CHALMERS
Department of Computer Science and Engineering

2021-09-22

DAT094
Introduction to Electronic System Design
Tutorial on the Vivado Integrated Logic Analyzer (ILA)

Introduction

Every now and then it happens that you do a VHDL-design, simulate it and the simulation results looks as expected but when you download the design to hardware it doesn´t work. The reason might be that you haven´t done a simulation that is thorough enough. It´s also true that the simulation is in most cases an idealized situation that doesn´t fully apply when we come to hardware.

In this situation we must try to debug the hardware and analyze the signals going in to and coming out of the design. If there are many input sources and output connectors it can be quite messy, and you will need a lot of signal sources and measuring instruments.

Even if you have this the debugging might not succeed. To find the problem you may need access to some of the signals hidden within the design. One solution is to bring these signals out to external device pins but then you will need even more test equipment.

Xilinx have a solution to this with the Vivado Integrated Logic Analyzer (ILA).This is not any external equipment but something we put in the FPGA on test.

What we do is that we add an extra block, an IP block, to our design and we can configure this to connect probes to the internal and external signals (ports) we want to look at and the signals values at these points are transferred to a GUI in the PC using the same USB connection at we use to program the FPGA device. The signals can be single values or vectors containing a number of bits. We can have up to 1024 probes. When we do the analyze we can set a number of different trigger conditions to the measured signals.

A slight limitation is that we can only measure logical values, so we can for example not look at states in an FSM unless we code the states using a std_logic_vector. There are as we shall see ways to do this.

Another restriction is that we can only connect vectors to the ILA. This means that if we have signals that are single std_logic bits we must put these bits in vectors with the size of one bit.

We must also remember that normally Vivado optimizes our design so some of the internal signals might have been optimized away. If any of these signals is connected to an ILA probe it can´t be optimized away and the design under test will not be fully optimized but the design really doesn´t have to be optimized when we debug it. When we have sorted out the problems, we can remove the ILA and let Vivado do a full optimization Which hopefully will give the same result.

As you will see much of the work here involves preparing the design for connection to the ILA. Running the test is less of a problem.

Example design

SPI_state

Instead of giving a general presentation we will introduce the ILA by going through an example. Let´s take SPI_state that we used in lab assignment 1. You find the code in *Appendix 1*.

We are now going to put this into hardware. To do this we need to connect some other parts to our SPI_state. This means that we need a top-level design, a *wrapper*, where we can connect the parts together.

In hardware we can only look at ports or signals that are at the top level of the design. In this case we would like to see the transitions of the signals next_state_signal and state_signal but these are hidden within SPI_state so we add two extra output ports to this design, and we use these to bring the requested signals up to the top level. These extra ports are added in the code in *Appendix 1*, highlighted in green.

Now what more do we need to get a design that functions in hardware?

System clock

We need a clock signal but there is a system clocks supplied on the board we will be using, the Nexys4 DDR so this is done.

SPI clock

We will also need the SPI_clock and the SPI_clock_anable signals. These signals are generated from the system clock and we add a module, `SPI_clock`, that does this. You can find the code in *Appendix 2*.

State conversion

We need one more piece of code. We would like to see our state signals but for the moment these are only symbolic names in a type specification, and we can only have binary signals in hardware, so we need to turn the symbolic names into binary vectors. The design has 19 states, so we need a five-bit vector to represent the states.

We do this using a function that we can use for both the next_state_signal and the state_signal. The function is outlined in a package. The old type_package with the type

declaration of the states have been augmented with this function and the package is renamed *type_state_package*. You can find the package in *Appendix 3*.

Top level design, wrapper

Now it´s time to tie the parts together in a top-level design, a so-called wrapper. You can find the code in *Appendix* .

We will now discuss the ports and signals that are needed to connect it all together. There are also a part of the wrapper that implements the ILA. In *Appendix 5* this part s are added to the code in *Appendix 4*.

We have selected to let the ILA monitor the signals

0 - Reset – from button

1 - Send – from button

2 - SCLK – the SPI clock

3 - SCLK_enable - SPI_clock_enable

4 - CS – Chip select

5 - SDI – serial output data

6 - DATA_in – set by switches

7 – next state

8 - state

We mentioned earlier that all signals that connect to the ILA must be vectors. We can see that signals 0 – 5 are single bits so they must be placed in single bit vectors. This is marked in blue in *Appendix 4*.

The functions to convert from states to binary vectors are called at the end of the code, marked in red.

It´s finally time to put this into hardware.

Going to hardware

Start Vivado and create an RTL project. Add the design files including the constraints file. Select the board Nexys4DDR.

Creating the ILA component

When the project opens up, we should start by creating the ILA component. This is an IP block, so we click on the IP Catalog in the Flow Navigator 

.

When the IP Catalog opens up search for the ILA, *Figure 1*.

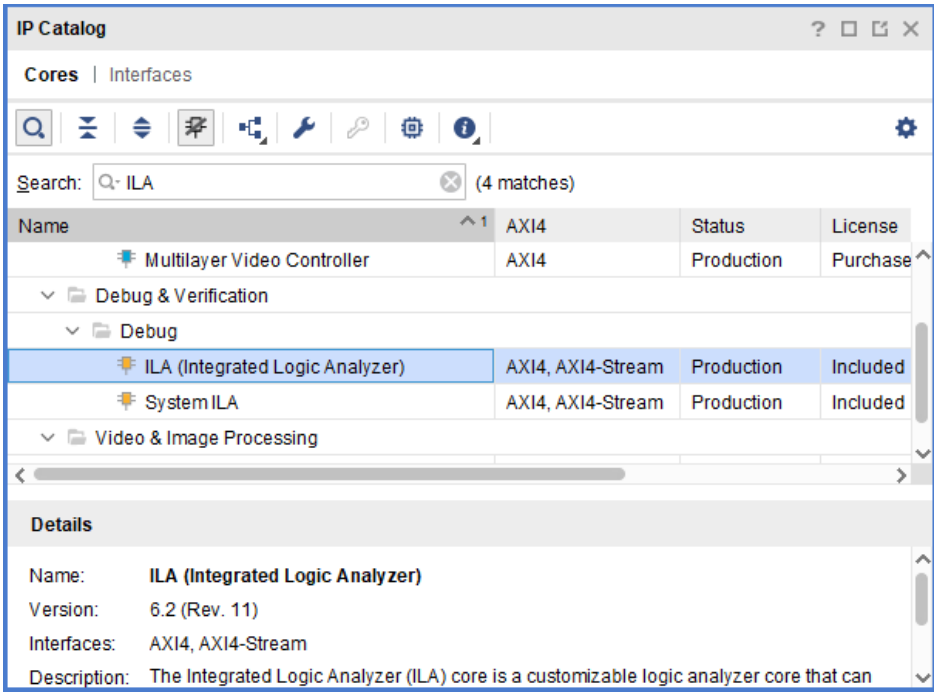

Figure 1 The IP catalog showing the ILA

Open up the ILA by double clicking on it. This opens up the window in *Figure 2*.

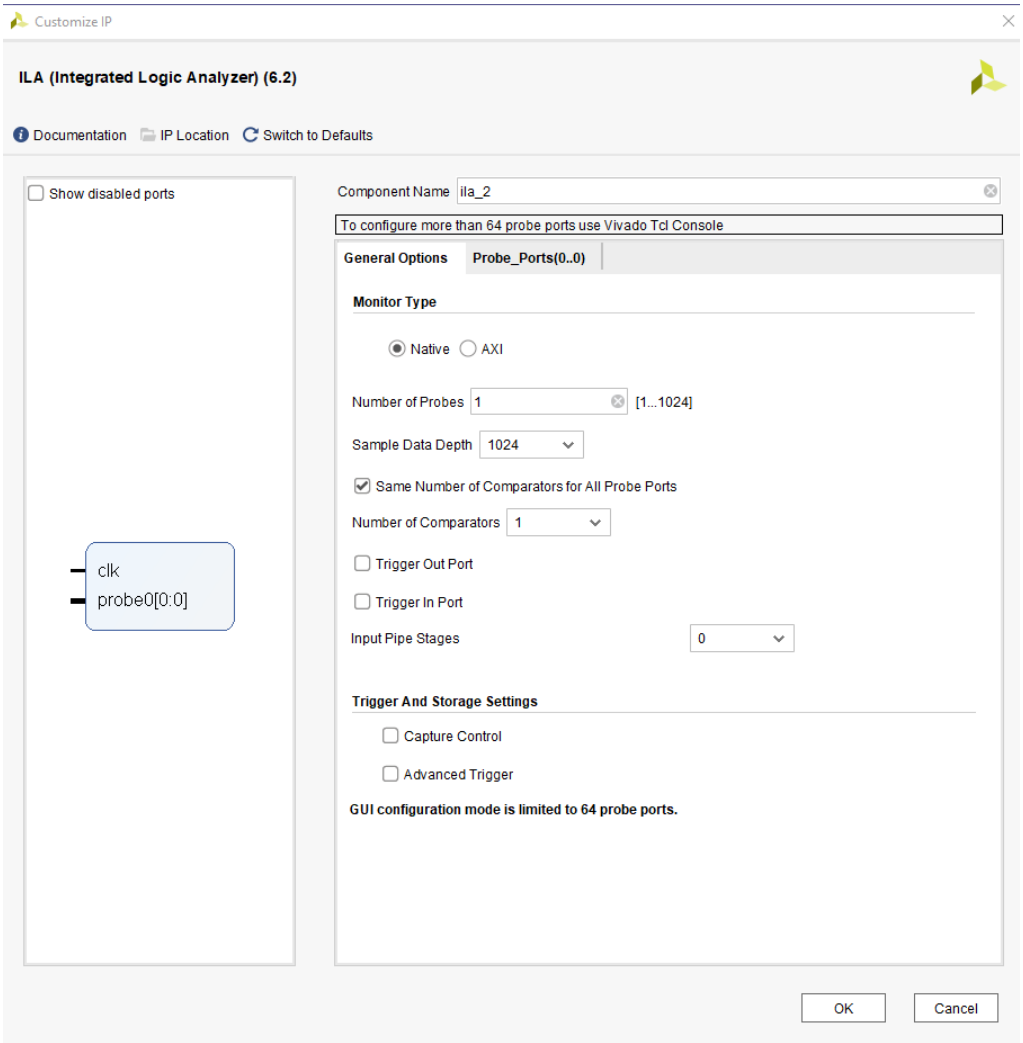

Figure 2 ILA configuration

There are a number of configurations here, but we will limit ourselves to two of them. We decided on nine probes earlier, so we set Number of probes to 9.

The monitoring is not continuous but done in sampling bursts where we can set the number of sample points, Sample Data Depth. This is by default set to 1024 but it is nice to have some more points, so we set it to 4096. We can always zoom in if we want to set fewer points. If we later want to increase the number of points generated, we need to resynthesize the ILA.

The configuration has now changed to *Figure 3*.

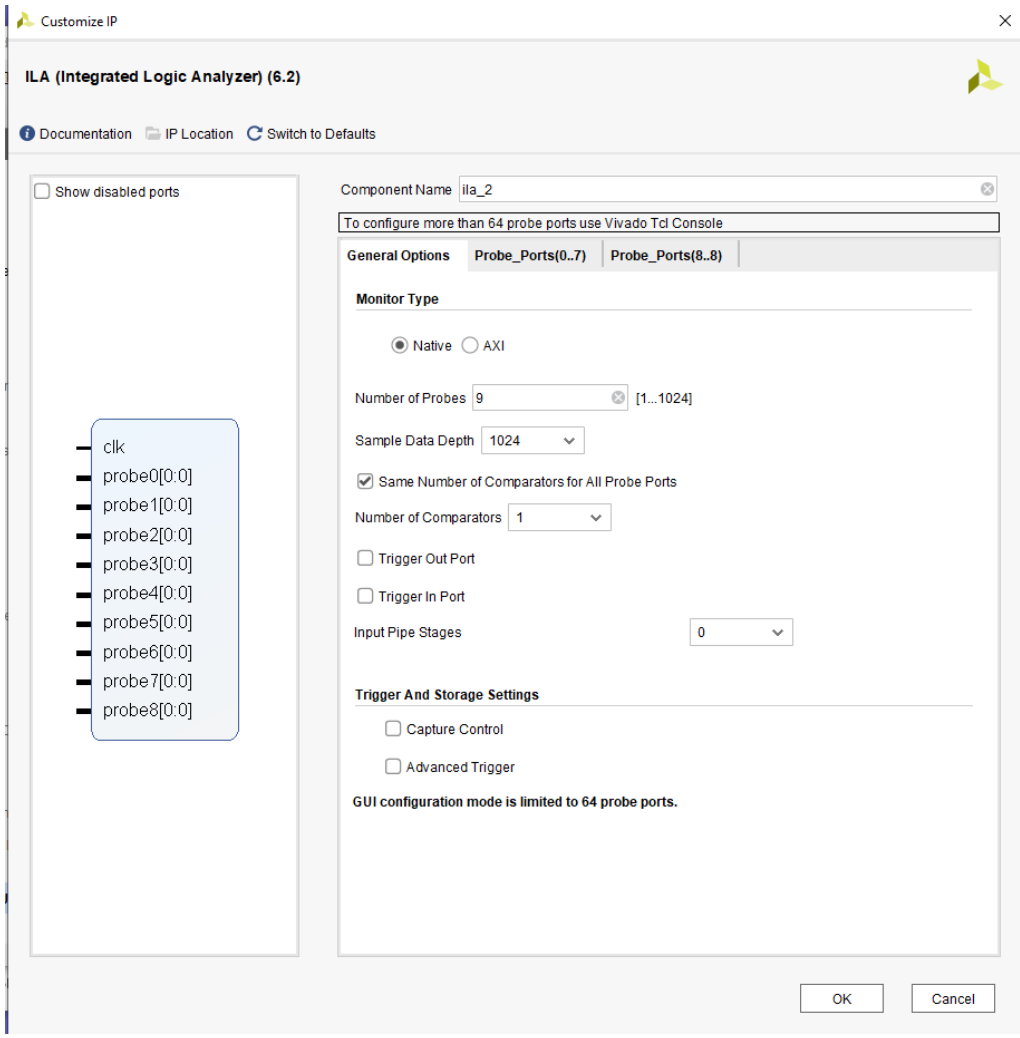

Figure 3 ILA first configuration

As we can see the symbol has been updated with the requested number of probes, the size of the probes are not set yet though. Since we have more than eight probs an extra tab has also been added.

Click on the Probe_Ports(0..7) tab and set the number of bits for each probe. Do the same for the last probe in the Probe_Ports(8..8) tab.

As a result, the ports in the symbol are updates to their correct sizes, *Figure 4*.

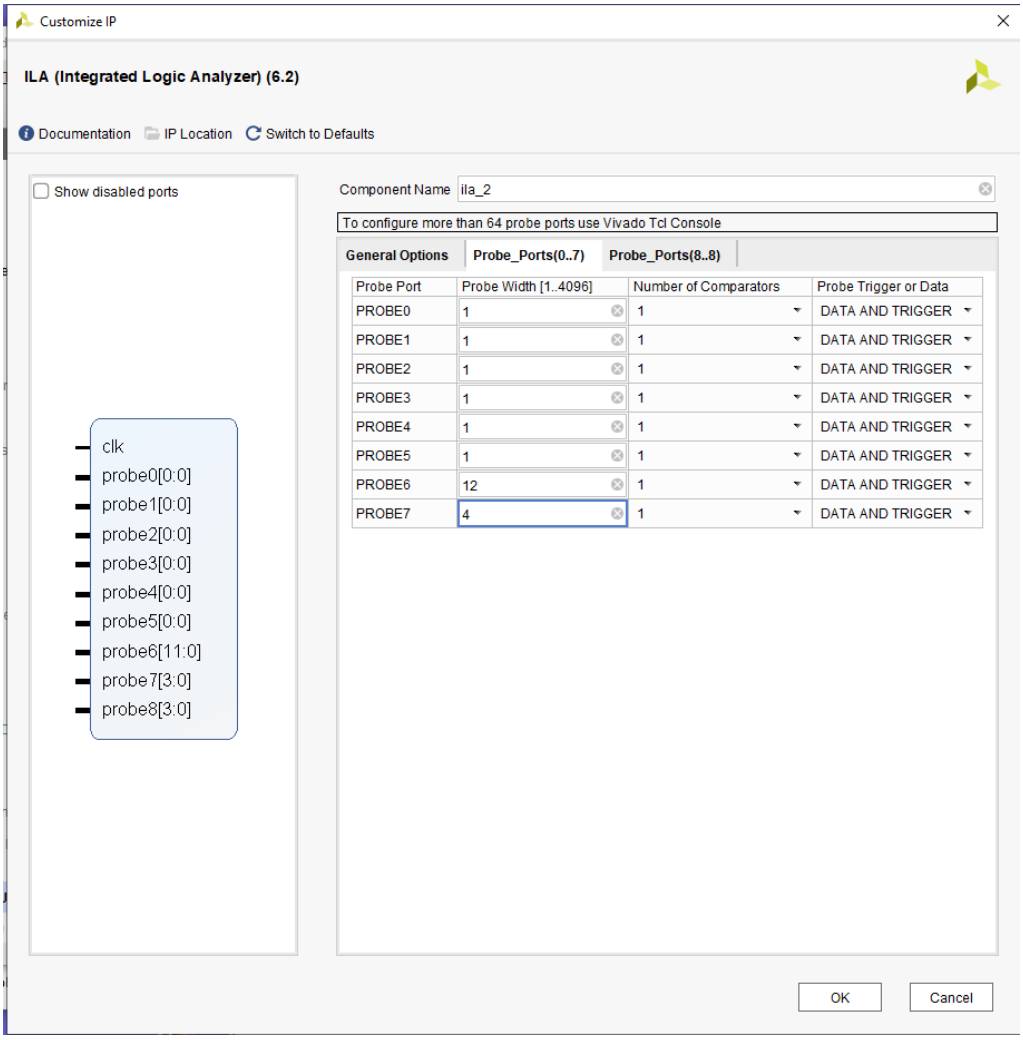

Figure 4 After ILA configuration

Finally click on OK and the IP block will be generated. This takes some time.

It starts with a window informing that output products will be generated, *Figure 5*. Leave this as it is and just click on Generate and the generation starts.

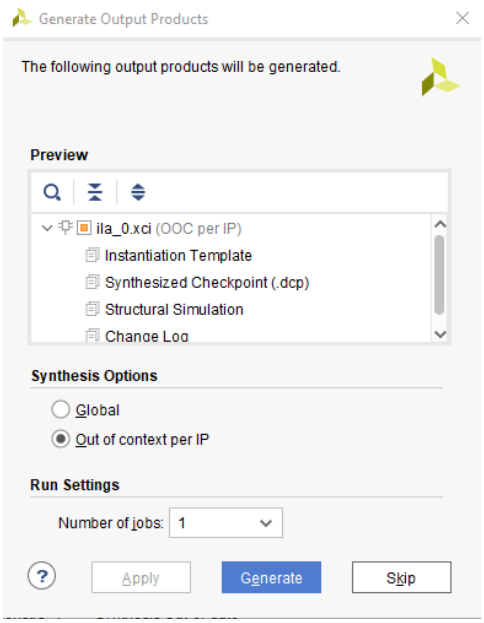

Figure 5 ILA Generate output products

When the generation is done, we get the information windows in *Figure 6*.

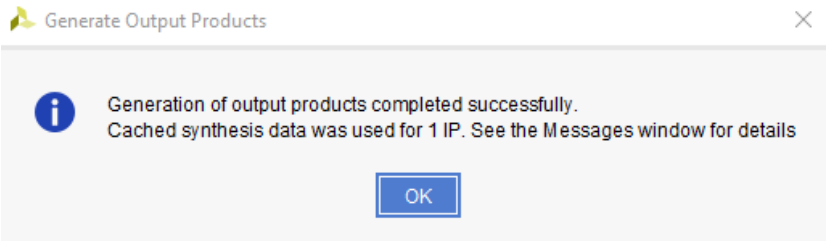

Figure 6 Finished ILA generation

Click on OK and we´re back in the Vivado GUI *Figure 7*.

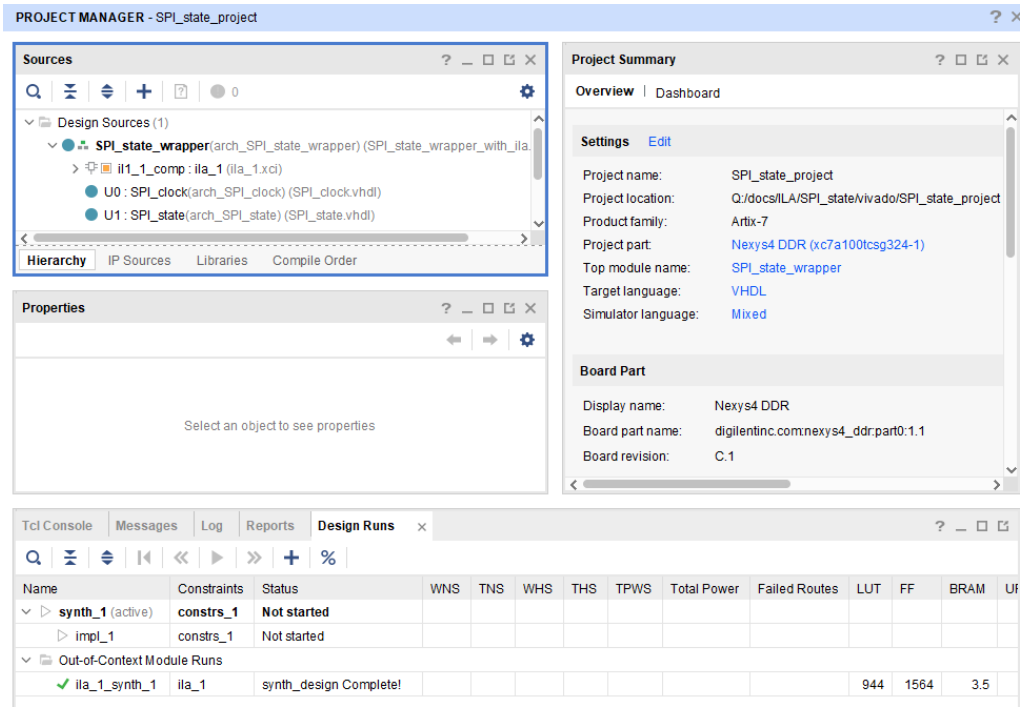

Figure 7 GUI after ILA generation

We can see that a new item has been added to the design sources. The component `il1_1_comp` is our newly generated ILA.

Editing the constraints file

Before we generate the hardware, we only need to edit the constraints file so get the connections we want, *Appendix 5*. Almost all lines are commended out by the hash sign (#), but I like to keep all the original lines since it´s hard to know how the line should look if you have erased it but want to put it back. This means that I keep all unused lines and makes copies of the lines I want to edit and keeps the original unchanged commented out by the hash sign. The active constraints are marked in blue.

The file in *Appendix 6* is somewhat edited, new lines have been inserted to make it a bit more readable.

Generate the hardware

It´s time to generate the hardware so click on the Generate Bitstream button Generate Bitstream in the Project Manager.

During generation you might get a couple critical warnings about `wr_clk_period` and the pin planning. You can ignore these warnings.

Download the design to hardware

When we have generated the hardware, and sorted out any errors, it´s time to download it to the FPGA.

Connect the Nexys4DDR board to your PC. Open the Hardware Manager and set up the connection to the board.

Program the device. In the popup window, *Figure 8*, there are now two items and not just one as before. The new entry is the ILA.

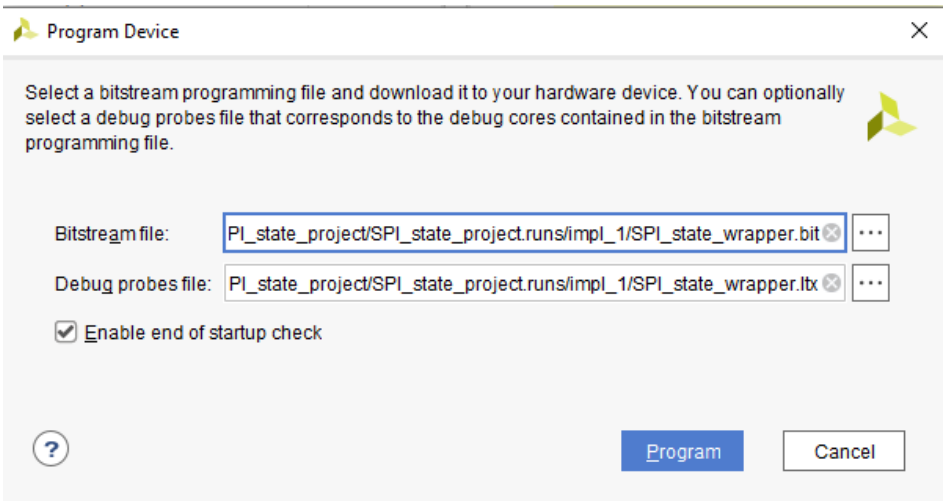

Figure 8 Popup window for programming the FPGA

When the Hardware Manager opens it will contain a new window, a waveform window that shows the signals we have assigned to the ILA probes, *Figure 9*.

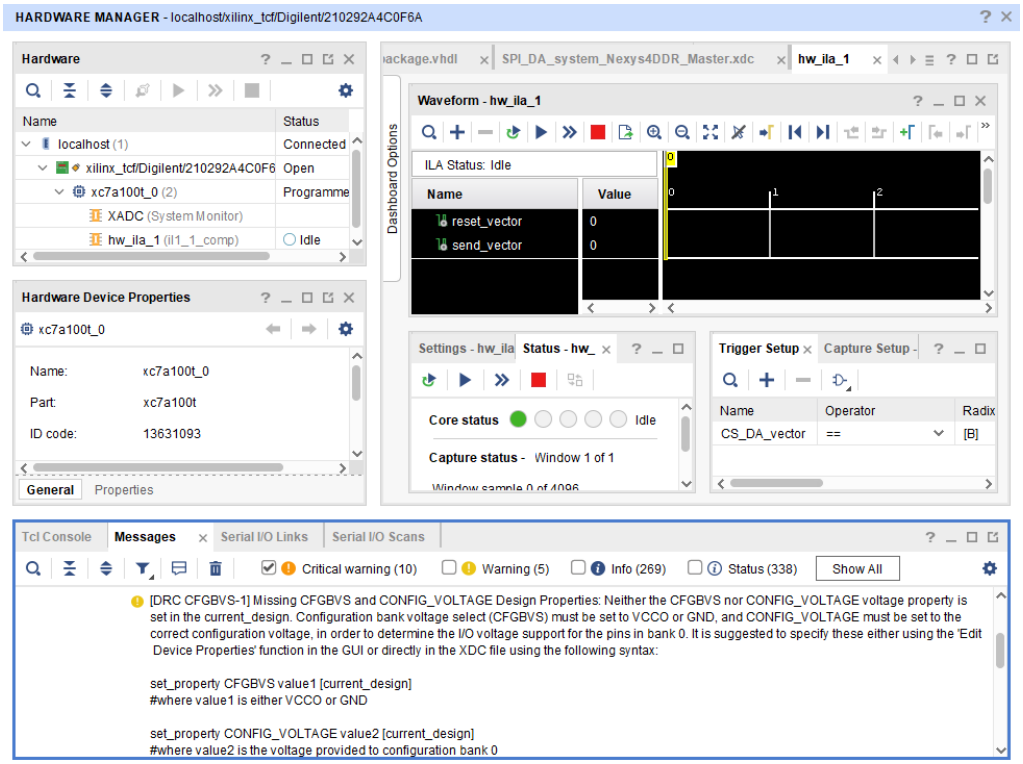

Figure 9 Hardware manager with ILA waveforms

The waveform window is too small to be useful so turn it into a floating window, *Figure 10*.

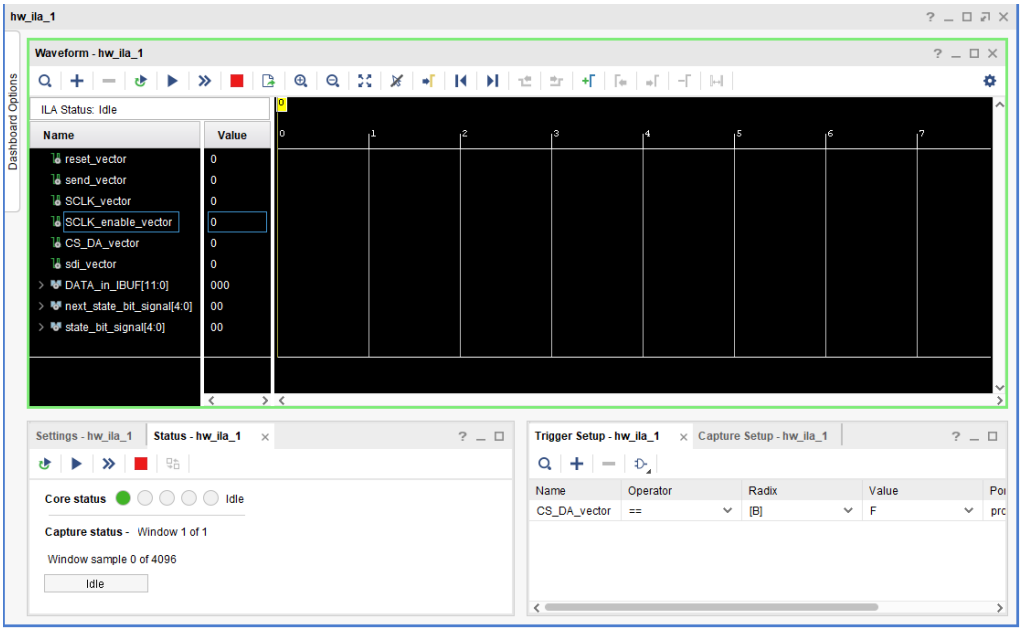

Figure 10 ILA waveforms

The waveform window is similar to the waveform window in QuestaSim. We see the signal names.

We see a column with the signal values. The displayed values are the values at the simulation time where the cursor is placed. By right clicking in the Value column, we can configure the look of the graphs. The most important settings are the signal radix and color of the graphs.

We have a frame where we can include some settings, *Figure 11*.

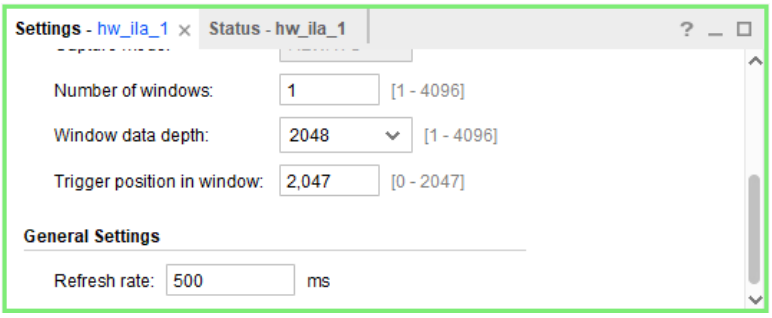

Figure 11 Waveform setting

We can set the number of displayed sampling points up to the number of points we set in the ILA configuration.

When we have set a trigger for the signal capture, we will get to this in a moment, we can the trigger position in the window. This means that we can display graphs with values both before and after the trigging point.

There are some more settings, but we leave them as they are.

In the Trigger Setup frame, *Figure 12*, we can set up the trigging conditions for the waveform capture.

If we click on the plus symbol  we will get a list of the signals, we have connected to the ILA and we can select which of these signals that should be the bases for our trigger conditions, and we add them to the list. We can take away a signal my activating it and click on the minus sign  .

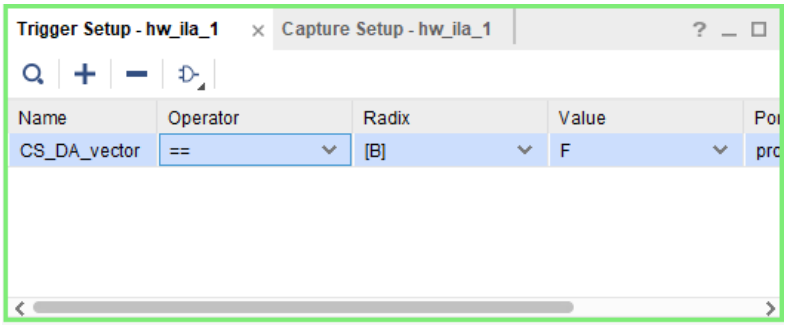

Figure 12 Trigger settings

In *Figure 10* the signal `CS_out_vector` is added. This is a good choice for a trigger signal here since a new SPI transmission starts with the chip select signal, CS, going low.

This trigging condition is set in the Value column where we can set the condition for the trigging to occur. The most used conditions are on a raising signal, 0- to-1 transition, or a falling signal, 1-to-0 transition.

To start the waveform capture we click on the  button at the top of the window. This will give one signal capture. If we first clock on  and then on  we will get a repeating capture that repeats with the rate set by the Refresh rate condition in the Settings frame. It doesn´t give a new sampling after this time but the ILA will be ready to be triggered again after this time.

To cancel a trigger session, we click on the  button.

Let´s put our design to test and look at the waveforms, *Figure 13*. Here we have set the capture conditions for one single capture with 4096 sampling points and the trigger position is in the middle of the graph, 2048 points. The trigger condition is a falling edge (F) on the CS signal.

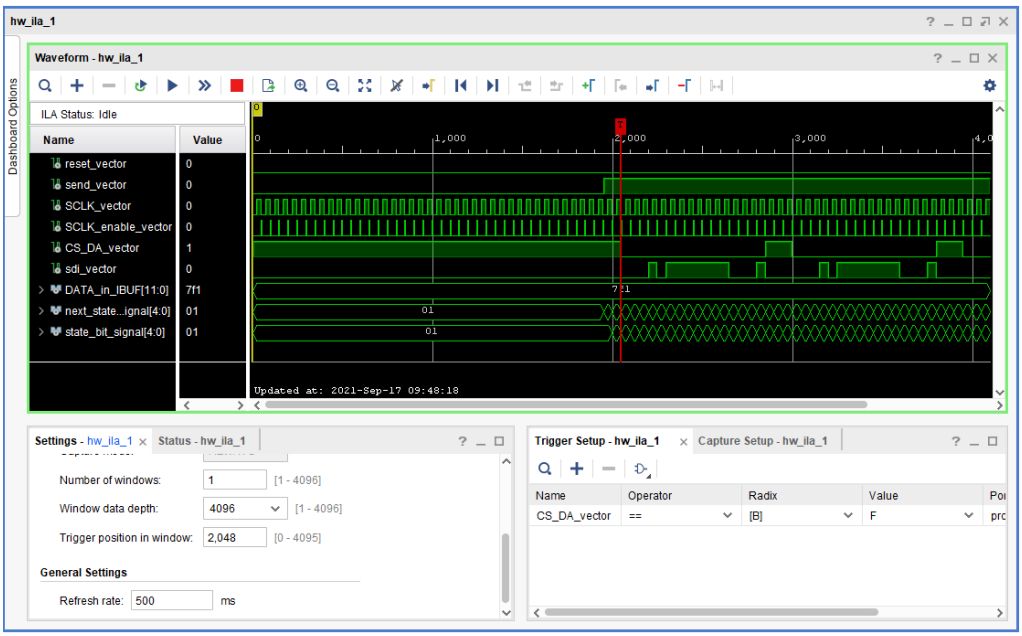

Figure 13 Waveform capture with trigging on falling edge on CS

Final comments

The ILA is easy to use but as we have seen it takes some work to set up our design for connection to the ILA. It also takes some practice to get used to the process.

When we discover an error in our design, we need to correct that in the code and recompile the sources. If we don´t change the number of ILA probes or change the number of bits in any probe we don´t have to regenerate the ILA. If we do one of these things, we must regenerate the ILA before we compile the code.

We have seen that Vivado is not that fast in bit file generation so it´s good to do all needed changes before you recompile.

Appendix 1 SPI_state

```
----------------------------------------
-- SPI_state.vhdl                      --
-- state version of SPI configuration --
-- of a MicroChip MCP4822 DAC          --
-- Sven Knutsson                       --
----------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE WORK.type_package.ALL;

ENTITY SPI_state IS
   PORT (reset:IN STD_LOGIC;
         clk:IN STD_LOGIC;
         start:IN STD_LOGIC;
         SPI_clk_enable:IN STD_LOGIC;
         A_B:IN STD_LOGIC;
         GA:IN STD_LOGIC;
         DATA_in:IN STD_LOGIC_VECTOR(11 DOWNTO 0);
         state:OUT state_type;
         next_state:OUT state_type;
         CS:OUT STD_LOGIC;
         SDI:OUT STD_LOGIC);
END SPI_state;

ARCHITECTURE arch_SPI_state OF SPI_state IS
   CONSTANT SHDN:STD_LOGIC:='1';
   SIGNAL state_signal:state_type;
   SIGNAL next_state_signal:state_type;

BEGIN

   state_transition_proc:
   PROCESS(reset,clk)
   BEGIN
      IF rising_edge(clk) THEN
         IF (reset='1') THEN
            state_signal<=idle_state;
         ELSIF (SPI_clk_enable = '1') THEN
            state_signal<=next_state_signal;
         ELSE
            state_signal <= state_signal;
         END IF;
      END IF;
```

```
END PROCESS state_transition_proc;

state_flow_proc:
PROCESS(state_signal,start)
BEGIN
   CASE state_signal IS
      WHEN idle_state =>
         IF (start = '1') THEN
            next_state_signal <= start_state;
         ELSE
            next_state_signal <= idle_state;
         END IF;
      WHEN start_state =>
         next_state_signal <= A_B_state;
      WHEN A_B_state =>
         next_state_signal <= zero_state;
      WHEN zero_state =>
         next_state_signal <= GA_state;
      WHEN GA_state =>
         next_state_signal <= SHDN_state;
      WHEN SHDN_state =>
         next_state_signal <= data_state_11;
      WHEN data_state_11 =>
         next_state_signal <= data_state_10;
      WHEN data_state_10 =>
         next_state_signal <= data_state_9;
      WHEN data_state_9 =>
         next_state_signal <= data_state_8;
      WHEN data_state_8 =>
          next_state_signal <= data_state_7;
       WHEN data_state_7 =>
          next_state_signal <= data_state_6;
       WHEN data_state_6 =>
         next_state_signal <= data_state_5;
      WHEN data_state_5 =>
          next_state_signal <= data_state_4;
       WHEN data_state_4 =>
          next_state_signal <= data_state_3;
       WHEN data_state_3 =>
         next_state_signal <= data_state_2;
       WHEN data_state_2 =>
         next_state_signal <= data_state_1;
      WHEN data_state_1 =>
          next_state_signal <= data_state_0;
       WHEN data_state_0 =>
          next_state_signal <= end_state;
      WHEN end_state =>
         next_state_signal <= idle_state;
```

```
   END CASE;
END PROCESS state_flow_proc;

assignment_proc:
PROCESS(state_signal,DATA_in,A_B,GA)
BEGIN
   SDI <= '0';
   CS <= '1';
   CASE state_signal IS
      WHEN idle_state =>
         SDI <= '0';
      WHEN start_state =>
         SDI <= '0';
      WHEN A_B_state =>
         CS <= '0';
         SDI <= A_B;
      WHEN zero_state =>
         CS <= '0';
         SDI <= '0';
      WHEN GA_state =>
         CS <= '0';
         SDI <= GA;
      WHEN SHDN_state =>
         CS <= '0';
         SDI <= SHDN;
      WHEN data_state_11 =>
         CS <= '0';
         SDI <= DATA_in(11);
      WHEN data_state_10 =>
         CS <= '0';
         SDI <= DATA_in(10);
      WHEN data_state_9 =>
         CS <= '0';
         SDI <= DATA_in(9);
      WHEN data_state_8 =>
         CS <= '0';
         SDI <= DATA_in(8);
      WHEN data_state_7 =>
         CS <= '0';
         SDI <= DATA_in(7);
      WHEN data_state_6 =>
         CS <= '0';
         SDI <= DATA_in(6);
      WHEN data_state_5 =>
         CS <= '0';
         SDI <= DATA_in(5);
      WHEN data_state_4 =>
         CS <= '0';
```

```
            SDI <= DATA_in(4);
         WHEN data_state_3 =>
            CS <= '0';
            SDI <= DATA_in(3);
         WHEN data_state_2 =>
            CS <= '0';
            SDI <= DATA_in(2);
         WHEN data_state_1 =>
            CS <= '0';
            SDI <= DATA_in(1);
         WHEN data_state_0 =>
            CS <= '0';
            SDI <= DATA_in(0);
         WHEN end_state =>
            CS <= '1';
      END CASE;
   END PROCESS assignment_proc;

END arch_SPI_state;
```

Appendix 2 SPI_clock

```
-----------------------------------------
-- SPI_clock.vhdl                       --
-- creation of a 2 MHz SPI_clock signal --
-- and SPI_clock enable signal          --
-- Sven Knutsson                        --
-----------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY SPI_clock is
    PORT(reset:IN STD_LOGIC;
          clk:IN STD_LOGIC;
          start_high:IN STD_LOGIC;
         SCLK:OUT STD_LOGIC;
          SCLK_enable:OUT STD_LOGIC);
END SPI_clock;

ARCHITECTURE arch_SPI_clock OF SPI_clock IS
--SPI clock frequency 100 MHz/25=2 MHz
CONSTANT PERIOD_constant:INTEGER:=50;
SIGNAL serial_count:INTEGER RANGE 0 TO PERIOD_constant;
SIGNAL SCLK_signal:STD_LOGIC;

BEGIN
   serialclock:
   PROCESS(reset,clk)
   BEGIN
      IF RISING_EDGE(clk) THEN
         IF (reset='1') THEN
            serial_count<=0;
            SCLK_enable<='0';
            SCLK_signal<='0';
         ELSE
            IF (serial_count=0) THEN
               SCLK_enable<='1';
               SCLK_signal<='0';
            ELSIF (serial_count=PERIOD_constant/2) THEN
               SCLK_enable<='0';
               SCLK_signal<='1';
            ELSE
                SCLK_enable<='0';
            END IF;
            serial_count<=serial_count+1;
            IF (serial_count=PERIOD_constant-1) THEN
```

```
                serial_count<=0;
            END IF;
         END IF;
      END IF;
   END PROCESS serialclock;

   SCLK<=SCLK_signal XOR start_high;

END arch_SPI_clock;
```

Appendix 3 type_state_package

```
-----------------------------
-- type_state_package.vhdl --
-- state type and          --
-- state conversion        --
-- SPI_state               --
-- Sven Knutsson           --
-----------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE type_state_package IS
   TYPE state_type IS (idle_state,start_state,A_B_state,
                       zero_state,GA_state,SHDN_state,
                       data_state_0,data_state_1,data_state_2,
                       data_state_3,data_state_4,data_state_5,
                       data_state_6,data_state_7,data_state_8,

data_state_9,data_state_10,data_state_11,
                       end_state);

   FUNCTION state_bit(state:IN state_type)
      RETURN STD_LOGIC_VECTOR;

END PACKAGE type_state_package;

PACKAGE BODY type_state_package IS

   FUNCTION state_bit(state:IN state_type)
      RETURN STD_LOGIC_VECTOR IS
   BEGIN
      CASE state IS
               WHEN idle_state =>
                  RETURN "00001";
               WHEN start_state =>
                  RETURN "00010";
               WHEN A_B_state =>
                  RETURN "00011";
               WHEN zero_state =>
                  RETURN "00100";
                WHEN GA_state =>
                  RETURN "00101";
               WHEN SHDN_state =>
                  RETURN "00110";
               WHEN data_state_11 =>
                  RETURN "00111";
```

```
                WHEN data_state_10 =>
                   RETURN "01000";
                WHEN data_state_9 =>
                   RETURN "01001";
                WHEN data_state_8 =>
                   RETURN "01010";
                WHEN data_state_7 =>
                   RETURN "01011";
                WHEN data_state_6 =>
                   RETURN "01100";
                WHEN data_state_5 =>
                   RETURN "01101";
                WHEN data_state_4 =>
                   RETURN "01110";
                WHEN data_state_3 =>
                   RETURN "01111";
                WHEN data_state_2 =>
                   RETURN "10000";
                WHEN data_state_1 =>
                   RETURN "10001";
                WHEN data_state_0 =>
                   RETURN "10010";
                WHEN end_state =>
                   RETURN "10011";
      END CASE;
   END;
END PACKAGE BODY type_state_package;
```

Appendix 4 SPI_state_wrapper

```
----------------------------
-- SPI_state_wrapper.vhdl --
-- top level design for   --
-- the SPI_state design   --
-- Sven Knutsson          --
----------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE WORK.type_state_package.ALL;

ENTITY SPI_state_wrapper is
    PORT(reset:IN STD_LOGIC;
         clk:IN STD_LOGIC;
         send:IN STD_LOGIC;
         start:OUT STD_LOGIC;
         DA_channel:IN STD_LOGIC;
         DA_gain:IN STD_LOGIC;
         DATA_in:IN STD_LOGIC_VECTOR(11 DOWNTO 0);
         SDI_out:OUT STD_LOGIC;
         CS_out:OUT STD_LOGIC;
         state_bits:OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
         SCLK_out:OUT STD_LOGIC;
         LDAC_out:OUT STD_LOGIC);
END SPI_state_wrapper;

ARCHITECTURE arch_SPI_state_wrapper OF SPI_state_wrapper IS

SIGNAL DATA_valid_DA_signal:STD_LOGIC;
SIGNAL reset_signal:STD_LOGIC;
SIGNAL state_signal:state_type;
SIGNAL next_state_signal:state_type;
SIGNAL SDI_out_signal:STD_LOGIC;
SIGNAL SCLK_signal:STD_LOGIC;
SIGNAL SCLK_enable_signal:STD_LOGIC;
SIGNAL SCLK_enable_vector:STD_LOGIC_VECTOR(0 To 0);
SIGNAL state_bit_signal:STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL next_state_bit_signal:STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL CS_out_signal:STD_LOGIC;

COMPONENT SPI_clock is
    PORT(reset:IN STD_LOGIC;
         clk:IN STD_LOGIC;
         start_high:IN STD_LOGIC;
```

```
         SCLK:OUT STD_LOGIC;
         SCLK_enable:OUT STD_LOGIC);
END COMPONENT SPI_clock;

COMPONENT SPI_state IS
   PORT (reset:IN STD_LOGIC;
         clk:IN STD_LOGIC;
         start:IN STD_LOGIC;
         SPI_clk_enable:IN STD_LOGIC;
         A_B:IN STD_LOGIC;
         GA:IN STD_LOGIC;
         DATA_in:IN STD_LOGIC_VECTOR(11 DOWNTO 0);
         state:OUT state_type;
         next_state:OUT state_type;
         CS:OUT STD_LOGIC;
         SDI:OUT STD_LOGIC);
END COMPONENT SPI_state;

BEGIN
   start <= send;
   SDI_out <= SDI_out_signal;
   CS_out <= CS_out_signal;
   LDAC_out <= '0';
   SCLK_out <= SCLK_signal;

   SPI_clock_comp:
   COMPONENT SPI_clock
      PORT MAP(reset=>reset,
               clk=>clk,
               start_high=>'0',
               SCLK=>SCLK_signal,
               SCLK_enable=>SCLK_enable_signal);

   SPI_state_comp:
   COMPONENT SPI_state
      PORT MAP(reset=>reset,
               clk=>clk,
               start=>send,
               SPI_clk_enable=>SCLK_enable_signal,
               A_B=>DA_channel,
               GA=>DA_gain,
               DATA_in=>DATA_in,
               state=>state_signal,
               next_state=>next_state_signal,
               CS=>CS_out_signal,
               SDI=>SDI_out_signal);

   state_bit_signal <= state_bit(state_signal);
```

```
   state_bits <= state_bit_signal;
   next_state_bit_signal <= state_bit(next_state_signal);

END arch_SPI_state_wrapper;
```

Appendix 5 SPI_state_wrapper with ILA

```
-------------------------------------
-- SPI_state_wrapper_with_ila.vhdl --
-- top level design for            --
-- the SPI_state design            --
-- with connection to a ILA        --
-- Sven Knutsson                   --
-------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE WORK.type_state_package.ALL;

ENTITY SPI_state_wrapper is
    PORT(reset:IN STD_LOGIC;
         clk:IN STD_LOGIC;
         send:IN STD_LOGIC;
         start:OUT STD_LOGIC;
         DA_channel:IN STD_LOGIC;
         DA_gain:IN STD_LOGIC;
         DATA_in:IN STD_LOGIC_VECTOR(11 DOWNTO 0);
         SDI_out:OUT STD_LOGIC;
         CS_out:OUT STD_LOGIC;
         state_bits:OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
         SCLK_out:OUT STD_LOGIC;
         LDAC_out:OUT STD_LOGIC);
END SPI_state_wrapper;

ARCHITECTURE arch_SPI_state_wrapper OF SPI_state_wrapper IS

SIGNAL DATA_valid_DA_signal:STD_LOGIC;
SIGNAL reset_signal:STD_LOGIC;
SIGNAL state_signal:state_type;
SIGNAL next_state_signal:state_type;
SIGNAL SDI_out_signal:STD_LOGIC;
SIGNAL reset_vector:STD_LOGIC_VECTOR(0 TO 0);
SIGNAL send_vector:STD_LOGIC_VECTOR(0 TO 0);
SIGNAL sdi_vector:STD_LOGIC_VECTOR(0 TO 0);
SIGNAL SCLK_signal:STD_LOGIC;
SIGNAL SCLK_vector:STD_LOGIC_VECTOR(0 TO 0);
SIGNAL SCLK_enable_signal:STD_LOGIC;
SIGNAL SCLK_enable_vector:STD_LOGIC_VECTOR(0 To 0);
SIGNAL state_bit_signal:STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL next_state_bit_signal:STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL CS_out_signal:STD_LOGIC;
```

```
SIGNAL CS_out_vector:STD_LOGIC_VECTOR(0 TO 0);

  COMPONENT ila_1 is
  Port (
    clk : in STD_LOGIC;
    probe0 : in STD_LOGIC_VECTOR ( 0 to 0 );
    probe1 : in STD_LOGIC_VECTOR ( 0 to 0 );
    probe2 : in STD_LOGIC_VECTOR ( 0 to 0 );
    probe3 : in STD_LOGIC_VECTOR ( 0 to 0 );
    probe4 : in STD_LOGIC_VECTOR ( 0 to 0 );
    probe5 : in STD_LOGIC_VECTOR ( 0 to 0 );
    probe6 : in STD_LOGIC_VECTOR ( 11 downto 0 );
    probe7 : in STD_LOGIC_VECTOR ( 4 downto 0 );
    probe8 : in STD_LOGIC_VECTOR ( 4 downto 0 )
  );
   end COMPONENT ila_1;

COMPONENT SPI_clock is
    PORT(reset:IN STD_LOGIC;
         clk:IN STD_LOGIC;
         start_high:IN STD_LOGIC;
         SCLK:OUT STD_LOGIC;
         SCLK_enable:OUT STD_LOGIC);
END COMPONENT SPI_clock;

COMPONENT SPI_state IS
   PORT (reset:IN STD_LOGIC;
         clk:IN STD_LOGIC;
         start:IN STD_LOGIC;
         SPI_clk_enable:IN STD_LOGIC;
         A_B:IN STD_LOGIC;
         GA:IN STD_LOGIC;
         DATA_in:IN STD_LOGIC_VECTOR(11 DOWNTO 0);
         state:OUT state_type;
         next_state:OUT state_type;
         CS:OUT STD_LOGIC;
         SDI:OUT STD_LOGIC);
END COMPONENT SPI_state;

BEGIN
   start <= send;
   reset_vector(0) <= reset;
   send_vector(0) <= send;
   SCLK_vector(0) <= SCLK_signal;
   SCLK_enable_vector(0) <= SCLK_enable_signal;
   sdi_vector(0) <= SDI_out_signal;
   SDI_out <= SDI_out_signal;
   CS_out <= CS_out_signal;
```

```
    CS_out_vector(0) <= CS_out_signal;
    LDAC_out <= '0';
    SCLK_out <= SCLK_signal;

    il1_1_comp:
    COMPONENT ila_1
    PORT MAP (
     clk => clk,
     probe0 => reset_vector,
     probe1 => send_vector,
     probe2 => SCLK_vector,
     probe3 => SCLK_enable_vector,
     probe4 => CS_out_vector,
     probe5 => SDI_vector,
     probe6 => DATA_in,
     probe7 => next_state_bit_signal,
     probe8 => state_bit_signal);

    SPI_clock_comp:
    COMPONENT SPI_clock
       PORT MAP(reset=>reset,
                clk=>clk,
                start_high=>'0',
                SCLK=>SCLK_signal,
                SCLK_enable=>SCLK_enable_signal);

    SPI_state_comp:
    COMPONENT SPI_state
       PORT MAP(reset=>reset,
                clk=>clk,
                start=>send,
                SPI_clk_enable=>SCLK_enable_signal,
                A_B=>DA_channel,
                GA=>DA_gain,
                DATA_in=>DATA_in,
                state=>state_signal,
                next_state=>next_state_signal,
                CS=>CS_out_signal,
                SDI=>SDI_out_signal);

    state_bit_signal <= state_bit(state_signal);
    state_bits <= state_bit_signal;
    next_state_bit_signal <= state_bit(next_state_signal);

END arch_SPI_state_wrapper;
```

Appendix 6 Constraints file

```
## This file is a general .xdc for the Nexys4 DDR Rev. C
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the
top level signal names in the project

## Clock signal

#set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
                                            [get_ports   {   CLK100MHZ   }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
                                                        [get_ports { clk }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}

[get_ports {CLK100MHZ}];
#set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets reset_IBUF]

##Switches

#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }

[get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
                                                      [get_ports { reset }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
                  [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
             [get_ports { DA_channel }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
                     [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
                   [get_ports { DA_gain }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
#set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
                        [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
                        [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
              [get_ports { DATA_in[0] }]; #IO_L12N_T1_MRCC_14 Sch=DATA_in[0]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
                         [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
                                                [get_ports { DATA_in[1] }];
#IO_L7N_T1_D10_14 Sch=DATA_in[1]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
                     [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
           [get_ports { DATA_in[2] }]; #IO_L17N_T2_A13_D29_14 Sch=DATA_in[2]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
                           [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
                [get_ports { DATA_in[3] }]; #IO_L5N_T0_D07_14 Sch=DATA_in[3]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 }
```

```
                                [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN T8     IOSTANDARD LVCMOS18 }
                    [get_ports { DATA_in[4] }]; #IO_L24N_T3_34 Sch=DATA_in[4]
#set_property -dict { PACKAGE_PIN U8     IOSTANDARD LVCMOS18 }
                                    [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN U8     IOSTANDARD LVCMOS18 }
                         [get_ports { DATA_in[5] }]; #IO_25_34 Sch=DATA_in[5]
#set_property -dict { PACKAGE_PIN R16    IOSTANDARD LVCMOS33 }
                  [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN R16    IOSTANDARD LVCMOS33 }
         [get_ports { DATA_in[6] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=DATA_in[6]
#set_property -dict { PACKAGE_PIN T13    IOSTANDARD LVCMOS33 }
                    [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN T13    IOSTANDARD LVCMOS33 }
            [get_ports { DATA_in[7] }]; #IO_L23P_T3_A03_D19_14 Sch=DATA_in[7]
#set_property -dict { PACKAGE_PIN H6     IOSTANDARD LVCMOS33 }
                            [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN H6     IOSTANDARD LVCMOS33 }
                    [get_ports { DATA_in[8] }]; #IO_L24P_T3_35 Sch=DATA_in[8]
#set_property -dict { PACKAGE_PIN U12    IOSTANDARD LVCMOS33 }
                    [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U12    IOSTANDARD LVCMOS33 }
            [get_ports { DATA_in[9] }]; #IO_L20P_T3_A08_D24_14 Sch=DATA_in[9]
#set_property -dict { PACKAGE_PIN U11    IOSTANDARD LVCMOS33 }
               [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN U11    IOSTANDARD LVCMOS33 }
    [get_ports { DATA_in[10] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=DATA_in[10]
#set_property -dict { PACKAGE_PIN V10    IOSTANDARD LVCMOS33 }
                        [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
set_property -dict { PACKAGE_PIN V10    IOSTANDARD LVCMOS33 }
              [get_ports { DATA_in[11] }]; #IO_L21P_T3_DQS_14 Sch=DATA_in[11]

## LEDs

#set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 }
                        [get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 }
                 [get_ports { state_bits[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
#set_property -dict { PACKAGE_PIN K15    IOSTANDARD LVCMOS33 }
                        [get_ports { LED[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN K15    IOSTANDARD LVCMOS33 }
                 [get_ports { state_bits[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
#set_property -dict { PACKAGE_PIN J13    IOSTANDARD LVCMOS33 }
                        [get_ports { LED[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN J13    IOSTANDARD LVCMOS33 }
                 [get_ports { state_bits[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
#set_property -dict { PACKAGE_PIN N14    IOSTANDARD LVCMOS33 }
                         [get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN N14    IOSTANDARD LVCMOS33 }
                   [get_ports { state_bit[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18    IOSTANDARD LVCMOS33 }
                         [get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN R18    IOSTANDARD LVCMOS33 }
                  [get_ports { state_bits[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
#set_property -dict { PACKAGE_PIN V17    IOSTANDARD LVCMOS33 }
                    [get_ports { LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17    IOSTANDARD LVCMOS33 }
                    [get_ports { LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
```

```
#set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 }
                    [get_ports { LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 }
                    [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 }
                      [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 }
                  [get_ports { LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 }
           [get_ports { LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 }
                    [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
#set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 }
                  [get_ports { LED[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 }
                  [get_ports { LED[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 }
              [get_ports { LED[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 }
                [get_ports { start }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

#set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 }
                      [get_ports { LED16_B }]; #IO_L5P_T0_D06_14 Sch=led16_b
#set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 }
                     [get_ports { LED16_G }]; #IO_L10P_T1_D14_14 Sch=led16_g
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 }
                    [get_ports { LED16_R }]; #IO_L11P_T1_SRCC_14 Sch=led16_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 }
                [get_ports { LED17_B }]; #IO_L15N_T2_DQS_ADV_B_15 Sch=led17_b
#set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 }
                                [get_ports { LED17_G }]; #IO_0_14 Sch=led17_g
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 }
                    [get_ports { LED17_R }]; #IO_L11N_T1_SRCC_14 Sch=led17_r

##7 segment display

#set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 }
                           [get_ports { CA }]; #IO_L24N_T3_A00_D16_14 Sch=ca
#set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 }
                                        [get_ports { CB }]; #IO_25_14 Sch=cb
#set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 }
                                        [get_ports { CC }]; #IO_25_15 Sch=cc
#set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 }
                                [get_ports { CD }]; #IO_L17P_T2_A26_15 Sch=cd
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 }
                              [get_ports { CE }]; #IO_L13P_T2_MRCC_14 Sch=ce
#set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 }
                           [get_ports { CF }]; #IO_L19P_T3_A10_D26_14 Sch=cf
#set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 }
                                [get_ports { CG }]; #IO_L4P_T0_D04_14 Sch=cg

#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 }
                          [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp

#set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 }
                      [get_ports { AN[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
#set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 }
                      [get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
```

```
#set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 }
                     [get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
#set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 }
                        [get_ports { AN[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 }
                         [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
#set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 }
                       [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
#set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 }
                              [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
#set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 }
                     [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]

##Buttons

#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 }
           [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

#set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
                             [get_ports { BTNC }]; #IO_L9P_T1_DQS_14 Sch=btnc
#set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
                         [get_ports { resetn }]; #IO_L9P_T1_DQS_14 Sch=btnc
#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 }
                             [get_ports { BTNU }]; #IO_L4N_T0_D05_14 Sch=btnu
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 }
                         [get_ports { BTNL }]; #IO_L12P_T1_MRCC_14 Sch=btnl
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 }
                          [get_ports { BTNR }]; #IO_L10N_T1_D15_14 Sch=btnr
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 }
                       [get_ports { BTND }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 }
                       [get_ports { send }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd

##Pmod Headers

##Pmod Header JA

#set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 }
                        [get_ports { JA[1] }]; #IO_L20N_T3_A19_15 Sch=ja[1]
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 }
                     [get_ports { JA[2] }]; #IO_L21N_T3_DQS_A18_15 Sch=ja[2]
#set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33
                        [get_ports { JA[3] }]; #IO_L21P_T3_DQS_15 Sch=ja[3]
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 }
                        [get_ports { JA[4] }]; #IO_L18N_T2_A23_15 Sch=ja[4]
#set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 }
                        [get_ports { JA[7] }]; #IO_L16N_T2_A27_15 Sch=ja[7]
#set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 }
                        [get_ports { JA[8] }]; #IO_L16P_T2_A28_15 Sch=ja[8]
#set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 }
                        [get_ports { JA[9] }]; #IO_L22N_T3_A16_15 Sch=ja[9]
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 }
                       [get_ports { JA[10] }]; #IO_L22P_T3_A17_15 Sch=ja[10]

##Pmod Header JB

#set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 }
                        [get_ports { JB[1] }]; #IO_L1P_T0_AD0P_15 Sch=jb[1]
```

```
#set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 }
                        [get_ports { JB[2] }]; #IO_L14N_T2_SRCC_15 Sch=jb[2]
#set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 }
                        [get_ports { JB[3] }]; #IO_L13N_T2_MRCC_15 Sch=jb[3]
#set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 }
                         [get_ports { JB[4] }]; #IO_L15P_T2_DQS_15 Sch=jb[4]
#set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 }
                        [get_ports { JB[7] }]; #IO_L11N_T1_SRCC_15 Sch=jb[7]
#set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 }
                         [get_ports { JB[8] }]; #IO_L5P_T0_AD9P_15 Sch=jb[8]
#set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 }
                                    [get_ports { JB[9] }]; #IO_0_15 Sch=jb[9]
#set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 }
                      [get_ports { JB[10] }]; #IO_L13P_T2_MRCC_15 Sch=jb[10]

##Pmod Header JC

#set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 }
                             [get_ports { JC[1] }]; #IO_L23N_T3_35 Sch=jc[1]
set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33
                          [get_ports { SCLK_out }]; #IO_L23N_T3_35 Sch=jc[1]
#set_property -dict { PACKAGE_PIN F6    IOSTANDARD LVCMOS33 }
                        [get_ports { JC[2] }]; #IO_L19N_T3_VREF_35 Sch=jc[2]
set_property -dict { PACKAGE_PIN F6    IOSTANDARD LVCMOS33 }
                      [get_ports { LDAC_out }]; #IO_L19N_T3_VREF_35 Sch=jc[2]
#set_property -dict { PACKAGE_PIN J2    IOSTANDARD LVCMOS33 }
                             [get_ports { JC[3] }]; #IO_L22N_T3_35 Sch=jc[3]
#set_property -dict { PACKAGE_PIN G6    IOSTANDARD LVCMOS33 }
                             [get_ports { JC[4] }]; #IO_L19P_T3_35 Sch=jc[4]
#set_property -dict { PACKAGE_PIN E7    IOSTANDARD LVCMOS33 }
                              [get_ports { JC[7] }]; #IO_L6P_T0_35 Sch=jc[7]
set_property -dict { PACKAGE_PIN E7    IOSTANDARD LVCMOS33 }
                             [get_ports { CS_out }]; #IO_L6P_T0_35 Sch=jc[7]
#set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 }
                             [get_ports { JC[8] }]; #IO_L22P_T3_35 Sch=jc[8]
set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 }
                           [get_ports { SDI_out }]; #IO_L22P_T3_35 Sch=jc[8]
#set_property -dict { PACKAGE_PIN J4    IOSTANDARD LVCMOS33 }
                         [get_ports { JC[9] }]; #IO_L21P_T3_DQS_35 Sch=jc[9]
#set_property -dict { PACKAGE_PIN E6    IOSTANDARD LVCMOS33 }
                      [get_ports { JC[10] }]; #IO_L5P_T0_AD13P_35 Sch=jc[10]

##Pmod Header JD

#set_property -dict { PACKAGE_PIN H4    IOSTANDARD LVCMOS33 }
                         [get_ports { JD[1] }]; #IO_L21N_T3_DQS_35 Sch=jd[1]
#set_property -dict { PACKAGE_PIN H1    IOSTANDARD LVCMOS33 }
                             [get_ports { JD[2] }]; #IO_L17P_T2_35 Sch=jd[2]
#set_property -dict { PACKAGE_PIN G1    IOSTANDARD LVCMOS33 }
                             [get_ports { JD[3] }]; #IO_L17N_T2_35 Sch=jd[3]
#set_property -dict { PACKAGE_PIN G3    IOSTANDARD LVCMOS33 }
                             [get_ports { JD[4] }]; #IO_L20N_T3_35 Sch=jd[4]
#set_property -dict { PACKAGE_PIN H2    IOSTANDARD LVCMOS33 }
                         [get_ports { JD[7] }]; #IO_L15P_T2_DQS_35 Sch=jd[7]
#set_property -dict { PACKAGE_PIN G4    IOSTANDARD LVCMOS33 }
                             [get_ports { JD[8] }]; #IO_L20P_T3_35 Sch=jd[8]
#set_property -dict { PACKAGE_PIN G2    IOSTANDARD LVCMOS33 }
                         [get_ports { JD[9] }]; #IO_L15N_T2_DQS_35 Sch=jd[9]
```

```
#set_property -dict { PACKAGE_PIN F3    IOSTANDARD LVCMOS33 }
                      [get_ports { JD[10] }]; #IO_L13N_T2_MRCC_35 Sch=jd[10]

##Pmod Header JXADC

#set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVDS     }
                  [get_ports { XA_N[1] }]; #IO_L9N_T1_DQS_AD3N_15 Sch=xa_n[1]
#set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVDS     }
                  [get_ports { XA_P[1] }]; #IO_L9P_T1_DQS_AD3P_15 Sch=xa_p[1]
#set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVDS     }
                    [get_ports { XA_N[2] }]; #IO_L8N_T1_AD10N_15 Sch=xa_n[2]
#set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVDS     }
                    [get_ports { XA_P[2] }]; #IO_L8P_T1_AD10P_15 Sch=xa_p[2]
#set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVDS     }
                     [get_ports { XA_N[3] }]; #IO_L7N_T1_AD2N_15 Sch=xa_n[3]
#set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVDS     }
                     [get_ports { XA_P[3] }]; #IO_L7P_T1_AD2P_15 Sch=xa_p[3]
#set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVDS     }
                   [get_ports { XA_N[4] }]; #IO_L10N_T1_AD11N_15 Sch=xa_n[4]
#set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVDS     }
                   [get_ports { XA_P[4] }]; #IO_L10P_T1_AD11P_15 Sch=xa_p[4]

##VGA Connector

#set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 }
                  [get_ports { VGA_R[0] }]; #IO_L8N_T1_AD14N_35 Sch=vga_r[0]
#set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 }
                   [get_ports { VGA_R[1] }]; #IO_L7N_T1_AD6N_35 Sch=vga_r[1]
#set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 }
                   [get_ports { VGA_R[2] }]; #IO_L1N_T0_AD4N_35 Sch=vga_r[2]
#set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 }
                  [get_ports { VGA_R[3] }]; #IO_L8P_T1_AD14P_35 Sch=vga_r[3]

#set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 }
                   [get_ports { VGA_G[0] }]; #IO_L1P_T0_AD4P_35 Sch=vga_g[0]
#set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 }
                [get_ports { VGA_G[1] }]; #IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
#set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 }
                  [get_ports { VGA_G[2] }]; #IO_L2N_T0_AD12N_35 Sch=vga_g[2]
#set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 }
                [get_ports { VGA_G[3] }]; #IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]

#set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 }
                  [get_ports { VGA_B[0] }]; #IO_L2P_T0_AD12P_35 Sch=vga_b[0]
#set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 }
                       [get_ports { VGA_B[1] }]; #IO_L4N_T0_35 Sch=vga_b[1]
#set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 }
                   [get_ports { VGA_B[2] }]; #IO_L6N_T0_VREF_35 Sch=vga_b[2]
#set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 }
                       [get_ports { VGA_B[3] }]; #IO_L4P_T0_35 Sch=vga_b[3]

#set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 }
                            [get_ports { VGA_HS }]; #IO_L4P_T0_15 Sch=vga_hs
#set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 }
                   [get_ports { VGA_VS }]; #IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs

##Micro SD Connector
```

```
#set_property -dict { PACKAGE_PIN E2    IOSTANDARD LVCMOS33 }
                  [get_ports { SD_RESET }]; #IO_L14P_T2_SRCC_35 Sch=sd_reset
#set_property -dict { PACKAGE_PIN A1    IOSTANDARD LVCMOS33 }
                     [get_ports { SD_CD }]; #IO_L9N_T1_DQS_AD7N_35 Sch=sd_cd
#set_property -dict { PACKAGE_PIN B1    IOSTANDARD LVCMOS33 }
                   [get_ports { SD_SCK }]; #IO_L9P_T1_DQS_AD7P_35 Sch=sd_sck
#set_property -dict { PACKAGE_PIN C1    IOSTANDARD LVCMOS33 }
                          [get_ports { SD_CMD }]; #IO_L16N_T2_35 Sch=sd_cmd
#set_property -dict { PACKAGE_PIN C2    IOSTANDARD LVCMOS33 }
                     [get_ports { SD_DAT[0] }]; #IO_L16P_T2_35 Sch=sd_dat[0]
#set_property -dict { PACKAGE_PIN E1    IOSTANDARD LVCMOS33 }
                     [get_ports { SD_DAT[1] }]; #IO_L18N_T2_35 Sch=sd_dat[1]
#set_property -dict { PACKAGE_PIN F1    IOSTANDARD LVCMOS33 }
                     [get_ports { SD_DAT[2] }]; #IO_L18P_T2_35 Sch=sd_dat[2]
#set_property -dict { PACKAGE_PIN D2    IOSTANDARD LVCMOS33 }
                [get_ports { SD_DAT[3] }]; #IO_L14N_T2_SRCC_35 Sch=sd_dat[3]

##Accelerometer

#set_property -dict { PACKAGE_PIN E15   IOSTANDARD LVCMOS33 }
                  [get_ports { ACL_MISO }]; #IO_L11P_T1_SRCC_15 Sch=acl_miso
#set_property -dict { PACKAGE_PIN F14   IOSTANDARD LVCMOS33 }
                   [get_ports { ACL_MOSI }]; #IO_L5N_T0_AD9N_15 Sch=acl_mosi
#set_property -dict { PACKAGE_PIN F15   IOSTANDARD LVCMOS33 }
                  [get_ports { ACL_SCLK }]; #IO_L14P_T2_SRCC_15 Sch=acl_sclk
#set_property -dict { PACKAGE_PIN D15   IOSTANDARD LVCMOS33 }
                    [get_ports { ACL_CSN }]; #IO_L12P_T1_MRCC_15 Sch=acl_csn
#set_property -dict { PACKAGE_PIN B13   IOSTANDARD LVCMOS33 }
               [get_ports { ACL_INT[1] }]; #IO_L2P_T0_AD8P_15 Sch=acl_int[1]
#set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33 }
               [get_ports { ACL_INT[2] }]; #IO_L20P_T3_A20_15 Sch=acl_int[2]

##Temperature Sensor

#set_property -dict { PACKAGE_PIN C14   IOSTANDARD LVCMOS33 }
                     [get_ports { TMP_SCL }]; #IO_L1N_T0_AD0N_15 Sch=tmp_scl
#set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33 }
                    [get_ports { TMP_SDA }]; #IO_L12N_T1_MRCC_15 Sch=tmp_sda
#set_property -dict { PACKAGE_PIN D13   IOSTANDARD LVCMOS33 }
                     [get_ports { TMP_INT }]; #IO_L6N_T0_VREF_15 Sch=tmp_int
#set_property -dict { PACKAGE_PIN B14   IOSTANDARD LVCMOS33 }
                       [get_ports { TMP_CT }]; #IO_L2N_T0_AD8N_15 Sch=tmp_ct

##Omnidirectional Microphone

#set_property -dict { PACKAGE_PIN J5    IOSTANDARD LVCMOS33 }
                                  [get_ports { M_CLK }]; #IO_25_35 Sch=m_clk
#set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33 }
                          [get_ports { M_DATA }]; #IO_L24N_T3_35 Sch=m_data
#set_property -dict { PACKAGE_PIN F5    IOSTANDARD LVCMOS33 }
                              [get_ports { M_LRSEL }]; #IO_0_35 Sch=m_lrsel

##PWM Audio Amplifier

#set_property -dict { PACKAGE_PIN A11   IOSTANDARD LVCMOS33 }
                         [get_ports { AUD_PWM }]; #IO_L4N_T0_15 Sch=aud_pwm
#set_property -dict { PACKAGE_PIN D12   IOSTANDARD LVCMOS33 }
                           [get_ports { AUD_SD }]; #IO_L6P_T0_15 Sch=aud_sd
```

```
##USB-RS232 Interface

#set_property -dict { PACKAGE_PIN C4    IOSTANDARD LVCMOS33 }
             [get_ports { UART_TXD_IN }]; #IO_L7P_T1_AD6P_35 Sch=uart_txd_in
#set_property -dict { PACKAGE_PIN D4    IOSTANDARD LVCMOS33 }
          [get_ports { UART_RXD_OUT }]; #IO_L11N_T1_SRCC_35 Sch=uart_rxd_out
#set_property -dict { PACKAGE_PIN D3    IOSTANDARD LVCMOS33 }
                  [get_ports { UART_CTS }]; #IO_L12N_T1_MRCC_35 Sch=uart_cts
#set_property -dict { PACKAGE_PIN E5    IOSTANDARD LVCMOS33 }
                  [get_ports { UART_RTS }]; #IO_L5N_T0_AD13N_35 Sch=uart_rts

##USB HID (PS/2)

#set_property -dict { PACKAGE_PIN F4    IOSTANDARD LVCMOS33 }
                    [get_ports { PS2_CLK }]; #IO_L13P_T2_MRCC_35 Sch=ps2_clk
#set_property -dict { PACKAGE_PIN B2    IOSTANDARD LVCMOS33 }
                 [get_ports { PS2_DATA }]; #IO_L10N_T1_AD15N_35 Sch=ps2_data

##SMSC Ethernet PHY

#set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33 }
                    [get_ports { ETH_MDC }]; #IO_L11P_T1_SRCC_16 Sch=eth_mdc
#set_property -dict { PACKAGE_PIN A9    IOSTANDARD LVCMOS33 }
                  [get_ports { ETH_MDIO }]; #IO_L14N_T2_SRCC_16 Sch=eth_mdio
#set_property -dict { PACKAGE_PIN B3    IOSTANDARD LVCMOS33 }
                 [get_ports { ETH_RSTN }]; #IO_L10P_T1_AD15P_35 Sch=eth_rstn
#set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 }
                 [get_ports { ETH_CRSDV }]; #IO_L6N_T0_VREF_16 Sch=eth_crsdv
#set_property -dict { PACKAGE_PIN C10   IOSTANDARD LVCMOS33 }
                [get_ports { ETH_RXERR }]; #IO_L13N_T2_MRCC_16 Sch=eth_rxerr
#set_property -dict { PACKAGE_PIN C11   IOSTANDARD LVCMOS33 }
              [get_ports { ETH_RXD[0] }]; #IO_L13P_T2_MRCC_16 Sch=eth_rxd[0]
#set_property -dict { PACKAGE_PIN D10   IOSTANDARD LVCMOS33 }
                                              19N_T3_VREF_16 Sch=eth_rxd[1]
#set_property -dict { PACKAGE_PIN B9    IOSTANDARD LVCMOS33 }
                                                 1_SRCC_16 Sch=eth_txen
#set_property -dict { PACKAGE_PIN A10   IOSTANDARD LVCMOS33 }
              [get_ports { ETH_TXD[0] }]; #IO_L14P_T2_SRCC_16 Sch=eth_txd[0]
#set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 }
              [get_ports { ETH_TXD[1] }]; #IO_L12N_T1_MRCC_16 Sch=eth_txd[1]
#set_property -dict { PACKAGE_PIN D5    IOSTANDARD LVCMOS33 }
              [get_ports { ETH_REFCLK }]; #IO_L11P_T1_SRCC_35 Sch=eth_refclk
#set_property -dict { PACKAGE_PIN B8    IOSTANDARD LVCMOS33 }
                  [get_ports { ETH_INTN }]; #IO_L12P_T1_MRCC_16 Sch=eth_intn

##Quad SPI Flash

#set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 }
           [get_ports { QSPI_DQ[0] }]; #IO_L1P_T0_D00_MOSI_14 Sch=qspi_dq[0]
#set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 }
            [get_ports { QSPI_DQ[1] }]; #IO_L1N_T0_D01_DIN_14 Sch=qspi_dq[1]
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 }
                [get_ports { QSPI_DQ[2] }]; #IO_L2P_T0_D02_14 Sch=qspi_dq[2]
#set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 }
                [get_ports { QSPI_DQ[3] }]; #IO_L2N_T0_D03_14 Sch=qspi_dq[3]
#set_property -dict { PACKAGE_PIN L13   IOSTANDARD LVCMOS33 }
                  [get_ports { QSPI_CSN }]; #IO_L6P_T0_FCS_B_14 Sch=qspi_csn
```