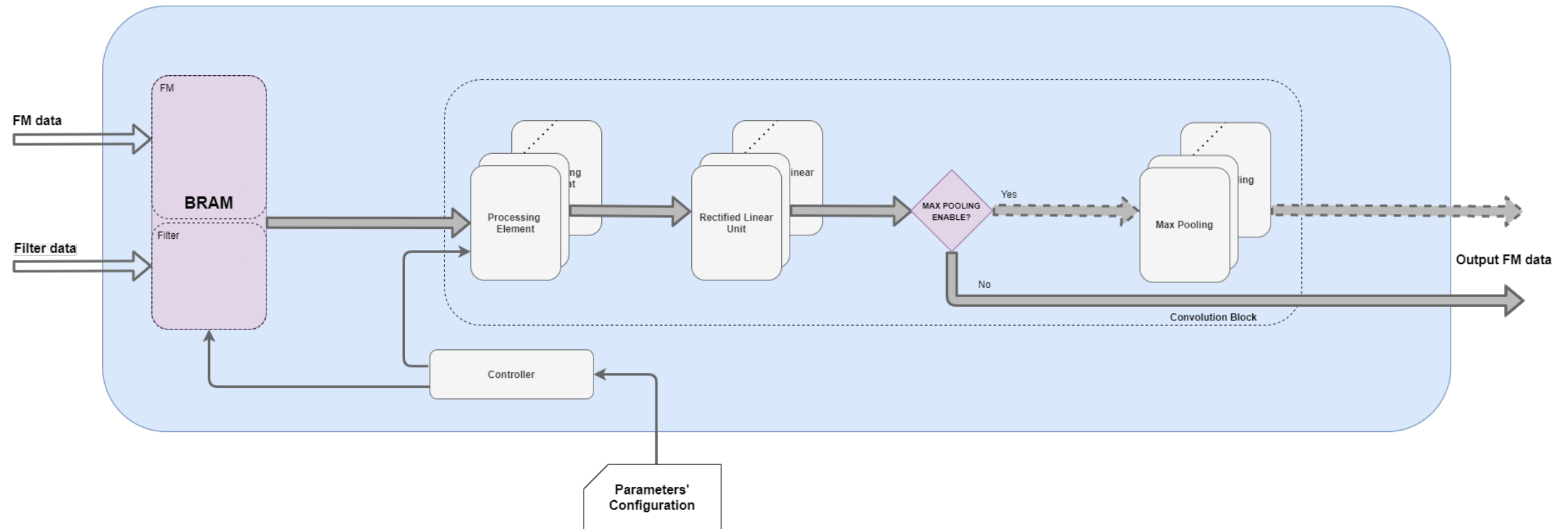


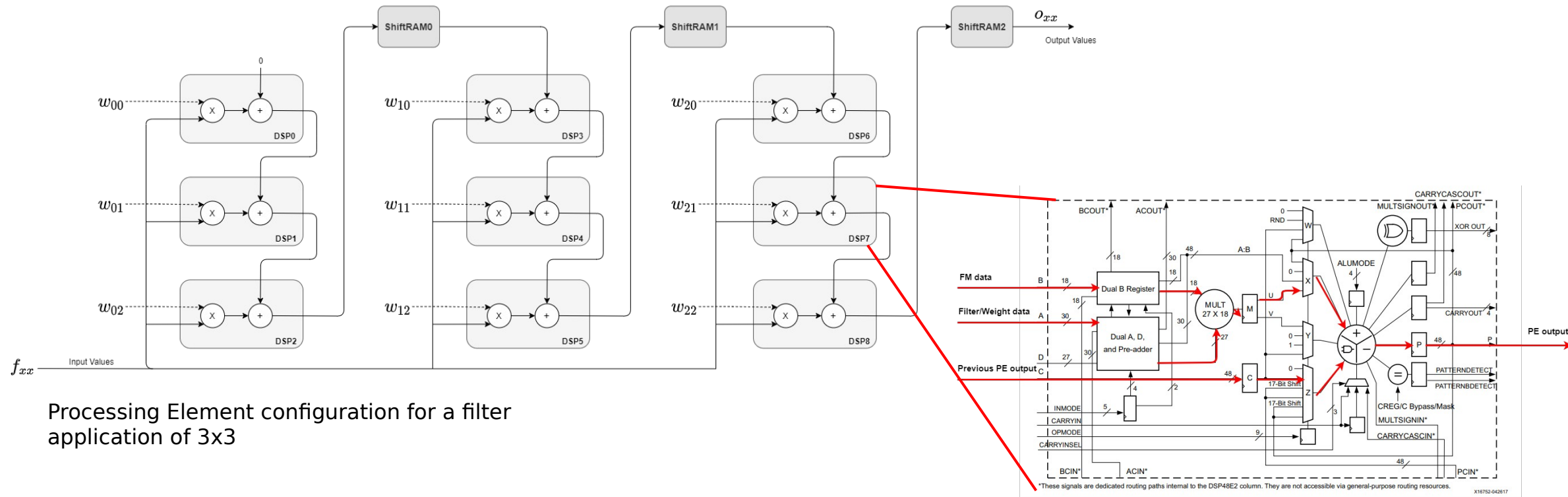
Convolution Module

Module architecture



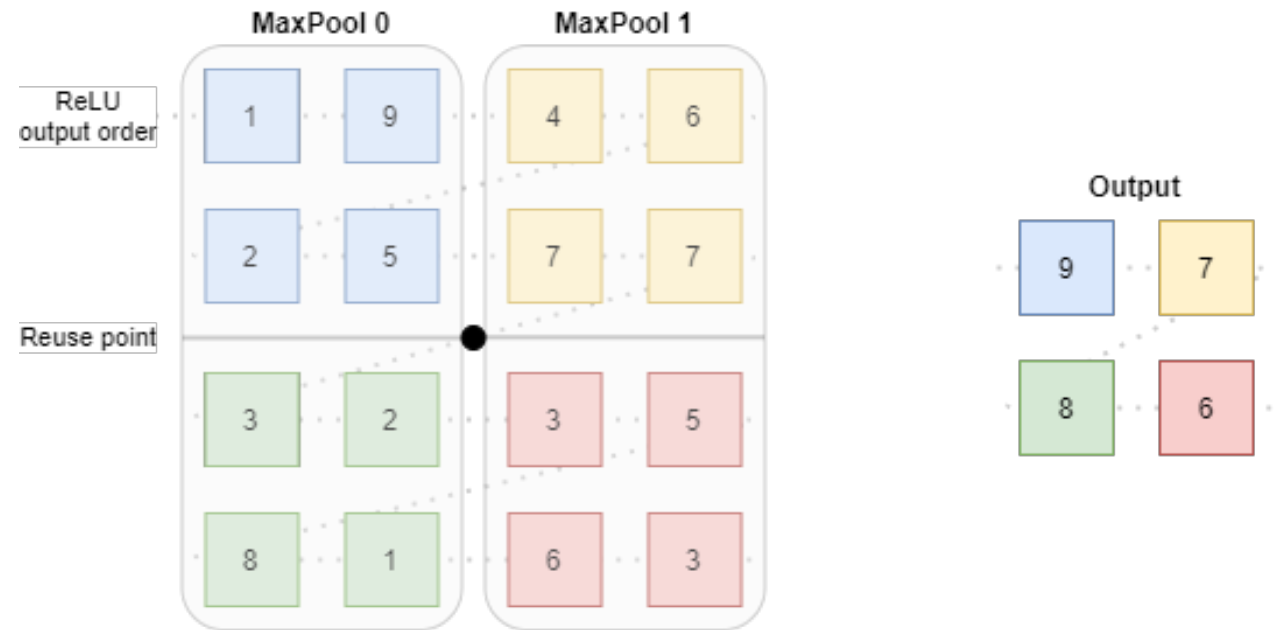
Processing Element

- 3x3 filter



Max Pooling

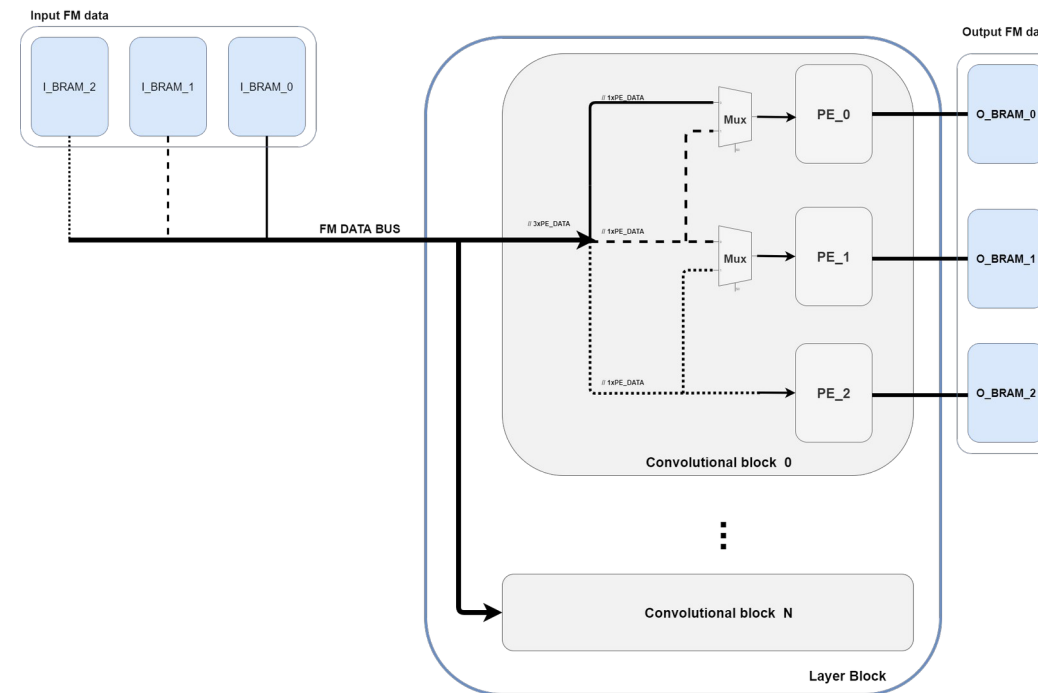
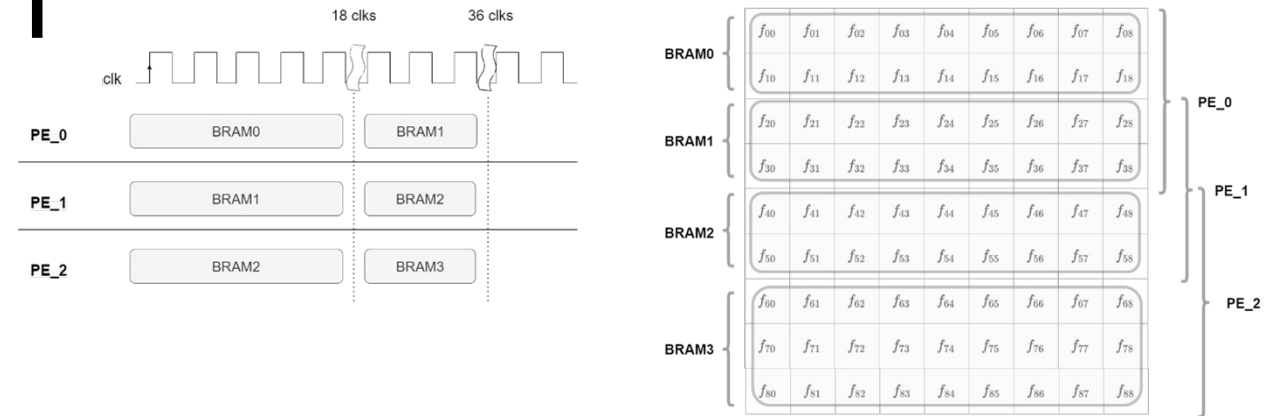
- Number of MaxPooling blocks depend on the IFM size
- In this example, the output of the ReLU is a 4x4 FM, so 2 MaxP blocks are instantiated
- Both blocks are reused after the first output line have been processed (values 9 and 7)



BRAM distribution

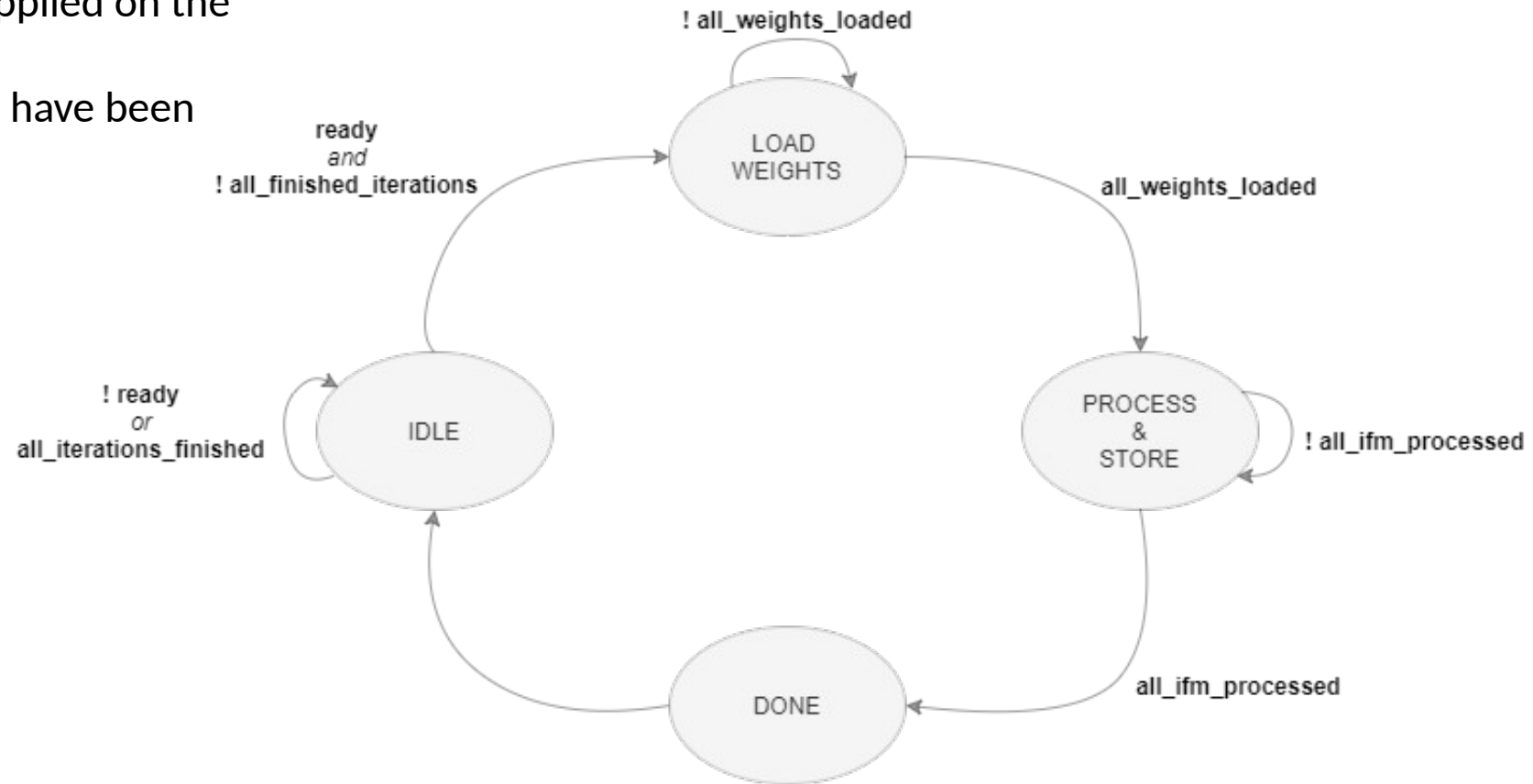
A FM of 9x9 with a filter application of 3x3, stride of 1.

- Figure shows maximum parallelism application
- Three PE blocks in parallel
- After 18 clks, each PE reads BRAM values from the following BRAM block
- Requires a MUX to control BRAM data flow



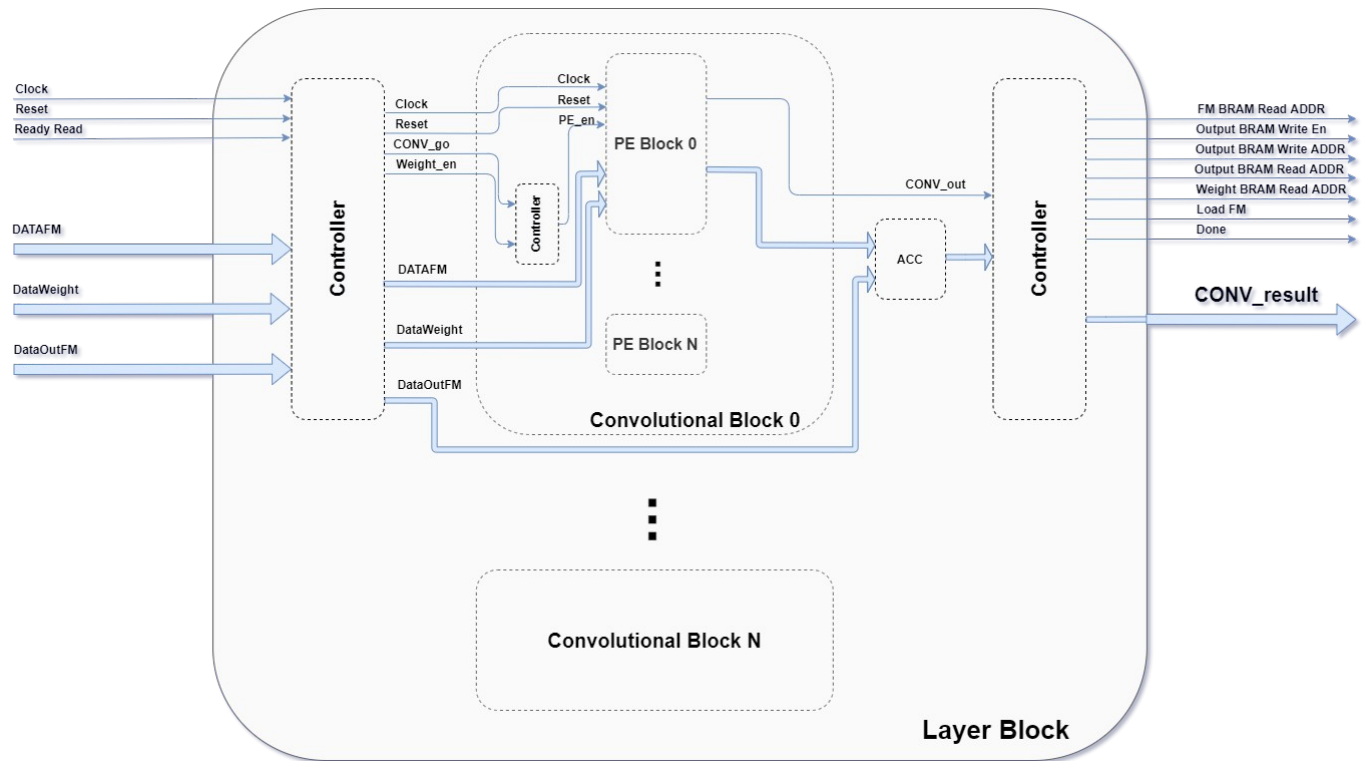
FSM

- According to the available resources, the processing required is divided into several iterations.
- Each iteration a new set of filters are applied on the IFM.
- The processing finished when all filters have been applied to all IFM



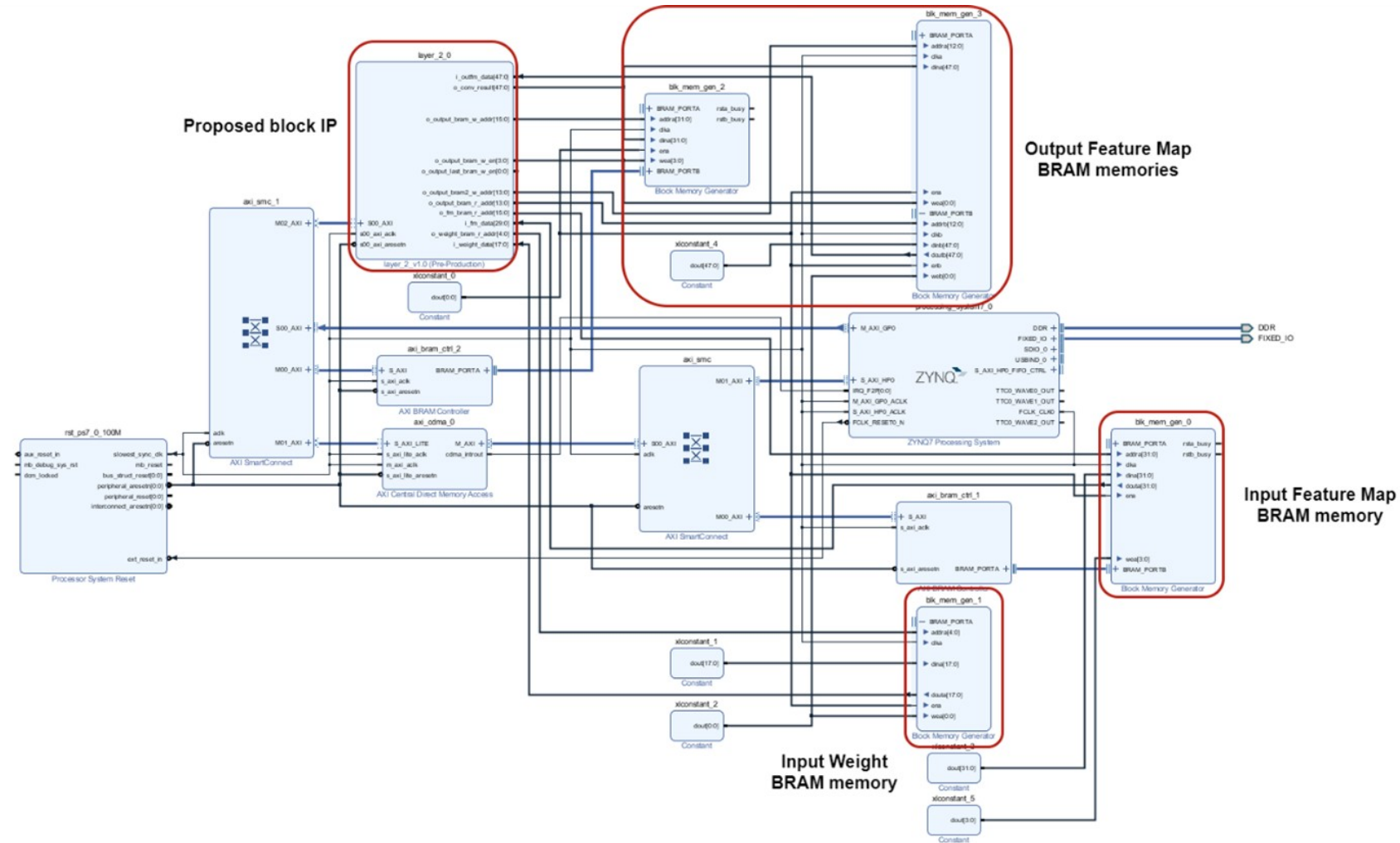
Layer block

- Inside the Layer block are possible to instantiate more than one Conv modules.
- Each Conv module is associated with a different filter.
- All inputs and outputs must be correctly assigned to BRAM blocks, otherwise data collision may occur.



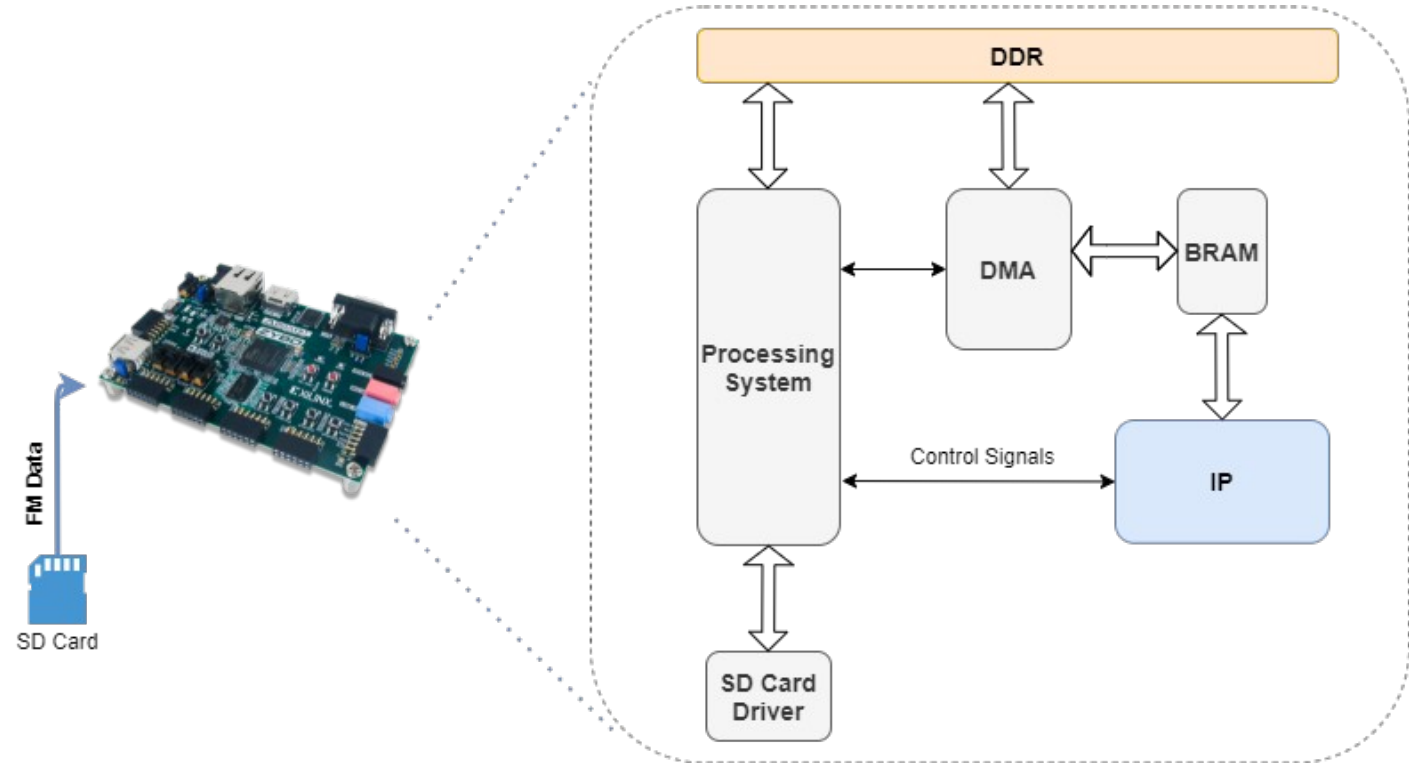
How to use - Block Design for validation

- 3 Block Rams (Filter, IFM, and OFM)
- Developed IP connected to all BRAMS
- OFM BRAMS with two ports
(convolution with each IFM channel is added)



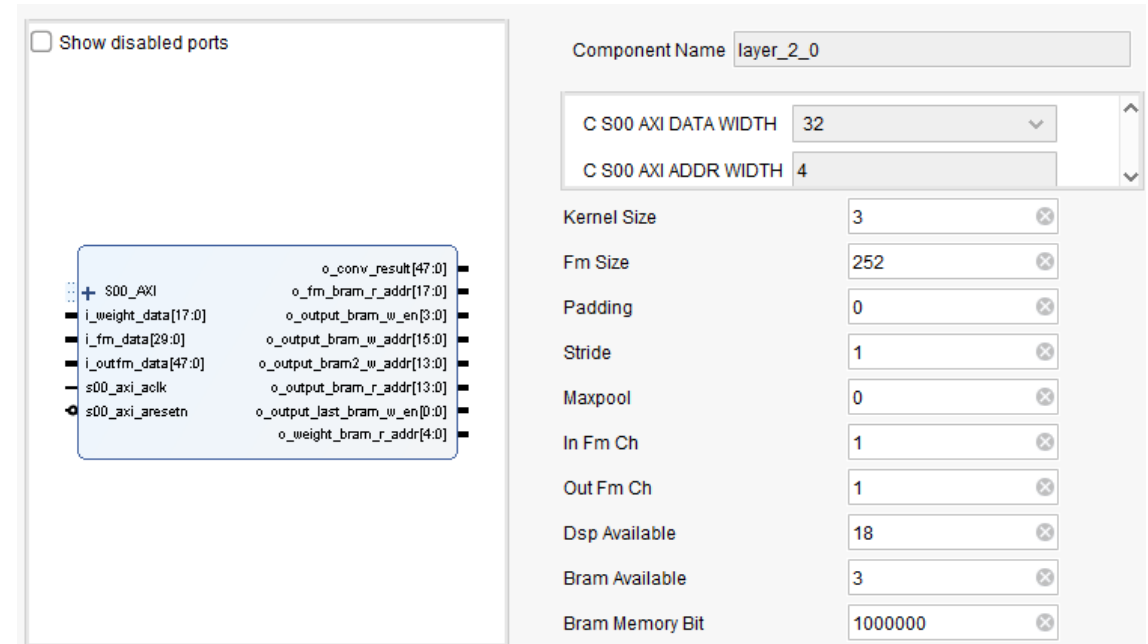
Test Setup

- The input image (FM) is stored in an SD card.
- The image is sent to DDR through a DMA module.
- Then when a signal is sent from the PS to our IP, using an AXI interconnect, the state machine collects data from BRAM block which was previously stored using the DMA module.
- After finishing the convolution process, the output FM is sent to SD card.



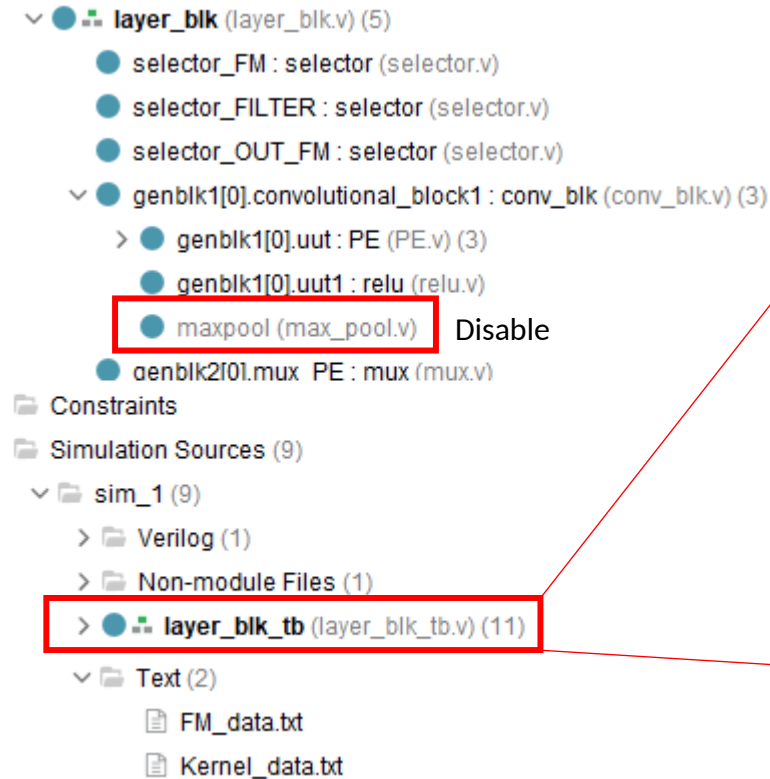
Layer block – IP interface

- [illegible]



Vivado Simulation

In test bench file, change parameters values



```
module layer_blk_tb #(
    parameter KERNEL_SIZE    = 3,
    parameter FM_SIZE        = 252,
    parameter PADDING        = 0,
    parameter STRIDE         = 1,
    parameter MAXPOOL        = 0,
    parameter DSP_AVAILABLE  = 36,
    parameter IN_FM_CH       = 1,
    parameter OUT_FM_CH      = 1,
    parameter BRAM_AVAILABLE = 12,
    parameter BRAM_MEMORY_BIT = 1000000,

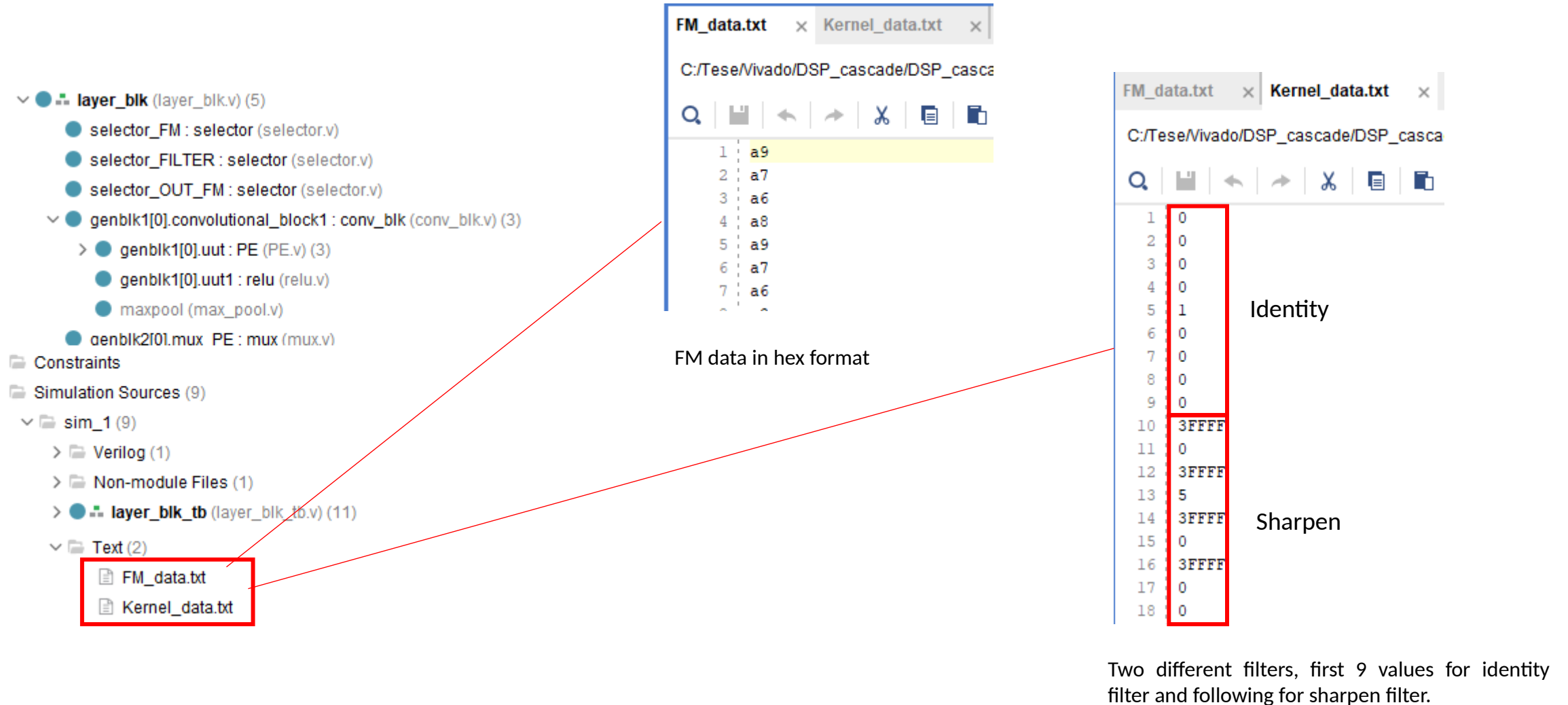
```

```
if(((output_bram_r_addr < (BRAM_SIZE*OUT_SIZE)+1) && out_file == 0) || ((output_bram_r_addr
if(output_bram_r_addr > 0) begin
    $fwrite(out_data,"%0d\n", output_bram_o_data[out_file*48 + 48*PE_TO_USE*0 +: 48]);

```

- Each \$fwrite saves the convolution output from one filter application.
- The output file is located on .../.sim/sim_x/.. dir from vivado project.

Vivado Simulation



The screenshot displays the Vivado IDE interface during a simulation. On the left, the Project Explorer shows a hierarchy of components: **layer_blk** (layer_blk.v) (5), which includes **selector_FM**, **selector_FILTER**, **selector_OUT_FM**, **genblk1[0].convolutional_block1** (conv_blk.v) (3), and **genblk2[0].mux** (mux.v). Below this, the **Simulation Sources** section lists **sim_1** (9), which contains **Verilog** (1), **Non-module Files** (1), **layer_blk_tb** (layer_blk_tb.v) (11), and **Text** (2). The **Text** folder contains **FM_data.txt** and **Kernel_data.txt**, which are highlighted with a red box. Two red lines originate from this box: one points to the **FM_data.txt** editor window, and the other points to the **Kernel_data.txt** editor window.

The **FM_data.txt** editor window shows the file path **C:/Tese/Vivado/DSP_cascade/DSP_casca** and contains the following hex data:

Line	Hex Value
1	a9
2	a7
3	a6
4	a8
5	a9
6	a7
7	a6

Below the table, the text **FM data in hex format** is displayed.

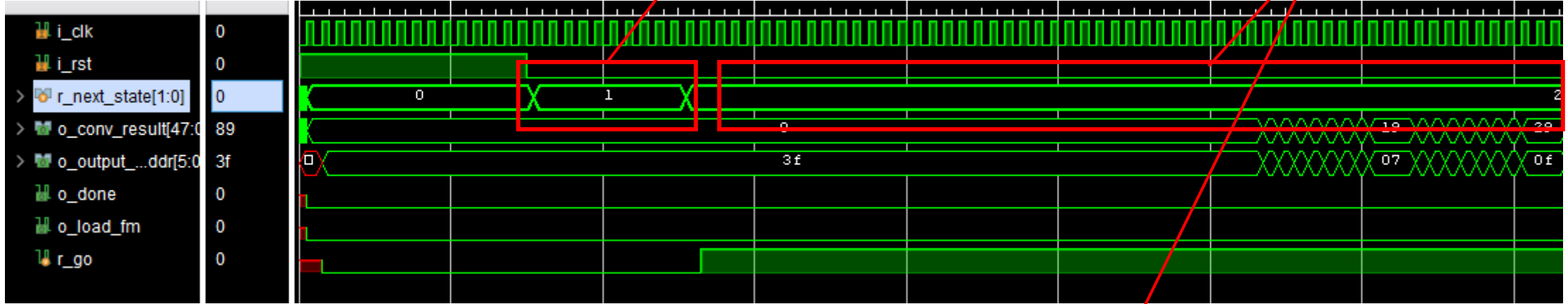
The **Kernel_data.txt** editor window shows the same file path and contains the following data:

Line	Value
1	0
2	0
3	0
4	0
5	1
6	0
7	0
8	0
9	0
10	3FFFF
11	0
12	3FFFF
13	5
14	3FFFF
15	0
16	3FFFF
17	0
18	0

To the right of the table, the text **Identity** is aligned with lines 1-9, and **Sharpen** is aligned with lines 10-18.

At the bottom right, a caption reads: **Two different filters, first 9 values for identity filter and following for sharpen filter.**

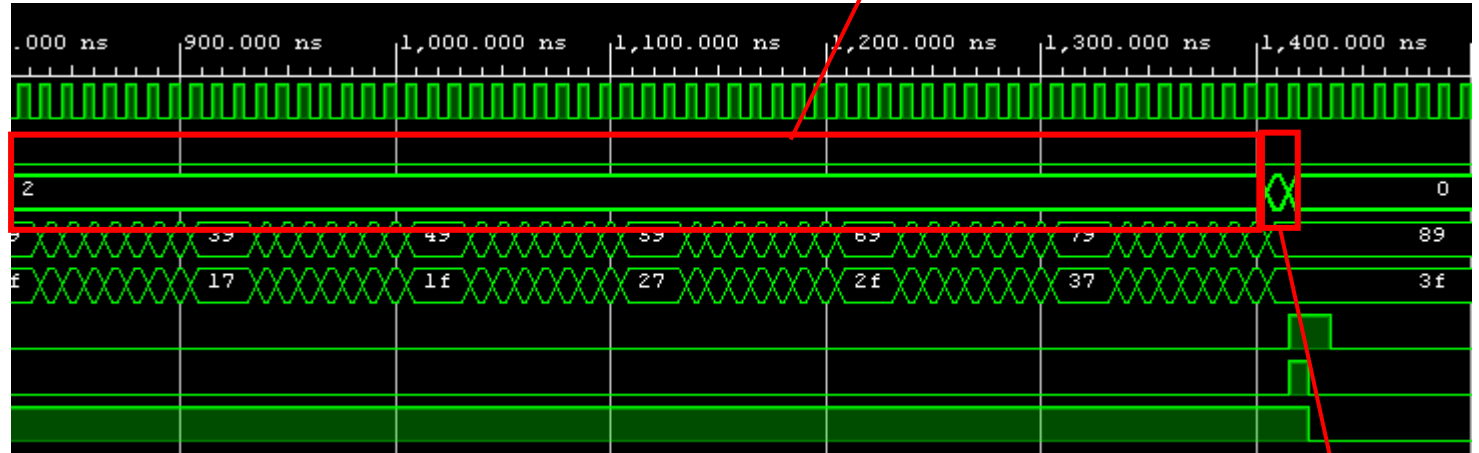
Vivado Simulation



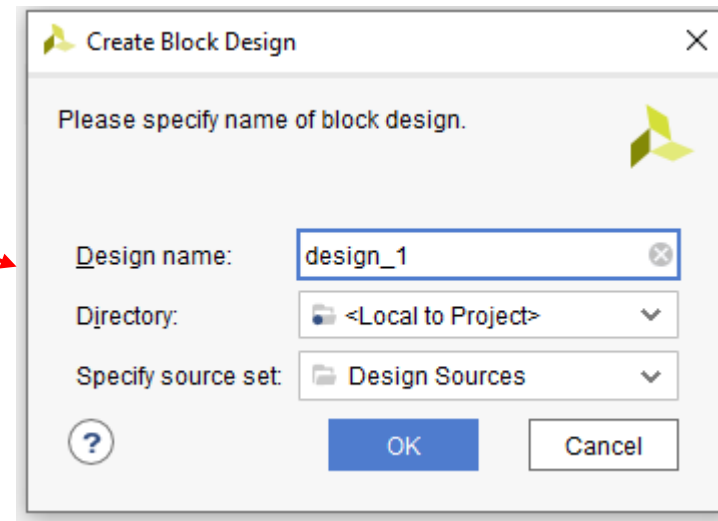
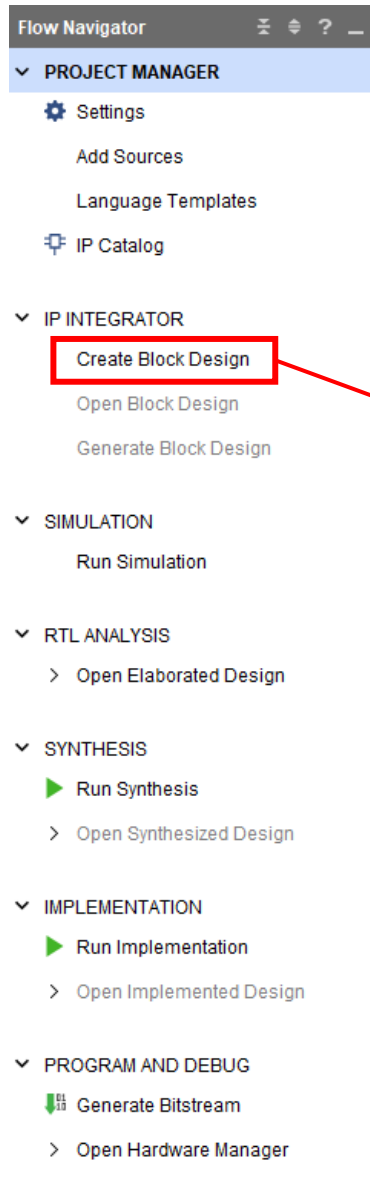
FM of 10x10, filter 3x3 with stride 1

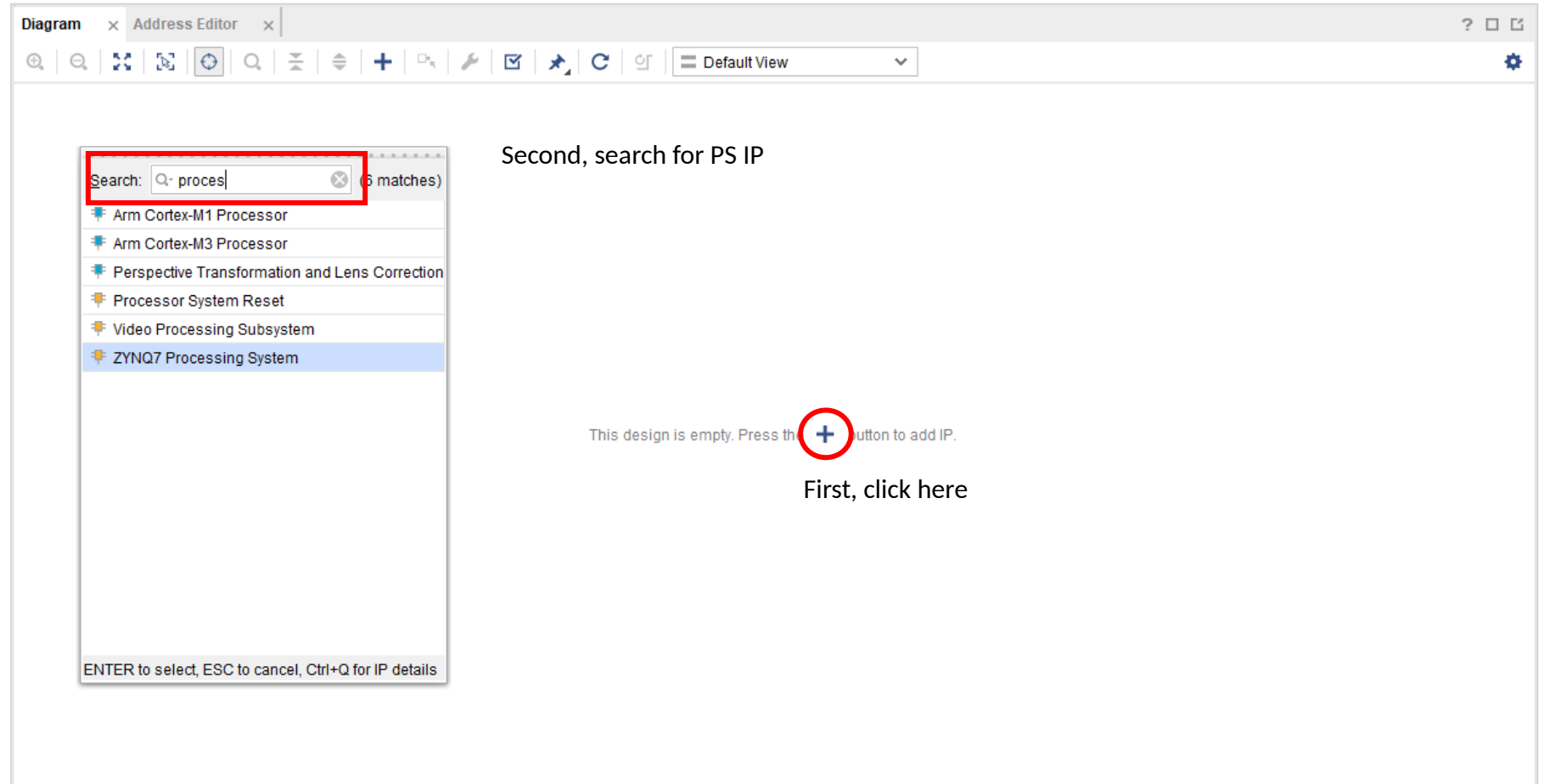
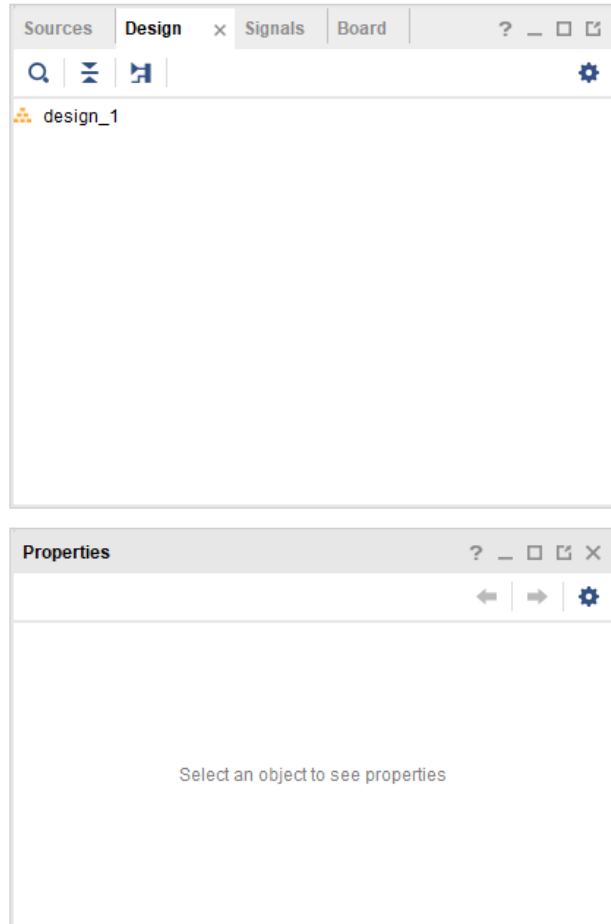
R_go enables convolution process

O_done signals the end of processing



Done





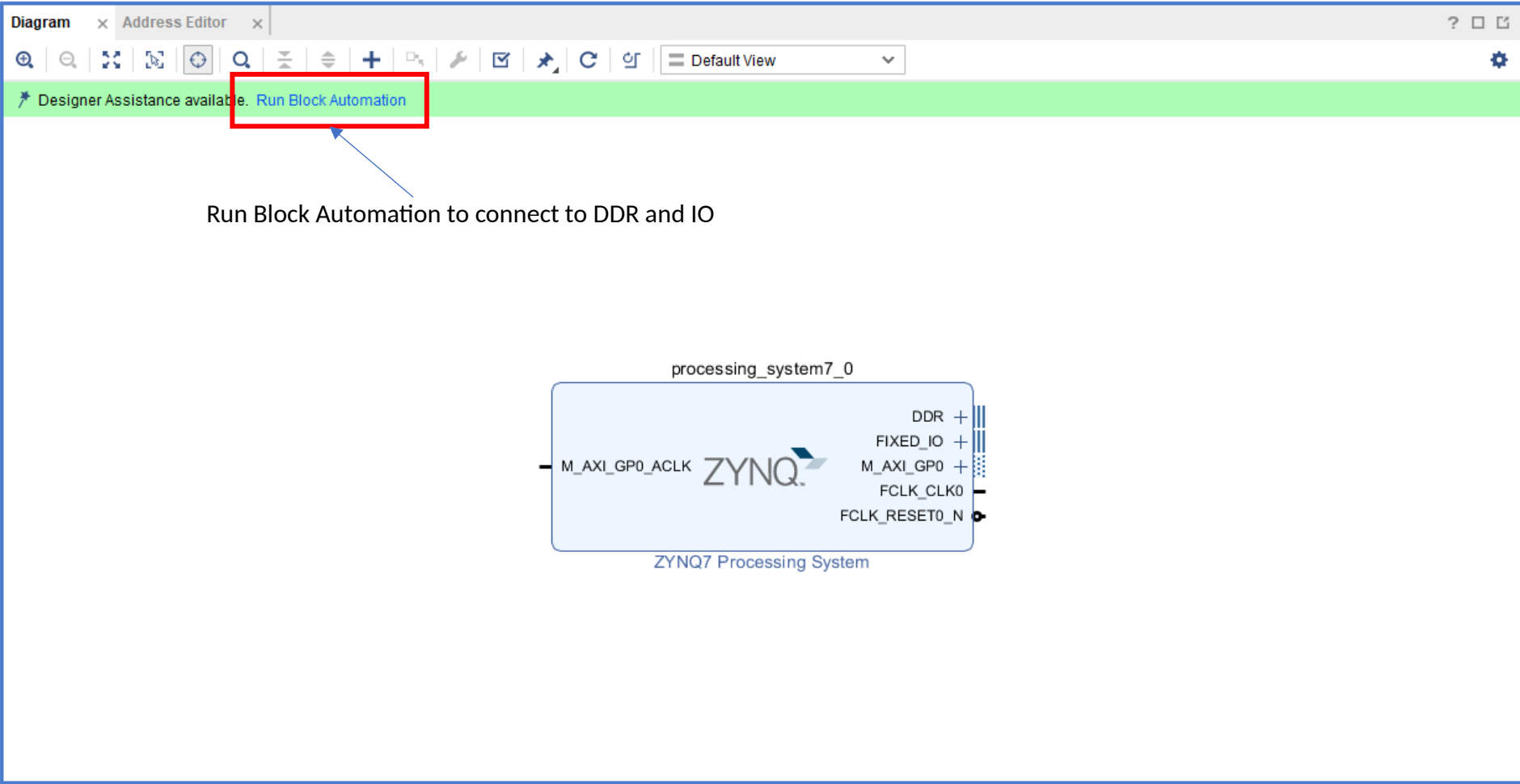
Sources Design x Signals Board ? _ □ □

design_1


> processing_system7_0 (ZYNQ7 Processing System:5.5)

Properties ? _ □ □ x

Select an object to see properties



Sources Design x Signals Board ? _ □ ▢



design_1

- > External Interfaces
- > Interface Connections
- > processing_system7_0 (ZYNQ7 Processing System:5.5)

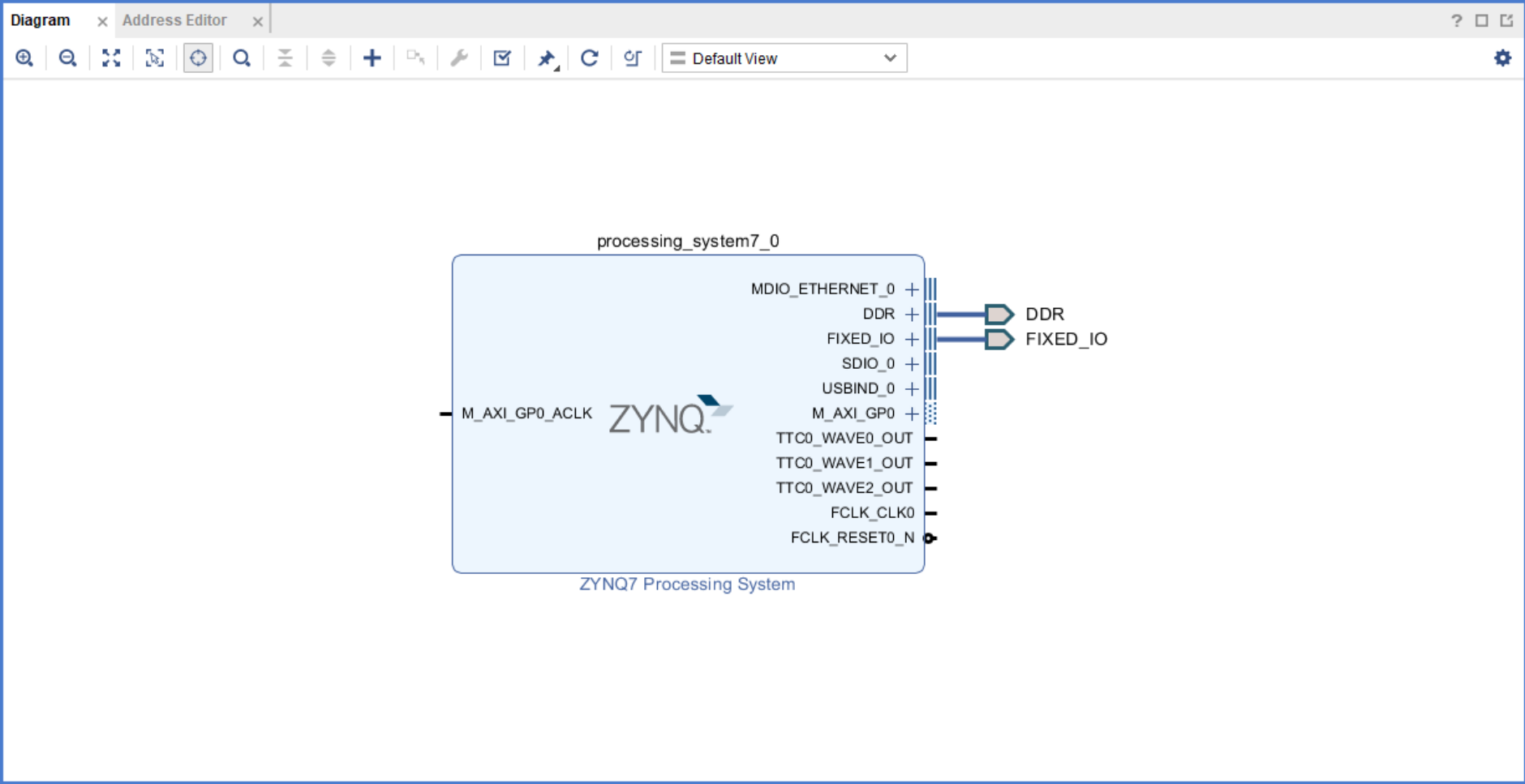
Block Properties ? _ □ ▢ x

processing_system7_0 ← → ⚙

Name: processing_system7_0

Parent name: design_1

General Properties IP



Add CDMA Module to read from input BRAM

The screenshot displays the Xilinx Block Design tool interface for a design named "design_1". The left sidebar shows the project hierarchy with "processing_system7_0 (ZYNQ7 Processing System:5.5)" selected. The bottom-left pane shows the "Block Properties" for "processing_system7_0", with fields for "Name" and "Parent name".

The main workspace is in "Diagram" view, showing a block representing the "ZYNQ7 Processing System" (processing_system7_0). The block has various input and output ports, including MDIO_ETHERNET_0, DDR, FIXED_IO, SDIO_0, USBIND_0, M_AXI_GP0, TTC0_WAVE0_OUT, TTC0_WAVE1_OUT, TTC0_WAVE2_OUT, FCLK_CLK0, and FCLK_RESET0_N. The "M_AXI_GP0_ACLK" port is highlighted with a red line.

A search window is open in the center, showing results for the search term "cdma". The search results list "AXI Central Direct Memory Access" and "AXI Multi Channel Direct Memory Access". A red box highlights the "AXI Central Direct Memory Access" entry, and a red arrow points from the "+" button in the top toolbar to this entry.

The bottom of the search window displays the instruction: "ENTER to select, ESC to cancel, Ctrl+Q for IP details".

Sources Design x Signals Board ? _ □ □

design_1

- > External Interfaces
- > Interface Connections
- > axi_cdma_0 (AXI Central Direct Memory Access:4.1)
- > processing_system7_0 (ZYNQ7 Processing System:5.5)

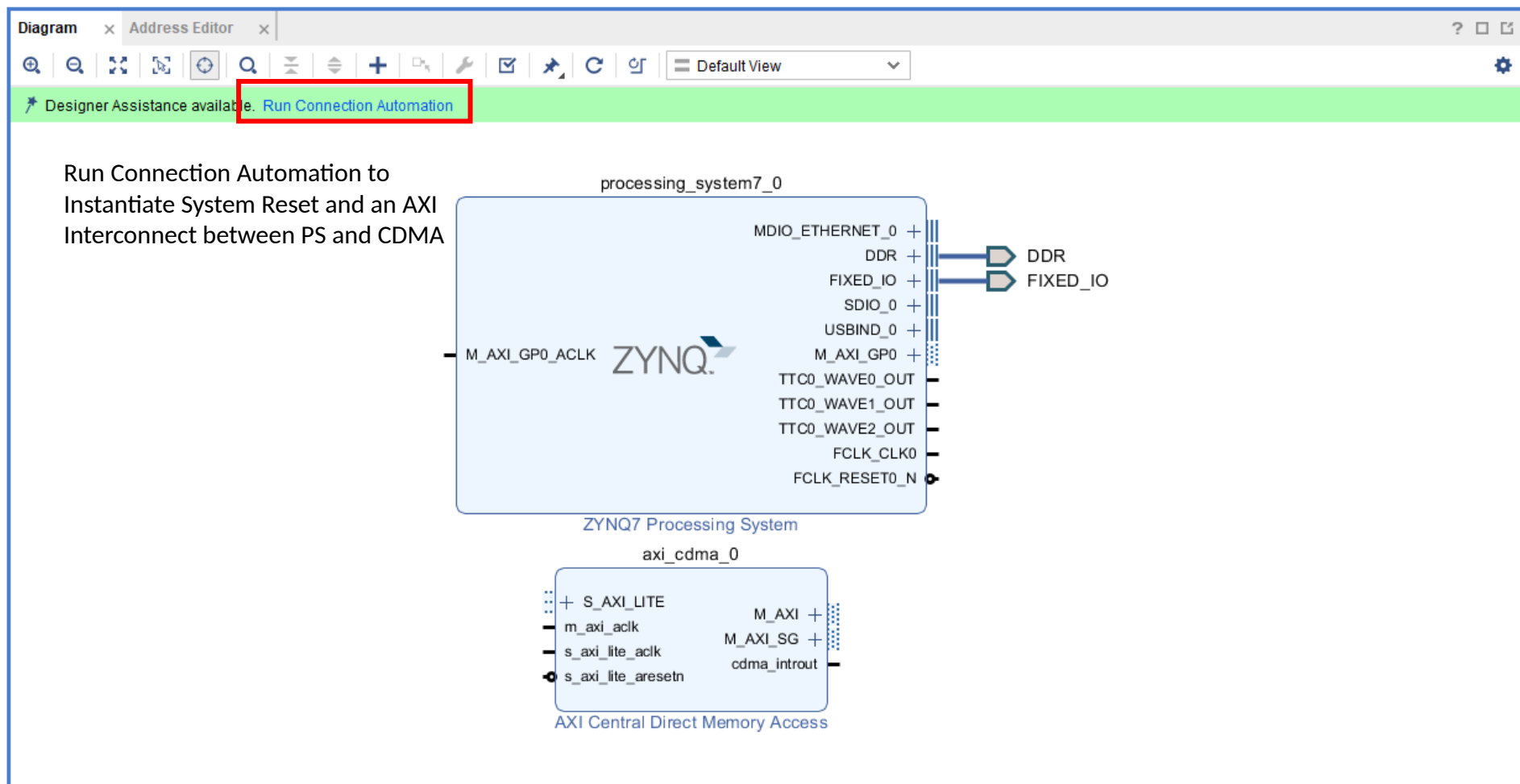
Block Properties ? _ □ □ x

processing_system7_0 ← → ⚙

Name: processing_system7_0

Parent name: design_1

General Properties IP



Sources Design x Signals Board ? _ □ ▢

design_1

- External Interfaces
- Interface Connections
 - processing_system7_0_DDR
 - processing_system7_0_FIXED_IO
 - processing_system7_0_M_AXI_GP0
 - ps7_0_axi_periph_M00_AXI
- Nets
- axi_cdma_0 (AXI Central Direct Memory Access:4.1)
 - M_AXI
 - M_AXI_SG
 - S_AXI_LITE
 - cdma_introut
 - m_axi_aclk
 - s_axi_lite_aclk

Block Interface Properties ? _ □ ▢ x

S_AXI_LITE ← → ⚙

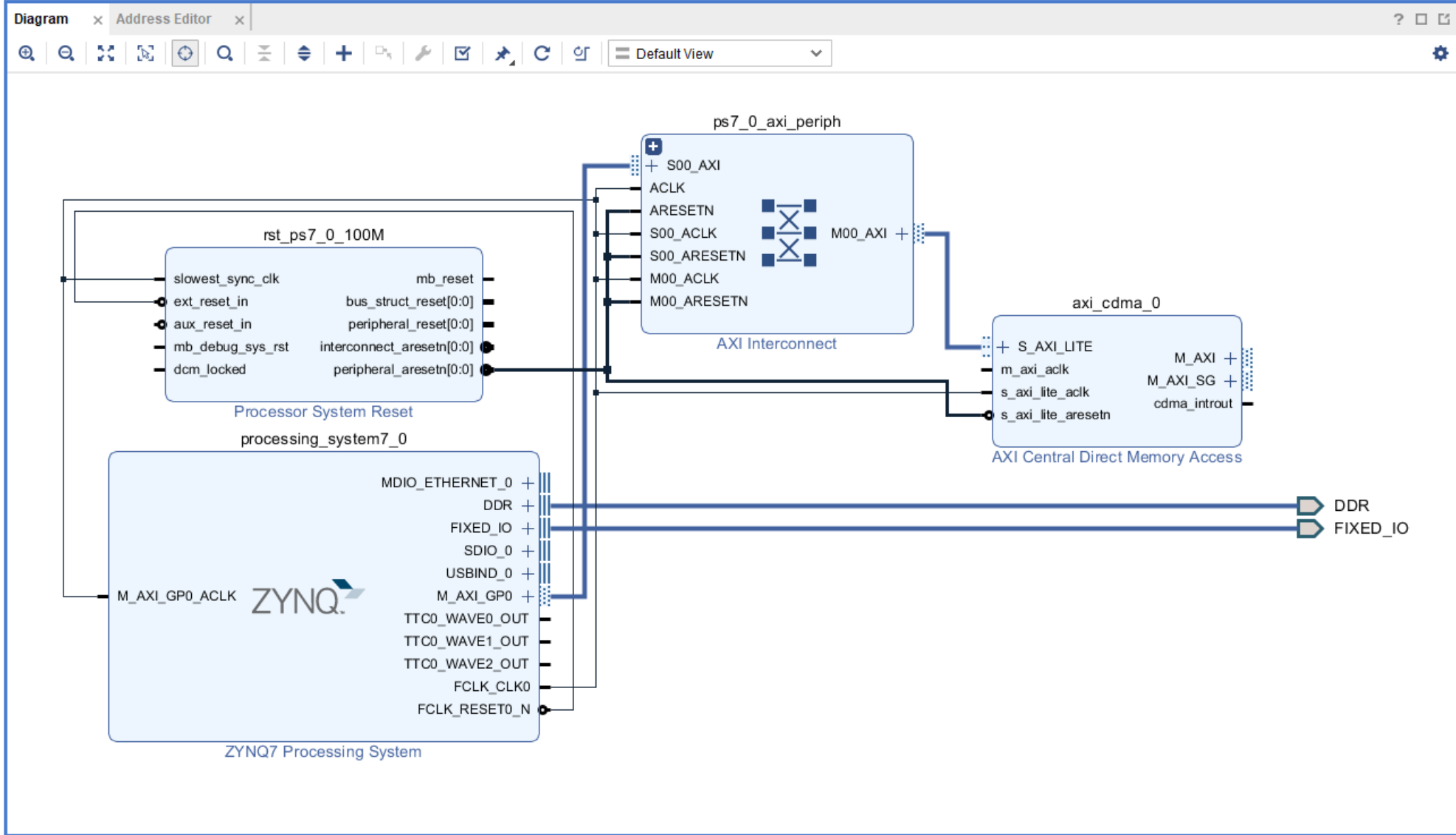
Name: S_AXI_LITE

Mode: SLAVE

Connection: ps7_0_axi_periph_M00_AXI

Associated clock: s_axi_lite_aclk

General Properties



Sources Design x Signals Board ? _ □ ▢

design_1

- External Interfaces
- Interface Connections
 - processing_system7_0_DDR
 - processing_system7_0_FIXED_IO
- Nets
- axi_cdma_0 (AXI Central Direct Memory Access:4.1)
 - M_AXI
 - M_AXI_SG
 - S_AXI_LITE
 - cdma_introut
 - m_axi_aclk
 - s_axi_lite_aclk
 - s_axi_lite_aresetn
- processing_system7_0 (ZYNQ7 Processing System:5)

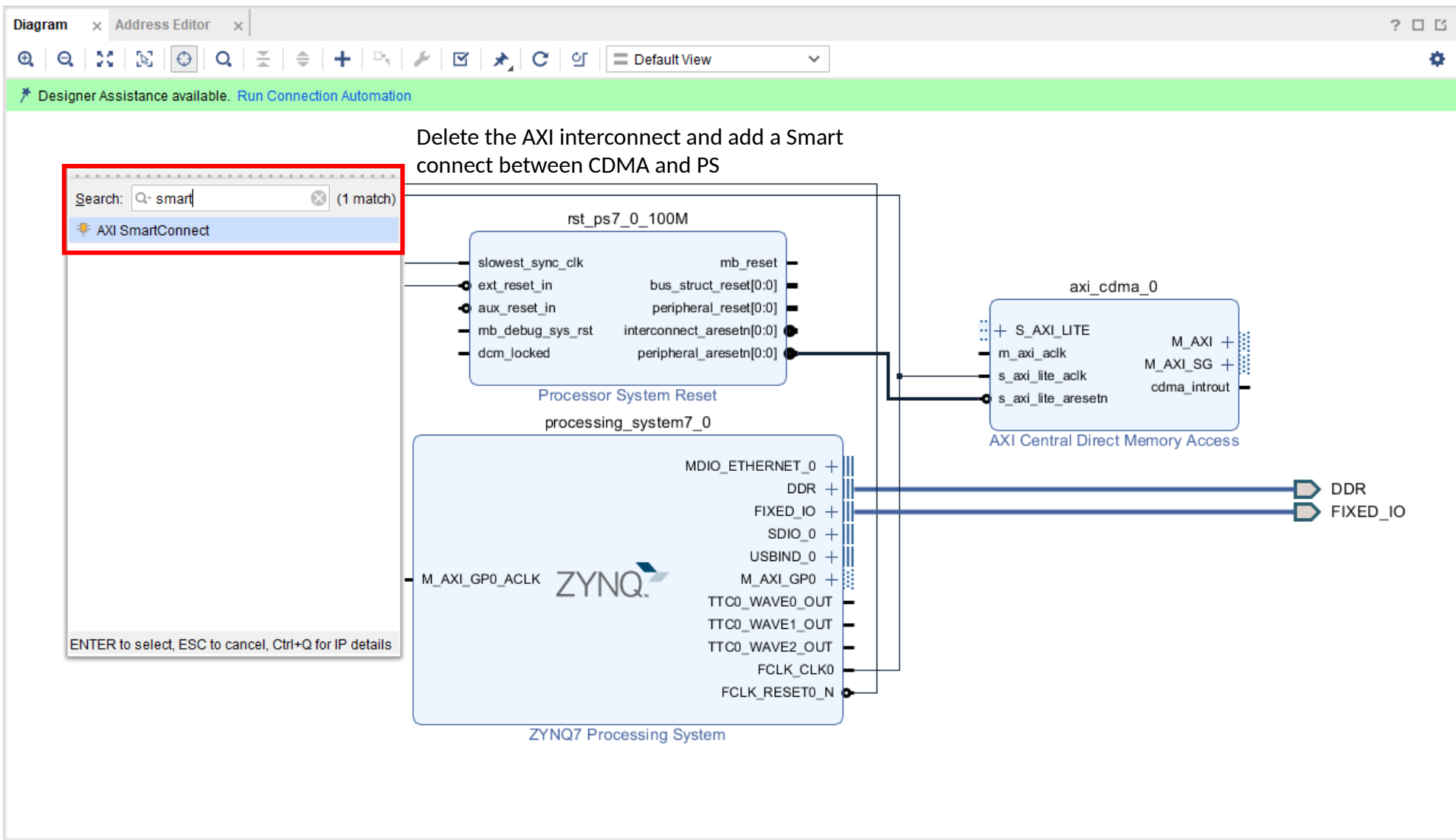
Block Properties ? _ □ ▢ x

processing_system7_0

Name: processing_system7_0

Parent name: design_1

General Properties IP



The diagram illustrates the ZYNQ7 Processing System architecture. The central component is the **ZYNQ7 Processing System**, which includes the **processing_system7_0** block. This block has various interfaces: **MDIO_ETHERNET_0**, **DDR**, **FIXED_IO**, **SDIO_0**, **USBIND_0**, **M_AXI_GP0**, **TTC0_WAVE0_OUT**, **TTC0_WAVE1_OUT**, **TTC0_WAVE2_OUT**, **FCLK_CLK0**, and **FCLK_RESET0_N**. The **M_AXI_GP0** interface is connected to the **smartconnect_0** block.

The **smartconnect_0** block, labeled **AXI SmartConnect**, has inputs **S00_AXI**, **S01_AXI**, **aclk**, and **aresetn**, and an output **M00_AXI**. The **aclk** input is connected to **FCLK_CLK0** from the processing system. The **aresetn** input is connected to **FCLK_RESET0_N** from the processing system. The **M00_AXI** output is connected to the **S_AXI_LITE** input of the **axi_cdma_0** block.

The **axi_cdma_0** block, labeled **AXI Central Direct Memory Access**, has inputs **S_AXI_LITE**, **m_axi_aclk**, **s_axi_lite_aclk**, and **s_axi_lite_aresetn**, and outputs **M_AXI** and **M_AXI_SG**. The **m_axi_aclk** input is connected to **FCLK_CLK0** from the processing system. The **s_axi_lite_aclk** input is connected to **FCLK_CLK0** from the processing system. The **s_axi_lite_aresetn** input is connected to **FCLK_RESET0_N** from the processing system. The **M_AXI** output is connected to **DDR** and **FIXED_IO** blocks. The **M_AXI_SG** output is connected to **cdma_introut**.

The **rst_ps7_0_100M** block, labeled **Processor System Reset**, has inputs **slowest_sync_clk**, **ext_reset_in**, **aux_reset_in**, **mb_debug_sys_rst**, and **dcm_locked**, and outputs **mb_reset**, **bus_struct_reset[0:0]**, **peripheral_reset[0:0]**, **interconnect_aresetn[0:0]**, and **peripheral_aresetn[0:0]**. The **slowest_sync_clk** input is connected to **FCLK_CLK0** from the processing system. The **ext_reset_in** input is connected to **FCLK_RESET0_N** from the processing system. The **aux_reset_in** input is connected to **FCLK_RESET0_N** from the processing system. The **mb_debug_sys_rst** input is connected to **FCLK_RESET0_N** from the processing system. The **dcm_locked** input is connected to **FCLK_RESET0_N** from the processing system. The **mb_reset** output is connected to **aresetn** of the **smartconnect_0** block. The **bus_struct_reset[0:0]** output is connected to **aresetn** of the **axi_cdma_0** block. The **peripheral_reset[0:0]** output is connected to **aresetn** of the **axi_cdma_0** block. The **interconnect_aresetn[0:0]** output is connected to **aresetn** of the **axi_cdma_0** block. The **peripheral_aresetn[0:0]** output is connected to **aresetn** of the **axi_cdma_0** block.

Connect the SmartConnect IP to M_AXI_GP0 on PS and S_AXI_LITE on CDMA

Enable Interrupt on PS and connect to CMDA

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator

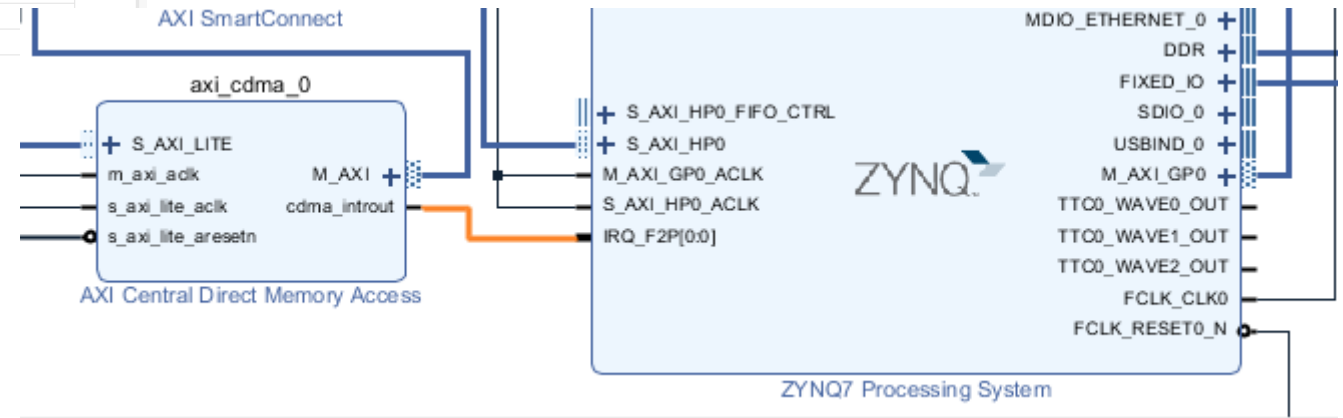
- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

Interrupts [Summary Report](#)

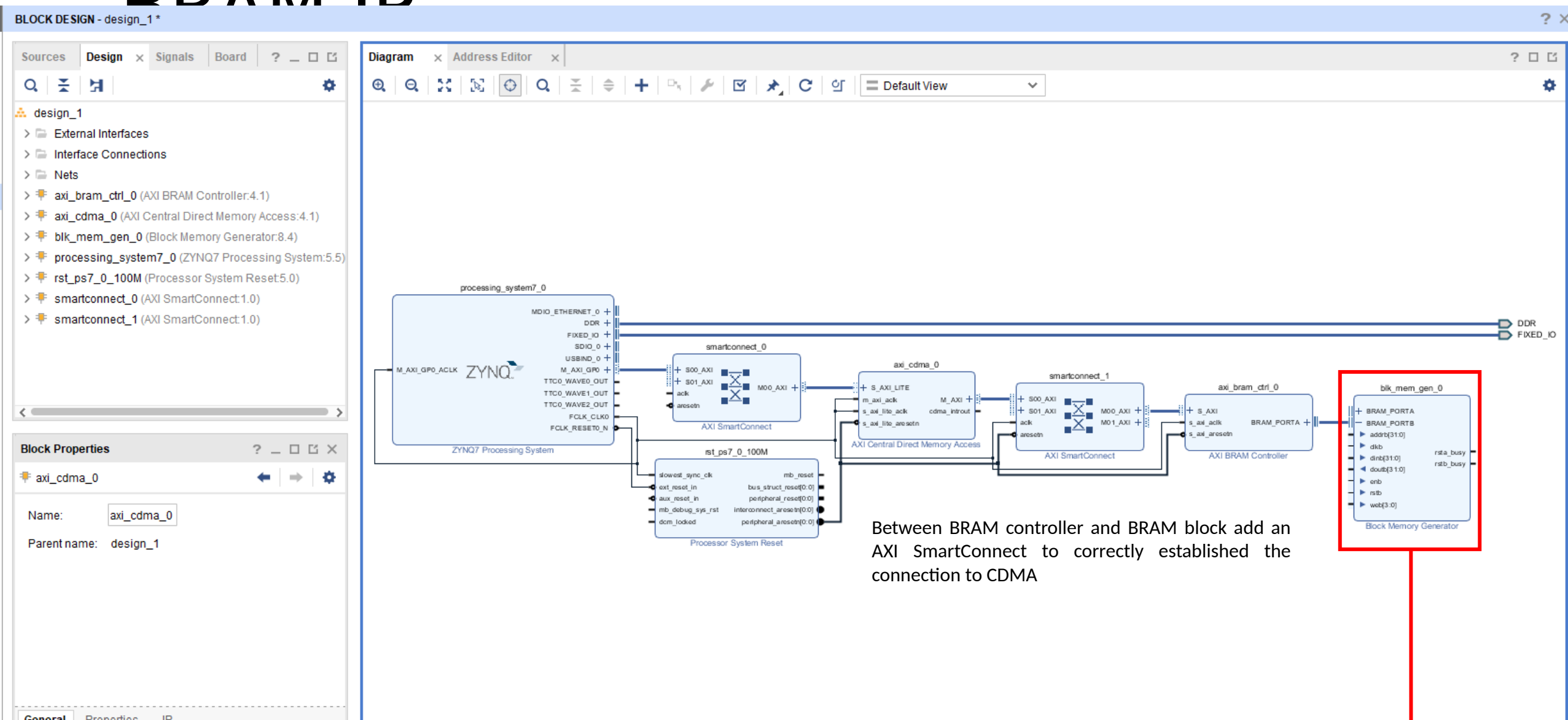
Search: Q-

Interrupt Port	ID	Description
<input checked="" type="checkbox"/> Fabric Interrupts		Enable PL Interrupts to PS and vice versa
<input checked="" type="checkbox"/> PL-PS Interrupt Ports		
<input checked="" type="checkbox"/> IRQ_F2P[15:0]	[91:84], [68:6]	Enables 16-bit shared interrupt port from the PL. MSB is assigned the hi
<input type="checkbox"/> Core0_nFIQ	28	Enables fast private interrupt signal for CPU0 from the PL
<input type="checkbox"/> Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the PL
<input type="checkbox"/> Core1_nFIQ	28	Enables fast private interrupt signal for CPU1 from the PL
<input type="checkbox"/> Core1_nIRQ	31	Enables private interrupt signal for CPU1 from the PL

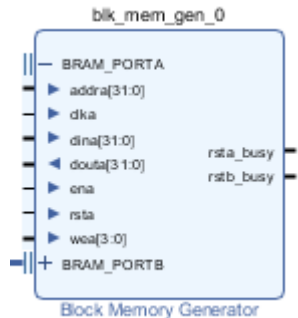
> PS-PL Interrupt Ports



ADD BRAM controller and input DDRAM ID



Change the BRAM parameters and BRAM controller for desired example



The diagram shows the Block Memory Generator component with the following ports and signals:

- Inputs: `BRAM_PORTA`, `addra[31:0]`, `dka`, `dina[31:0]`, `douta[31:0]`, `ena`, `rstb`, `wea[3:0]`, `BRAM_PORTB`.
- Outputs: `rstb_busy`, `rstb_busy`.

Block Memory Generator

Component Name `blk_mem_gen_0`

Basic | Port A Options | Port B Options | Other Options | Summary

Mode: **BRAM Controller** ☒ Generate address interface with 32 bits

Memory Type: **True Dual Port RAM** ☐ Common Clock

ECC Options

ECC Type: **No ECC** ☐ Error Injection Pins: **Single Bit Error Injection**

Write Enable

☒ Byte Write Enable

Byte Size (bits): **8**

Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives. Refer datasheet for more information.

Algorithm: **Minimum Area**

Primitive: **8kx2**

Component Name `blk_mem_gen_0`

Basic | Port A Options | Port B Options | Other Options | Summary

Memory Size (in words)

Write Width: **32** Range: 32 to 1024 (bits)

Read Width: **32**

Write Depth: **16384** Range: 2 to 1048576

Read Depth: **16384**

Operating Mode: **Write First** Enable Port Type: **Use ENA Pin**

Port A Optional Output Registers

☐ Primitives Output Register ☐ Core Output Register

☐ SoftECC Input Register ☐ REGCEA Pin

Port A Output Reset Options

☒ RSTA Pin (set/reset pin) Output Reset Value (Hex): **0**

☐ Reset Memory Latch Reset Priority: **CE (Latch or Register Enable)**

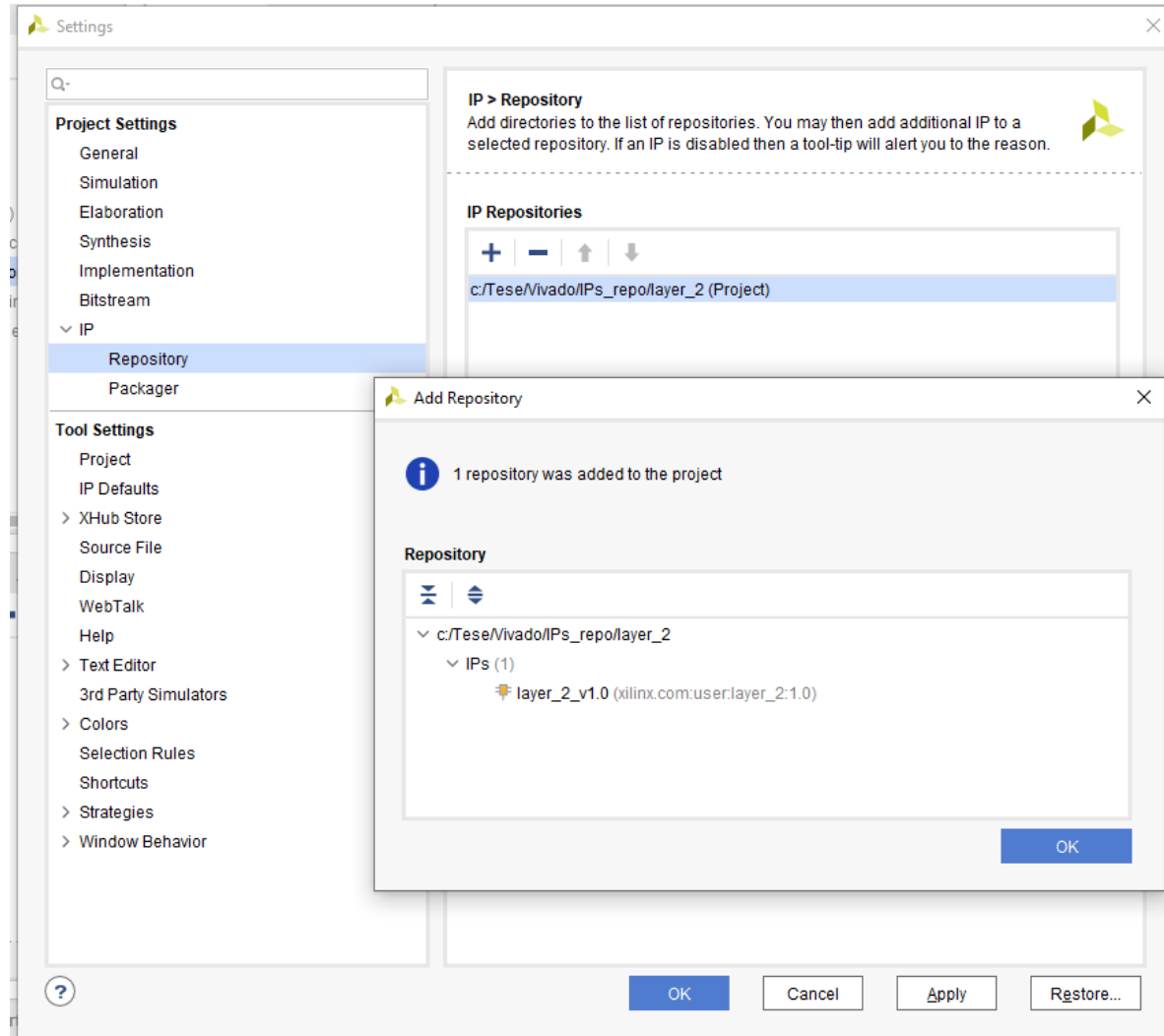
READ Address Change A

☐ Read Address Change A

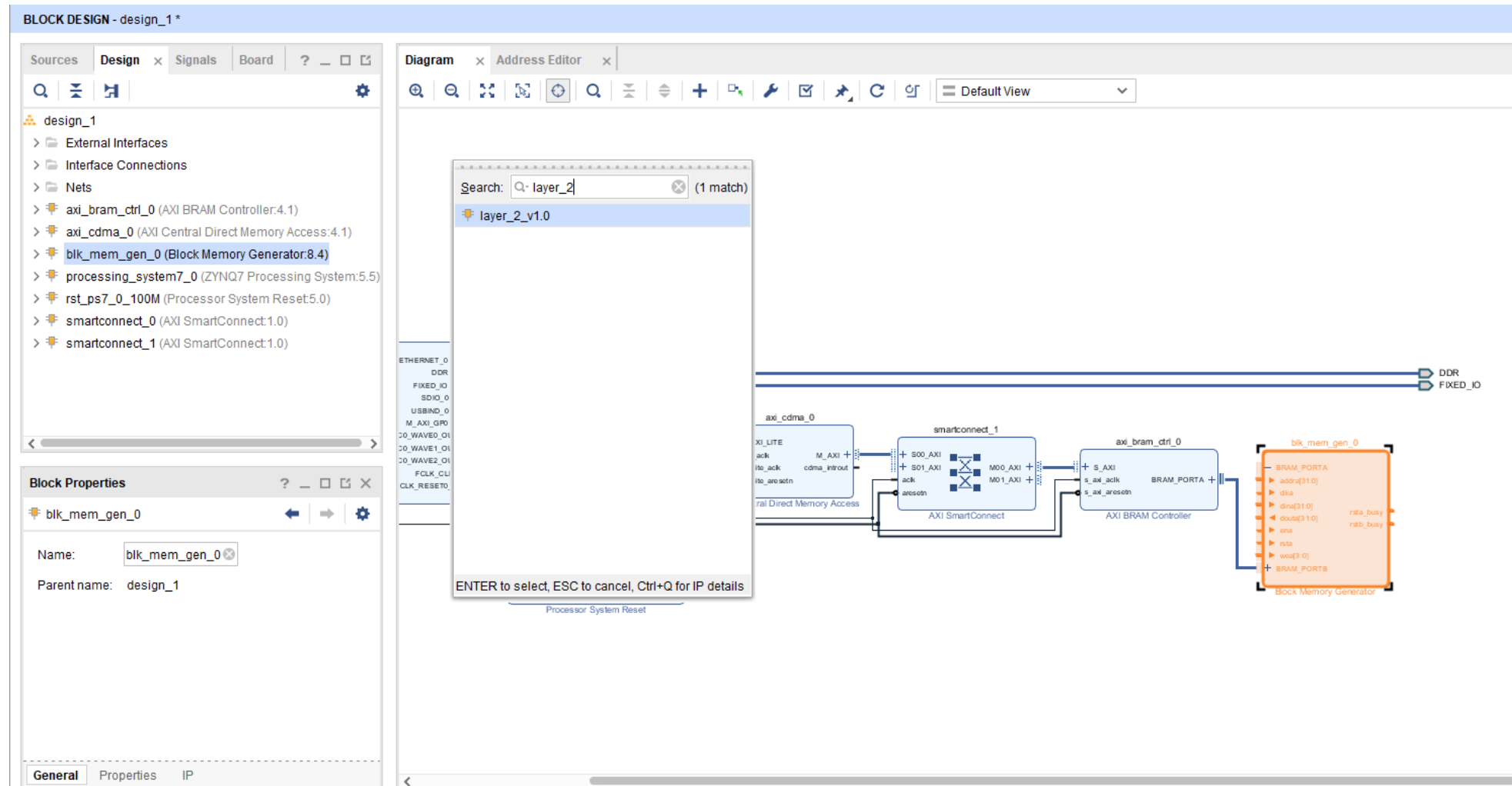
In this case is using a True Dual port, Port B is connected to CMDA and Port A will be connected to our IP.

To add a custom IP go to Setting and..

- Choose correctly the path where the IP is located
- After, add it to IP repository of the current project



Search for IP on Block Design and add it



Sources

- design_1
 - External Interfaces
 - Interface Connections
 - axi_bram_ctrl_0_BRAM_PORTA
 - axi_cdma_0_M_AXI
 - processing_system7_0_DDR
 - processing_system7_0_FIXED_IO
 - processing_system7_0_M_AXI
 - smartconnect_0_M00_AXI
 - smartconnect_0_M01_AXI
 - smartconnect_1_M00_AXI

Nets

- blk_mem_gen_0_douta
- layer_2_0_o_fm_bram_r_addr
- processing_system7_0_FCLK_0

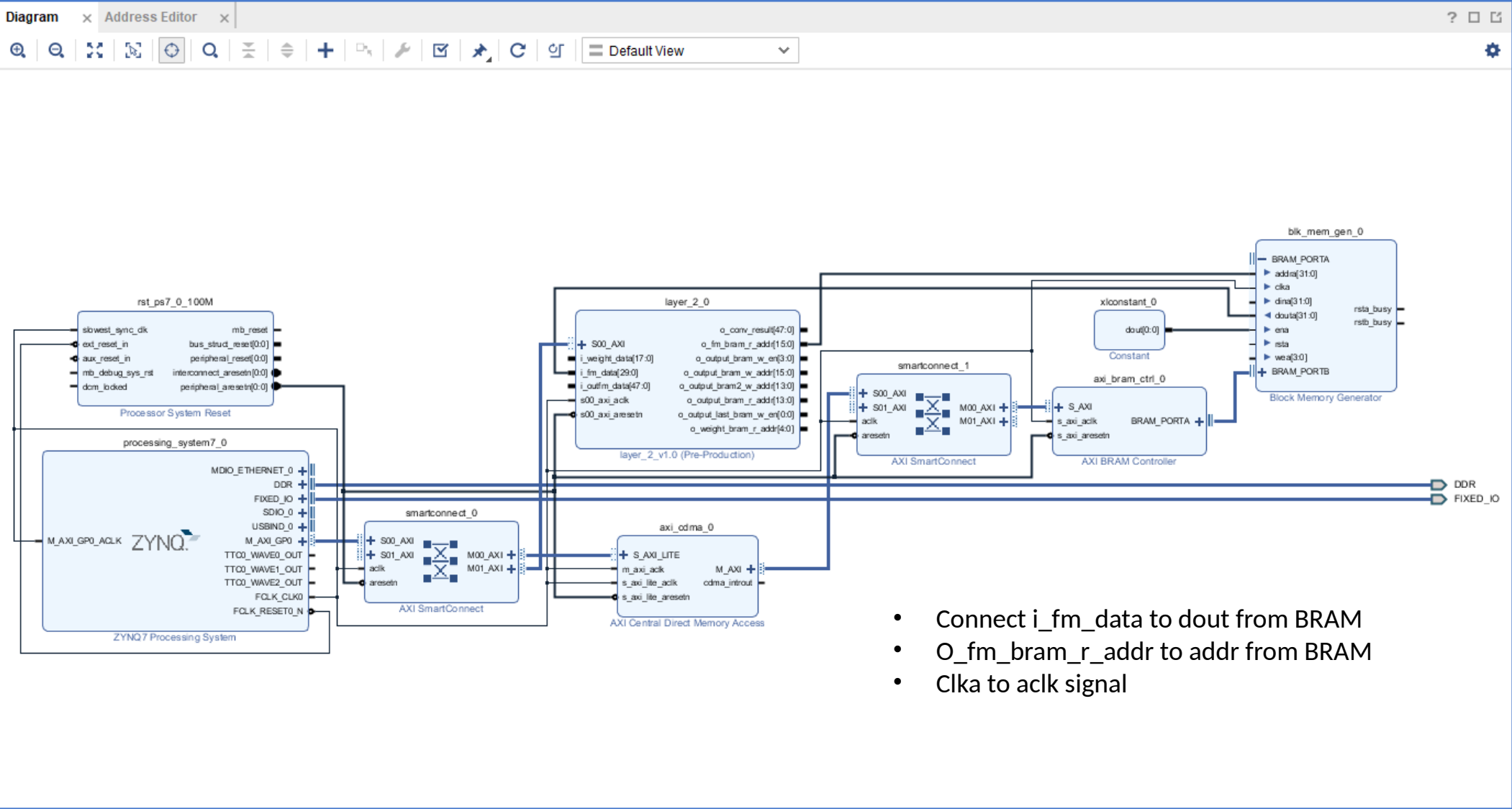
System Net Property

Name: xlconstant_0_dout

Parent name: design_1

Driver: xlconstant_0/dout

General Properties Pins



- Connect i_fm_data to dout from BRAM
- O_fm_bram_r_addr to addr from BRAM
- Clka to aclk signal

Add a BRAM to hold weight values

The image shows the Vivado IP configuration interface for the `blk_mem_gen_1` component. The interface is divided into several sections:

- IP Symbol:** Includes a checkbox for "Show disabled ports" and a button labeled "+ BRAM_PORTA".
- Component Name:** Set to `blk_mem_gen_1`.
- Basic Tab:**
 - Mode:** Set to "Stand Alone".
 - Memory Type:** Set to "Single Port RAM".
 - ECC Options:** "ECC Type" is set to "No ECC". "Error Injection Pins" is set to "Single Bit Error Injection".
 - Write Enable:** "Byte Write Enable" is unchecked. "Byte Size (bits)" is set to 9.
 - Algorithm Options:** "Algorithm" is set to "Minimum Area". "Primitive" is set to "8kx2".
- Port A Options Tab:**
 - Memory Size:** "Write Width" is 18, "Read Width" is 18, "Write Depth" is 18, and "Read Depth" is 18.
 - Operating Mode:** Set to "Write First". "Enable Port Type" is set to "Use ENA Pin".
 - Port A Optional Output Registers:** "Primitives Output Register" is checked. "Core Output Register" is unchecked. "SoftECC Input Register" is unchecked. "REGCEA Pin" is unchecked.
 - Port A Output Reset Options:** "RSTA Pin (set/reset pin)" is unchecked. "Reset Memory Latch" is unchecked. "Output Reset Value (Hex)" is 0. "Reset Priority" is set to "CE (Latch or Register Enable)".
 - READ Address Change A:** "Read Address Change A" is unchecked.

For a straightforward implementation, use a COE file to preload the BRAM memory with weight values

Add a BRAM to hold weight values

Basic Port A Options Other Options Summary

Pipeline Stages within Mux: 0 Mux Size: 1x1

Memory Initialization

☒ Load Init File

Coe File:

☐ Fill Remaining Memory Locations

Remaining Memory Locations (Hex):

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

Collision Warnings:

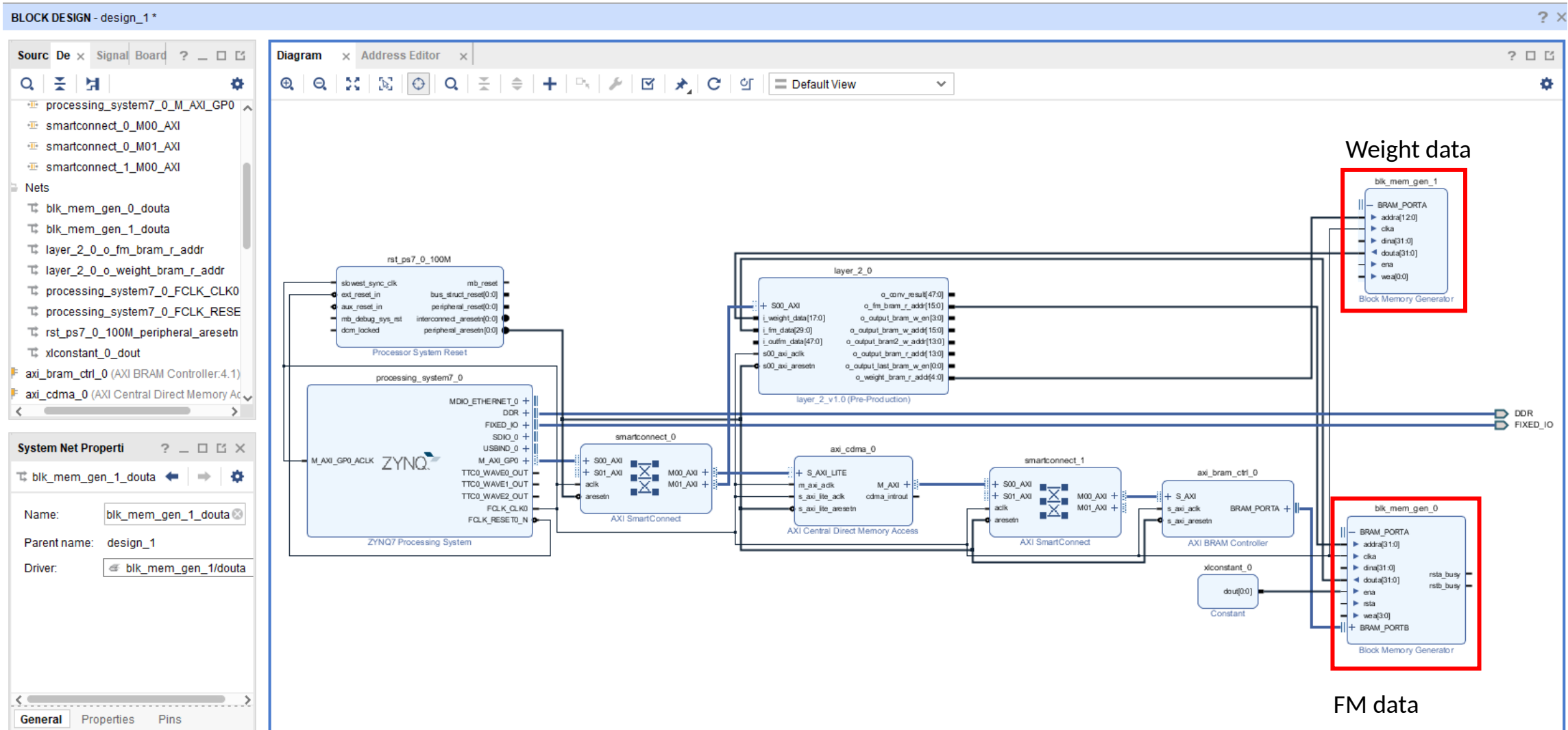
Behavioral Simulation Model Options

☐ Disable Collision Warnings ☐ Disable Out of Range Warnings

```
1 memory_initialization_radix=2;  
2 memory_initialization_vector=0 0 0 0 1 0 0 0 0;  
3
```

Identity filter used in filtro_2.coe file

Do the connections to layer IP

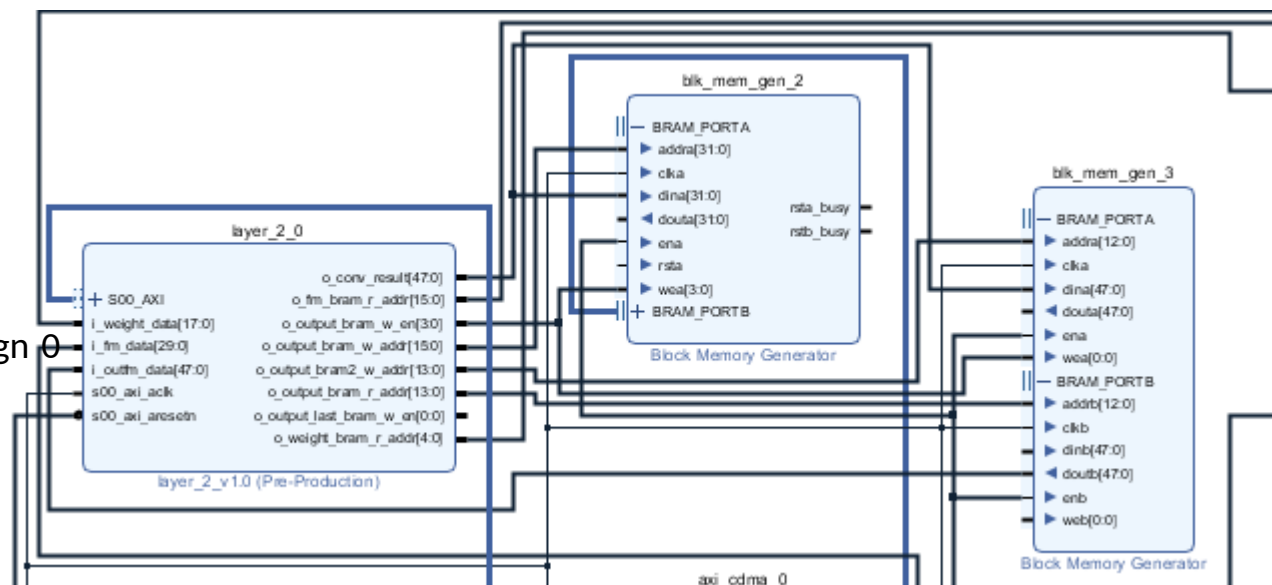


Add two output BRAMs

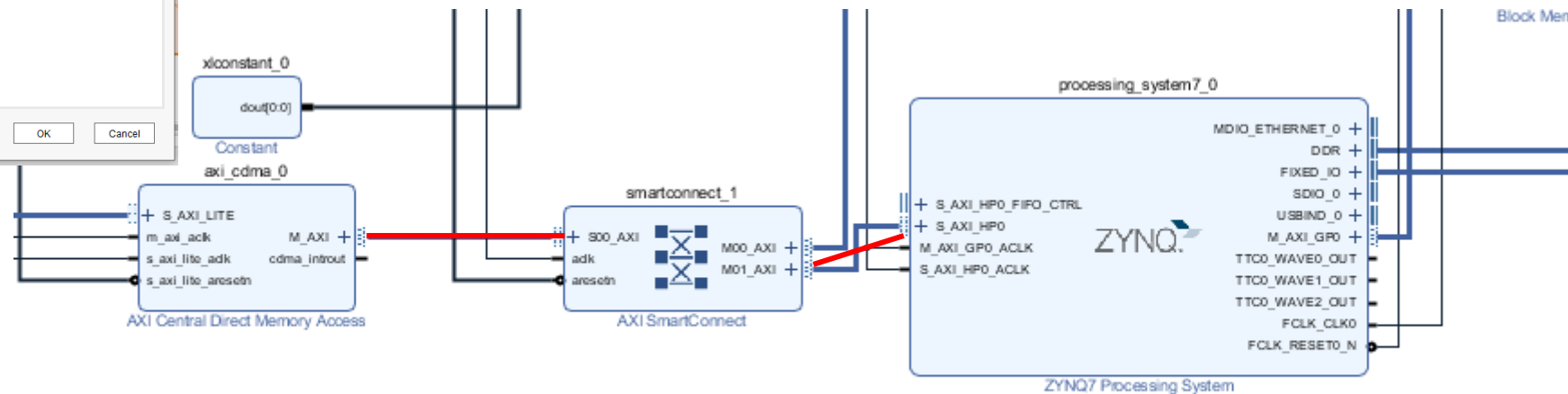
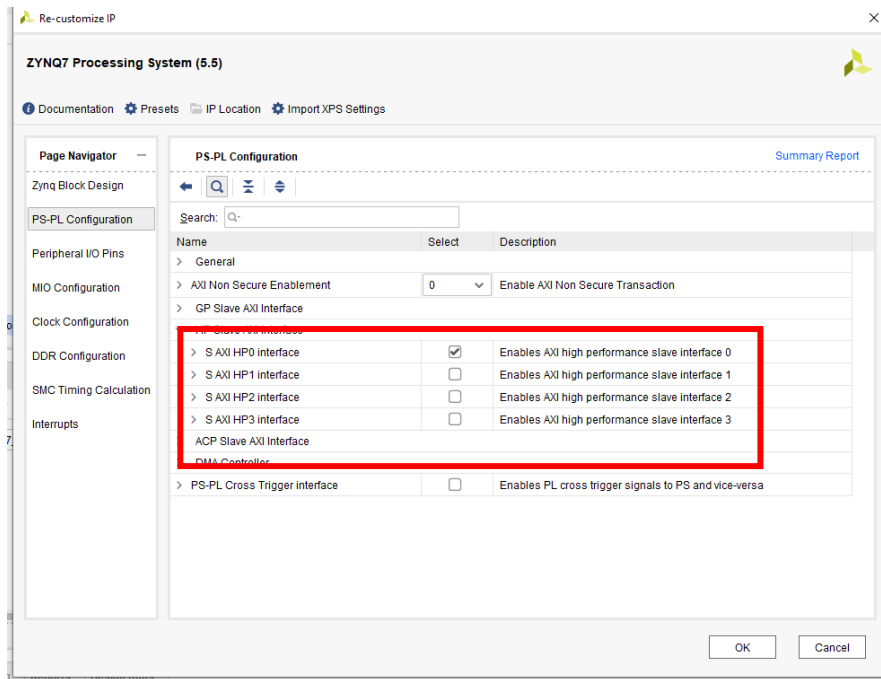
For a correct work of convolution process it is only necessary one BRAM block, in this case to read back and write on SD card it is required a connection from BRAM block and PS using a BRAM controller.

Correctly connect each input/output from layer ip with BRAM modules.

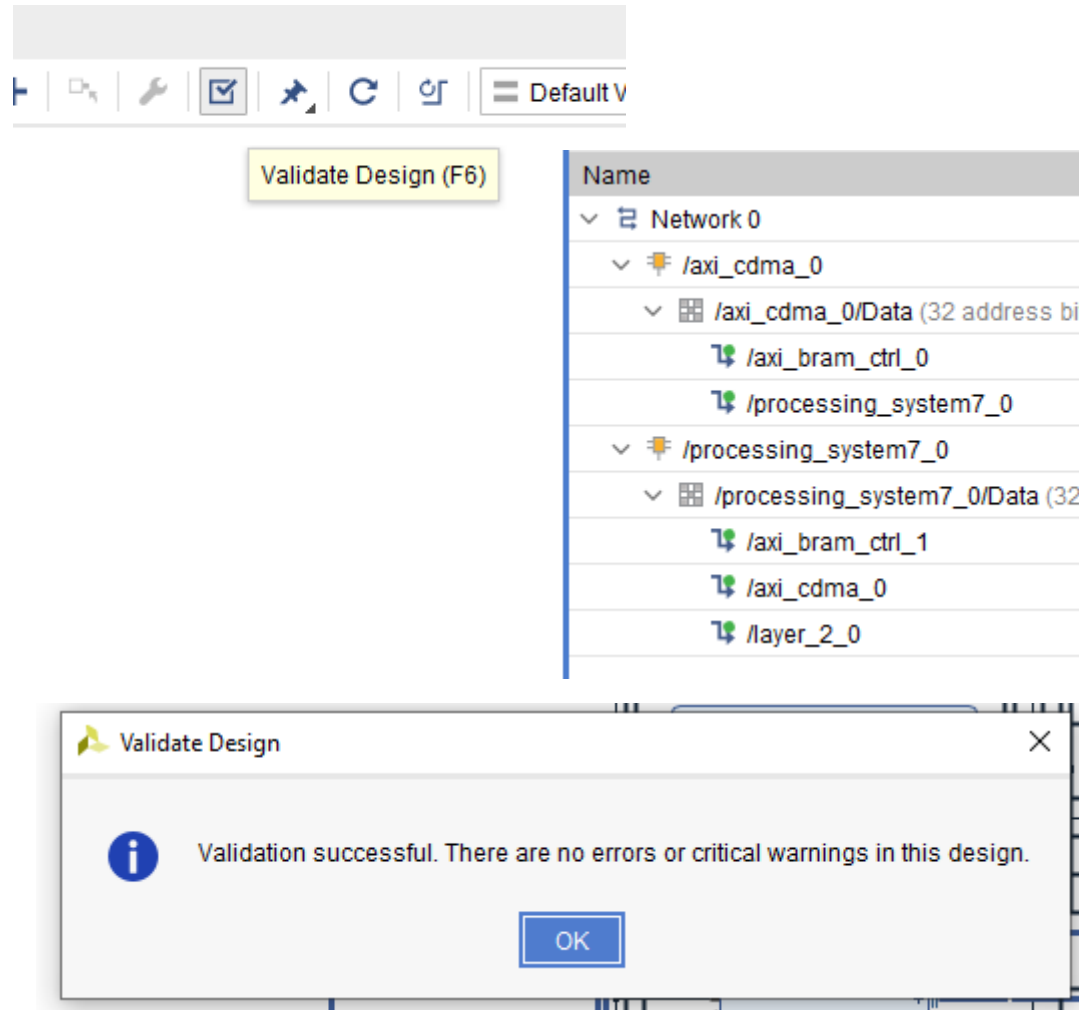
In case of incorrect work of BRAM blocks, use const IP to assign 0 value to dinb on blk_mem_gen3



Connect CDMA to High Performance on PS



Run Validate Desing



The screenshot shows the 'Validate Design (F6)' button in the top toolbar. Below it, a table displays the design hierarchy and memory addresses. The table has columns: Name, Interface, Slave Segment, Master Base Address, Range, and Master High Address. The design hierarchy includes Network 0, /axi_cdma_0, /axi_cdma_0/Data (32 address bits : 4G), /axi_bram_ctrl_0, /processing_system7_0, /processing_system7_0/Data (32 address bits : 0x40000000 [1G]), /axi_bram_ctrl_1, /axi_cdma_0, and /layer_2_0. The table lists the Master Base Address, Range, and Master High Address for each component.

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/axi_cdma_0					
/axi_cdma_0/Data (32 address bits : 4G)					
/axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	64K	0xC000_FFFF
/processing_system7_0	S_AXI_HP0	HP0_DDR_LOW	0x0000_0000	512M	0x1FFF_FFFF
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/axi_bram_ctrl_1	S_AXI	Mem0	0x4000_0000	64K	0x4000_FFFF
/axi_cdma_0	S_AXI_LITE	Reg	0x7E20_0000	64K	0x7E20_FFFF
/layer_2_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF

The 'Validate Design' dialog box shows a message: 'Validation successful. There are no errors or critical warnings in this design.' with an 'OK' button.

Should appear not errors, if so check address editor tab for possible addresses collisions.

The range value should be updated for different sizes of image being processed. Larger images requires more data consumption.

After that..

- Generate output products
- Create HDL Wrapper
- Run synthesis
- Run implementation
- Generate bitstream

“Export SDK” after bitstream has been generated

The screenshot shows the Vivado IDE interface. The 'File' menu is open, and the 'Export' option is selected, which has opened a sub-menu with the following options:

- Export Hardware...
- Export Block Design...
- Export Bitstream File...
- Export Simulation...

The 'Address Editor' window is open, showing a table of hardware resources. The table has the following columns: Name, Interface, Slave Segment, Master Base Address, Range, and Master High Address. The data is organized into a tree structure under 'Network 0'.

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/axi_cdma_0					
/axi_cdma_0/Data (32 address bits : 4G)					
/axi_bram_ctrl_1	S_AXI	Mem0	0x4110_0000	64K	0x4110_FFFF
/processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/axi_bram_ctrl_2	S_AXI	Mem0	0x4200_0000	64K	0x4200_FFFF
/axi_cdma_0	S_AXI_LITE	Reg	0x7E20_0000	64K	0x7E20_FFFF
/myip_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF

At the bottom of the IDE, the 'Design Runs' window is open, showing a table of design runs. The table has the following columns: Name, Constraints, Status, WNS, TNS, WHS, THS, TPWS, Total Power, Failed Routes, LUT, FF, BRAM, URAM, DSP, Start, Elapsed, Run Strategy, and Report Strategy.

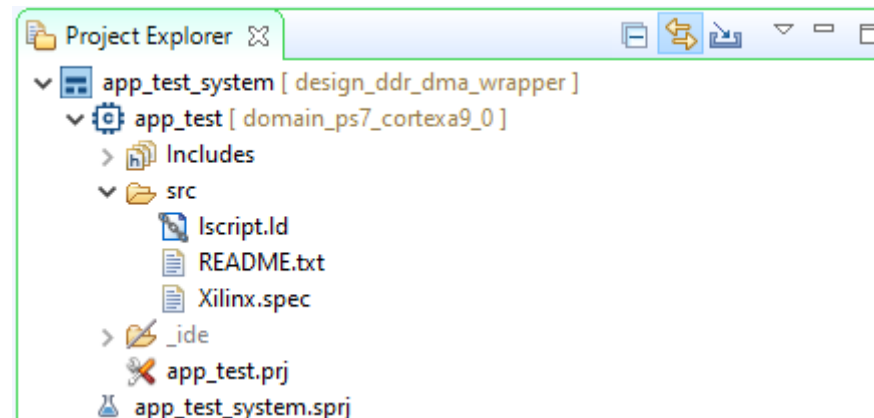
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.0	0	0	7/28/21, 3:26 PM	00:01:15	Vivado Synthesis Defaults (Vivado Synthesis 2020)	Vivado Synthesis Default Reports (Vivado Synthesis 2020)
impl_1	constrs_1	route_design Complete!	1.308	0.000	0.023	0.000	0.000	1.812	0	6621	8190	56.0	0	9	7/28/21, 3:27 PM	00:05:08	Vivado Implementation Defaults (Vivado Implementation 2020)	Vivado Implementation Default Reports (Vivado Implementation 2020)
synth_1_copy_1	constrs_1	Not started															Vivado Synthesis Defaults* (Vivado Synthesis 2020)	Vivado Synthesis Default Reports (Vivado Synthesis 2020)
Out-of-Context Module Runs																		
design_1		Submodule Runs Complete													3/6/21, 12:39 PM	3,457:46:33		

Export hardware platform

- Platform type = Fixed
- Output = Include bitstream
- Choose the name of the hardware platform type and the location where the xsa file will be stored – default option is viable
- Finish to export the platform

Xilinx Vitis

- Create new application project
- Select the .xsa file previously generated
- Write the name of the application
- For the template, select the “Empty Application” option.
- The application structure should look similar to this:



Add source files

- From path: "C:\Xilinx\Vitis\2020.1\data\embeddedsdsw\XilinxProcessorIPLib\drivers\axicdma_v4_7\examples" copy to src the file: "xaxicdma_example_simple_intr.c"
- This file is used since some functions are already implemented
- Nonetheless we need to edit this file.
- "xparameters.h" file is where all the macros with the base address for the DDR memory and the BRAM controllers are defined.

Includes

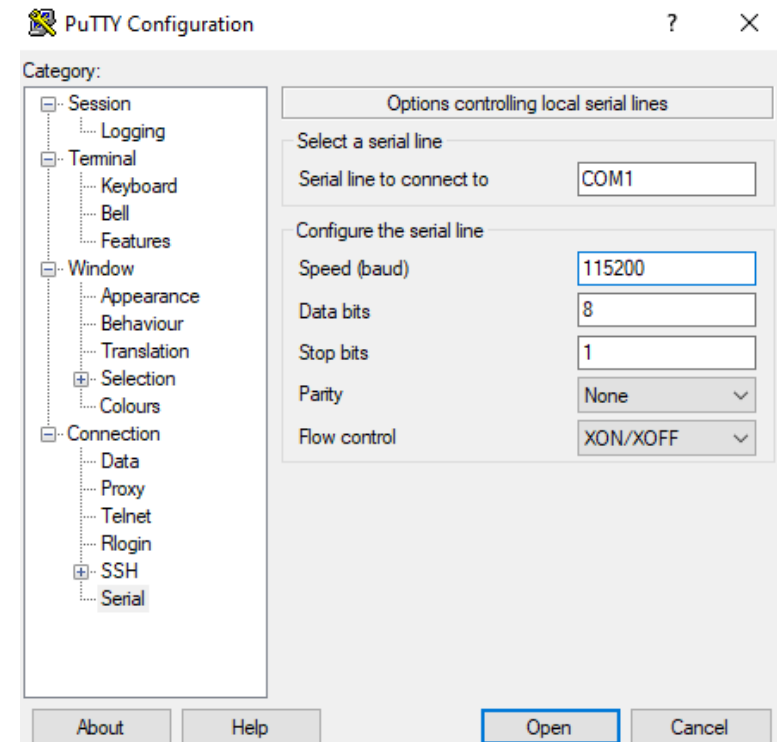
- Make sure to include this set of libraries.
As we pretend to transfer data from the Sdcard to the DDR memory, some functions will be used for that purpose.
- Depending on the data size used inside the hardware module, the functions may need to be edited to work with a certain data width.

```
#include "xparameters.h"  
#include "SDCARD.h"  
#include "CONFIGS.h"  
#include "stdio.h"  
#include "math.h"
```

```
int sdCardDriverInit();  
int writeFramesToSDCard(char* SD_File, u8 *FrameBuffer, u32 SizeBuffer, u32 offset);  
int readFramesFromSDCard(char* SD_File, u16 n_frames, u8 *FrameBuffer, u32 SizeBuffer);  
int read8FramesFromSDCard(char* SD_File, u8 *FrameBuffer, u32 SizeBuffer, u32 offset);
```


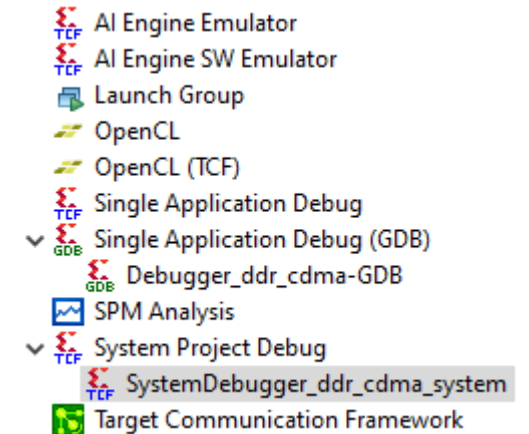
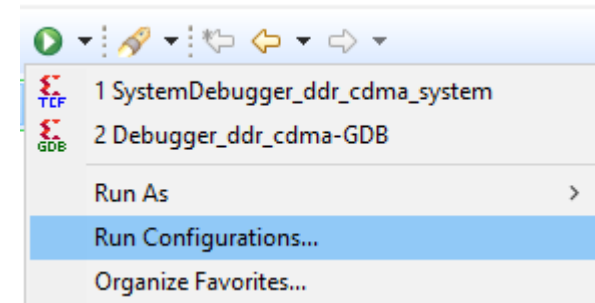

Program the FPGA

- Select “Program the FPGA”
- Select the correct bitstream and then select “Program”
- Before Run the application you may want to setup the connection to the target platform first.
- In this case, PuTTY was used
- Set the port used and the correct baudrate



Run the application

- Click on the created application
- Select “Run configurations” from the top-bar
- Select “System debugger”
- Hit run. In the terminal the values sent from the DDR to the BRAM will be displayed as well as the output of processing.
- Depending on the IFM provided, the output values will vary.



```
after: Pos (42000000) value: 0
after: Pos (42000008) value: 214
after: Pos (42000010) value: 216
after: Pos (42000018) value: 132
after: Pos (42000020) value: 136
after: Pos (42000028) value: 193
after: Pos (42000030) value: 196
after: Pos (42000038) value: 199
after: Pos (42000040) value: 200
after: Pos (42000048) value: 202
after: Pos (42000050) value: 205
after: Pos (42000058) value: 205
after: Pos (42000060) value: 204
after: Pos (42000068) value: 207
after: Pos (42000070) value: 209
after: Pos (42000078) value: 208
after: Pos (42000080) value: 212
after: Pos (42000088) value: 215
after: Pos (42000090) value: 217
after: Pos (42000098) value: 214
```