

JOINT INSTITUTE FOR NUCLEAR RESEARCH

Dzhelepov Laboratory of Nuclear Problems

**FINAL REPORT ON THE**

**START PROGRAMME**

*Intel FPGA training. Introduction in development of FPGA based readout systems for Timepix series detectors*

**Supervisor:**

Aleksandr Lapkin

**Student:**

Viktoria Makarova, Russia  
Kazan Federal University

**Participation period:**

October 02 – November 12,

Summer Session 2022

Dubna, 2022

**Abstract**

Currently, the Laboratory of Nuclear Problems is developing a multi-energy X-ray tomograph. It differs from the usual one in that it can separate substances by changing the absorption of X-ray radiation at different energies. Multi-energy X-ray tomograph requires a detector that can determine the energy and position of each absorbed photons (single photon counting). Detectors need readout systems, that connect to computer and detector, as well as software for controlling the detector. For these purposes, it is planned to use a new model of the Timepix4 detector of the Medipix family. Therefore, it is necessary to develop a readout system for Timepix4 based on FPGAs.

**Acknowledgments**

I want to thank my supervisor Alexander Lapkin for valuable advice and continuous mentoring throughout the practice. Special thanks to the organizing committee for the opportunity to participate in the program. It was a great chance to work on an interesting project and gain unique experience in solving engineering tasks under the guidance of specialists.

**Introduction**

Currently, computed tomography is one of the most common imaging and diagnostic methods. This method is based on the fact that materials have different X-ray transmittance in different energy of radiation. The X-ray beam passes through the tissue layer in different directions. Then the transmitted radiation is detected and the information received is converted into an image. Currently, the Laboratory of Nuclear Problems is developing a multi-energy X-ray tomograph. It differs from the usual one in that it can separate substances by changing the absorption of X-ray radiation at different energies. For these purposes, it is planned to use semiconductor hybrid pixel detectors Timepix 4 of the Medipix family as a detector. The very first Medipix chip was released in the 1990s [1]. The development of Timepix4 has been launched in 2016 [2]. The main feature of Timepix is that detectors based on it allow not only to detect the presence of certain sources of radiation, but also to determine the energy of the particles emitted by them. Timepix4 has a larger pixel matrix and can handle higher data rates than other Medipix family devices [3]. It also has a higher energy and spatial resolution. This allows obtaining X-ray images, which make it possible to determine the chemical composition of samples on tomograms, study the microstructure of tissues of living organisms and determine the proportions of various substances in them. To use such detectors, it is necessary to develop a readout system. FPGA is a high-performance device, so it can be used to transfer and process large amounts of data at high speed. Thus, FPGA is a good option for implementing a readout system. For these purposes, the Quartus project design and debugging environment can be used. Therefore, the main purpose of this work is to improve skills in working with the Quartus Prime18.1 software and the Development Kit and be prepared for the next work. To improve skills FPGAs working 5 tasks were completed.

**Task 1. Quartus basic functions.**

The first step was to install the Quartus Prime 19.1 software. For this software to work correctly, the USB-Blaster II driver must have been installed. The appearance of the Programmer window when the Development Kit is connected DE 1\_SoC without the driver installed is shown in Figure 1.1.

|  |
| --- |
|  |
| Figure 1.1 |

At the same time, an unknown device was detected in the Windows Device Manager.  To install the driver, it is needed to update this device and specify the full path to the folder where the driver is located (drivers\usb-blaster-ii\x64). Then the USB-Blaster II driver will be installed. Also, it is needed to have disabled driver signature verification using the boot parameters. Having performed these actions, the Programmer window looks like in Figure 1.2.

|  |
| --- |
|  |
| Figure 1.2 |

The next step was to create a project in Quartus Prime 19.1. It was necessary to set the parameters such as project name, file name, project type and also to find the FPGA model – Cyclone V SoC 5CSEMA5F31 in the user's guide.

Then a schematic file was created and PLL was placed in it. PLL – is an automatic control system used for converting the clock signal, as well as for phase adjustment. It was necessary to setup the PLL: the input frequency was set to 50 MHz, the desired output frequency was set to 16.777408 MHz. Then this diagram was placed in a free field of the diagram. After that, a source file (Verilog HDL File) was created and the code is shown in Figure 1.3 was written in it.

|  |
| --- |
|  |
| Figure 1.3 |

The first block of code specifies the input and output ports of this circuit. The output is 10 LEDs. The Reg type, which is used in this code file, is a data object that stores the value from one procedure assignment to the next and is used only in various functions and procedure blocks. The initial block is used to initialize variables, so it is used for setting input actions, i.e., initial values.

The procedure block described with always @(posedge Clock or posedge Reset) is triggered on the positive edge of the Clock or Reset signal. Therefore, at the rising edge of the clock signal, the next value will be written to the counter registers, which is equal the previous one plus one. This system works synchronously, but with asynchronous reset, i.e., not related to the clock frequency. To implement a synchronous reset scheme, it’s necessary to set always @(posedge Clock) in this block.

Then a schematic file was created. It contains the PLL element, 2 input ports, 1 output port, including 10 pins, a counter (described above), and an inverter (Not element). After that, all the blocks were connected to each other. The resulted scheme is shown in Figure 1.4.

|  |
| --- |
|  |
| Figure 1.4 |

The next step was to work with Pin Planner. It was needed to specify the pin designations used in this diagram. The pin designations can be found in the user's manual [4]. An example is shown in Figure 1.5.

|  |
| --- |
|  |
| Figure 1.5 |

The result after entering the required data in the Location column and assigning the value of I/O Standart 3.3 V was shown in Figure 1.6. It is also worth to mention that the Reset input was connected to the "0" button.

|  |
| --- |
|  |
| Figure 1.6 |

The next step was setting time constraints. It is necessary to describe the external clock signal in the created time constraints file. The clock period was set to 20 ns. The derive\_pll\_clock command was used to automatically generate clock cycles for each PLL output. The time constraints file is shown in Figure 1.7.

|  |
| --- |
|  |
| Figure 1.7 |

Then the project compiling can be started. After successfully completing this step, the Programmer window located on the toolbar is needed to open. It was necessary to select the Development Kit – DE1\_SoC and the compiled project file. The programming can be started by clicking Start button.

The result of this program is an alternating flashing of the LEDs corresponding to the addition of 1, starting from 0 to 1023. A demonstration how the algorithm works is shown in Figure 1.8.

|  |  |
| --- | --- |
|  |  |
| Figure 1.8 | |

Also, by changing the range of values in line 10 of the code shown in Figure 1.3, you can change the frequency of flashing LEDs. By changing this range to [30:21], the LED blinking frequency increased by ~ 8 times. The corrected version of the code is shown in Figure 1.9.

|  |
| --- |
|  |
| Figure 1.9 |

The next task was to transfer the reset function to the "3" button of the Development Kit. To do this, it is necessary to change the output for Reset in Pin Planner. The result of these actions is shown in Figure 1.10.

|  |
| --- |
|  |
| Figure 1.10 |

On the next stage, the project was modeled. The University Program VWF file was created for this purpose. In the field on the left, you need to add registers, inputs and outputs. After that, a clock signal of 50 MHz (20 ns) was applied to the Clock input using Overwrite Clock. Then high logic level was applied to the Reset input using the Forcing High button. In order to run the functional simulation correctly, it is necessary to correct the full path to the previously saved modeling1.vwf file, as well as to the modeling1.vwf.vt file. This file obtained as a result of running ModelSim TestBench and Script. The full path is needed to be corrected in all lines that appear on the screen in the Simulation Settings options. Figure 1.11 shows the fields where it is needed to replace the full path to the required file. In the first line, correct the current path for vector\_source\_source to the path through the output\_files folder\_files, and for testbench\_file to the path through the modelsim folder. It is necessary to add the full path to the modeling1.vwf.vt file in the highlighted line 3. Then copy the modeling1.vwf.vt file from the modelsim folder to the qsim folder.

|  |
| --- |
|  |
| Figure 1.11 |

After that, functional modeling was started, the results of which are shown in Figure 1.12.

|  |
| --- |
|  |
| Figure 1.12 |

The state of the LEDs does not change when the clock pulse arrives. This fact can be explained because of selecting the [33:24] range of value in the code shown in Figure 1.3. Thus, the LEDs are in the "0" state, because the counter [33: 24] is in the "0" state for the set End time = 1 ms (while the first 4 digits are in the Z state). To observe the change in LED states, the range of these values can be changed, for example, to [9:0]. The result of the modified code is shown in Figure 1.13. The End time also can be increasing so the entire counter range [33:24] will change its state to "0" or "1" (but there is a limit of 100 microseconds).

|  |
| --- |
|  |
| Figure 1.13 |

Then a time analysis of the project was performed, which resulted in the value Fmax=405.02 MHz. The maximum frequency value also changing by changing the range shown in Figure 1.3. By changing the range to [30:21], the Fmax becomes equal to 374.67 MHz.

The next step was to work with Chip Planner, which displayed the distribution of the clock signal from the PLL to the project implementation blocks using the Generate Fan-In Connection, Generate Fan-Out Connection and Expand Connection buttons. The resulting signal distribution result is shown in Figure 1.14.

|  |
| --- |
|  |
| Figure 1.14 |

If the previously added PLL output is set as Region Clock by Assigment Editor, the scheme will change, as well as the value of Fmax. Figure 1.15 shows the distribution of the clock signal from PLL to project implementation blocks after previously made changes. Fmax in this case is 334.78 MHz.

|  |
| --- |
|  |
| Figure 1.15 |

**Conclusion:** as a result of this project, the Quartus Prime 19.1 software was installed, the writing code skills in the Verilog language were obtained, as well as Quartus Prime 19.1, Pin Planner and ModelSim working skill. The result of this work is an alternating flashing of the LEDs corresponding to the addition of 1, starting from 0 to 1023, and also their flashing frequency can be changed.

**Task №2. Project with LEDs, buttons and switches.**

The first step was to create a project in Quartus Prime 19.1. It was necessary to set the parameters such as project name, file name, project type and also to find the FPGA model – Cyclone V SoC 5CSEMA5F31 in the user's guide.

Then the SystemVerilog HDL file was created. The code shown in Figure 2.1 was added to this file. The current file with the code was added to the project.

|  |
| --- |
|  |
| Figure 2.1 |

The next step was to create a top-level file in the hierarchy (Verilog HDL File). The code shown in Figure 2.2 was added to this file. This code demonstrates which LEDs will light up when any of the 4 buttons are pressed. Then the current file with the code was also added to the project.

|  |  |
| --- | --- |
|  |  |
| Figure 2.2 | |

The next step after successful Analysis and Synthesis was to work with Pin Planner. It was needed to specify the pin designations used in this diagram. The pin designations can be found in the user's manual. All the pins used in this diagram are shown in Figure 2.3. After entering the required data in the Location column, the I/O Standart column was set to 3.3 V, and the Current Strength column –was set to 16 mA.

|  |
| --- |
|  |
| Figure 2.3 |

The next step was setting time constraints. It is necessary to describe the external clock signal in the created time constraints file. The clock period was set to 20 ns. The derive\_pll\_clock command was used to automatically generate clock cycles for each PLL output. The time constraints file is shown in Figure 2.4. Then the current file was added to the project.

|  |
| --- |
|  |
| Figure 2.4 |

Then the project compilation can be started. After successfully completing this step, the Programmer window located on the toolbar is needed to open. It was necessary to select the Development Kit – DE1\_SoC and the compiled project file. The final view of the Programmer window is shown in Figure 2.5. The programming can be started by clicking Start button.

|  |
| --- |
|  |
| Figure 2.5 |

The result of this scheme is shown in Figures 2.6-2.9. The value "0101101001" was set by using the switches. The status of the LEDs in Figure 2.6 corresponds to the "3" button pressed. When the button is pressed, all 10 LEDs are turned on. This case will be discussed in more details later. The status of the LEDs in Figure 2.7 corresponds to the "2" button pressed. The glow of the LEDs depended on the state of the switch, i.e., LEDs with the switch in the "1" state were turned on. The status of the LEDs in Figure 2.8 corresponds to the "1" button pressed. This state is an inversion of the previous case, i.e., LEDs with the switch in the "0" state were turned on. The status of the LEDs in Figure 2.9 corresponds to the "0" button pressed. When the button is pressed, all 10 LEDs are turned off regardless of the switch position.

|  |
| --- |
|  |
| Figure 2.6 |
|  |
| Figure 2.7 |
|  |
| Figure 2.8 |
|  |
| Figure 2.9 |

When the "3" button is pressed, the brightness of the LEDs depends on the position of the switch. Figure 2.10 shows an example where all switches correspond to the "0" position, in this case the LEDs are turned off. As the position of the switches is changed to "1", the brightness of the LEDs increases. This fact is shown in Figures 2.11-2.13. At the same time, the leftmost switch provides the highest brightness of the LEDs, and the rightmost switch provides the lowest.

|  |
| --- |
|  |
| Figure 2.10 |
|  |
| Figure 2.11 |
|  |
| Figure 2.12 |
|  |
| Figure 2.13 |

Then it was necessary to work with RTL Viewer. This is a utility that allows to see the logical implementation of a project in graphical form. To run this utility, user need to select Netlist Viewers from the Tools menu and then choose RTL Viewer. The result of running RTL Viewer is shown in Figure 2.14.

|  |
| --- |
|  |
| Figure 2.14 |

The next step was to work with Chip Planner, which displayed the distribution of the clock signal from the PLL to the project implementation blocks using the Generate Fan-In Connection, Generate Fan-Out Connection and Expand Connection buttons. The resulting signal distribution result is shown in Figure 2.15. The result obtained by using the Expand Connection button, which allows to display all the components of the selected connection, is shown in Figure 2.16.

|  |
| --- |
|  |
| Figure 2.15 |
|  |
| Figure 2.16 |

Then the SignalTap Logic Analyser, located in the Tools tab was used. It is necessary to add a clock signal (located in the right panel of the window) and tracking signals (located in the left panel). Basic OR is needed to be select in the Trigger Conditions column, and Falling Edge – for led\_driver\_driver:led\_driver\_inst\_driver\_inst|Reset. After that, it is necessary to compile the project, program the FPGA and get the data once. The set switches position and the result of pressing the KEY2 button are shown in Figure 2.17. The results obtained during these actions are shown in Figure 2.18.

|  |
| --- |
|  |
| Figure 2.17 |
|  |
| Figure 2.18 |

**Conclusion:** as a result of this project, RTL Viewer and SignalTap Logic Analyser working skill were obtained. Also, Pin Planner and Chip Planner skills were improved. The result of this work is the LEDs flashing, which depends on which button is pressed.

**Task 3. Quartus debugging tools.**

The first step was to create a project in Quartus Prime 19.1. It was necessary to set the parameters such as project name, file name, project type and also to find the FPGA model – Cyclone V SoC 5CSEMA5F31 in the user's guide.

Then a schematic file was created and PLL was placed in it. PLL – is an automatic control system used for converting the clock signal, as well as for phase adjustment. It was necessary to setup the PLL: the input frequency was set to 50 MHz, the desired output frequency was set to 16.777408 MHz. Then this diagram was placed in a free field of the diagram. After that, a code file (Verilog HDL File) was created and the code is shown in Figure 3.1 was written in it.

|  |
| --- |
|  |
| Figure 3.1 |

The first block of code specifies the input and output ports of this circuit. The output is 10 LEDs. The Reg type, which is used in this code file, is a data object that stores the value from one procedure assignment to the next and is used only in various functions and procedure blocks. The initial block is used to initialize variables, so it is used for setting input actions, i.e. initial values.

The procedure block described with always @(posedge Clock or posedge Reset) is triggered on the positive edge of the Clock or Reset signal. Therefore, at the rising edge of the clock signal, the next value will be written to the counter registers, which is equal the previous one plus one. This system works synchronously, but with asynchronous reset, i.e. not related to the clock frequency. To implement a synchronous reset scheme, it’s necessary to set always @(posedge Clock) in this block.

Then a schematic file was created. It contains the PLL element, 2 input ports, 1 output port with 10 pins, a counter (described above), and an inverter (Not element) and an OR element with 2 inputs (OR2). After that Intel FPGA In-System Sources & Probes was added under the name sys\_probe. The Probe Port Width [0..512] parameter was set to 34. Then it is necessary to add the generated sys\_probe.qip file to the project, and also place this probe on the empty space of the circuit. All the blocks were connected to each other. Then all I/Os were renamed as Clock, Reset, Led[33..24]. The bus from the Led output of the cnt module to the Probe input of the sys\_probe module should be renamed as Led[33..0], as well as the bus leading to the Led pin[33..24] as Led[33..24]. The resulted scheme is shown in Figure 3.2.

|  |
| --- |
|  |
| Figure 3.2 |

The next step was to work with Pin Planner. It was needed to specify the pin designations used in this diagram. The pin designations can be found in the user's manual. An example is shown in Figure 3.3.

|  |
| --- |
|  |
| Figure 3.3 |

The result after entering the required data in the Location column and assigning the value of I/O Standart 3.3 V was shown in Figure 3.4. It is also worth to mention that the Reset input was connected to the "0" button.

|  |
| --- |
|  |
| Figure 3.4 |

The next step was setting time constraints. It is necessary to describe the external clock signal in the created time constraints file. The clock period was set to 20 ns. The derive\_pll\_clock command was used to automatically generate clock cycles for each PLL output. The time constraints file is shown in Figure 3.5.

|  |
| --- |
|  |
| Figure 3.5 |

Then the project compilation can be started. After successfully completing this step, the Programmer window located on the toolbar is needed to open. It was necessary to select the Development Kit – DE1\_SoC and the compiled project file. The programming can be started by clicking Start button.

The result of this program is an alternating flashing of the LEDs corresponding to the addition of 1, starting from 0 to 1023. An example of how the algorithm works is shown in Figure 3.6.

|  |  |
| --- | --- |
|  |  |
| Figure 3.6 | |

The next step was to work with System Probe. It was necessary to run the In-System Sources and Probes Editor located in the Tools tab, then select the previously programmed FPGA (Hardware: DE\_SoC [USB X], Device: @2:5CSE(BA5|MA5)...). The probe polling is needed to be enable by using the Continuously Read Probe Data. The result of these actions is shown in Figure 3.7. Signals with probe[33..24] represent an addition of 1 in binary code, the remaining signals are noise.

|  |
| --- |
|  |
| Figure 3.7 |

In-System Sources and Probes Editor extends the set of verification tools and provides a dynamic debugging environment, as well as allows to control the internal signal. This way, data from all sources and samples in the project can be viewed. Continuously Read Probe Data allows to continuously sample data from the probe. The In-System Sources and Probes Editor allows to store up to 128k samples. This parameter is set in the Maximum Size line (in this project, it was chosen to be 2k).

The next step was to turn off the source probe, i.e. in the Data tab it was set to "0". After that, the probe polling was started again. The result is shown in Figure 3.8. The probe[33..0], which are project outputs, have a value of "0" while Source value is also "0". So, the Source is used to control the input data.

|  |
| --- |
|  |
| Figure 3.8 |

Unfortunately, the In-System Sources and Probes Editor debugging tool has a drawback – a non-constant frequency of polling that cannot be adjusted by the user. This disadvantage is demonstrated in Figure 3.9. It’s clearly shows how much the frequency changes on probe[33..0] in the absence of any changes in the program code and settings.

|  |
| --- |
|  |
| Figure 3.9 |

The next step was to use the SignalTap Logic Analyser located in the Tools tab. It is necessary to add a clock signal (in the Clock field in the right panel of the window) and tracking signals (cnt:inst2|Reset and cnt:inst2|counter[33..0] located in the left panel). Basic OR is needed to be select in the Trigger Conditions column, and Falling Edge – for cnt:inst2|Reset. After that, it is necessary to compile the project, program the FPGA and read the data once. The results obtained during these actions are shown in Figure 3.10.

|  |
| --- |
|  |
| Figure 3.10 |

The SignalTap logic Analyzer is a debugging tool that displays signal behavior in real time, allowing to observe the interaction between hardware and software in projects [5]. This software allows to choose which signals to capture, when signal capture starts, and how many data samples to capture.

The next step was to configure SignalTap, so that data collection occurs after the counter reaches the value 0X1234567. To do this, in the Trigger Conditions column for cnt:inst2|counter[33..0] is needed to be set the "0X1234567" value, and in the line cnt:inst2 /Reset – "Don’'t care". Also the value for Trigger Conditions must be replaced with Basic AND. The view of the Setup window in SignalTap with the changes is shown in Figure 3.11. The result of these operations is shown in Figure 3.12. This figure shows that the beginning of the reference, i.e. zero value on the scale, which located at the top, corresponds to the state of the LEDs "1001000110100010101100111" (which is the number "1234567" in the hexadecimal numeral system).

|  |
| --- |
|  |
| Figure 3.11 |
|  |
| Figure 3.12 |

**Conclusion:** As a result of this project, debugging tools In-System Sources and Probes Editor and SignalTap Logic Analyser working skills were improved. In-System Sources and Probes Editor is a simple tool for setting and managing internal signals. Unfortunately, this debugging tool is not suitable for monitoring high frequency signals. SignalTap Logic Analyzer records FPGA logic signals with the frequency of the selected clock signal and gives user the ability to monitor the signals in real time. Also, worth to mention that SignalTap Logic Analyzer consumes FPGA resources. These debugging tools help clearly demonstrate how the code works.

**Task №4. SDRAM module testing.**

The first stage was the creation of a project, which includes specifying the project name, the model of the FPGA used, etc. Then it was necessary to launch Platform Designer, located in the Tools tab. The Platform Designer window view is shown in Figure 4.1.

|  |
| --- |
|  |
| Figure 4.1 |

Then it was necessary to place the SDRAM Controller, located in the Memory Interfaces and Controllers → SDRAM → SDRAM Controller Intel FPGA IP tab, the PLL (Basic Function → Clock, PLL and Resets → PLL Intel FPGA IP) and the JTAG – Avalon bridge (Basic Functions → Bridges and Adapters → Memory Mapped → JTAG to Avalon Master Bridge). You also need to configure all the elements described above. The first step was to set the requirement parameters, which are shown in Figure 4.2, in the Memory Profile and Timing tabs for the SDRAM Controller.

|  |
| --- |
|  |
| Figure 4.2 |

Then the PLL was configured, the parameters of which are shown in Figure 4.3. The reference clock frequency was set to 50 MHz, and the number of Clock generators was set to 2: outclk0 and outclk1.

|  |
| --- |
|  |
| Figure 4.3 |

After that, by double-clicking in the Export window the necessary elements were exported – wire (named sdram), outclk1 (named sdram\_clock) and locked (named locked). The appearance of the Platform Designer window after making the changes described earlier is shown in Figure 4.4.

|  |
| --- |
|  |
| Figure 4.4 |

Then the code was generated using Generate HDL. After that, the sdram.qip file was added to the project from its folder by using the Add/Remove Files. The next step was to write the code in the previously created DE1\_SoC\_sdram.v file. The Verilog HDL File with the code is shown in Figure 4.5.

|  |
| --- |
|  |
| Figure 4.5 |

At first, input, output and inout (bidirectional) ports were set. Then the single-bit reset and locked circuits were announced. Then the example shown in Figure 4.6 was copied in the Instantiation Templates tab of the Platform Designer window for later declaring SDRAM module instances.

|  |
| --- |
|  |
| Figure 4.6 |

After that, you can proceed to creating the time constraints file, which is shown in Figure 4.7.

|  |
| --- |
|  |
| Figure 4.7 |

Clock frequencies are described using the create\_clock command. In this case, a generator with a frequency of 100 MHz is used, the source of which is the DRAM\_CLK port. A second clock frequency with a period of 20 ns was also set. To use the get\_ports function, when you select Insert Constraint, you must use the Create Timing Netlist tab after running the Timing Analyzer. Then you need to set input and output delays of 3 ns in the Synopsys Design Constraint File File(.sdc) using the set\_input\_delay and set\_output\_delay commands. The next step was to set the maximum and minimum time for input and output delays for clk\_dram, the source of which is the previously announced output and bidirectional ports. These data were obtained from the Development Kit descriptions and manuals.

After successful Analysis and Synthesis, you can start working with Pin Planner. In the user's manual, you need to specify the pin designations used in this diagram. All the pins used in this diagram are shown in Figures 4.8, 4.9 and 4.10.

|  |
| --- |
|  |
| Figure 4.8 |
|  |
| Figure 4.9 |
|  |
| Figure 4.10 |

After entering the required data in the Location column, the I/O Standart column was set to 3.3 V, and the Current Strength column –was set to 16 mA. The final view of the Pin Planner window after making all the changes is shown in Figure 4.11.

|  |
| --- |
|  |
| Figure 4.11 |

The next step was to create a Tcl Script file containing the code shown in Figure 4.12. This code is used for memory testing. Lines 4 and 5 contain the initial values of the first elements of the Fibonacci sequence. Using the while loop, we described filling of rows, in which each next number is equal to the sum of the previous two. These steps are repeated until the number exceeds 0xffffffff value (i.e., 4294967295 in decimal).

|  |
| --- |
|  |
| Figure 4.12 |

After that, you can start compiling the project. After successfully completing this step, you need to open the Programmer window located on the toolbar. You need to select the programmable board – DE1\_SoC and the compiled project file. The final view of the Programmer window is shown in Figure 4.13. After that, you can click Start button to start programming.

|  |
| --- |
|  |
| Figure 4.13 |

After finishing programming, the system console was launched (Tools → System Debugging Tools → System Console). The string "source DE1\_SoC\_sdram\_test.tcl" was inserted in the Tcl Console window. Figure 4.14 shows the result that appears on the screen after running the test.

|  |
| --- |
|  |
| Figure 4.14 |

It is also worth to mention the problems that occurred while working with the Tcl Console. The first result of the program launching was the "test is unsuccessful" message on the display, which indicates problems with memory. After that, to test the internal memory, the On Chip Memory Intel FPGA IP element (Basic Function → On Chip Memory) was added to the project in Platform Designer. The scheme of its connection is shown in Figure 4.15. Also in this figure, you can see that the start and end address values for On Chip Memory were accordingly changed to 0x04000000 and 0x04000fff in the Address Map tab.

|  |
| --- |
|  |
| Figure 4.15 |

After compiling and running the project with the changes made on the Development Kit, the lines shown in Figure 4.16 were entered in System Console. Thus, it was concluded that the internal memory of this board works without errors, since it was possible to write a number into it and then read it. But at the same time, when writing the number "2" to the address 0x0, the external memory reads it as the value 0xffffffff, which corresponds to the number "-1" in the additional code.

|  |
| --- |
|  |
| Figure 4.16 |

**Conclusions:** as a result of this project, the skills of working with Platform Designer and System Console were obtained. The SDRAM module was tested by using the code, which represents the Fibonacci sequence. Also, the operability of internal and external memory was checked.

**Task №5. Nios II module testing.**

The first stage was the creation of a project, which includes specifying the project name, the model of the FPGA used, etc. Then it was necessary to launch Platform Designer, located in the Tools tab. The Platform Designer window view is shown in Figure 5.1.

|  |
| --- |
|  |
| Figure 5.1 |

Then it was necessary to place elements such as the Nios II processor, located in the Processors and Peripherals tab, memory (Basic Function → On Chip Memory → On Chip Memory (RAM or ROM) Intel FPGA IP), System ID (Basic Functions → Simulation; Debug and Verification → Debug and Performance → System ID Peripheral Intel FPGA IP) and UART via JTAG. Also 2 parallel I/O – PIOs (Processors and Peripherals → Peripherals → PIO) should be placed. These PIOs should be renamed as switch, and the other as led, and then export external\_connection for them with the appropriate names. Then all the elements described above are need to be configured. The first step was to set the necessary parameters for Nios II, which are shown in Figure 5.2. But it is worth noting that Nios II was configured after the elements were connected, since it is impossible to specify memory vectors otherwise [6].

|  |
| --- |
|  |
| Figure 5.2 |

For On Chip Memory the Total memory size parameter was set to 204800 bytes. Then the switch and led PIOs were configured: the Width parameter was set to 10, the Direction parameter value for switch – Input, and for led – Output. For the System ID, the value was set equal to 0x00c00000.

The appearance of the Platform Designer window is shown in Figure 5.3. It is also worth to mention that the Assign Base Addresses function was used to set the start and end address values for all elements.

|  |
| --- |
|  |
| Figure 5.3 |

Then the code was generated by using Generate HDL. After that the DE 1\_SoC\_nios.qip file was added to the project by using the Add/Remove Files option. The next step was to write the code in the previously created DE1\_SoC\_nios\_tr.v file. The Verilog HDL File with the code is shown in Figure 5.4.

|  |
| --- |
|  |
| Figure 5.4 |

The next step after successful Analysis and Synthesis was to work with Pin Planner. It was needed to specify the pin designations used in this diagram. The pin designations can be found in the user's manual. All the pins used in this diagram are shown in Figures 5.5.

|  |
| --- |
|  |
| Figure 5.5 |

It is necessary to install the Eclipse development environment to continue working with the project. The file containing Eclipse must be unzipped to the Quartus program folder ".../nios2eds/bin". Then rename the resulted folder to "eclipse\_nios2". Also, for correct operation, Eclipse requires the installation of Windows System for Linux and Ubuntu 18. Then the following command "wsl-set-default-version 1","sudo apt install wsl","sudo apt install dos2unix","sudo apt install make" should be entered on the command line. The Perspective value must be switched to Nios II when the Nios II Software Build Tool for Eclipse is running.

Then it is necessary to run the Eclipse program from the Nios II Command Shell.bat command line. The next step was to create a project (Nios II Application and BSP from Template), and select Hello World as the template. The program window with all the settings is shown in Figure 5.6.

|  |
| --- |
|  |
| Figure 5.6 |

Then the code is shown in Figure 5.7 must be added to a hello\_world.c file in the tab DE1\_SoC\_nios2\_tr\_software.

|  |
| --- |
|  |
| Figure 5.7 |

Then all Makefile in the DE1\_SoC\_nios2\_tr\_software and DE1\_SoC\_nios2\_tr\_software\_bsp folders must be deleted in the Project. The folder with created BSP must be open in the Eclipse Nios II Command Shell.bat. The next stage is running the "dos2unix create-this-bsp" and ". /create-this-bsp --cpu-name nios2\_gen2\_0 --no-make" command. Then similar actions should be performed with the folder of created application DE1\_SoC\_nios2\_tr\_software by using the commands "dos2unix create-this-app" and ". /create-this-app --cpu-name nios2\_gen2\_0 --no-make". The "dos2unix" command implement instantly converting to Unix encoding. The command line containing all the commands described earlier is shown in Figure 5.8.

|  |
| --- |
|  |
| Figure 5.8 |

The next step was to use the BSP Editor. Its settings are shown in Figure 5.9. The «enable\_lightweight\_device\_driver\_api» setting must be disabled. This option allows to reduce the amount of code and data.

|  |
| --- |
|  |
| Figure 5.9 |

After that, you need to open the Makefile created earlier in Eclipse, in which you need to replace 329 and 135 lines: APP\_LDFLAGS += -msys-lib=$(call adjust-path-mixed,$(SYS\_LIB)) with APP\_LDFLAGS += -msys-lib=hal\_bsp, and BUILD\_PRE\_PROCESS := on BUILD\_PRE\_PROCESS := touch $(ELF). srec, respectively. Without these corrections the project building and the elf file generation were unsuccessful. The changes made with the Makefile are shown in Figures 5.10 and 5.11.

|  |
| --- |
|  |
| Figure 5.10 |
|  |
| Figure 5.11 |

Use the context menu to build the BSP and application by using the Build Project command.

Then the project compiling can be started. After successfully completing this step, the Programmer window located on the toolbar is needed to open. It was necessary to select the Development Kit – DE1\_SoC and the compiled project file. The final view of the Programmer window is shown in Figure 5.12. The programming can be started by clicking Start button.

|  |
| --- |
|  |
| Figure 5.12 |

Then it is necessary to run the application in the Eclipse program. The Target Connection tab appears after selecting Run as Nios II Hardware for DE1\_SoC\_nios2\_tr\_software. So it is needed to click Refresh Connections, and then 2 data-filled lines shown in Figure 5.13 will appear.

|  |
| --- |
|  |
| Figure 5.13 |

Then by using the System ID Properties button, user need to check the values of Expected System ID and Connected System ID – they must match. These values are shown in Figure 5.14. Then the application can be started by using the Run button.

|  |
| --- |
|  |
| Figure 5.14 |

The following switch position was set on Development Kit (shown in Figure 5.15), which is equal to the number 0101001011 in binary numeral system.

|  |
| --- |
|  |
| Figure 5.15 |

The result of the program was the illumination of those LEDs for which the switch was in the logical unit position, as well as the Nios II Console screen message shown in Figure 5.16. The number that appears on the screen corresponds to the number set using the switches, but in decimal number system.

|  |
| --- |
|  |
| Figure 5.16 |

In order to increase the processor response time by 2 times, the number of repetitions in the loop for the variable j can be increased by 2 times. This result is shown in Figure 5.17. In practice, it was tested successful, that the processor response time increased by 2 times – from 1 to 2 minutes.

|  |
| --- |
|  |
| Figure 5.17 |

**Conclusions:** as a result of this project, the skills of working with Eclipse development environment were obtained. The Nios II module was tested by using the code. The result of the program was the illumination of those LEDs for which the switch was in the logical unit position.

**References**

1. Medipix\_CERN [http://www.jinr.ru/wp-content/uploads/2016/03/ Medipix\_CERN.pdf](http://www.jinr.ru/wp-content/uploads/2016/03/%20Medipix_CERN.pdf)
2. Ballabriga R., Campbell M., Llopart X. An introduction to the Medipix family ASICs // Radiation Measurements. – 2020. – Т. 136. – С. 106271.
3. Llopart X. et al. Timepix4, a large area pixel detector readout chip which can be tiled on 4 sides providing sub-200 ps timestamp binning // Journal of Instrumentation. – 2022. – Т. 17. – №. 01. – С. C01044.
4. DE1\_SoC User manual <http://download.terasic.com/downloads/cd-rom/de1-soc/DE1-SoC_v.5.1.3_HWrevF.revG_SystemCD.zip>
5. Quartus II Handbook, V.3 [https://hamblen.ece.gatech.edu/ UP3/quartusii\_handbook.pdf](https://hamblen.ece.gatech.edu/%20UP3/quartusii_handbook.pdf)
6. Основы разработки встраиваемых систем на ПЛИС с использованием процессора NIOS® II : учеб. пособие / Д.С. Смирнов, И.Г. Дейнека, А.С. Алейник, И.А. Шарков.. – СПб: Университет ИТМО, 2019. – 95 с.