ICAT3170, SOC-FPGA

Total system

Laboratory exercise: Image classification

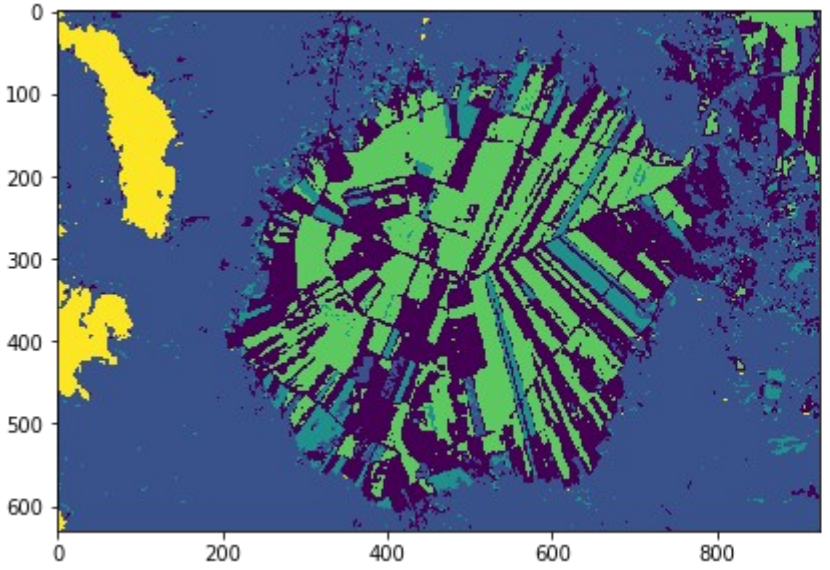
Vaasan yliopisto
UNIVERSITY OF VAASA

- Satellite images are new free underutilized resources, which can be used for many Earth observation tasks. They are already used for example land use analysis
- The analysis of large image basis takes a lot of time and energy
- Also the possibilities to store images in the satellite and transmit them back to Earth are limited
- Therefore it could be usefull to analyze the images already in space
- The processing capabilities in space are usually power limited, so FPGA:s are a perfect match for space based edge computing

Söderfjärden satellite image classification

Vaasan yliopisto
UNIVERSITY OF VAASA

- First we need to train the classifier using training data.
- This part of outside the scope of this course and the ready trained classifier will be provided
- See example



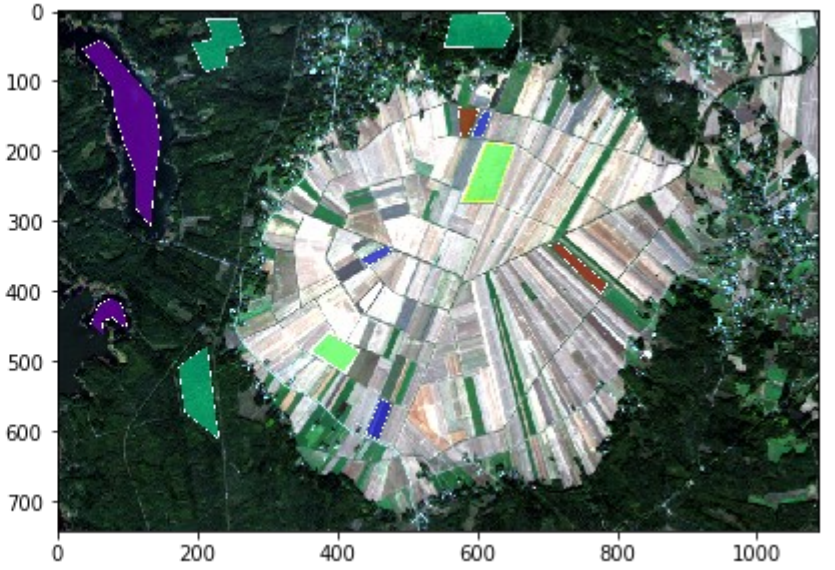
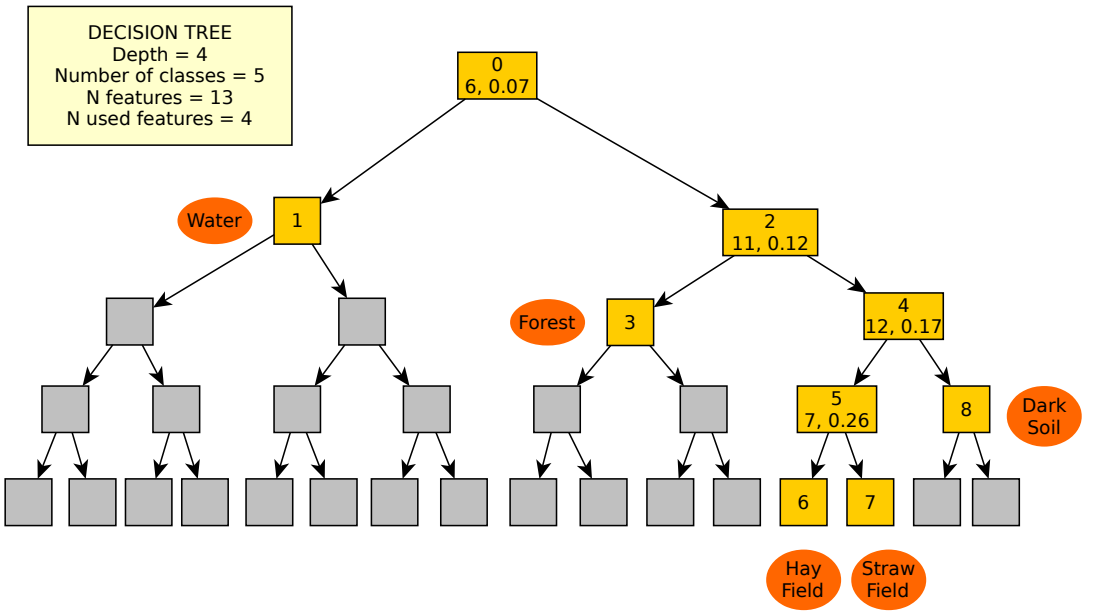


Image classification, decision trees

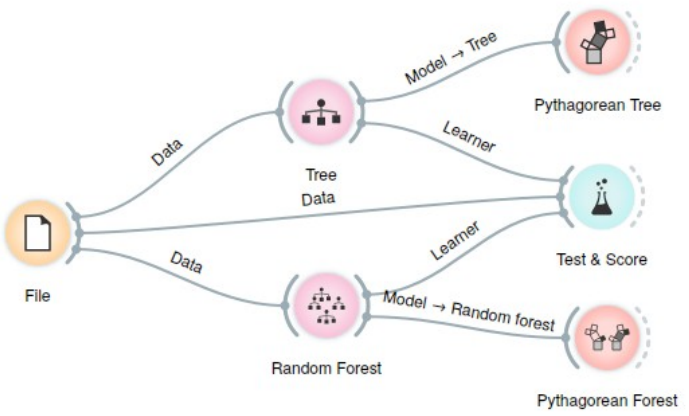
Vaasan yliopisto
UNIVERSITY OF VAASA

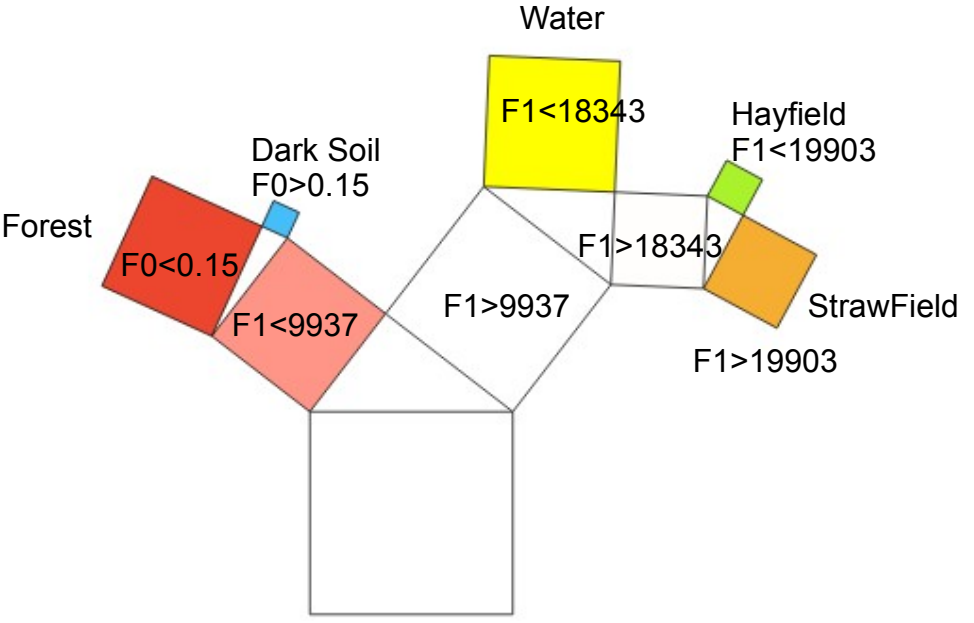
- Decision treess are very important type of machine learning methods used for classification and regression
- Nowadays so called Ensemble Methods, which consists of a bag of simple classifiers are proven very versatile
- Those simple classifiers are often simple decision trees: Random forest, (Extremely random) Extratrees and gradient boosted trees are such methods. The final output is reached by majority voting
- The bag of simple classifiers, sounds like it is a perfect match for FPGA. You need to implement just one simple decision tree and multiply many copies of it. Each copy may use different input data and different threshold, so you only need to implement a method to parametri



Classifier training

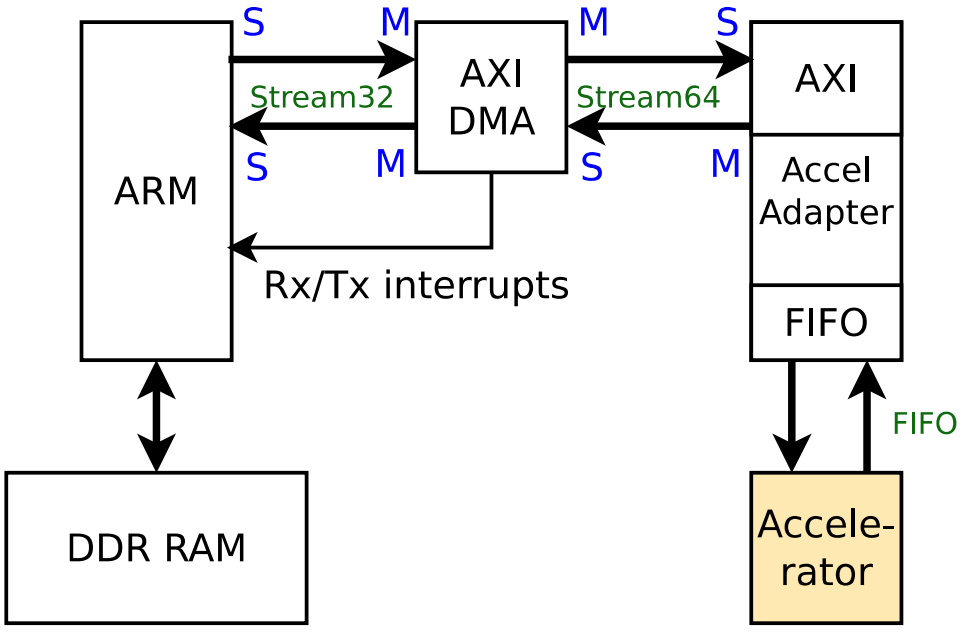
- Classifier can be trained for example using a software called Orange, or by using Python and Scikit Learn machine learning libraries





The processing system

- The data is located first in the RAM
- The Linux side open the file and configures AXI DMA to transfer the data to the Accelerator (TX) using Accelerator adapter, and transferring the results back to the RAM using the other channel of AXI DMA (RX)
- When the DMA finishes the transfer TX or RX, it sends an interrupt to the CPU to inform about it
- The accelerator can receive the data from the buffer of Accelerator adapter using simple FIFO interface



Building the FPGA side step by step

Vaasan yliopisto
UNIVERSITY OF VAASA

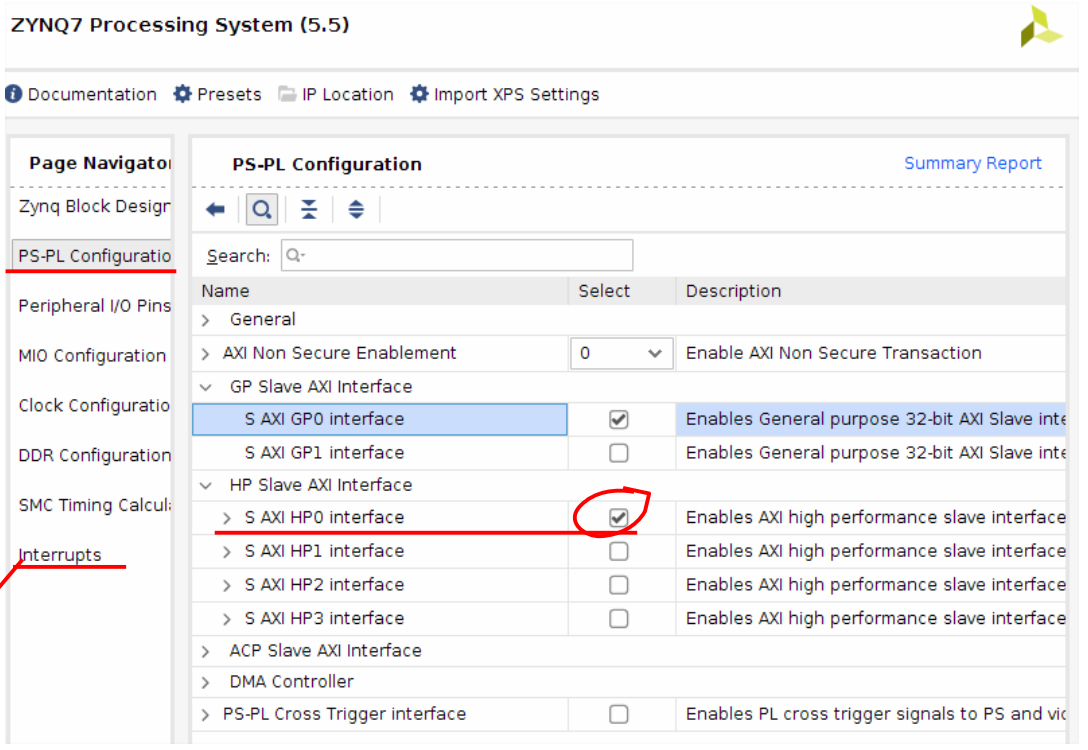
- Create new application project in Vivado
- Insert and configure the processing system
- Insert and connect AXI-DMA
- Insert and configure accelerator adapter
- Implement and connect accelerator adapter

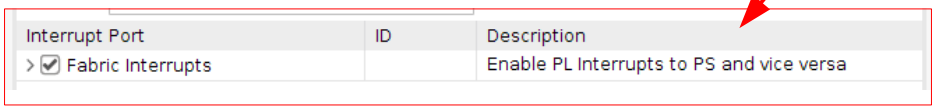
- The implementation of the accelerator needs to be carefully tested before integration, because the debugging of the whole system is extremely difficult
- JTAG testing adapters will be also really useful

Insert and configure the processing system

Vaasan yliopisto
UNIVERSITY OF VAASA

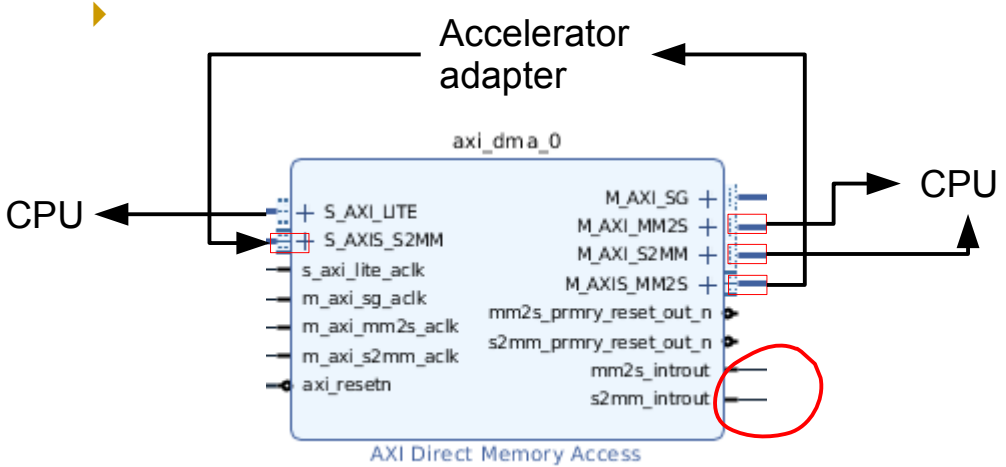
- Insert Zynq processing system in the block diagram as usual
- Enable high performance slave axi interface
- Enable interrupts from PL

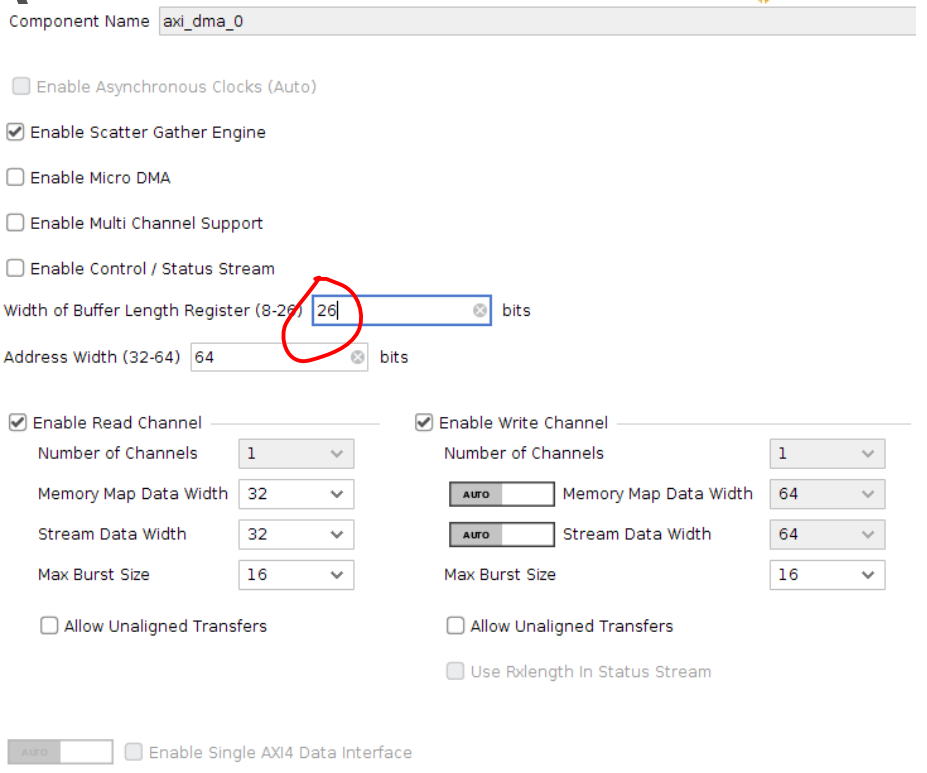




Insert and connect AXI-DMA

- Insert AXI DMA component and configure it according to the instructions
- S_AXI_Lite is used for configuring the axi_dma. It will be seen in the memory address space of the processor
-

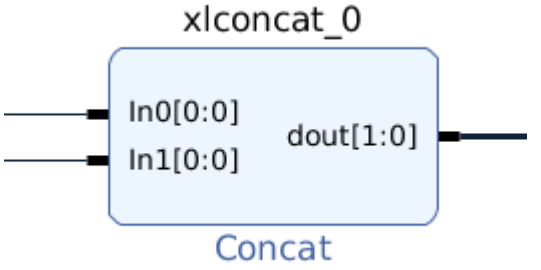


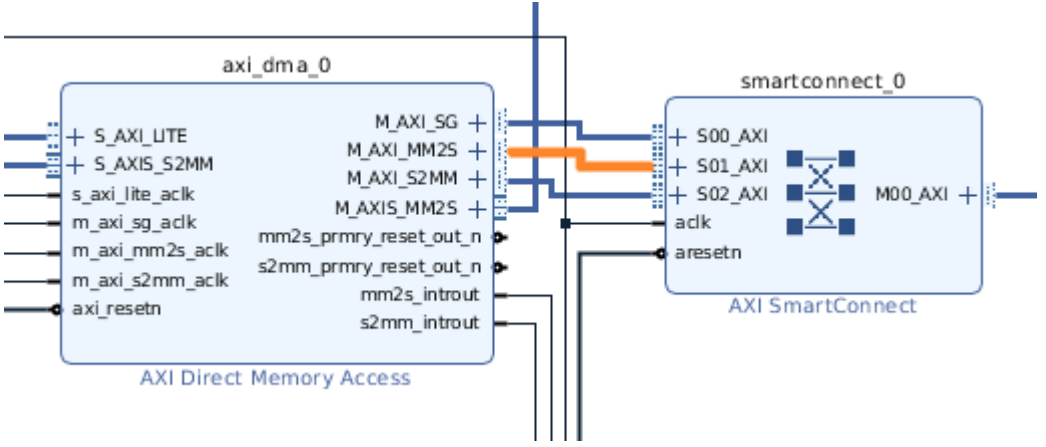


Cell	Slave Interface	Slave Segment	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
axi_dma_0					

AXI DMA connections

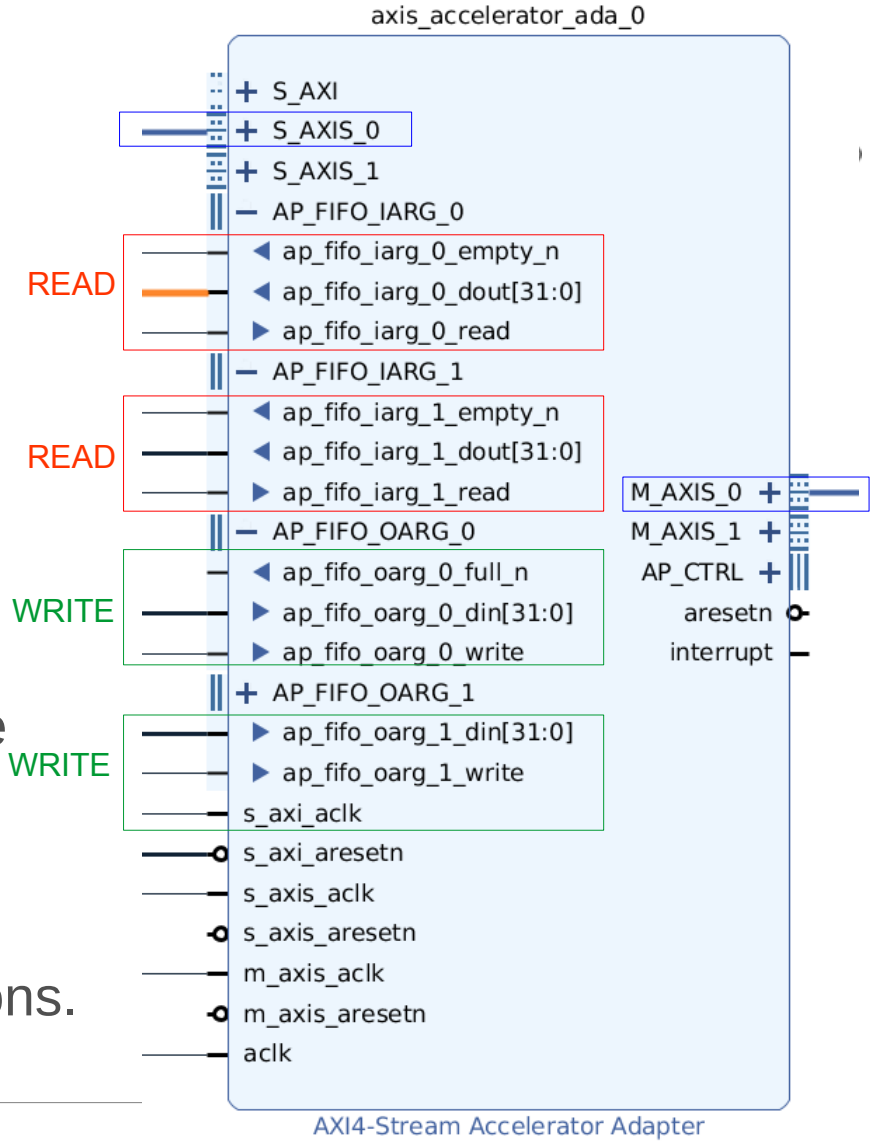
- Insert xlconcat component in the desing, and let it concatenate the tx and rx interrupts from the AXI_DMA to one array, which is then connected to the ZYNQ
- Insert smartconnect component to connect AXI buses together.
- Connect AXI_DMA to it
- Connect M00_AXI interface to the S_AXI_HP0 connection in th ZYNQ





AXI Stream accelator adapter

- The adapter is connected to the AXI stream slave bus for receiving data from AXI_DMA
- The AXI stream master is connected to he AXI_DMA to send the results to the DMA
- The AP_FIFO_IARG_n provides FIFO read interface with three signals: data out, empty indicator and read enable
- The AP_FIFO_OARG_n provides FIFO write interface with three signals: data in, full indicator and write enable
- Internally accelerator contains RX and TX FIFO:s, which buffer the data in both directions.



Accelerator FIFO signals

Vaasan yliopisto
UNIVERSITY OF VAASA

- Accelerator adapter is a FIFO, which provides the standard FIFO interface signals as shown above.

Accelerator Input Argument FIFO Interface (0 to 7)				
ap_fifo_iarg_n_read	AP_FIFO_IARG_n	I	–	FIFO interface read enable
ap_fifo_iarg_n_dout	AP_FIFO_IARG_n	O	0x0	FIFO interface read data
ap_fifo_iarg_n_empty_n	AP_FIFO_IARG_n	O	0x0	FIFO interface FIFO empty indicator
Accelerator Output Argument FIFO Interface (0 to 7)				
ap_fifo_oarg_n_write	AP_FIFO_OARG_n	I	–	FIFO interface write enable
ap_fifo_oarg_n_din	AP_FIFO_OARG_n	I	–	FIFO interface write Data
ap_fifo_oarg_n_full_n	AP_FIFO_OARG_n	O	0x0	FIFO interface FIFO full indicator

FIFO interface, Clock diagram

Vaasan yliopisto
UNIVERSITY OF VAASA

- Writing to the FIFO
  - Put the data in the data bus
    `wr_data <= data;`
  - Activate write enable:
    `wr_en <= '1'`
  - Pause writing if `full='1'`
- Reading from the FIFO
  - Activate read enable:
    `rd_en <= '1'`
  - Read the data from the data bus
    `data <= rd_data;`
  - Pause reading if `empty='1'`

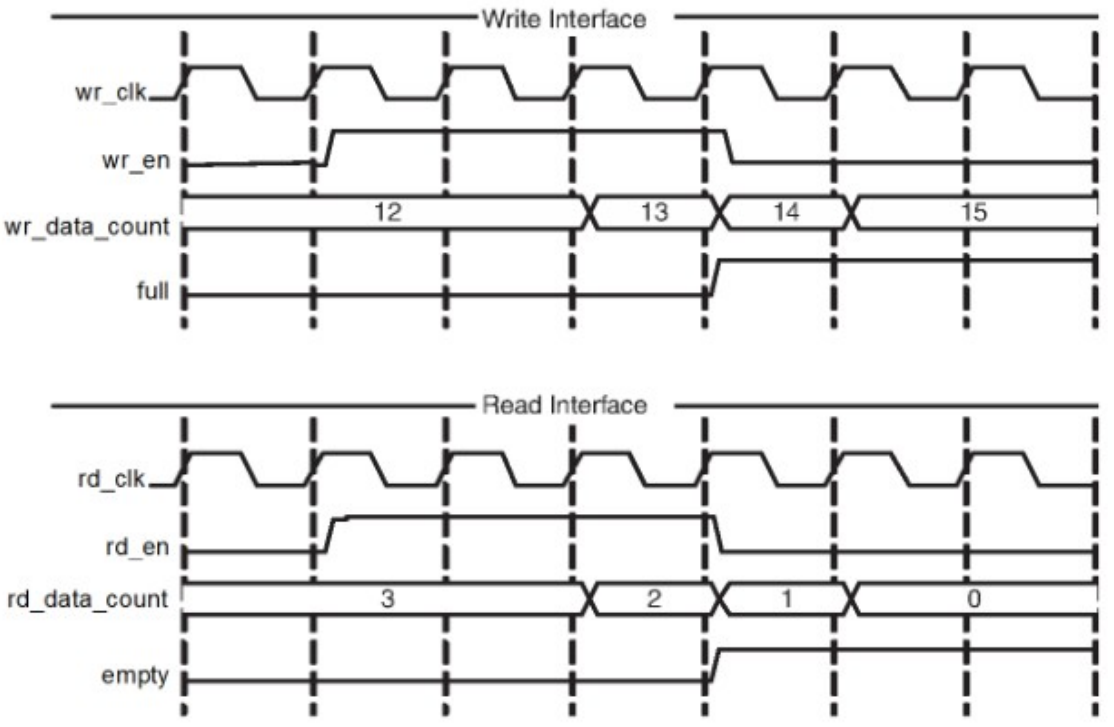
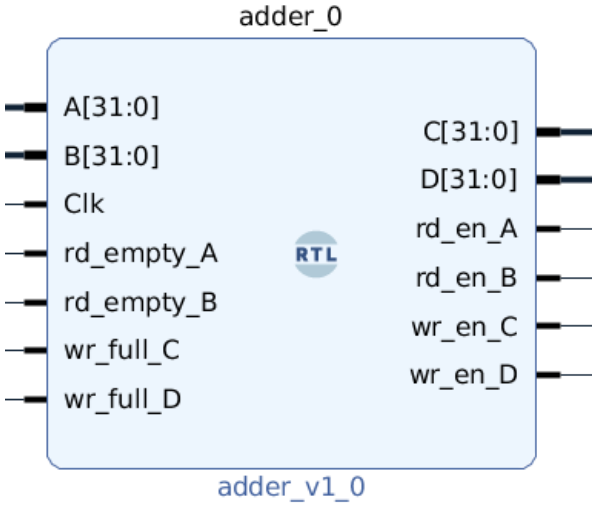


Figure 3-12: **Write and Read Data Counts for FIFO with Independent Clocks**

Custom accelerator with FIFO interface

- An adder implemented to fit in the FIFO interface




```
architecture RTL of adder is
    signal enabled : std_logic := '0';
begin
    process (clk)
    begin
        if(rising_edge(clk))
        then
            -- Check that we have input data and we have
            if (rd_empty_A='0') and (wr_full_C='0')
            then
                rd_en_A <= '1';
                wr_en_C <= '1';
                enabled <= '1';
            else
                wr_en_C <= '0';
                rd_en_A <= '0';
                enabled <= '0';
            end if;

            if enabled = '1' then
                C <= std_logic_vector(signed(A) + 1);
            else
                C <= (others => '0');
            end if;
        end if;
    end process;
end RTL;
```

Finally

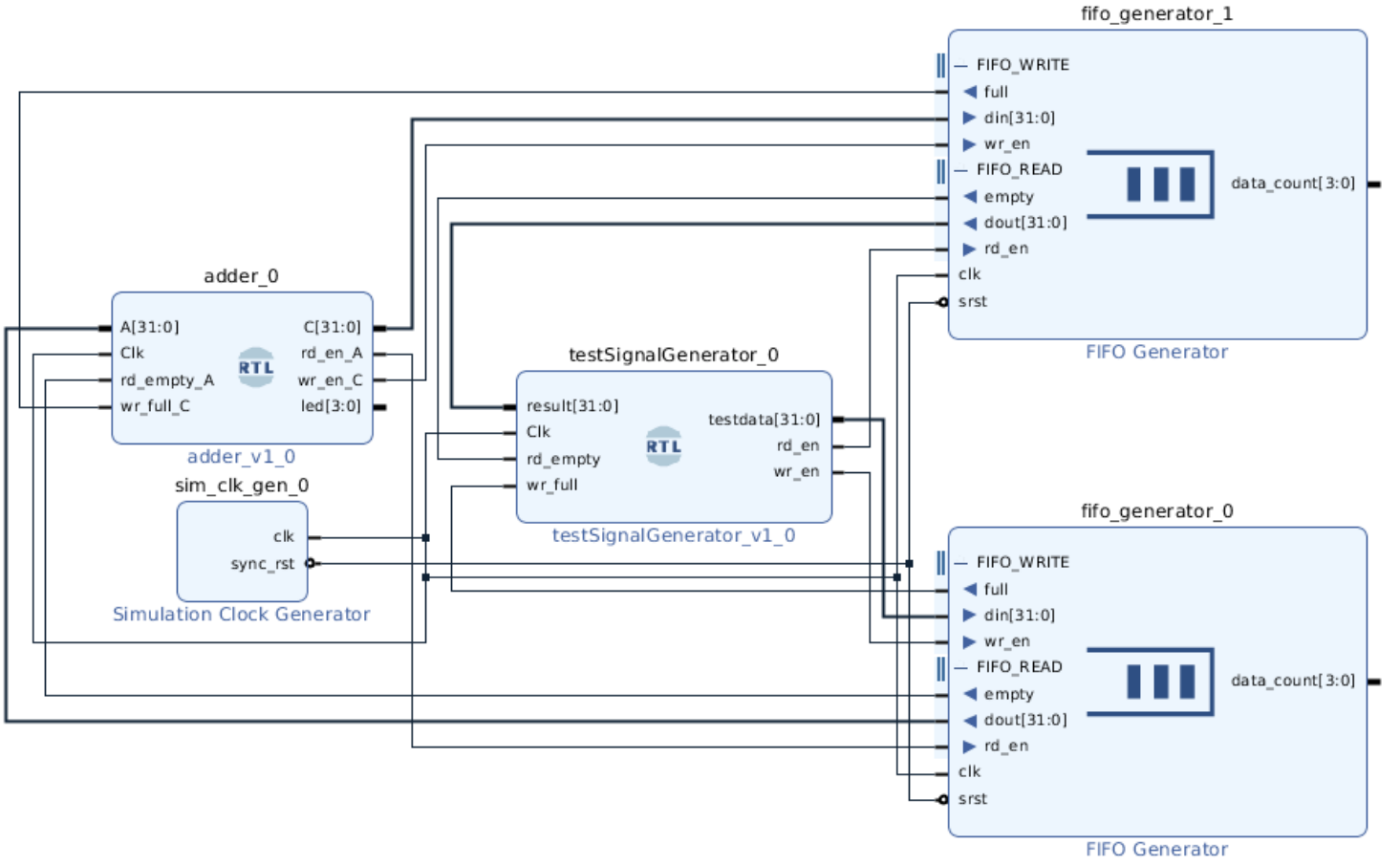
Vaasan yliopisto
UNIVERSITY OF VAASA

- Let connection assistant make the rest of the connections
- Some clock signals needs to be connected as well, connect them all the same 50 MHz clock source from the CPU
- Click validate
- Synthesize the design and make the bitstream
- Export HW including the bitstream
- Then you are ready for software development

Test circuit for simulation

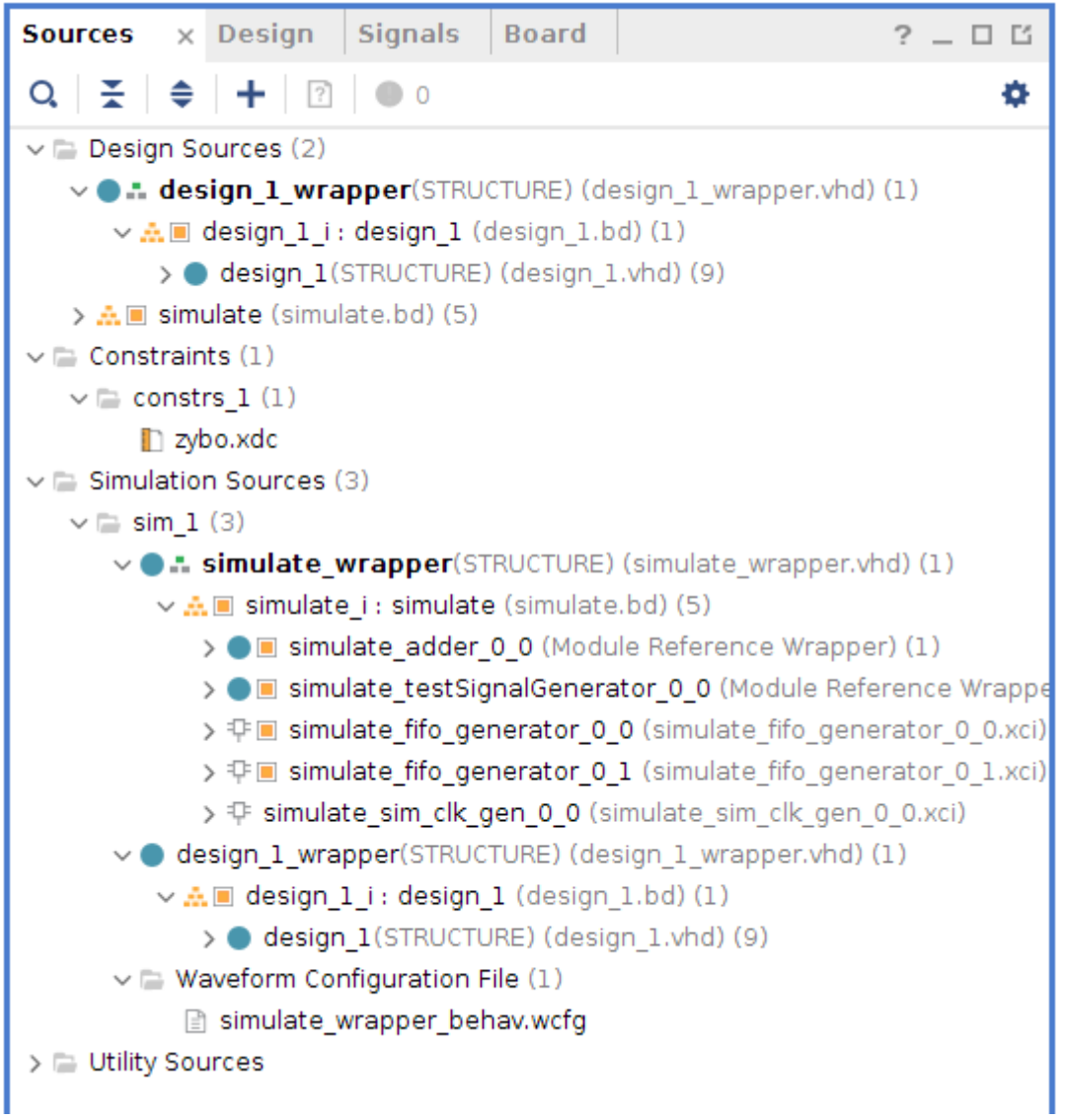
Vaasan yliopisto

- To make sure that the module (adder) will behave properly with FIFO interfaces, a test circuit only for simulation can be made
- Here test signal generator feeds and reads data to/from adder through FIFO interfaces



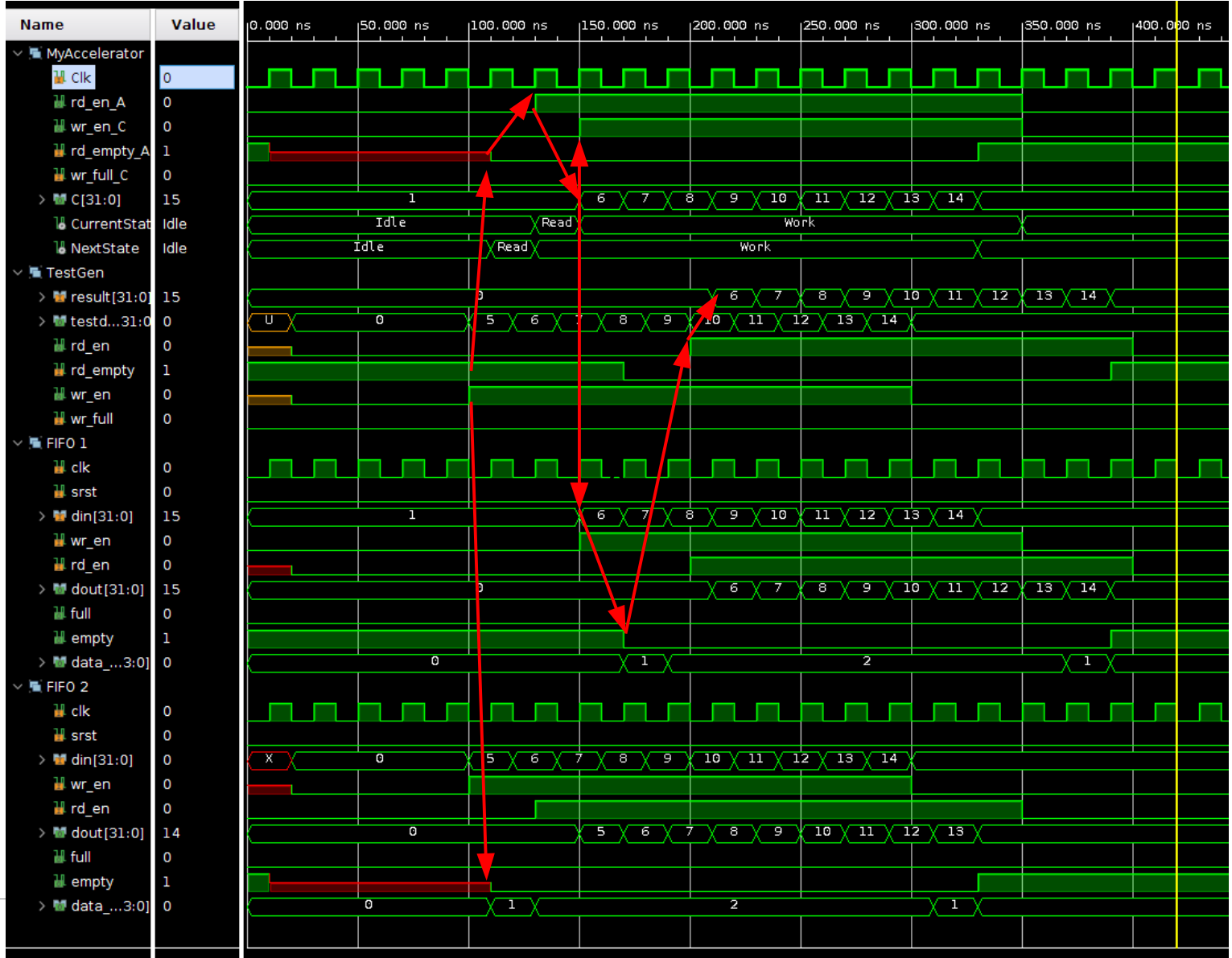
Add simulation

- Add another block design and create a HDL wrapper of it
- Right click the new source file and select Move to simulation sources
- Make it as a simulation top level
- When you now select Run simulation from the flow manager, it will run the simulation top level
- In this way you can have a separate simulation top level in your project

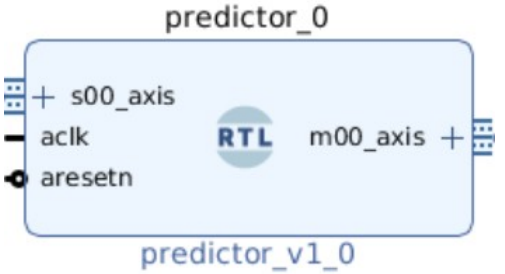


Waveform

- Here we can see that the accelarator behaves correctly with a FIFO interface
- Studying these kinds of simulations is rather hard, and I don't want to do this many times
- But it is still hundred times easier than debugging it in the HW



Implementing the predictor



Vaasan yliopisto
UNIVERSITY OF VAASA

```
entity predictor is
port (
  aclk    : in std_logic;
  aresetn    : in std_logic;

  -- Ports of Axi Slave Bus Interface S00_AXIS
  s00_axis_tready    : out std_logic;
  s00_axis_tdata    : in std_logic_vector(32-1 downto 0);
  --s00_axis_tstrb    : in std_logic_vector((C_S00_AXIS_TDA
  s00_axis_tlast    : in std_logic;
  s00_axis_tvalid    : in std_logic;

  -- Ports of Axi Master Bus Interface M00_AXIS
  m00_axis_tvalid    : out std_logic;
  m00_axis_tdata    : out std_logic_vector(32-1 downto 0);
  --m00_axis_tstrb    : out std_logic_vector((C_M00_AXIS_TDA
  m00_axis_tlast    : out std_logic;
  m00_axis_tready    : in std_logic
  );
end predictor;
```

- Clock and reset signals
- AXI Stream slave bus for data input
- AXI Stream master bus for data output

State machine for AXI Stream protocol

Vaasan yliopisto
UNIVERSITY OF VAASA

- If the system can perform it's task in one clock cycle, only two states are necessary. Here they are IDLE and WORK
- The system is in WORK state when data is received from the slave bus and our slave in the master bus is ready to receive more data

```
process(CurrentState, s00_axis_tvalid, m00_axis_tready)
begin
  s00_axis_tready<=m00_axis_tready;
  case (CurrentState) is
      when IDLE     =>
        if (S00_AXIS_TVALID = '1') and (m00_axis_tready='1') then
          NextState <= WORK;
        else
          NextState <= IDLE;
        end if;
      when WORK     =>
        if (S00_AXIS_TVALID = '1') and (m00_axis_tready='1') then
          NextState <= WORK;
        else
          NextState <= IDLE;
        end if;
      end case;
end process;
```

The output

Vaasan yliopisto
UNIVERSITY OF VAASA

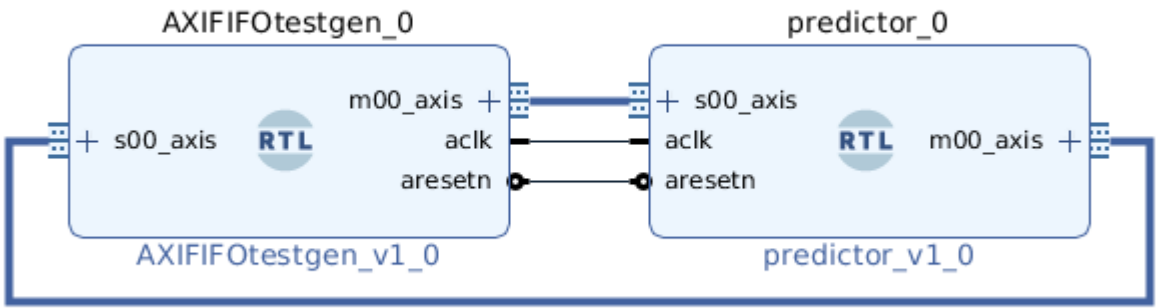
- Our example system is so simple that we can implement it in couple of concurrent assignment statetements
- We need to signal to our slave that the data written in the data bus is valid when the system is in WORK state by assignin ‘1’ in the tvalid signal
- The data is fed to our slave by just adding a 100 into the input data. Here you want to implement something more challenging.
- We just copy the possible last byte signal from our input to our output

```
m00_axis_tvalid<='1' when CurrentState=WORK else '0';
m00_axis_tdata <= std_logic_vector(signed(s00_axis_tdata) + 100)
    when CurrentState=WORK else (others => '0');
m00_axis_tlast <= s00_axis_tlast;
```

Test bench for simulation

Vaasan yliopisto
UNIVERSITY OF VAASA

- Here is the block diagram for simulating the simple predictor
- The predictor implements AXI Stream buses: Slave for data input and Master for data output
- The AXI FIFO testgen is another VHDL module which feeds in the test signals to the predictor and studies the output for testing
- Actually the predictor is now so simple, that the correctness is only studied from the simulation waveform
- Do not try to implement even this simple module without simulation! It takes time to implement a testbench, but it will take more time to debug it in the system considering all synthesize-program-test cycles.



Test bench entity

Vaasan yliopisto
UNIVERSITY OF VAASA

```
entity AXIFIFOtestgen is
port (
  aclk    : out std_logic;
  aresetn    : out std_logic;

  -- Ports of Axi Slave Bus Interface S00_AXIS
  s00_axis_tready    : out std_logic;
  s00_axis_tdata    : in std_logic_vector(32-1 downto 0);
  --s00_axis_tstrb    : in std_logic_vector((C_S00_AXIS_TDAT
  s00_axis_tlast    : in std_logic;
  s00_axis_tvalid    : in std_logic;

  -- Ports of Axi Master Bus Interface M00_AXIS
  m00_axis_tvalid    : out std_logic;
  m00_axis_tdata    : out std_logic_vector(32-1 downto 0);
  --m00_axis_tstrb    : out std_logic_vector((C_M00_AXIS_TDA
  m00_axis_tlast    : out std_logic;
  m00_axis_tready    : in std_logic
  );
end AXIFIFOtestgen;
```

- Clock and reset signals
- AXI Stream slave bus for data input
- AXI Stream master bus for data output

Test bench implementation

Vaasan yliopisto
UNIVERSITY OF VAASA

```
process(Clk)
    variable n: Integer :=0;
begin
    if (falling_edge(clk))
    then
        s00_axis_tready <='1';
        m00_axis_tdata <= (others => '0');
        m00_axis_tvalid <= '0';

        if (m00_axis_tready='1') then
            n:=n+1;
            s00_axis_tready <='1';
            if (n>=5) and (n<15) then
               m00_axis_tvalid <='1';
               m00_axis_tdata <= std_logic_vector(to_unsigned(n, 32));
            end if;
        end if;
     end if;
end process;
```

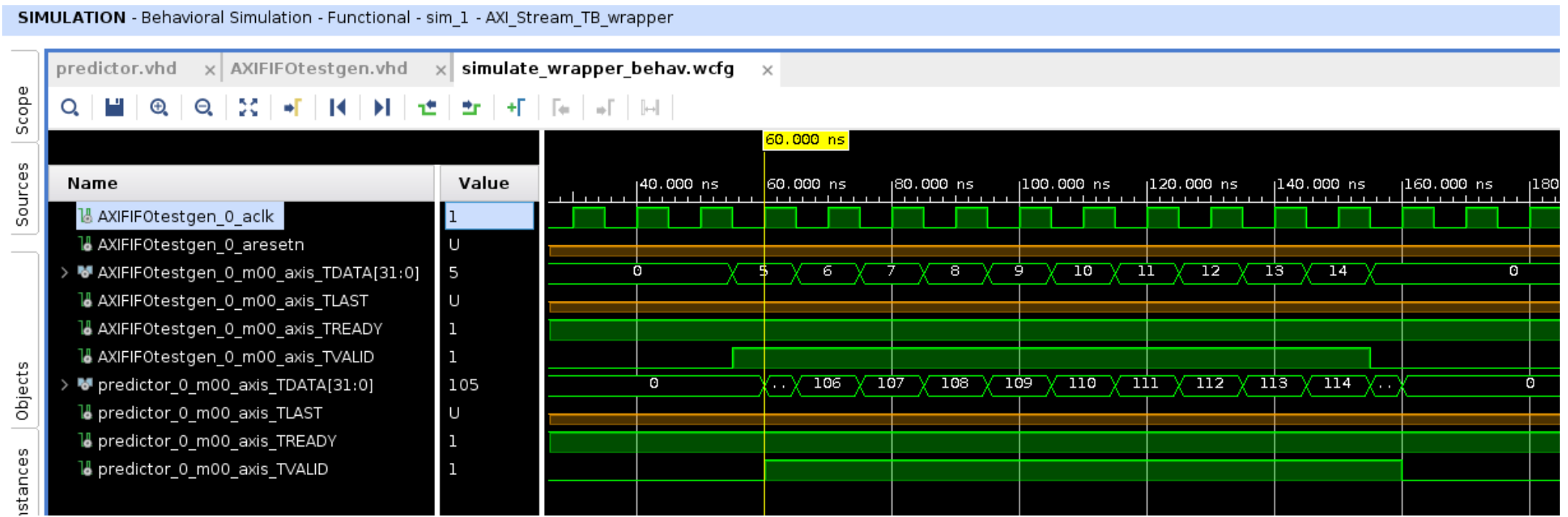
```
clkgen: process
begin
    Clk <= not Clk;
    wait for 5 ns;
end process clkgen;

aclk <= Clk;
```

- The test bench include a clock generation process and a test signal driver process
- The purpose is to send some data to the system to be tested and check that the output is correct
- It also checks that the handshake signals work correcty in the DUT

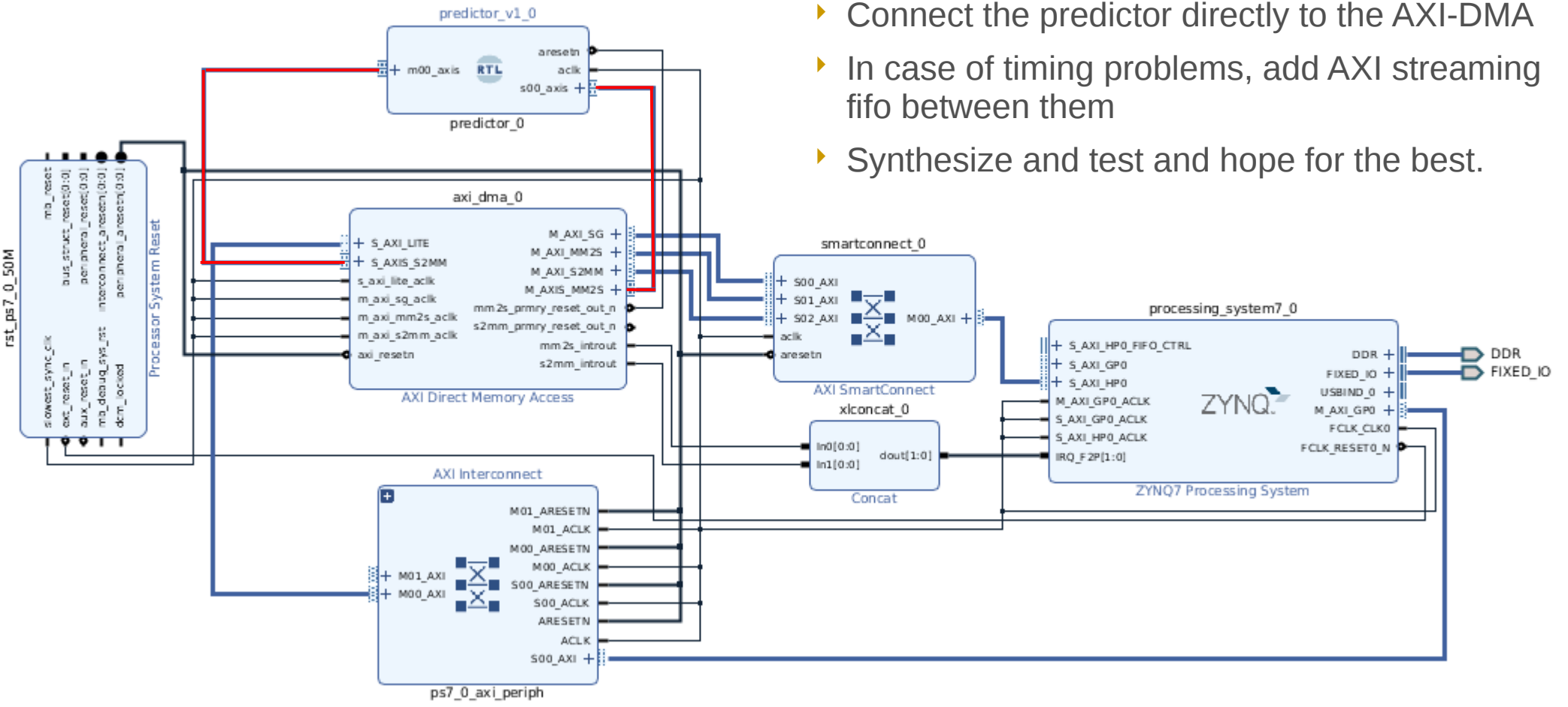
Simulation results

Vaasan yliopisto
UNIVERSITY OF VAASA

- The simulation shows how the test generator starts feeding the data at 55 ns. It is read by the detector at the next rising edge, at 60 ns
- Predictor adds 100 to the value and the result is shown immediately
- The TVALID signals show when the data in the buses can be read



Integration:



- Connect the predictor directly to the AXI-DMA
- In case of timing problems, add AXI streaming fifo between them
- Synthesize and test and hope for the best.