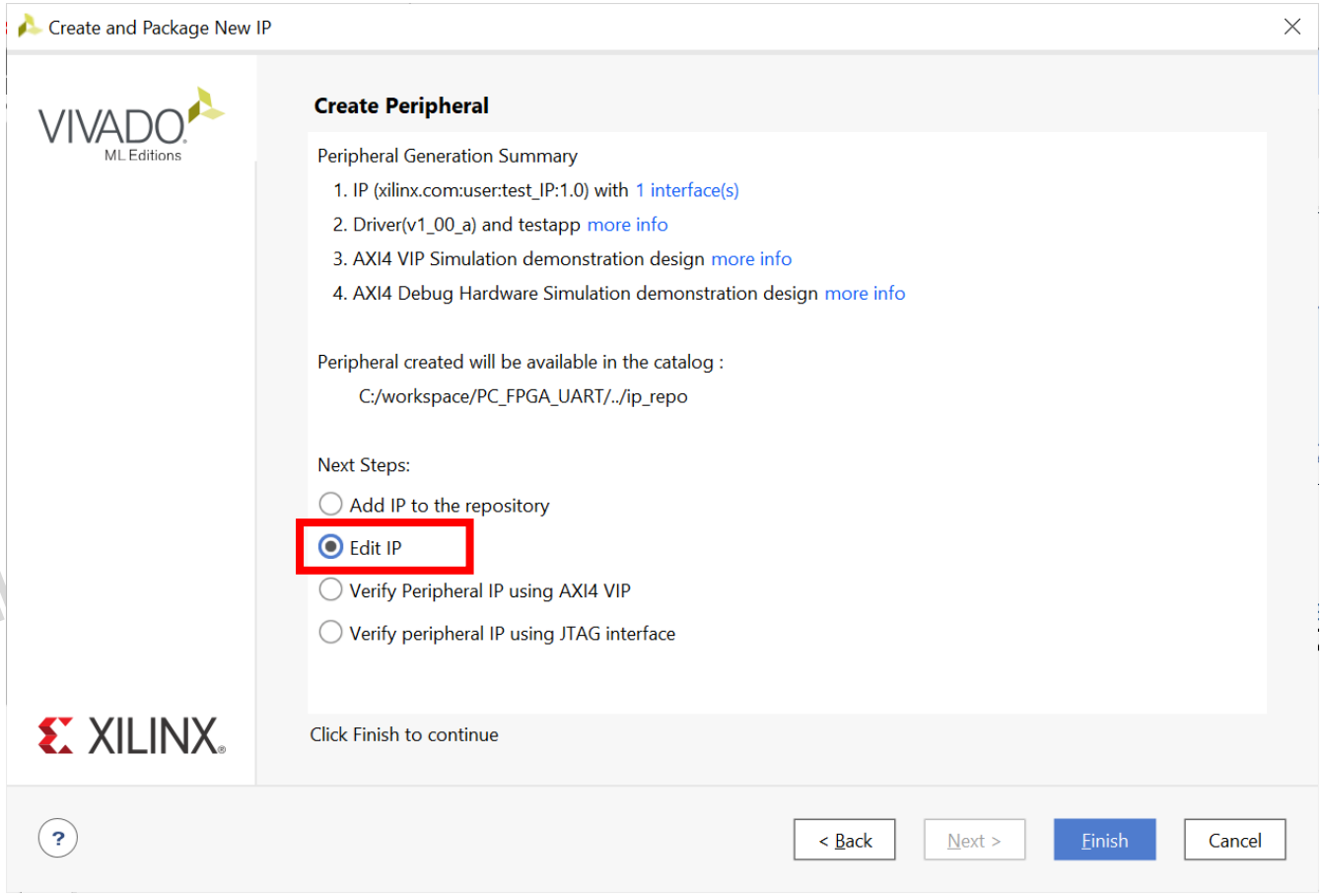
Creating a custom IP

- Here, select the settings as below.
- We will have 8 registers, one for each 32-bit port in our module(test_logic.v)



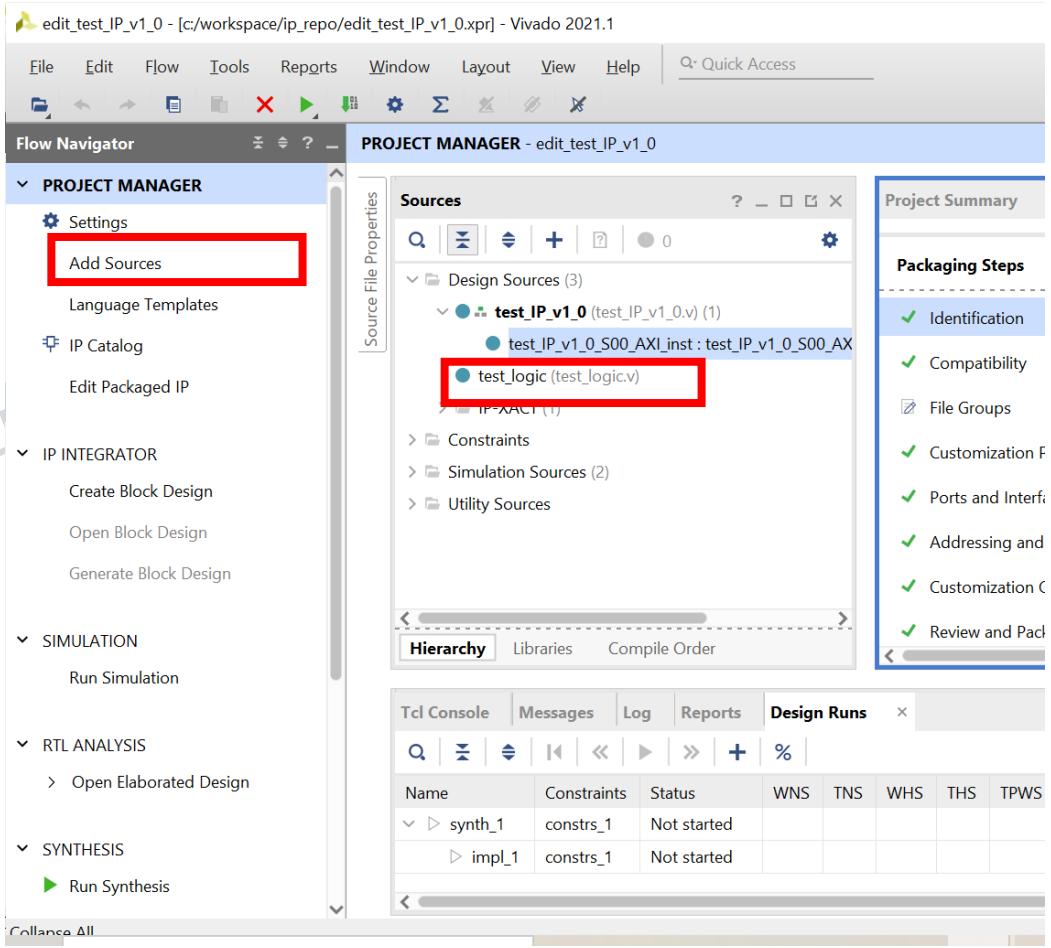
Creating a custom IP

- Now, select Edit IP and click Finish.
- A new project will open.



Creating a custom IP

- Now, go to Add sources -> add or create design sources.
  - Add test_logic.v
- Check test_logic.v appeared in the Sources tab.
- Then, double click on test_IP_v1_0_S00_AXI_inst
- We will edit this file



Creating a custom IP

- Get to the bottom of the file, and edit this part as follows:
- This will add our module test_logic in the new IP.

Project Summary | Package IP - test_IP | test_IP_v1_0_S00_AXI.v

c:/workspace/ip_repo/test_IP_1.0/hdl/test_IP_v1_0_S00_AXI.v

```
429         begin
430           axi_rdata  <= 0;
431         end
432       else
433         begin
434           // When there is a valid read address (S_AXI_ARVALID) with
435           // acceptance of read address by the slave (axi_arready),
436           // output the read dada
437           if (slv_reg_rden)
438             begin
439               axi_rdata <= reg_data_out;     // register read data
440             end
441         end
442     end
443
444   // Add user logic here
445
446   // User logic ends
447
448 endmodule
449
```

```
443
444 // Add user logic here
445
446 wire [31:0] sum, difference, bitwiseAnd, bitwiseOr, bitwiseXor, bitwiseXNor;
447
448 test_logic u_test_logic(
449     .a(slv_reg0),
450     .b(slv_reg1),
451     .sum(sum),
452     .difference(difference),
453     .bitwiseAnd(bitwiseAnd),
454     .bitwiseOr(bitwiseOr),
455     .bitwiseXor(bitwiseXor),
456     .bitwiseXNor(bitwiseXNor)
457 );
458 // User logic ends
459
460 endmodule
461
```

Creating a custom IP

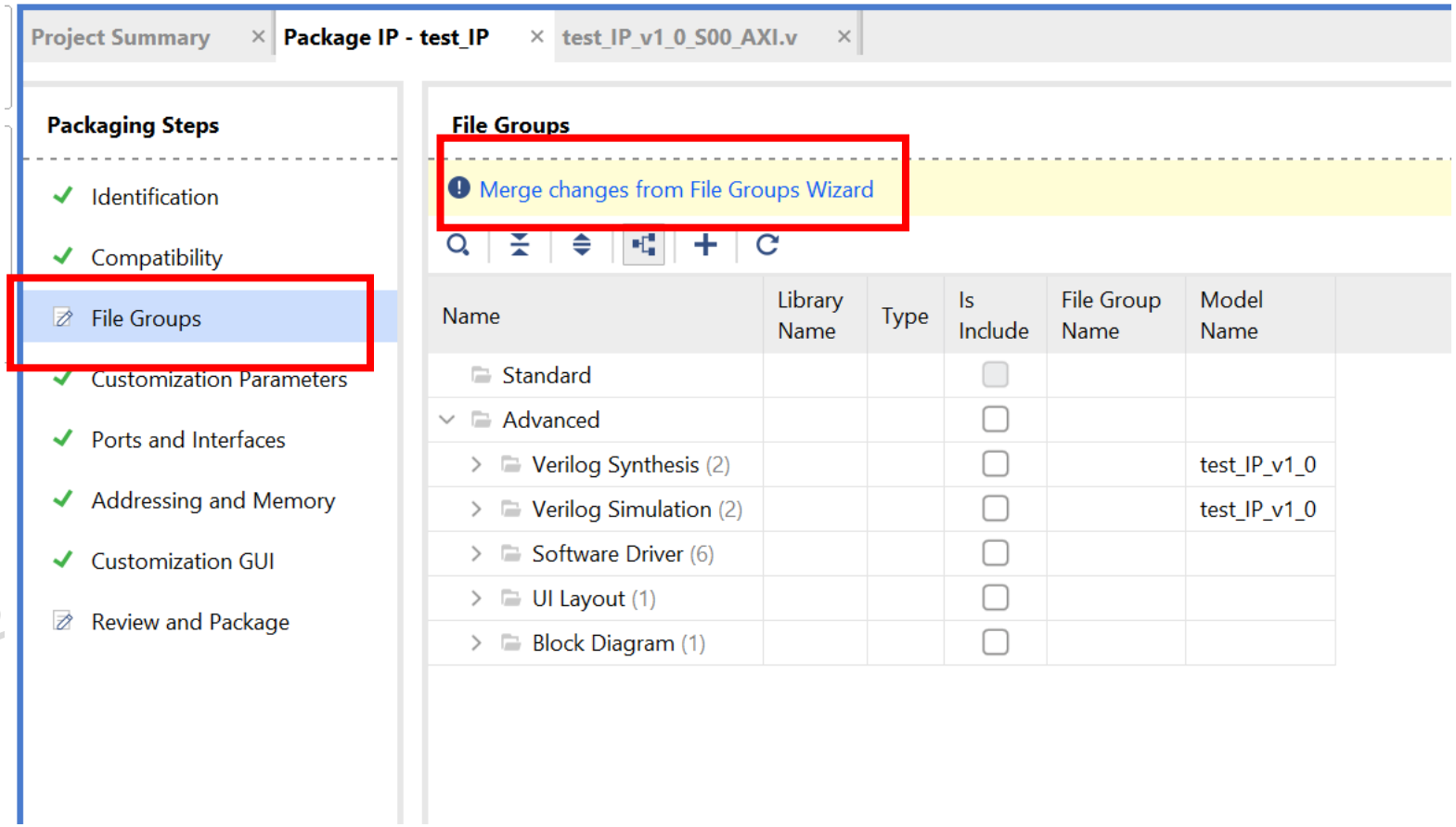
- Then, find the part where "if (slv_reg_wren)" is. Here, modify the code as below.

```
223     always @( posedge S_AXI_ACLK )
224     begin
225       if ( S_AXI_ARESETN == 1'b0 )
226         begin
227           slv_reg0 <= 0;
228           slv_reg1 <= 0;
229           slv_reg2 <= 0;
230           slv_reg3 <= 0;
231           slv_reg4 <= 0;
232           slv_reg5 <= 0;
233           slv_reg6 <= 0;
234           slv_reg7 <= 0;
235         end
236       else begin
237         if (slv_reg_wren)
238           begin
239             case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
240               3'h0:
241                 for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
242                   if ( S_AXI_WSTRB[byte_index] == 1 ) begin
243                     // Respective byte enables are asserted as per write strobes
```

```
224     begin
225       if ( S_AXI_ARESETN == 1'b0 )
226         begin
227           slv_reg0 <= 0;
228           slv_reg1 <= 0;
229           slv_reg2 <= 0;
230           slv_reg3 <= 0;
231           slv_reg4 <= 0;
232           slv_reg5 <= 0;
233           slv_reg6 <= 0;
234           slv_reg7 <= 0;
235         end
236       else begin
237           slv_reg2 <= sum;
238           slv_reg3 <= difference;
239           slv_reg4 <= bitwiseAnd;
240           slv_reg5 <= bitwiseOr;
241           slv_reg6 <= bitwiseXor;
242           slv_reg7 <= bitwiseXNor;
243         if (slv_reg_wren)
244           begin
```

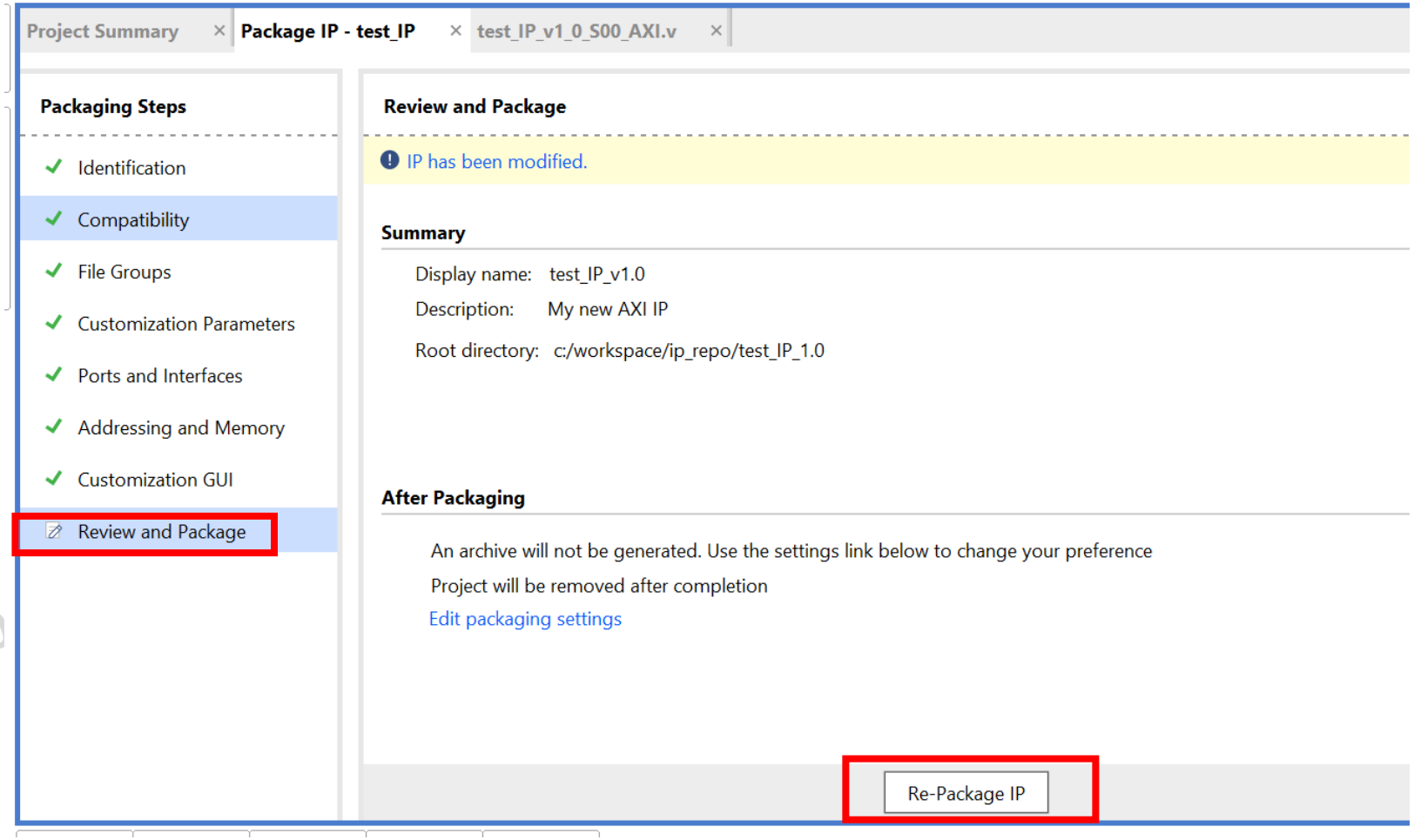
Creating a custom IP

- Now, save the file and go to Package IP – test_IP tab.
- Go to File groups tab and press Merge changes from File Groups Wizard.



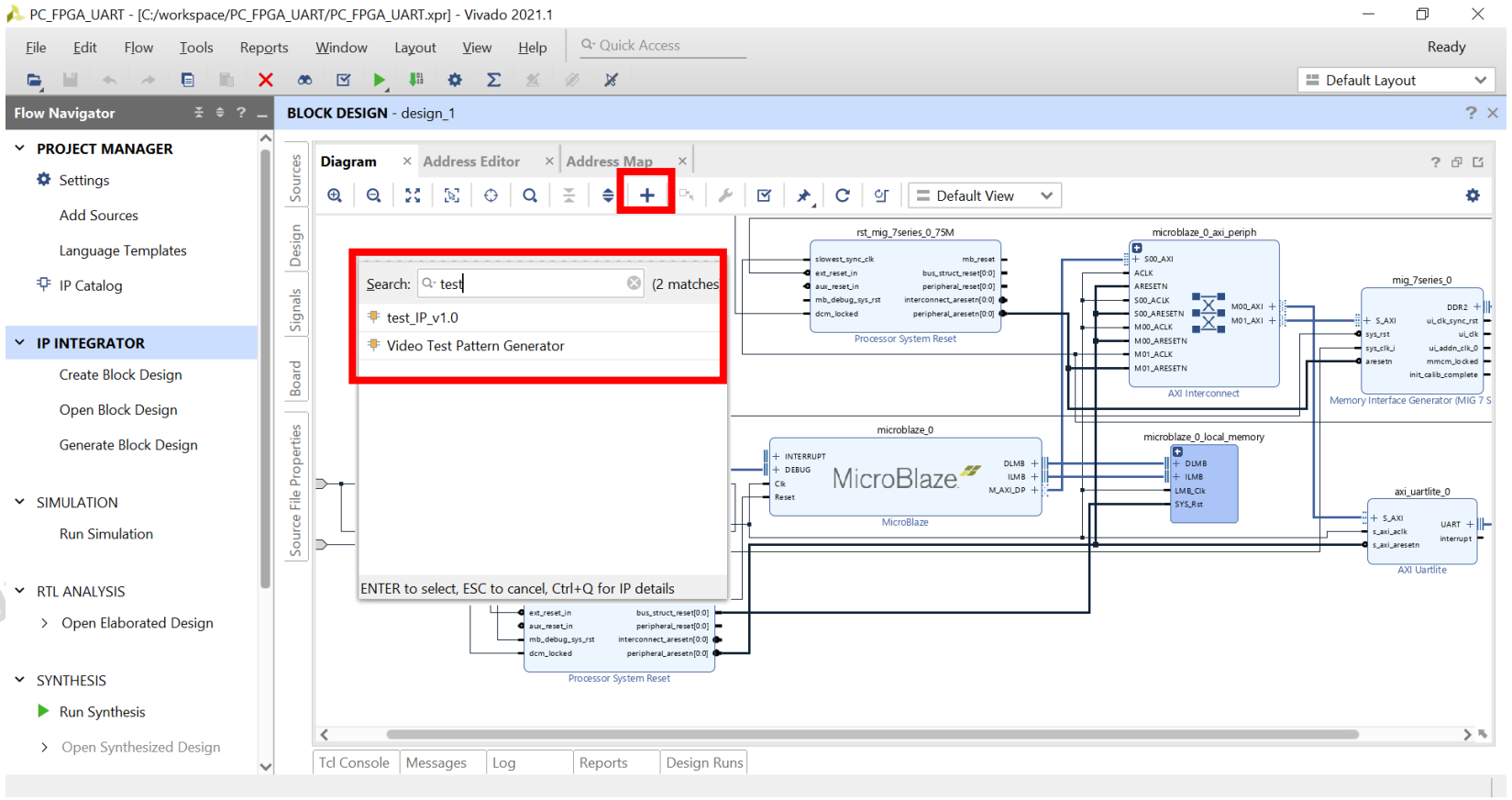
Creating a custom IP

- Then, go to Review and Package, and press Re-Package IP.



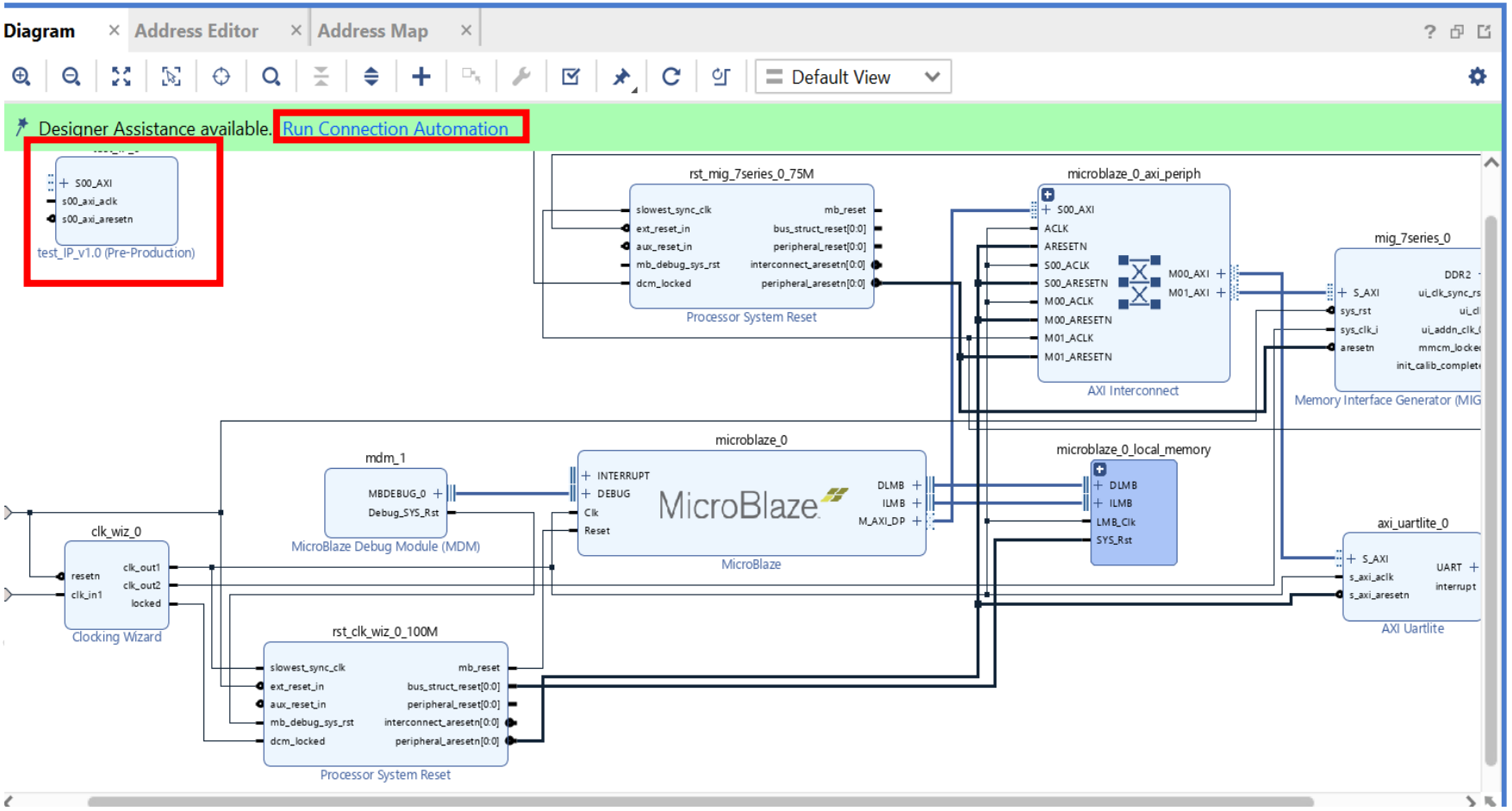
Creating a custom IP

- Get back to the PC_FPGA_UART project
- Now, we can add our IP to the design. Press the "+" button and search for test_IP



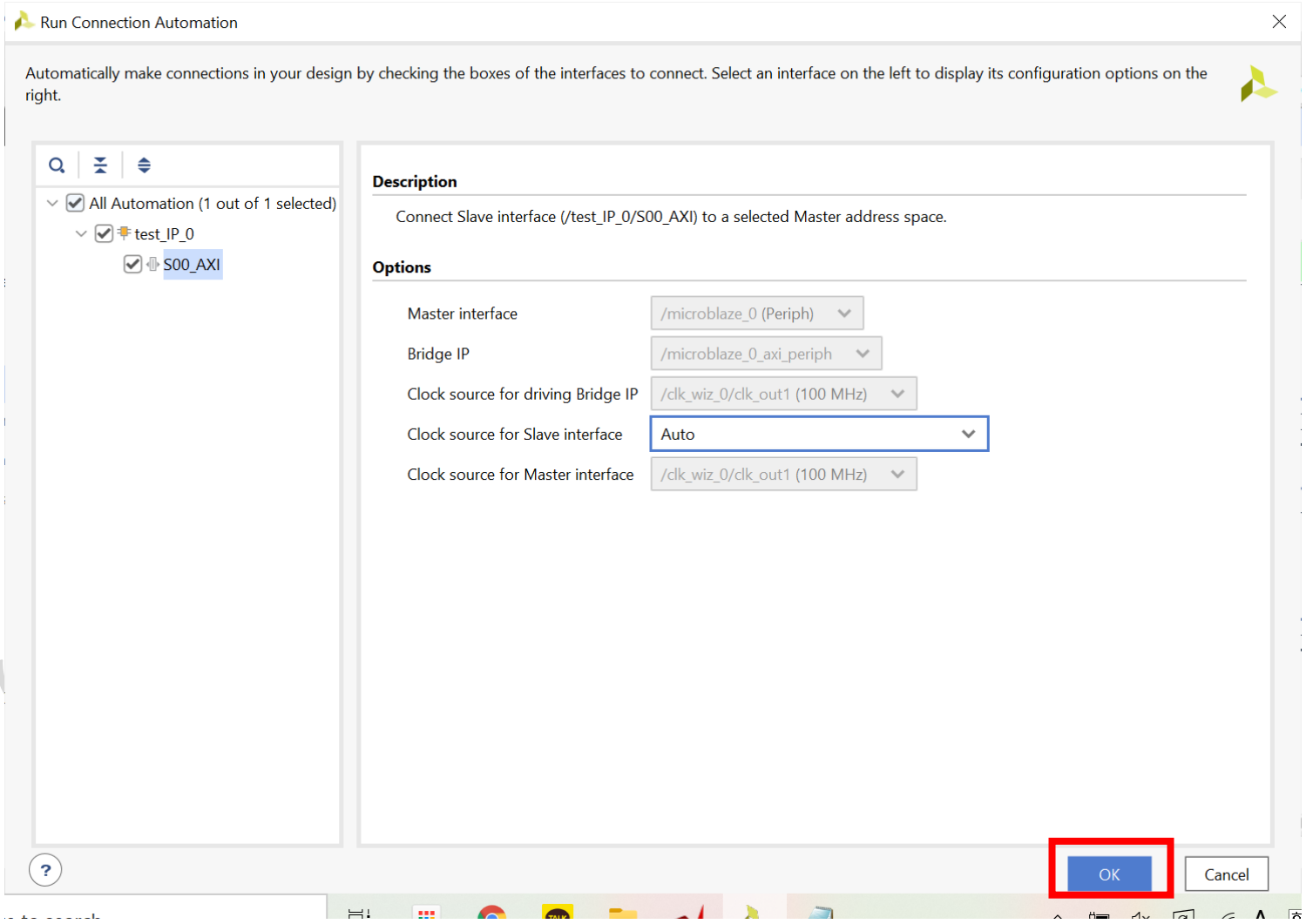
Creating a custom IP

- We can check that our IP is added in the design. Press Run Connection Automation.



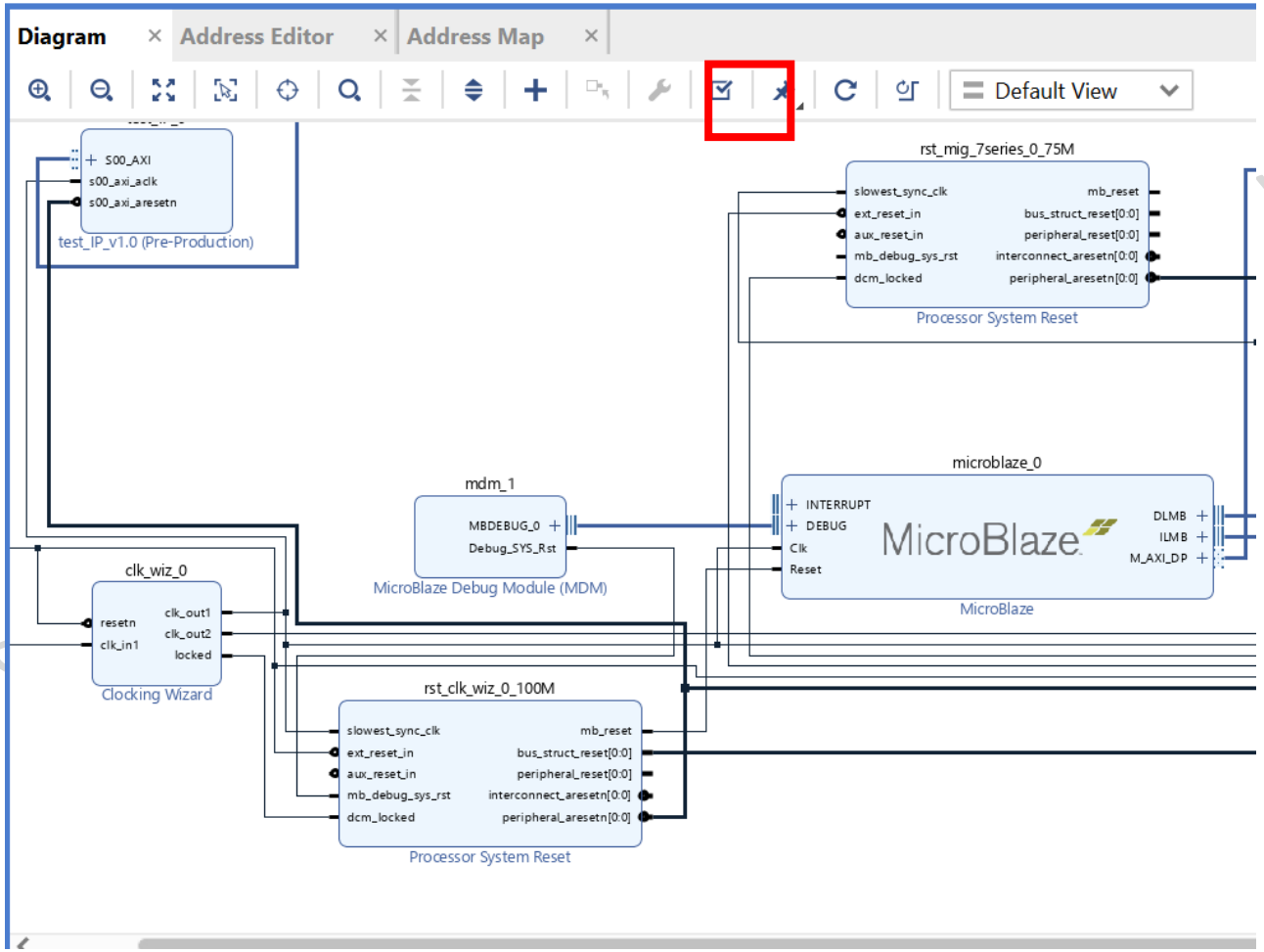
Creating a custom IP

- Here, press OK.



Creating a custom IP

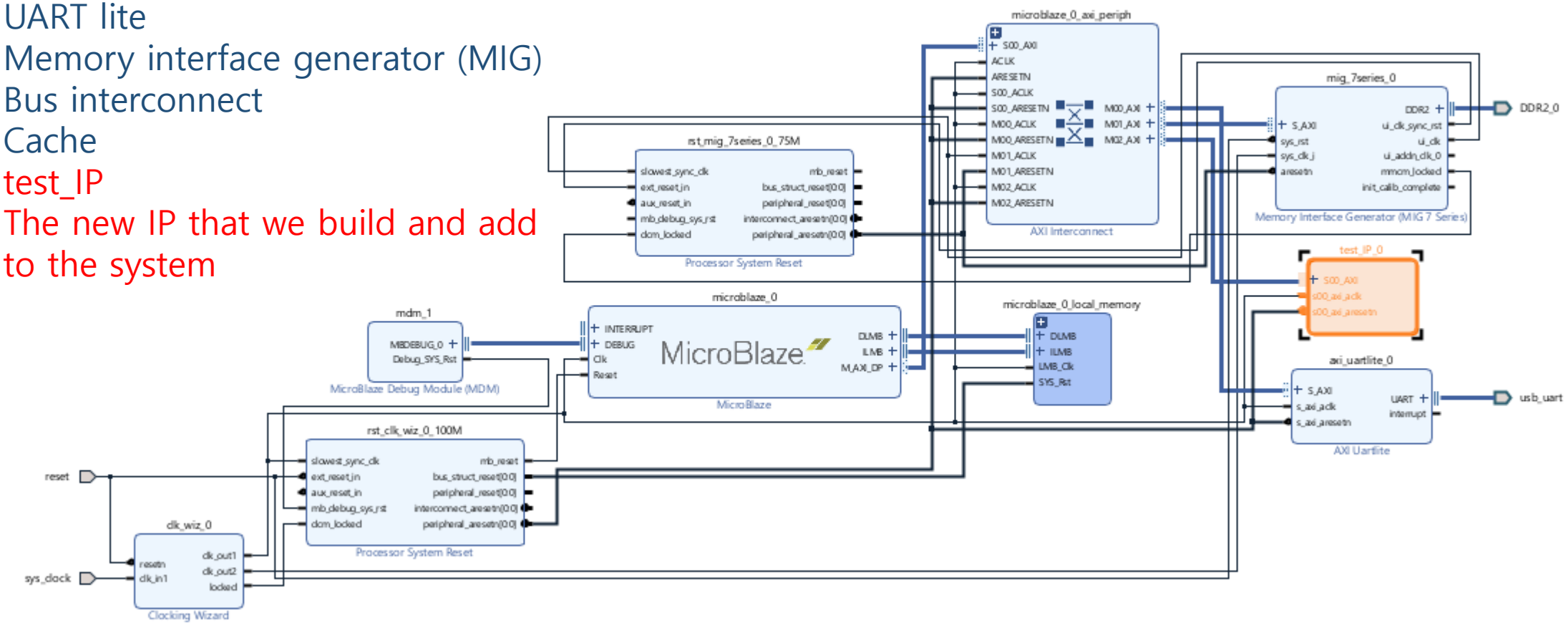
- Now, our IP is connected with other IPs. Press the validate design button.



Block diagram

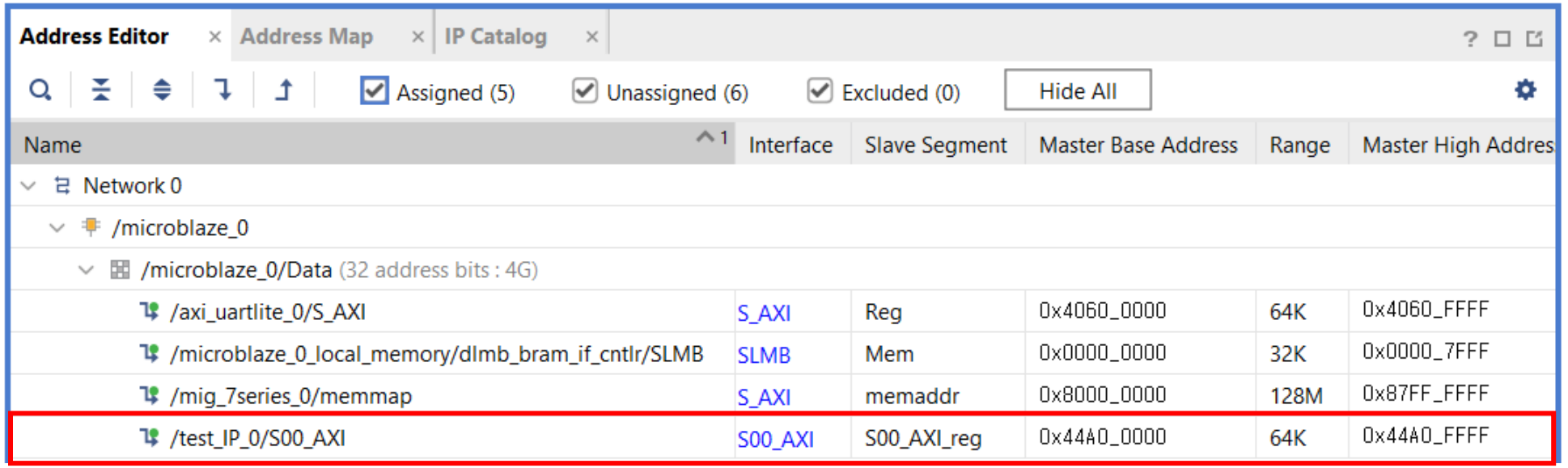
- MicroBlaze
- UART lite
- Memory interface generator (MIG)
- Bus interconnect
- Cache
- test_IP

⇒ The new IP that we build and add to the system



Address mapping

- Mapping addresses for masters and slaves
  - Base addresses and ranges
  - Example
    - UART: 0x4060_0000~0x4060_FFFF (64K)
    - MIG: 0x8000_0000~0x87FF_FFFF (128M)
    - S00_AXI: 0x44A0_0000~0x44A0_FFFF (64K)



Name	Interface	Slave Segment	Master Base Address	Range	Master High Addres
Network 0					
/microblaze_0					
/microblaze_0/Data (32 address bits : 4G)					
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
/mig_7series_0/memmap	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
/test_IP_0/S00_AXI	S00_AXI	S00_AXI_reg	0x44A0_0000	64K	0x44A0_FFFF

Firmware

```
86      else if(RecvBuffer[0] == 0x05){ //store to testip
87          //send response
88          uart_send(cWaiting, 16);
89
90          //receive offset and data
91          uart_recv(8);
92          uint32_t offset = (RecvBuffer[3] << 24) + (RecvBuffer[2] << 16) + (RecvBuffer[1] << 8) + RecvBuffer[0];
93
94          write_addr(XPAR_TEST_IP_0_S00_AXI_BASEADDR + offset, RecvBuffer + 4, 4);
95
96          //send response
97          uart_send(cComplete, 16);
98      }
99      else if(RecvBuffer[0] == 0x06){ //load from testip
100         //send response
101         uart_send(cWaiting, 16);
102
103         //receive offset
104         uart_recv(4);
105         uint32_t offset = (RecvBuffer[3] << 24) + (RecvBuffer[2] << 16) + (RecvBuffer[1] << 8) + RecvBuffer[0];
106
107         //load data from testip
108         uint8_t temp[4];
109         read_addr(XPAR_TEST_IP_0_S00_AXI_BASEADDR + offset, temp, 4);
110
111         //send via UART
112         uart_send(temp, 4);
113     }
```

Write inputs to the IP

Read outputs from the IP

To do …

- For FPGA verification, you should build the AXI IP for the CNN accelerator
- Example: sending configurations for the controller via AXI
  - q_width, q_height, q_frame_size
  - Synchronization delay
    - Frame/layer synchronization (vsync_delay)
    - Row/line synchronization (hsync_delay)

```
1   `timescale 1ns / 1ps
2
3   module cnn_ctrl(
4   clk,
5   rstn,
6   // Inputs
7   q_width,
8   q_height,
9   q_vsync_delay,
10  q_hsync_delay,
11  q_frame_size,
12  q_start,
13  //output
14  o_ctrl_vsync_run,
15  o_ctrl_vsync_cnt,
16  o_ctrl_hsync_run,
17  o_ctrl_hsync_cnt,
18  o_ctrl_data_run,
19  o_row,
20  o_col,
21  o_data_count,
22  o_end_frame
23  );
```