Faculty of Engineering
Department of Electrical and Electronics Engineering

EE 402 PROGRESS REPORT

Spring 2022

Tolga GÜMÜŞÇÜ
M. İbrahim KILIÇ

Multicore FPGArduino

Supervisor:	**H.Fatih Uğurdağ**	______________________
Jury Member 1:	**Cenk Demiroğlu**	______________________
Jury Member 2:	**Kadir Durak**	______________________

ABSTRACT

FPGArduino allows you to develop on FPGA development boards using Integrated Development Environment for Arduino. To achieve this, FPGArduino provides prebuilt software tools and FPGA configuration bitstreams for Arduino IDE. For the purpose of the bitstream files, we can use the FPGA as if it were an Arduino board, so that without mastering languages such as VHDL or Verilog, they can develop FPGA programs with a less complex language such as Arduino language. Therefore, when someone who develops an FPGA program is faced with a problem, they will be able to look for a solution about the Arduino language, which has more research and documentation about it.

FPGArduino is intended to be an alternative for those who do not have the necessary experience in FPGA development or are more comfortable developing with C and C++ than Verilog and VHDL. The system is built around a CPU core that executes subsets of RISC-V or MIPS instruction sets, and we plan to develop it and make a project that works beyond simple sample code. In the project, 32-bit pipelined f32c processor core with open-source MIPS and RISC-V instruction set support was used.

We started to do research to choose the FPGA hardware suitable for us for the FPGArduino project, we decided to use Xilinx Artix 7-FPGA Basys 3 for the beginning of the project, then we completed our configurations on the Arduino IDE as a result of our research. To test the compatibility of our bitstream files, we wrote demo codes and checked for compatibility. After our testing phases, we continued our research to develop the project and started planning for the future of the project.

As the last stage of the project, the FPGArduino will be converted to a multicore system, the CPU and memory of the FPGA board we use for this will be divided into two, allowing it to act as if they are two different systems, so that it can run two different jobs in a synchronous manner.

ACKNOWLEDGMENT

We would like to express our greatest appreciation to Prof. Hasan Fatih Uğurdağ for his extensive efforts in making this project with his knowledge and experience. We would also like to thank Özgür Ozan Yılmaz and Seçkin İpek for their technical assistance and guidance in our project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ASIC	Application-Specific Integrated Circuit
BRAM	Block Ram
CLB	Configurable Logic Block
CPU	Central Processing Unit
EBR	Embedded Block Ram
FPGA	Field-Programmable Gate Array
IDE	Integrated Development Environment
ISA	Instruction Set Architecture
MIPS	Microprocessor without Interlocked Pipeline Stages
OTP	One-Time Programmable
RISC	Reduced Instruction Set Computer
SIMD	Single Instruction Multiple Data
SOC	System On a Chip
SREC	Motorola S-record is a file format
VHDL	VHSIC Hardware Description Language

1 INTRODUCTION

The aim of this report is to explain Multi-core FPGArduino in detail, which is our EE402 Senior Project. FPGArduino project was done by Marko Zec and his colleague Vordah at the University of Zagreb in 2016. Our aim is to bring this project to a multi-core structure. In this report, we will talk about the working principles and the problems we encounter in implementing them.

We will start by explaining the basic concepts of our project. FPGA’s, Arduino, Arduino Ide and multicore processors.

Field Programmable Gate Arrays (FPGAs) [5] are semiconductor devices based on a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs are basically like a processor that can program the hardware, rather than write software to run a predefined processor. This indicates that FPGAs can be reprogrammed after production to the desired application or functionality requirements. This prominent feature distinguishes FPGAs from Application-Specific Integrated Circuits (ASICs), which perform specific design tasks for a single application and cannot be changed after the hardware side goes into the field. Although SRAM-based FPGAs are reprogrammable, one-time programmable (OTP) FPGAs exist, they are not the dominant types.

Arduino is [6] a concept of hardware and software ecosystem which was launched by people who build cheap prototype boards with extremely cheap 8-bit microcontrollers and has a built-in graphical user interface for making it easy to program. Today Arduino provides an open-source electronics platform based on easy-to-use hardware and software with more powerful but still cheap microcontrollers. The main attraction of Arduino is simplicity and quick results.

Arduino Integrated Development Environment (IDE) is a cross-platform application written in C and C++ that can be used to write and upload programs to Arduino compatible boards as well as 3rd party development boards.

We have the FPGArduino project, which is the main source for our project, as a reference, but as another sample project we have a project which is not similar to our project, but we can refer to the navigation platform comparison using FPGA and Arduino to help explain the application reason. In this study [1], FPGA provided more accuracy but complexity is more as compared to Arduino. We want to observe if we can achieve FPGA accuracy using Arduino simplicity.

FPGA chips are programmable logic devices that can be modified by users to meet their own requirements. This allows any user-oriented hardware data processing architecture to be built, developed, and tested on a single integrated circuit. The required configuration can be specified using an electronic circuit diagram or a hardware description language such as VHDL, Verilog, or others. FPGA technology's massively parallel computing feature allows designers to speed up a data-processing technique by separating any single computing activity into several independent concurrent threads. There are several computing issues that are straightforward to parallelize by their very nature. FPGA chips demonstrate their significant benefits in all these situations [16].

Each core has its own ALU, registers, and control unit. As a result, they are self-sufficient. By intelligently partitioning the process among each core, the process' time consumption can be dramatically lowered. SIMD is a way for breaking down a huge process into smaller sub-processes and using each individual core for one of them [15].

Gantt Chart:

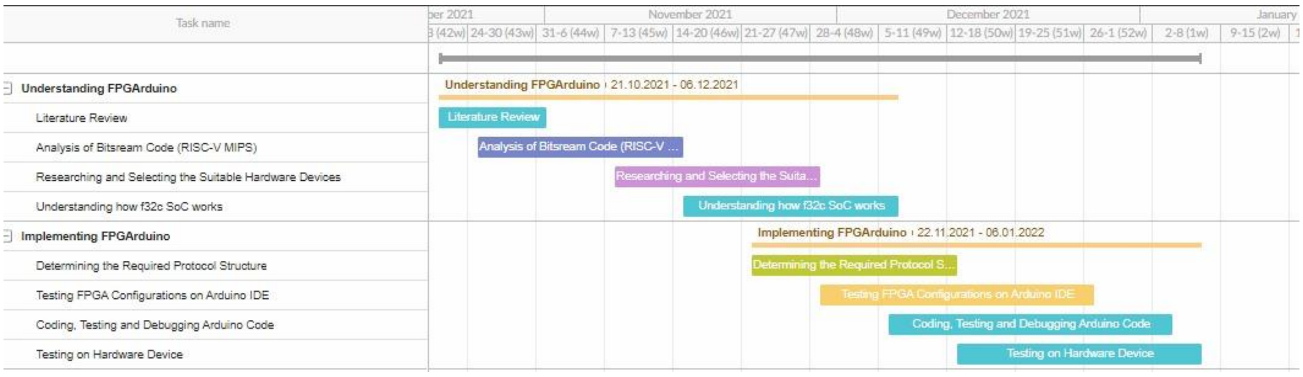
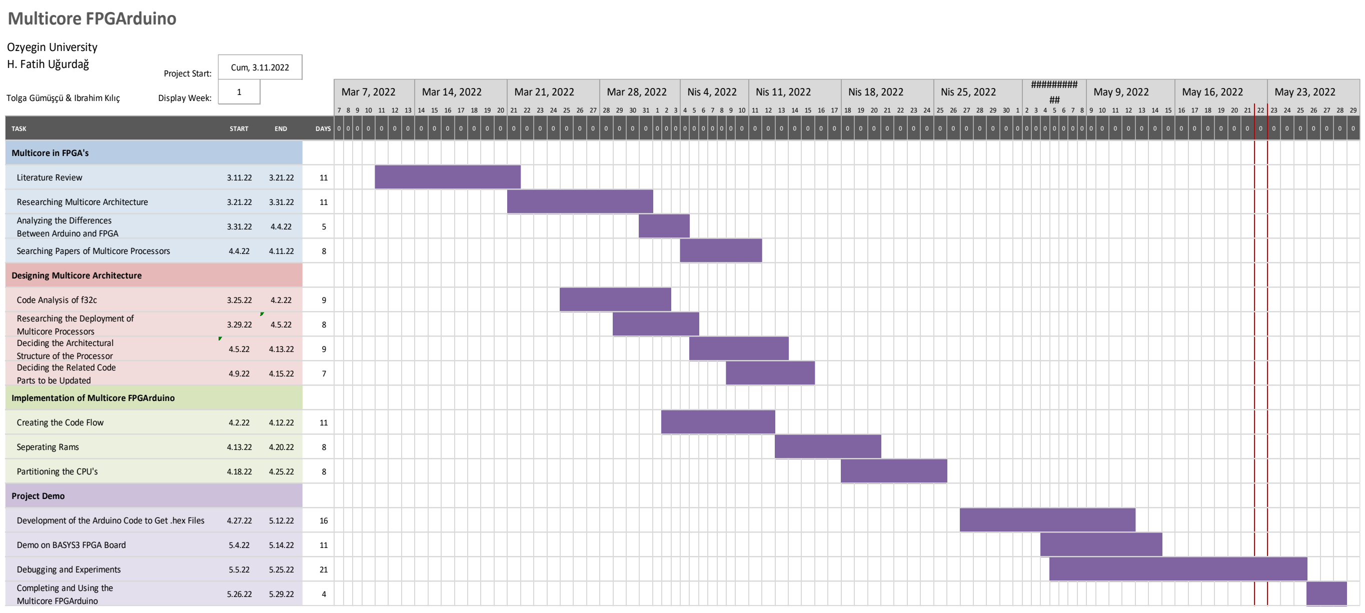


Figure 1: *Gantt Chart for First Semester Project*



Multicore FPGArduino

Ozyegin University
H. Fatih Uğurdağ

Tolga Gümüşçü & Ibrahim Kılıç

Project Start: Cum, 3.11.2022

Display Week: 1

TASK	START	END	DAYS
Multicore in FPGA's			
Literature Review	3.11.22	3.21.22	11
Researching Multicore Architecture	3.21.22	3.31.22	11
Analyzing the Differences Between Arduino and FPGA	3.31.22	4.4.22	5
Searching Papers of Multicore Processors	4.4.22	4.11.22	8
Designing Multicore Architecture			
Code Analysis of f32c	3.25.22	4.2.22	9
Researching the Deployment of Multicore Processors	3.29.22	4.5.22	8
Deciding the Architectural Structure of the Processor	4.5.22	4.13.22	9
Deciding the Related Code Parts to be Updated	4.9.22	4.15.22	7
Implementation of Multicore FPGArduino			
Creating the Code Flow	4.2.22	4.12.22	11
Seperating Rams	4.13.22	4.20.22	8
Partitioning the CPU's	4.18.22	4.25.22	8
Project Demo			
Development of the Arduino Code to Get .hex Files	4.27.22	5.12.22	16
Demo on BASYS3 FPGA Board	5.4.22	5.14.22	11
Debugging and Experiments	5.5.22	5.25.22	21
Completing and Using the Multicore FPGArduino	5.26.22	5.29.22	4

Figure 2*: Gantt Chart for Second Semester Project*

EE401:

1. Understanding how the FPGArduino project was developed and trying to determine the purpose of the project, experiments were made on how the project could be used from scratch.
2. The advantages and disadvantages of this project were explained in a cause-effect relationship.
3. Research has been done on the processor, which can also be coded using VHDL and RTL, which supports RISC-V and MIPS architectures running the f32c processor core, and the working methodologies of UART and SoC modules, GPIO, and PWM outputs are examined.
4. Using the Arduino language, test codes were developed, and board functions were tested, and we researched which sectors and areas the project could contribute to in the future.

EE402:

1. More complex Arduino codes will be developed to test the FPGArduino project so that it can be used as an Arduino with buttons, LEDs, switches, and more inputs without using an external component.
2. Sample applications provided for this project developed at the University of Zagreb will be run, and tests will be conducted to see if the library is fully compatible with Basys 3.
3. In addition to these, our own peripheral will be developed for the continuation of the project, thanks to which new applications can be created.
4. Improvements has been made to convert the FPGArduino project to a Multicore system and run multiple jobs in parallel at the same time.

2 FPGArduino

The FPGArduino project allows us to program FPGA developer boards using the Arduino development environment using pre-built software tools and FPGA configuration bitstreams. Arduino is a nice platform that provides hooks to different architectures embedded in the toolchain. What stands out for us is that Arduino is easy to program. In order to use this easy programmability on FPGA boards, Marko Zec from the University of Zagreb and his colleague Vordah made the FPGArduino software/hardware. The system on chip f32c they created for this project could get work done on many FPGA targets. We will explain f32c in this part in detail.

They provide us with a small set of starter proteins that set up the runtime environment and then Arduino will give us the rest we need. The project has IDE extensions in these we can see; ISA(RV32I/MIPS), memory size, mappings, FPGA board name and its details, SoC features it supports. There are also prebuilt FPGA bitstreams like opencd, xc3sprog, ujprog etc. Program uploads in asynchronous serial Motorola SREC hex or binary (ujprog).

F32c:

f32c is a flexible, open-source, pipelined, 32-bit open-source processor engine with support for MIPS and RISC-V ISA. The analysis of this processor architecture has been examined under 3 headings. These titles are "High-Level Architecture", "f32c core architecture" and "SoC component architectures" respectively.

"top_bram" module provides serial communication interface (RS-232), Switch, Button and Led connections as input/output. The module named "glue_bram" is used as a submodule under this module. The "glue_bram" module is the module that carries external peripherals to the f32c core architecture.

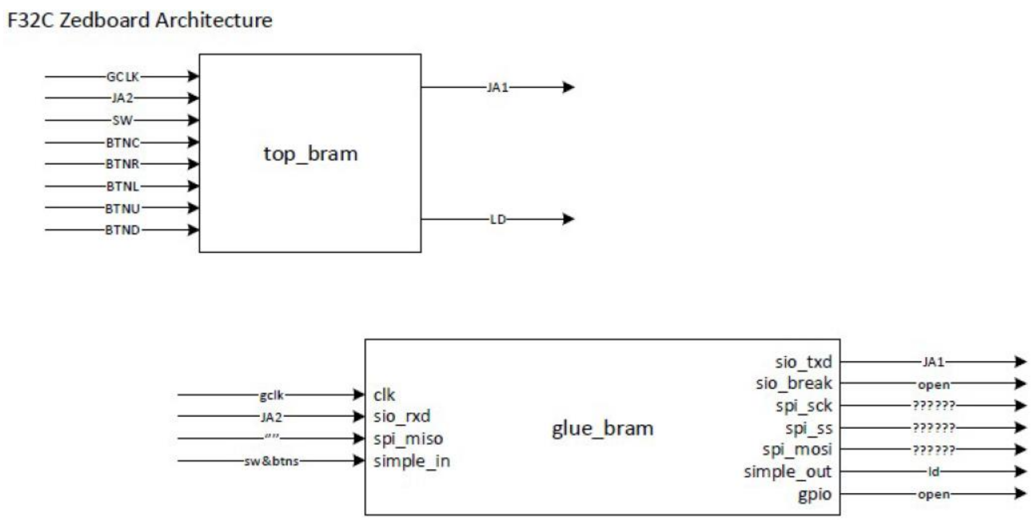


Figure 3: *Top_BRAM and Glue_BRAM [12]*

The logic architecture design of the "glue_bram" module is examined in detail below. Here, the connection logics of the signals between the Pipelined f32c core, BRAM and SoC components under the "glue_bram" module are extracted in detail. In order to show the tracking of the f32c core bus signals, different colors were painted.

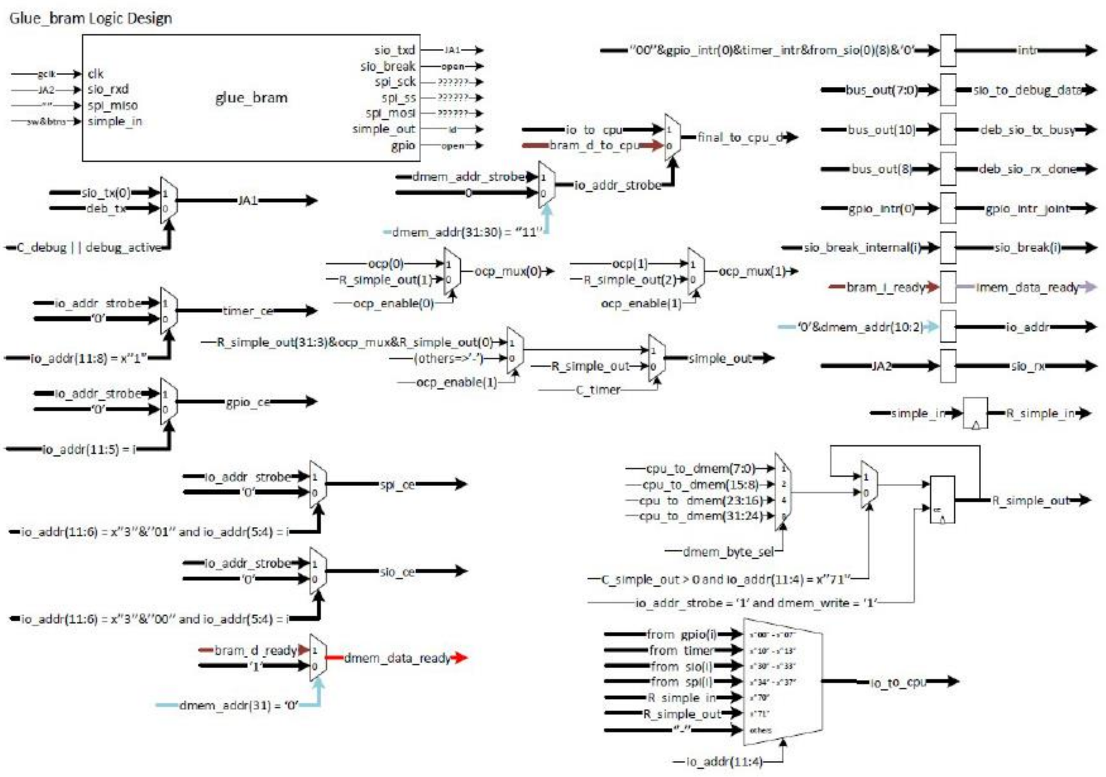


Figure 4: *Glue_BRAM [12]*

The f32c Core Architecture was analyzed on a block basis with the BRAM module. In the figure below, the connected signals and bus signals in the previous section are shown in different colors.

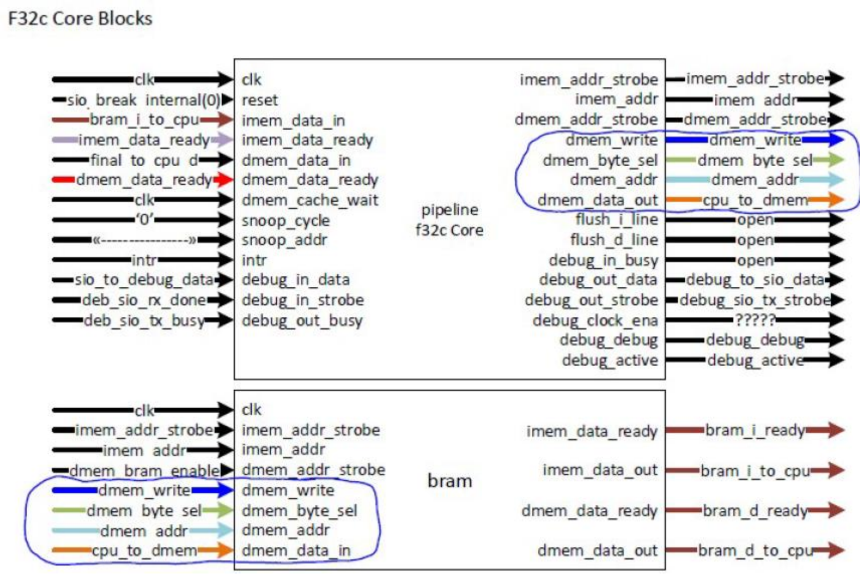


Figure 5: *f32c Core Blocks [12]*

We can define our FPGArduino project as a project that we are trying to code with Arduino IDE and Arduino language. While performing this process, we need to make a few changes on the FPGA, we need to upload the bitstream file necessary to implement this process on our board.

3 IMPLEMENTING FPGArduino

By definition, the bitstream can be defined as the transfer of information consisting of binary (1's and 0's) bits from one device to another. Thanks to this file, the device we install behaves like any other hardware, we can define the bitstream as an executable file, each bitstream is coded specifically for itself. Bitstream is a file that can be understood when it is read, so we can understand by examining the steps of the conversion process.

After transferring the bitstream file to the FPGA using the necessary applications (Adept, Vivado...), we make a C++ language-based code that can be run on the FPGA board with the configurations made on the Arduino IDE. After this stage, some configurations should be made on the Arduino IDE, these settings are as follows:

1) Board Type

2) CPU Architecture (RISC-V or MIPS)

3) SOC Type (Minimal and Vector FPU)

4) RAM Size (128KB BRAM Emulated External and 16KB BRAM Internal)

5) Protocol (HEX 115.2 kbit/s no verify RS232 and Binary 115.2 kbit/s CRC R232)

We are using BASYS 3 FPGA Board in our project, The Basys 3 is an entry-level FPGA development board, it is a ready-to-use digital circuit development platform based on the most recent Artix-7TM technology.

RISC-V: An open standard instruction set architecture based on embedded reduced instruction set computing principles. Thanks to RISC-V, we can design chips, chips, GPUs and the like, but also, since it is open-source, it can be customized for completely personal purposes.

MIPS: Microprocessor without Interlocked Pipeline Stages is a reduced instruction set type microprocessor architecture. Every command it contains is the same size, making it easy to decipher by computer hardware.

Where RISC-V is Superior to MIPS:

a) It is more modern; it includes learning & ideas that MIPS did not.

b) It is more versatile & expandable.

c) It is open source (not a technical difference but crucially important). [3]

SoC: These components almost always include a central processing unit (CPU), memory, input/output ports and secondary storage. [4]

BRAM: A Block RAM (sometimes called embedded memory, or Embedded Block RAM (EBR)), is a discrete part of an FPGA, meaning there are only so many of them available on the chip. Block RAMs are used for storing large amounts of data inside of your FPGA. [11]

CRC: Excellent in error detection and easy to implement in hardware, cyclic redundancy check (CRC) is an important error detection method widely used in network data transmission. [12]

After completing the configurations, we select our FPGA Board with the bitstream file loaded on the Arduino IDE, verify our code on the IDE, upload the project to the port to which our FPGA is connected, and our program is completed.

3.1 Methodology

The FPGA Arduino venture gives an environment that changes over FPGA development boards into programmable microprocessors utilizing the Arduino IDE. To begin with, the essential plan for the FPGA should be made and the FPGA should be modified with the bit record to be created. A while later, the code composed for the microprocessors to be run on the FPGA is compiled utilizing the precompiled gcc-based toolchain with the Arduino IDE. At last, the FPGA is modified by means of the serial port, much appreciated to the bootloader included within the FPGA design. 32-bit pipelined f32c processor center with open-source MIPS or RISC-V instruction set support is utilized within the venture.

In this project, we want to implement the system made using the f32c processor core. To do this, we followed the steps on the site given in [8], respectively. The system did not run smoothly, we encountered errors. We transferred the .bit files that were given to us to Basys3 using Artix-7 at the beginning, to Basys3 using Adept. Then we selected Enter automatic installation URL in the field from the Preferences section of the Arduino IDE and pasted this link "http://www.nxlab.fer.hr/fpgarduino/package_f32c_core_index.json" in the Additional

Boards Manager URLs section. Next, we selected the required boards from the Boards Manager in the IDE and loaded them. Normally, after doing these simple operations, the codes we compile and upload from the IDE should work directly in Basys3, which is connected to the computer, but it did not work.

Then we searched the GitHub codes of the f32c CPU core [7] and tried to understand why it didn't work. We got help from our advisor to understand what did what in the files on Github and we did research on the subject. From the codes, we found the necessary options that will work fully compatible with the board we have from the files with the .vhd extension. We couldn't do this without understanding how the f32c processor core works. We would have run it only by trial-and-error method, but we could not understand why it worked in those options.

3.2 Tools

We can divide the devices we use into two as hardware and software. As hardware, we used Digilent Basys 3 (Artix7) and Terasic DE0-Nano-SOC (Altera Cyclone-V) FPGA boards. There are also JTAG and micro-USB cables for connecting to the computer and giving power. In addition, we aim to use a monitor for VGA and video tests, and then sensors that can be connected to boards to run and test different projects.

Bays 3 details from Digilent [9]:

- Features the Xilinx Artix-7 FPGA: XC7A35T-1CPG236C
- 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)
- 1,800 Kbits of fast block RAM
- Five clock management tiles, each with a phase-locked loop (PLL)
- 90 DSP slices
- Internal clock speeds exceeding 450 MHz
- On-chip analog-to-digital converter (XADC)
- Digilent USB-JTAG port for FPGA programming and communication
- Serial Flash
- USB-UART Bridge
- 12-bit VGA output
- USB HID Host for mice, keyboards and memory sticks
- 16 user switches

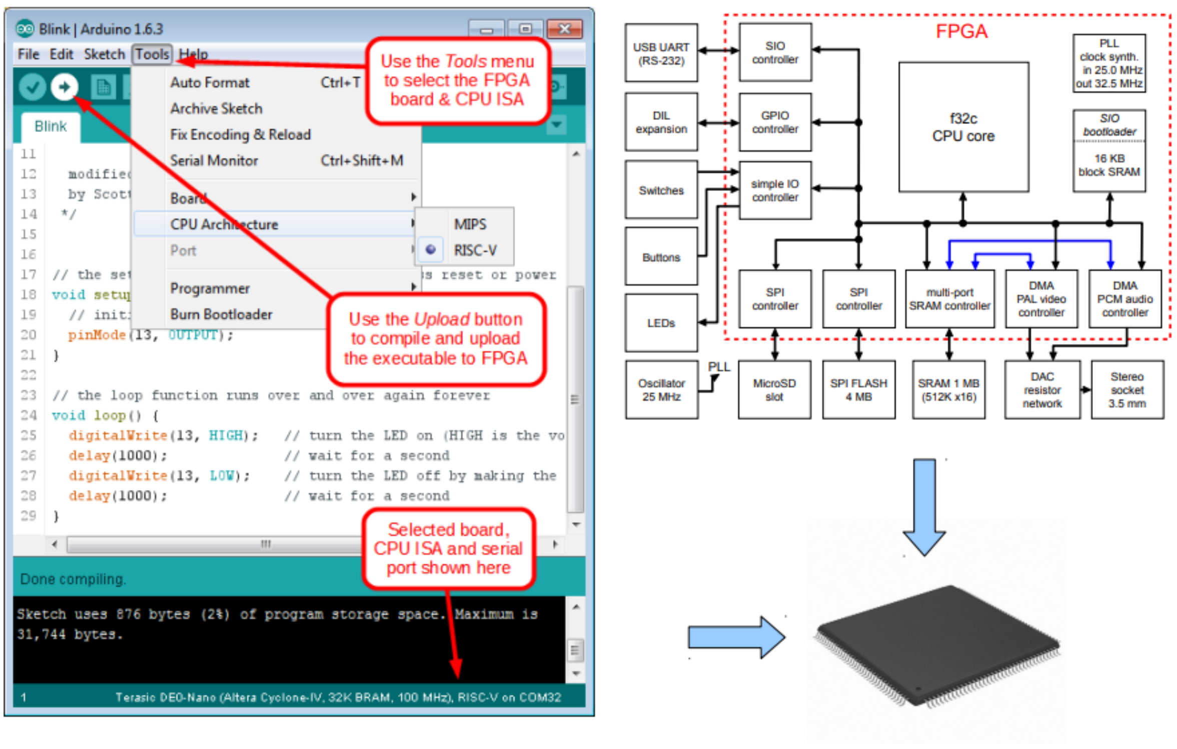
- 16 user LEDs
- 5 user pushbuttons
- 4-digit 7-segment display
- 4 Pmod ports: 3 Standard 12-pin Pmod ports, 1 dual-purpose XADC signal / standard Pmod port

DE0-Nano-SOC details from Terasic [10]:

- Altera Cyclone® V SE 5CSEMA4U23C6N device
- Serial configuration device – EPCS
- USB-Blaster II onboard for programming; JTAG Mode
- 2 push-buttons
- 4 slide switches
- 8 green user LEDs
- Three 50MHz clock sources from the clock generator
- Two 40-pin expansion header
- One Arduino expansion header (Uno R3 compatibility), can connect with Arduino shields.
- One 10-pin Analog input expansion header. (Shared with Arduino Analog input)
- A/D converter, 4-pin SPI interface with FPGA
- 925MHz Dual-core ARM Cortex-A9 processor
- 1GB DDR3 SDRAM (32-bit data bus)
- 1 Gigabit Ethernet PHY with RJ45 connector
- USB OTG Port, USB Micro-AB connector
- Micro SD card socket
- Accelerometer (I2C interface + interrupt)
- UART to USB, USB Mini-B connector
- Warm reset button and cold reset button
- One user button and one user LED
- LTC 2x7 expansion header

As software, we use Xilinx's Vivado Design Suite to code Basys 3 and Digilent Adept for JTAG configuration and data transfer. We are using Quartus II Web Edition release 13.0sp1 for DE0-Nano-SOC. Finally, we use the Arduino IDE (versions 1.6.0 or later.), which is one of the main protagonists of the project.

Finally, the f32c CPU core we used for the project. We have explained this in detail. You can also find the image provided by its makers in Figure 6.



```
   modifie
   by Scott
 */

// the set                                    s reset or power
void setup
  // init
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the vo
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the
  delay(1000);              // wait for a second
}
```

Done compiling.

```
Sketch uses 876 bytes (2%) of program storage space. Maximum is
31,744 bytes.
```

Terasic DE0-Nano (Altera Cyclone-IV, 32K BRAM, 100 MHz), RISC-V on COM32

Figure 6: *Arduino interface after uploading to the FPGA and connections with f32c [8]*

Table 1: *Arduino and Basys 3 Assigned Ports [8]*

Arduino	0	1	2	3	4	5	6	7
Basys	btn_right	btn_left	btn_down	btn_up	btn_center			
Arduino	8	9	10	11	12	13	14	15
Basys	LED #0	LED #1	LED #2	LED #3	LED #4	LED #5	LED #6	LED #7
Arduino	16	17	18	19	20	21	22	23
Basys	switch #0	switch #1	switch #2	switch #3	switch #4	switch #5	switch #6	switch #7
Arduino	24	25	26	27	28	29	30	31
Basys								
Arduino	32	33	34	35	36	38	38	39
Basys	JA(0)	JA(1)	JA(2)	JA(3)	JA(4)	JA(5)	JA(6)	JA(7)
Arduino	40	41	42	43	44	45	46	47
Basys	JB(0)	JB(1)	JB(2)	JB(3)	JB(4)	JB(5)	JB(6)	JB(7)
Arduino	48	49	50	51	52	53	54	55
Basys	JC(0)	JC(1)	JC(2)	JC(3)	JC(4)	JC(5)	JC(6)	JC(7)
Arduino	56	57	58	59	60	61	62	63
Basys								

4 MULTI-CORE PROCESSOR

4.1 Overview

A multicore architecture is one in which a single physical processor contains the core logic of many processors. These processors are packaged or held on a single integrated circuit. Die refers to these single integrated circuits. Multiple processor cores are grouped together as a single physical processor in multicore architecture. The goal is to develop a system that can handle more jobs at once, resulting in improved overall system performance [14].

When it comes to multi-core processors, there are various concepts that must be mentioned:

- Process: It is the running program thread.
- Thread (Subprocess): Process allocates tasks to threads.
- Threading: It is the ability to process more than one thread at the same time.
- Multi-Processing: All the multi-processing with threading and cores.
- Multi-Tasking: It enables more than one program to be run at the same time.

4.2 Multi-Core F32c

In the original version of F32c, there is a register file, ALU and controller in the pipeline. In the case of multi-core, all these features are used in the same way. As a result of this situation, they work as two different processors independently of each other. In this way, the time spent on many operations can be easily reduced. These two processors are connected to the block rams separately, and first, there is the Top module that connects them. Figure 7 shows the diagram of this connection.

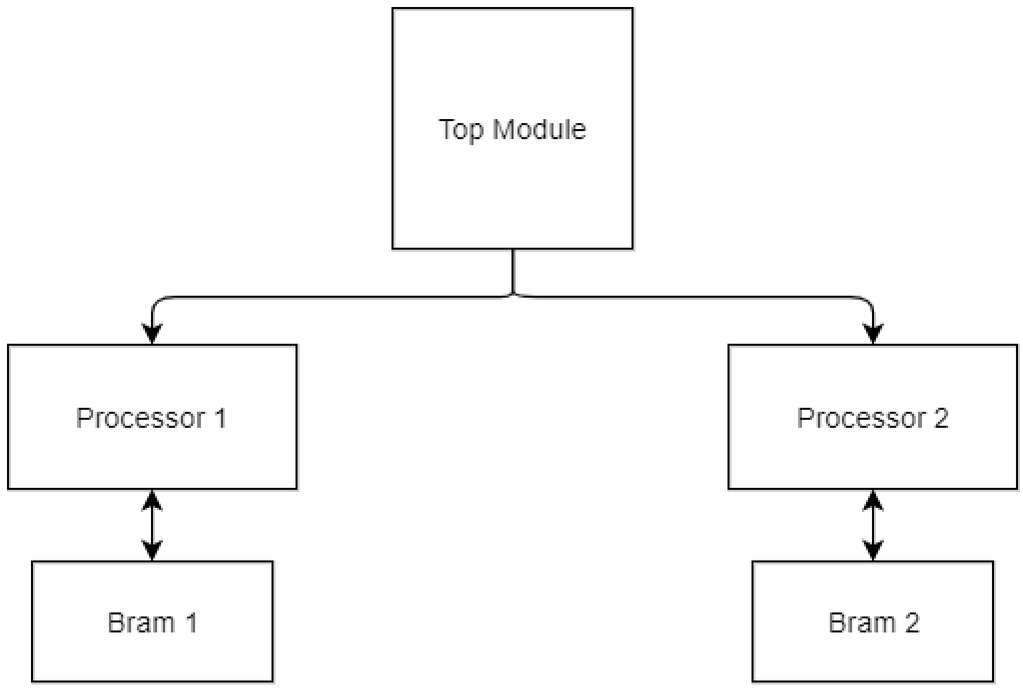


Figure 7*: Multi-core f32c Architecture*

FPGArduino converts the code compiled from Arduino Ide into a file with .hex extension and transfers it to FPGA with JTAG thanks to f32c. In the multi-core system, two separates .hex files are transferred to two processors through a single top module. In this way, after uploading a single bitstream file to the FPGA, Arduino codes could run parallel on a single FPGA.

Normally, when we upload the code we wrote from Arduino IDE, our FPGA works directly. In our multicore system, we have two separate processors that we control with the switch (0) on the FPGA. When SW (0) is "0" and we upload code from Arduino IDE our first processor works. Likewise, When SW (0) is "1" and we upload code from Arduino IDE our second processor works. These two continue to work in parallel. The detailed RTL schematic of this multicore system is in Figure 8.

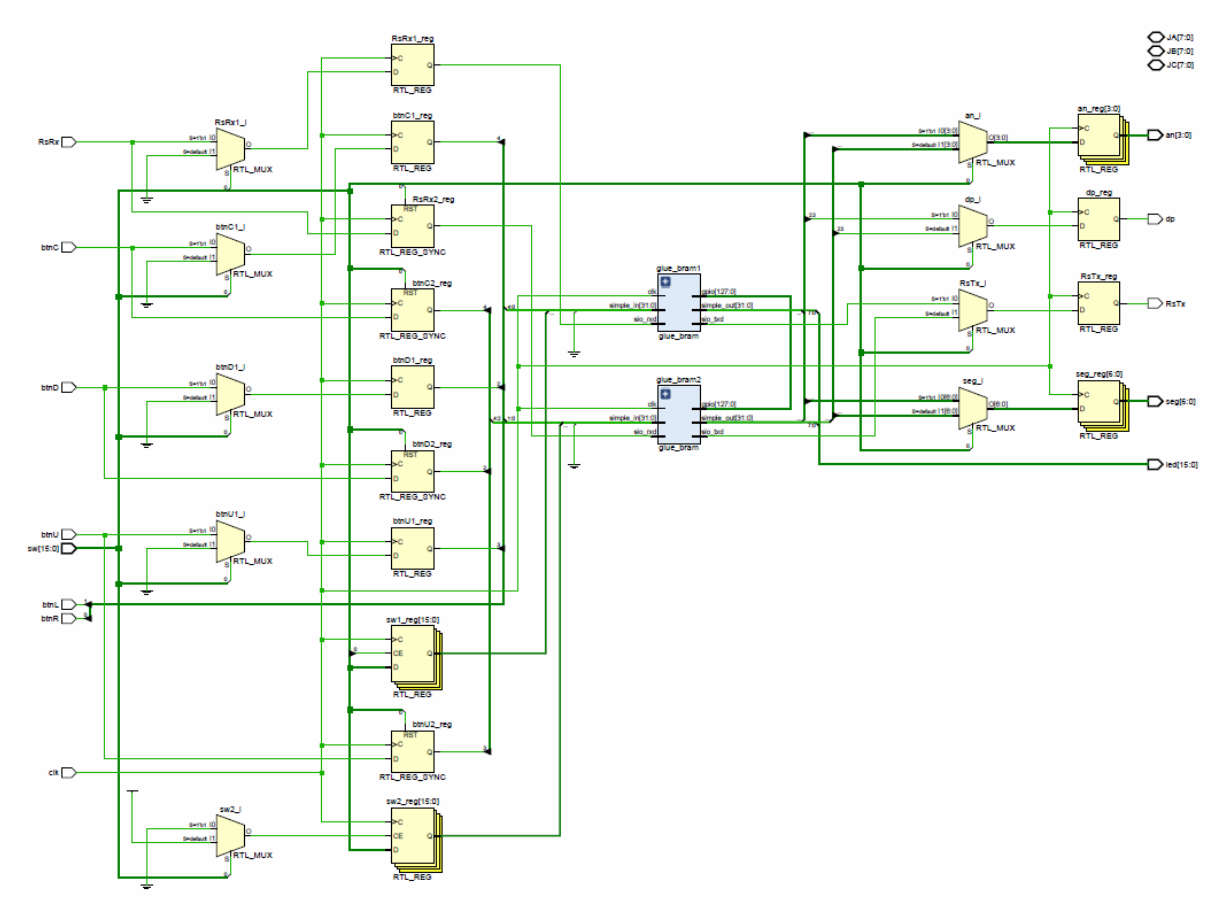
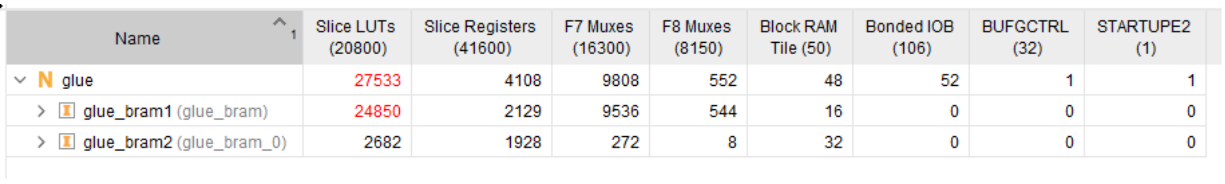


__Figure 8:__ RTL Schematic

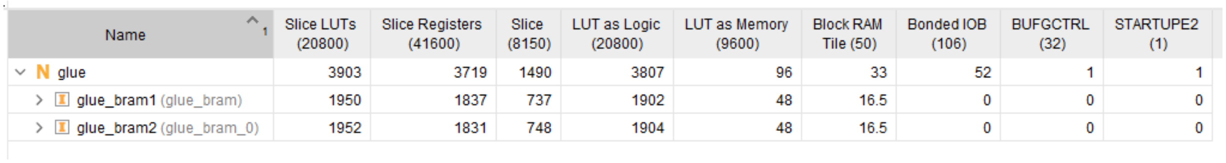
In this process, we encountered two main problems. First, when they used two of the same processors, they exceeded the 50 Block RAM limit of the Artix-7 XC7A35T, the FPGA we have, due to the amount of BRAM they used (the amount used by a single one of 32). Because of this overflow, they use LUT as Distributed Ram, but this is not enough eighter. Therefore, we have reduced the use of block RAM. You can view the state before editing in Figure 9 and the state after editing in Figure 10.



Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	STARTUPE2 (1)
glue	27533	4108	9808	552	48	52	1	1
glue_bram1 (glue_bram)	24850	2129	9536	544	16	0	0	0
glue_bram2 (glue_bram_0)	2682	1928	272	8	32	0	0	0

__Figure 9:__ Usage of FPGA Features Before Decreasing the BRAM Usage

As seen in Figure 9 both glue_bram’s are trying to use 32 BRAM’s but the limit of 50 won’t let them. Therefore, one of them is using LUT as Distrubited RAM but its also not enough.



Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	STARTUPE2 (1)
glue	3903	3719	1490	3807	96	33	52	1	1
glue_bram1 (glue_bram)	1950	1837	737	1902	48	16.5	0	0	0
glue_bram2 (glue_bram_0)	1952	1831	748	1904	48	16.5	0	0	0

***Figure 10**: Usage of FPGA Features After Decreasing the BRAM Usage*

Second problem we encounter was the STARUPE usage of our top module top_bram. In the original one we had 1 STARUPE and when we have two processors, we thought that we will need two of them. However, it turns out that our FPGA Artix-7 XC7A35T has only one STARUPE2. Therefore, we used only one of them in the top module and arranged them so that both are not needed. Even if we do not use this feature in many of the applications we use, the system works without using STARTUPE in any way, as there is no noticeable situation.

4.3 Results

Thanks to the feature of using FPGA over Arduino, provided by the FPGarduino project, we have achieved a quality that cannot normally be obtained on Arduino boards. We have created a new coding environment on Arduino, which can work with multi-core processors and where we can use the advantage of FPGAs over ASIC processors. A core has been created that can be easily coded via the Arduino IDE, and whose peripherals can be edited, constantly changed, and shaped according to the intended use.

4.4 Discussion

This multicore system we have obtained is different from traditional multicore systems. Moore's law suggested that the number of transistors in a processor would double every two years, thus increasing performance and reducing cost. But no more transistors can be squeezed into a processor anymore. Therefore, to achieve higher speeds, the industry that produces transistors for desktops has started to increase the number of cores. The reason they do this is to increase the clock frequency and run it faster with parallel operations. What we achieve with the multicore system we built is a much simpler version. The multicore system that emerged as a result of our project can only perform parallel operations.

5 CONCLUSION AND FUTURE WORKS

In order to make the FPGArduino project multicore, we tried to understand how a chip works multicore and how its architecture should be designed. Most processors around us today work using multi-core power, so we get performance and power efficiency by dividing it into multiple processor powers. The part that we had difficulty in developing our project was about dividing the memories with sufficient capacity. In order to solve this issue, we understood how to divide our memory as a result of the experiments we carried out depending on the outputs of the simulations we made. Another problem that needed to be solved was when to send the required Arduino project to which processor. To solve this issue, we used the switches on our FPGA Board, we developed the algorithm to send the necessary program to the 1st or 2nd processor by looking at whether the switch is 1 or 0. As future works, we can run more jobs in parallel by dividing it into more than two cores like today's modern processors, and a comparison of performance efficiency and power consumption between a single core processor and a multicore processor can be made.

REFERENCES

[1] S. A. Anil and R. K. Megalingam, "A comparative analysis of a navigation platform designed in Arduino and FPGA," 2016 International Conference on Communication and Signal Processing (ICCSP), 2016, pp. 1306-1310, doi: 10.1109/ICCSP.2016.7754364.

[2] M. Zec, and D. Jandrevic, "FPGArduino: a cross-platform RISC-V IDE for masses", 4th RISC-V Workshop, pp. 9-11, July, 2016.

[3] R. Baines, “What are some notable difference between MIPS and RISC V ...,” What are some notable difference between MIPS and RISC V? [Online]. Available:

https://www.quora.com/What-are-some-notable-difference-between-MIPS-and-RISC-V.

[4] Y.-L. S. Lin, “Essential issues in soc design,” Google Books. [Online]. Available: https://books.google.com.tr/books?id=7OV9lEn9LiQC&pg=PA176&redir_esc=y#v=onepage&q&f=false.

[5] “What is an FPGA? Field Programmable Gate Array?.” Internet: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html [Dec. 06, 2021].

[6] “What is Arduino?.” Internet:

https://www.arduino.cc/en/Guide/Introduction [Dec. 06, 2021].

[7] M. Zec “f32c RISC-V / MIPS ISA retargetable CPU core.” Internet:

https://github.com/f32c, Jul. 1, 2020 [Nov. 17, 2021].

[8] M. Zec, Vordah. “FPGArduino.” Internet:

“http://www.nxlab.fer.hr/fpgarduino/”, Marc. 28, 2015 [Nov. 17, 2021].

[9] “Basys 3 Artix-7 FPGA Trainer Board: Recommended for Introductory Users.” Internet: “https://digilent.com/shop/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/” [Dec. 06, 2021].

[10] “DE0-Nano-SoC Kit.” Internet: “https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=941&PartNo=2” [Dec. 06, 2021].

[11] "What is a block Ram (BRAM) in an FPGA? tutorial for beginners," What is a Block RAM in an FPGA? For Beginners. [Online]. Available:

https://www.nandland.com/articles/block-ram-in-fpga.html.

[12] W. Chuxiong and S. Haifeng, "(PDF) design and implementation of parallel CRC algorithm for Fibre Channel on FPGA," ResearchGate, 02-May-2019. [Online]. Available: https://www.researchgate.net/publication/336419367_Design_and_implementation_of_parallel_CRC_algorithm_for_fibre_channel_on_FPGA.

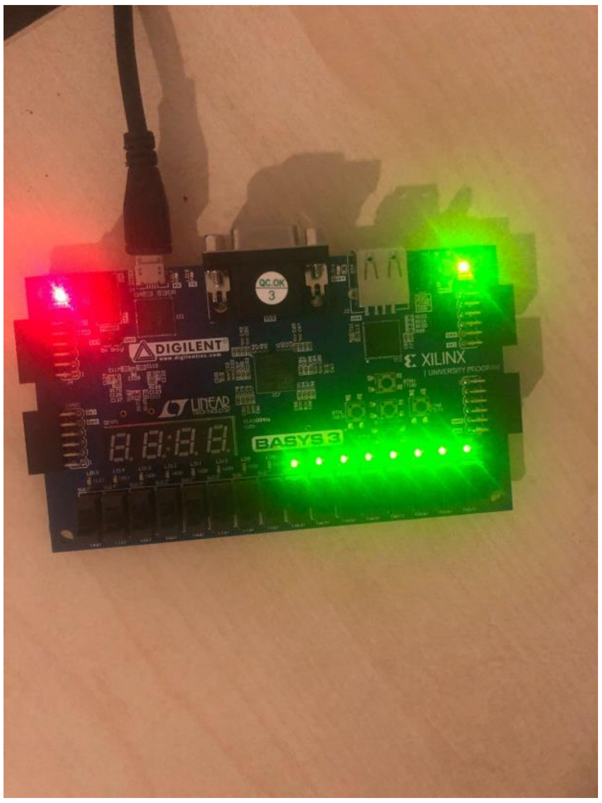
[13] M. Tosun. "FPGAArduino"(PDF) presented at Ozyegin University, Istanbul, Turkey, 2018

[14] "What is multicore?," Techopedia.com, 05-Feb-2019. [Online]. Available: - https://www.techopedia.com/definition/5305/multicore#:~:text=Multicore%20refers%20to%20an%20architecture,are%20known%20as%20a%20die.

[15] D. J. Samarawickrama, R. T. Hithanadura, K. H. M. T. M. S. Samarakoon, D. T. D. Vithanage, and M. N. C. Waas, University of Moratuwa, Moratuwa, rep., 2021.

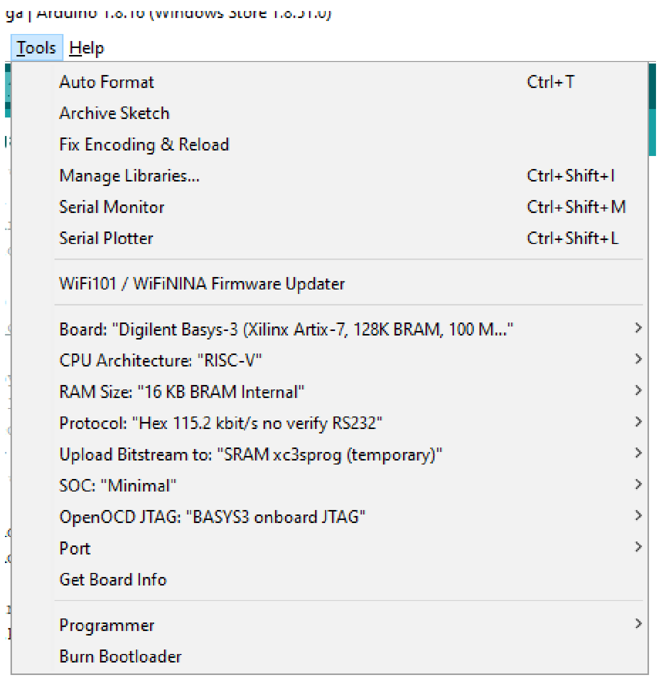
[16] X. Liang, "Ascend AI Processor Architecture and Programming," *Field Programmable Gate Arrays - an overview | ScienceDirect Topics*, 2020. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/field-programmable-gate-arrays. [Accessed: 22-May-2022].

APPENDICES



***Appendix 1**: When the RISCV Bitstream First Uploaded*

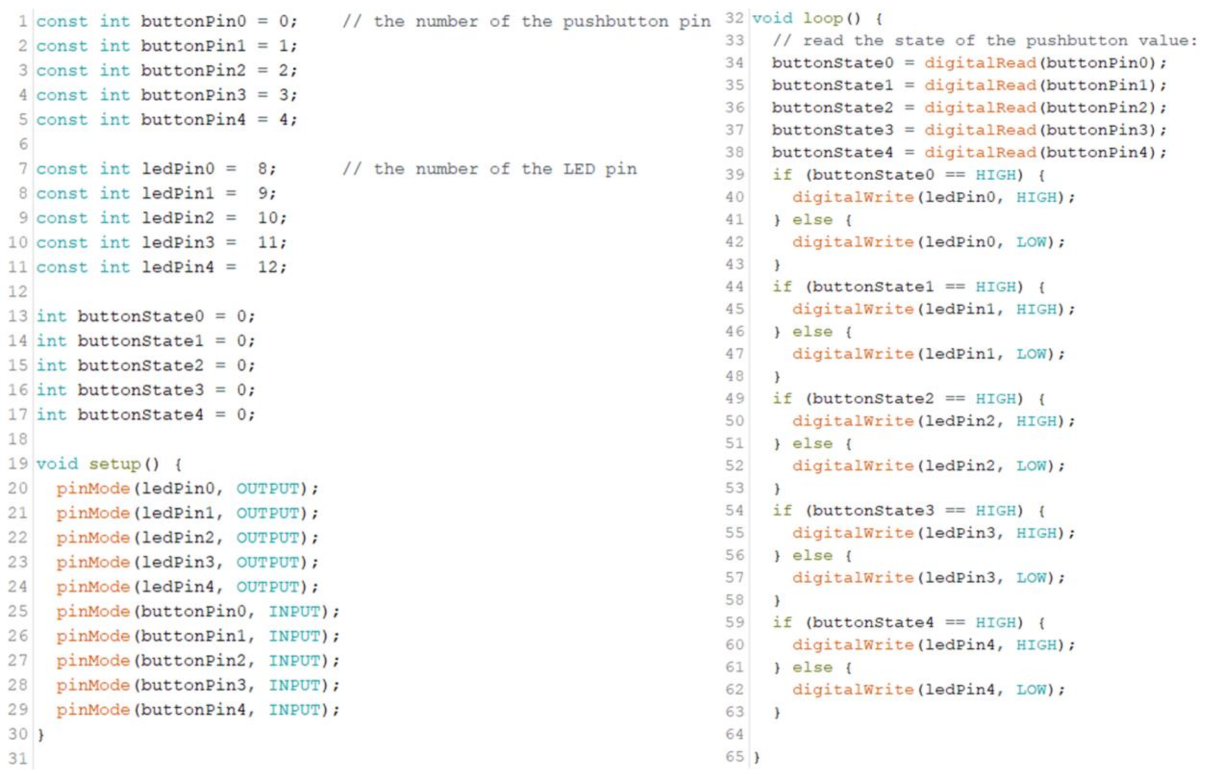
(basys3_riscv_100mhz.bit)


ga | Arduino 1.8.16 (Windows Store 1.8.51.0)

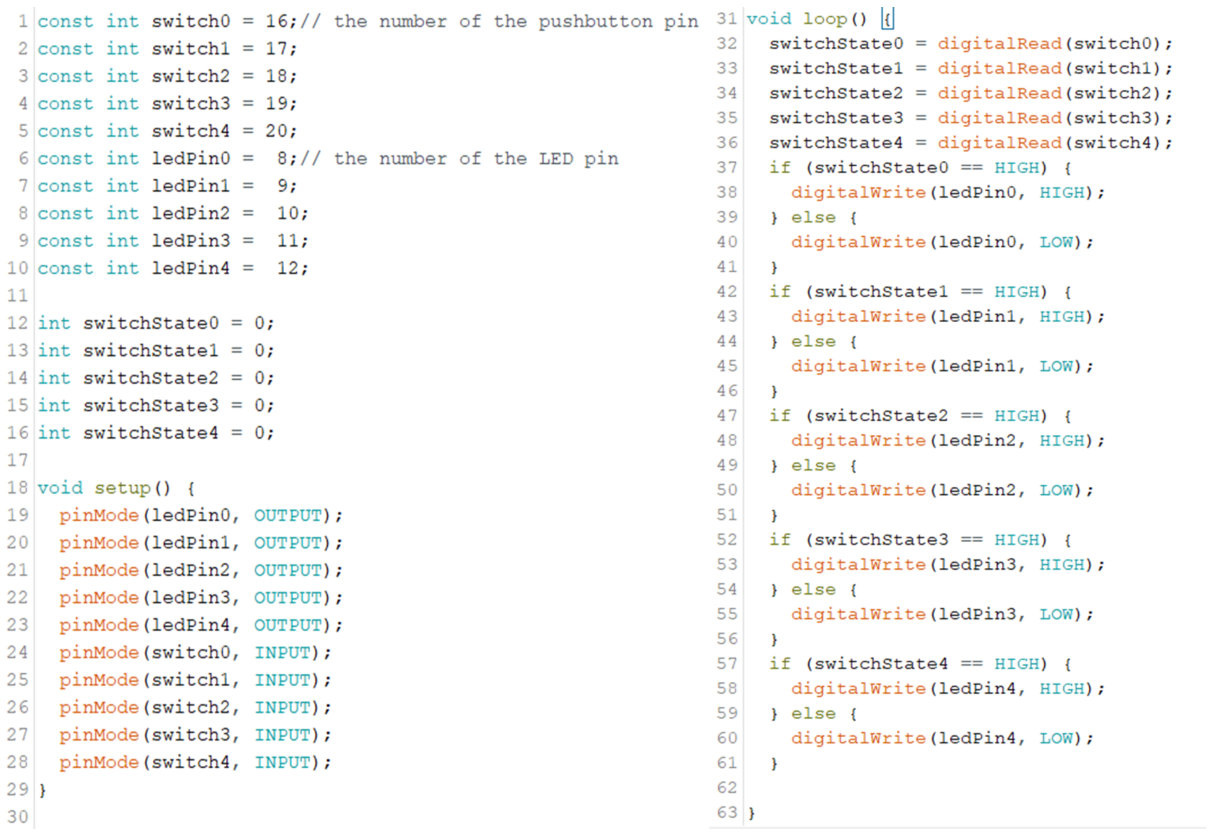
Tools Help

Auto Format	Ctrl+T
Archive Sketch	
Fix Encoding & Reload	
Manage Libraries...	Ctrl+Shift+I
Serial Monitor	Ctrl+Shift+M
Serial Plotter	Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater	
Board: "Digilent Basys-3 (Xilinx Artix-7, 128K BRAM, 100 M..."	
CPU Architecture: "RISC-V"	
RAM Size: "16 KB BRAM Internal"	
Protocol: "Hex 115.2 kbit/s no verify RS232"	
Upload Bitstream to: "SRAM xc3sprog (temporary)"	
SOC: "Minimal"	
OpenOCD JTAG: "BASYS3 onboard JTAG"	
Port	
Get Board Info	
Programmer	
Burn Bootloader	

Appendix 2: *Arduino IDE Tools Menu Selections for the Project*

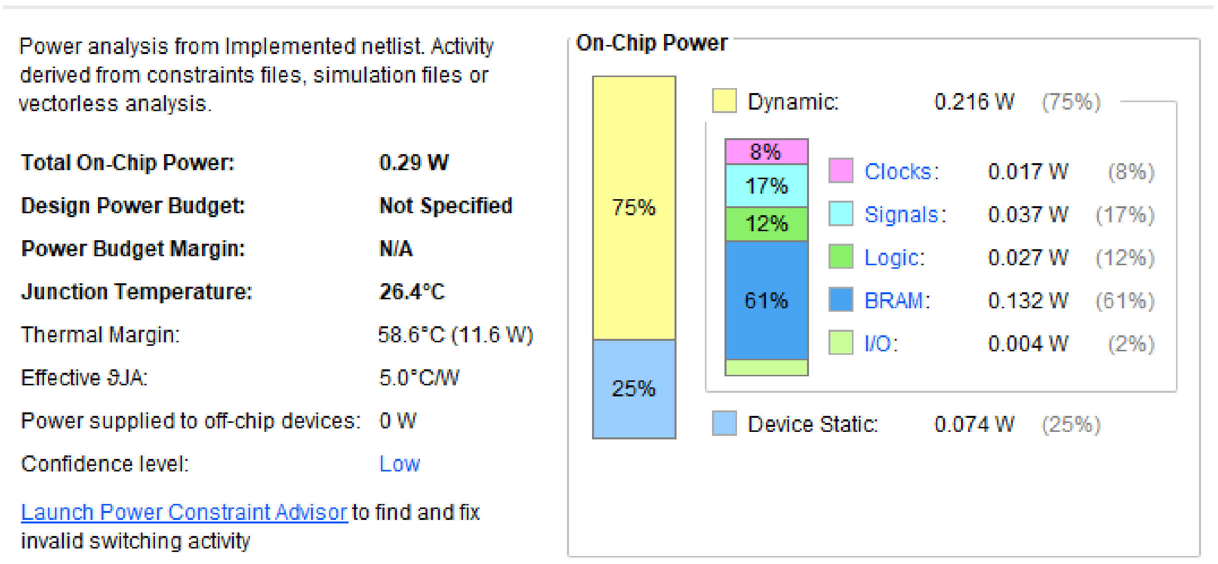

```
const int buttonPin0 = 0;     // the number of the pushbutton pin
const int buttonPin1 = 1;
const int buttonPin2 = 2;
const int buttonPin3 = 3;
const int buttonPin4 = 4;

const int ledPin0 =  8;       // the number of the LED pin
const int ledPin1 =  9;
const int ledPin2 =  10;
const int ledPin3 =  11;
const int ledPin4 =  12;

int buttonState0 = 0;
int buttonState1 = 0;
int buttonState2 = 0;
int buttonState3 = 0;
int buttonState4 = 0;

void setup() {
  pinMode(ledPin0, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(ledPin4, OUTPUT);
  pinMode(buttonPin0, INPUT);
  pinMode(buttonPin1, INPUT);
  pinMode(buttonPin2, INPUT);
  pinMode(buttonPin3, INPUT);
  pinMode(buttonPin4, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState0 = digitalRead(buttonPin0);
  buttonState1 = digitalRead(buttonPin1);
  buttonState2 = digitalRead(buttonPin2);
  buttonState3 = digitalRead(buttonPin3);
  buttonState4 = digitalRead(buttonPin4);
  if (buttonState0 == HIGH) {
    digitalWrite(ledPin0, HIGH);
  } else {
    digitalWrite(ledPin0, LOW);
  }
  if (buttonState1 == HIGH) {
    digitalWrite(ledPin1, HIGH);
  } else {
    digitalWrite(ledPin1, LOW);
  }
  if (buttonState2 == HIGH) {
    digitalWrite(ledPin2, HIGH);
  } else {
    digitalWrite(ledPin2, LOW);
  }
  if (buttonState3 == HIGH) {
    digitalWrite(ledPin3, HIGH);
  } else {
    digitalWrite(ledPin3, LOW);
  }
  if (buttonState4 == HIGH) {
    digitalWrite(ledPin4, HIGH);
  } else {
    digitalWrite(ledPin4, LOW);
  }

}
```

Appendix 3: *Controlling LEDs With Push Buttons (Arduino, C Language)*



```
const int switch0 = 16;// the number of the pushbutton pin
const int switch1 = 17;
const int switch2 = 18;
const int switch3 = 19;
const int switch4 = 20;
const int ledPin0 =  8;// the number of the LED pin
const int ledPin1 =  9;
const int ledPin2 =  10;
const int ledPin3 =  11;
const int ledPin4 =  12;

int switchState0 = 0;
int switchState1 = 0;
int switchState2 = 0;
int switchState3 = 0;
int switchState4 = 0;

void setup() {
  pinMode(ledPin0, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(ledPin4, OUTPUT);
  pinMode(switch0, INPUT);
  pinMode(switch1, INPUT);
  pinMode(switch2, INPUT);
  pinMode(switch3, INPUT);
  pinMode(switch4, INPUT);
}

void loop() {
  switchState0 = digitalRead(switch0);
  switchState1 = digitalRead(switch1);
  switchState2 = digitalRead(switch2);
  switchState3 = digitalRead(switch3);
  switchState4 = digitalRead(switch4);
  if (switchState0 == HIGH) {
    digitalWrite(ledPin0, HIGH);
  } else {
    digitalWrite(ledPin0, LOW);
  }
  if (switchState1 == HIGH) {
    digitalWrite(ledPin1, HIGH);
  } else {
    digitalWrite(ledPin1, LOW);
  }
  if (switchState2 == HIGH) {
    digitalWrite(ledPin2, HIGH);
  } else {
    digitalWrite(ledPin2, LOW);
  }
  if (switchState3 == HIGH) {
    digitalWrite(ledPin3, HIGH);
  } else {
    digitalWrite(ledPin3, LOW);
  }
  if (switchState4 == HIGH) {
    digitalWrite(ledPin4, HIGH);
  } else {
    digitalWrite(ledPin4, LOW);
  }

}
```

***Appendix 4**: Controlling LEDs With Switches (Arduino, C Language)*



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

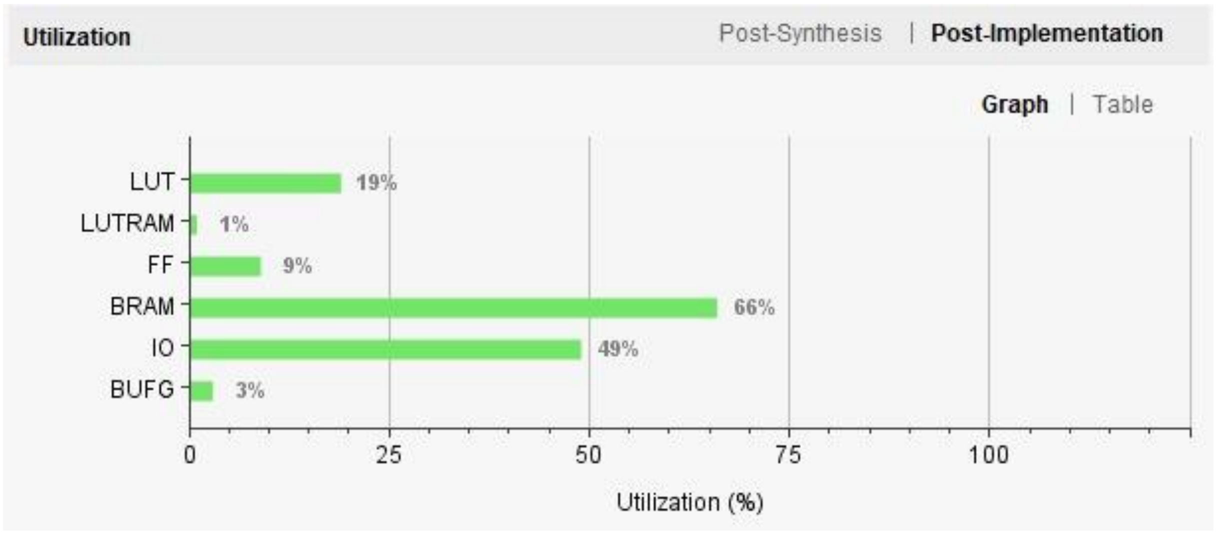
Total On-Chip Power:	**0.29 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.4°C**
Thermal Margin:	58.6°C (11.6 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

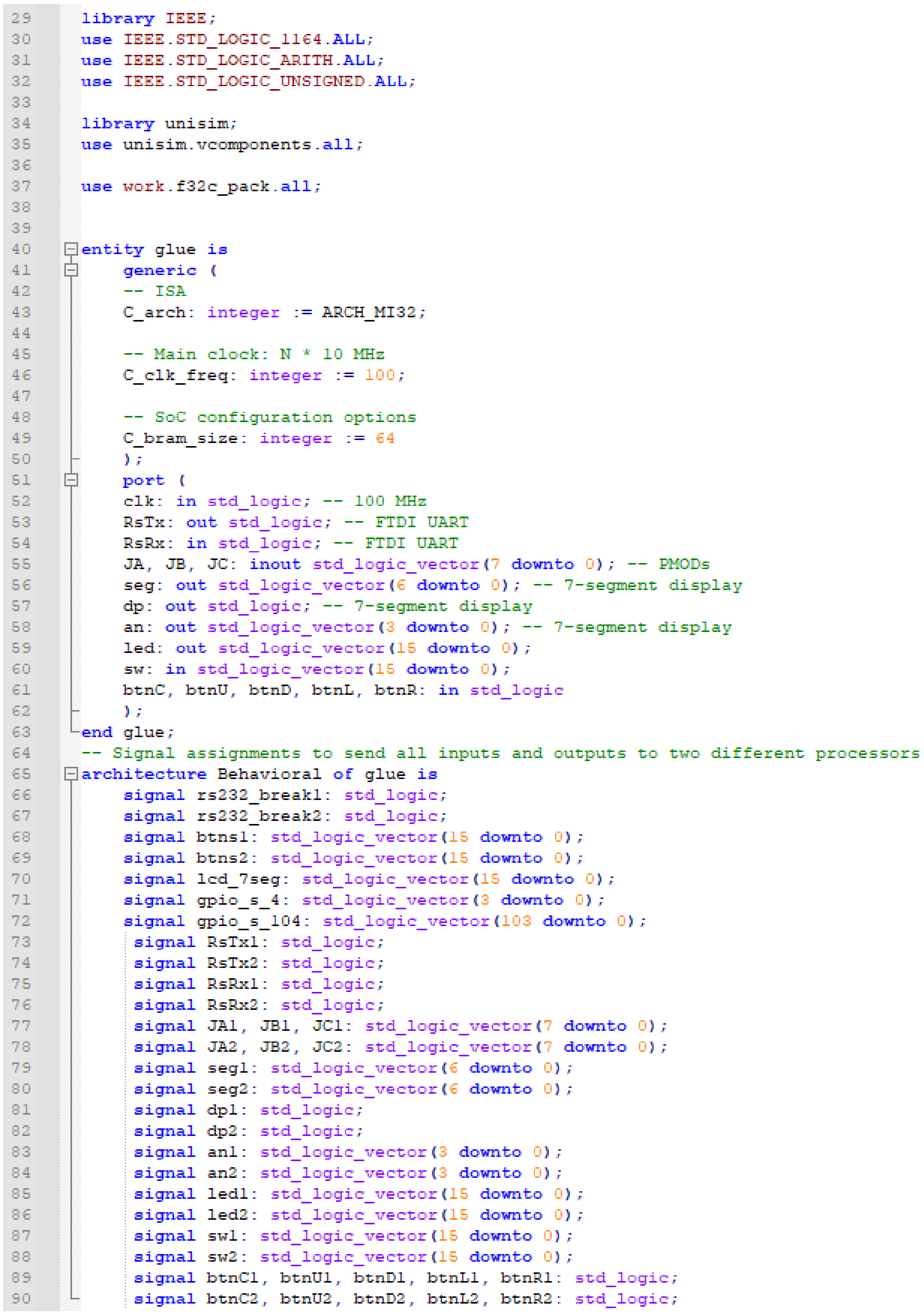
On-Chip Power

🟨	Dynamic:	0.216 W	(75%)
🟪	Clocks:	0.017 W	(8%)
🟦	Signals:	0.037 W	(17%)
🟩	Logic:	0.027 W	(12%)
🟦	BRAM:	0.132 W	(61%)
🟩	I/O:	0.004 W	(2%)
🟦	Device Static:	0.074 W	(25%)

***Appendix 5**: Power on-chip*

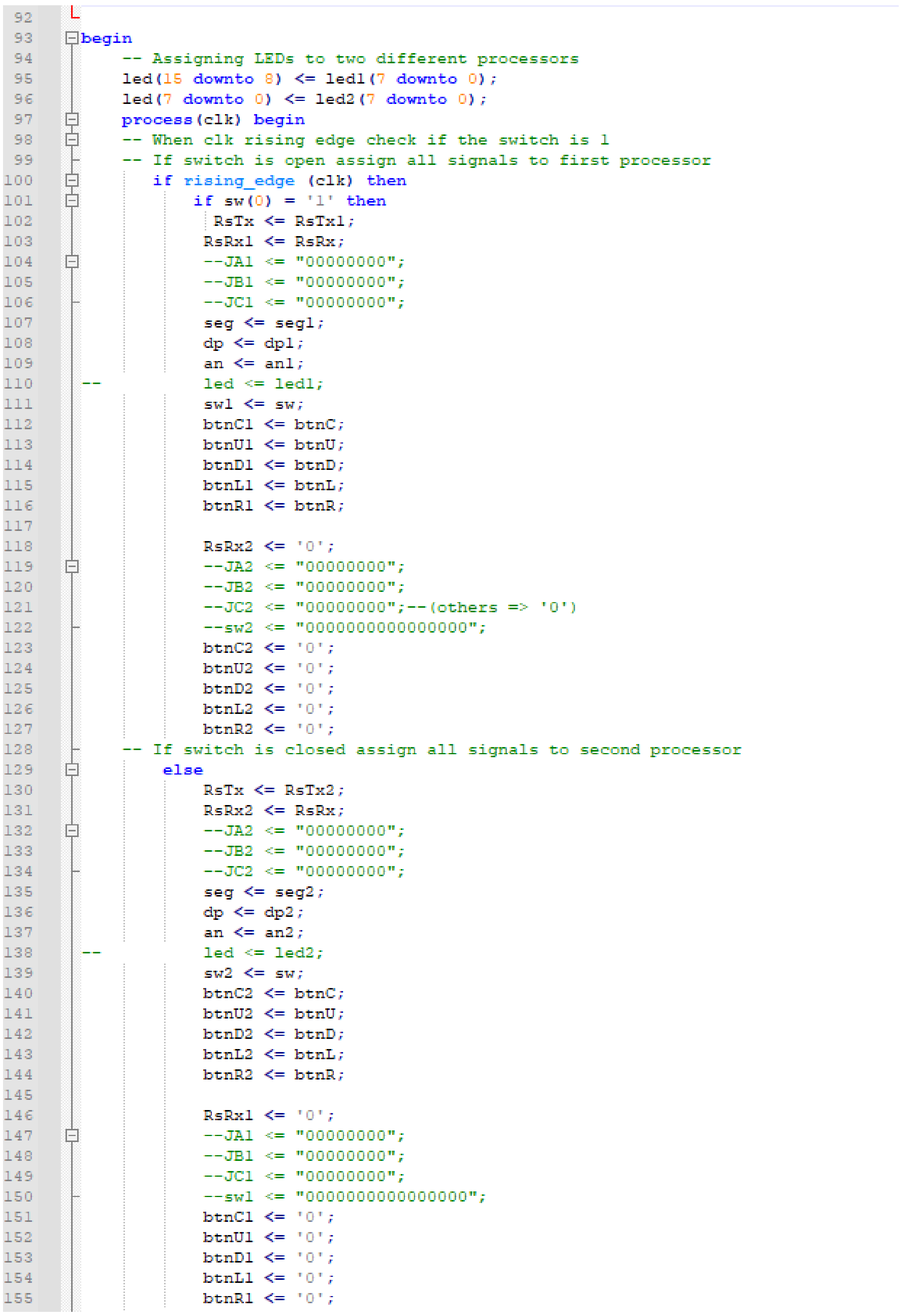


***Appendix 6**: Post implementation Utilization*



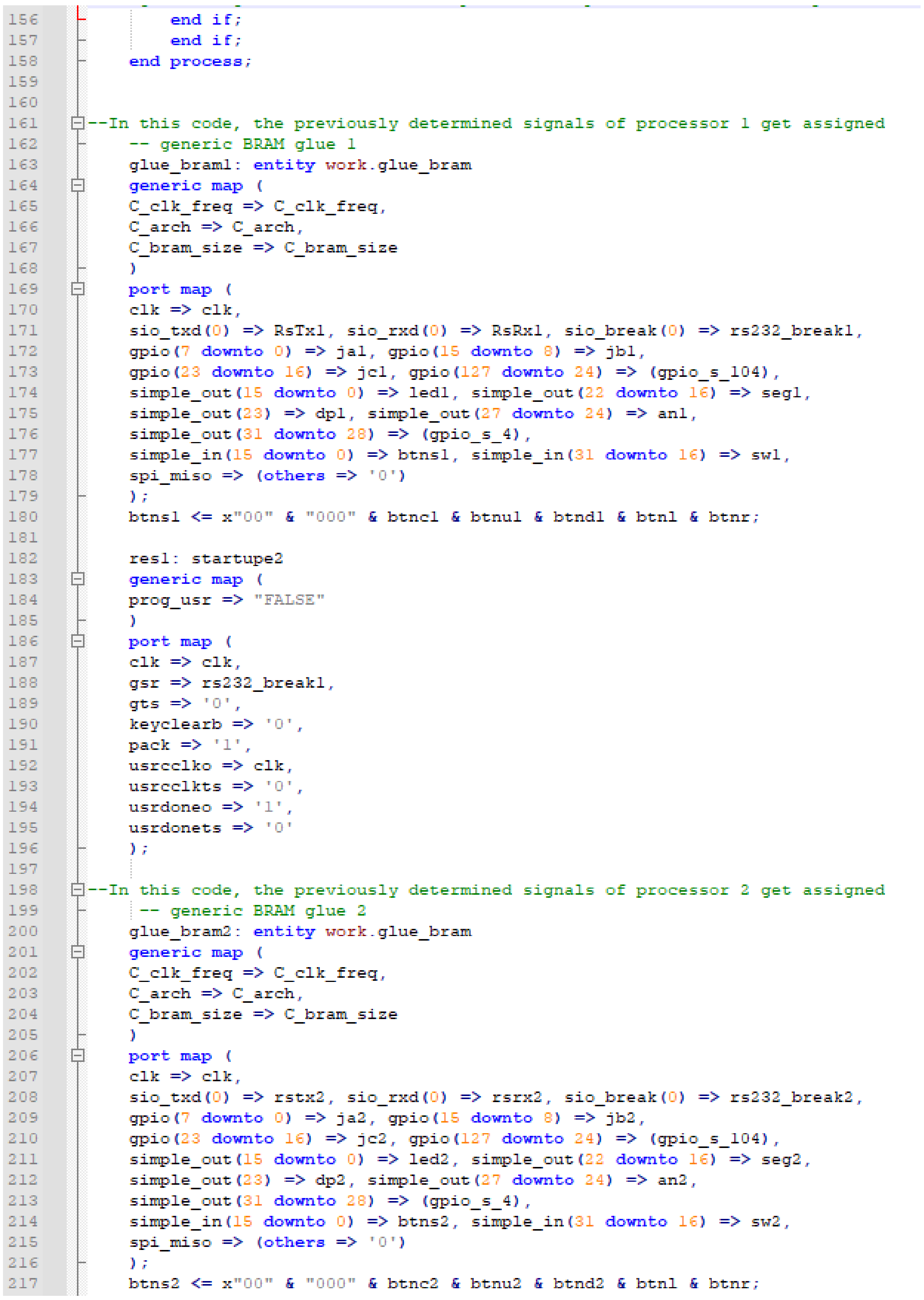
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library unisim;
use unisim.vcomponents.all;

use work.f32c_pack.all;


entity glue is
    generic (
    -- ISA
    C_arch: integer := ARCH_MI32;

    -- Main clock: N * 10 MHz
    C_clk_freq: integer := 100;

    -- SoC configuration options
    C_bram_size: integer := 64
    );
    port (
    clk: in std_logic; -- 100 MHz
    RsTx: out std_logic; -- FTDI UART
    RsRx: in std_logic; -- FTDI UART
    JA, JB, JC: inout std_logic_vector(7 downto 0); -- PMODs
    seg: out std_logic_vector(6 downto 0); -- 7-segment display
    dp: out std_logic; -- 7-segment display
    an: out std_logic_vector(3 downto 0); -- 7-segment display
    led: out std_logic_vector(15 downto 0);
    sw: in std_logic_vector(15 downto 0);
    btnC, btnU, btnD, btnL, btnR: in std_logic
    );
end glue;
-- Signal assignments to send all inputs and outputs to two different processors
architecture Behavioral of glue is
    signal rs232_break1: std_logic;
    signal rs232_break2: std_logic;
    signal btns1: std_logic_vector(15 downto 0);
    signal btns2: std_logic_vector(15 downto 0);
    signal lcd_7seg: std_logic_vector(15 downto 0);
    signal gpio_s_4: std_logic_vector(3 downto 0);
    signal gpio_s_104: std_logic_vector(103 downto 0);
     signal RsTx1: std_logic;
     signal RsTx2: std_logic;
     signal RsRx1: std_logic;
     signal RsRx2: std_logic;
     signal JA1, JB1, JC1: std_logic_vector(7 downto 0);
     signal JA2, JB2, JC2: std_logic_vector(7 downto 0);
     signal seg1: std_logic_vector(6 downto 0);
     signal seg2: std_logic_vector(6 downto 0);
     signal dp1: std_logic;
     signal dp2: std_logic;
     signal an1: std_logic_vector(3 downto 0);
     signal an2: std_logic_vector(3 downto 0);
     signal led1: std_logic_vector(15 downto 0);
     signal led2: std_logic_vector(15 downto 0);
     signal sw1: std_logic_vector(15 downto 0);
     signal sw2: std_logic_vector(15 downto 0);
     signal btnC1, btnU1, btnD1, btnL1, btnR1: std_logic;
     signal btnC2, btnU2, btnD2, btnL2, btnR2: std_logic;
```

Appendix 7*: Top Module code part I*



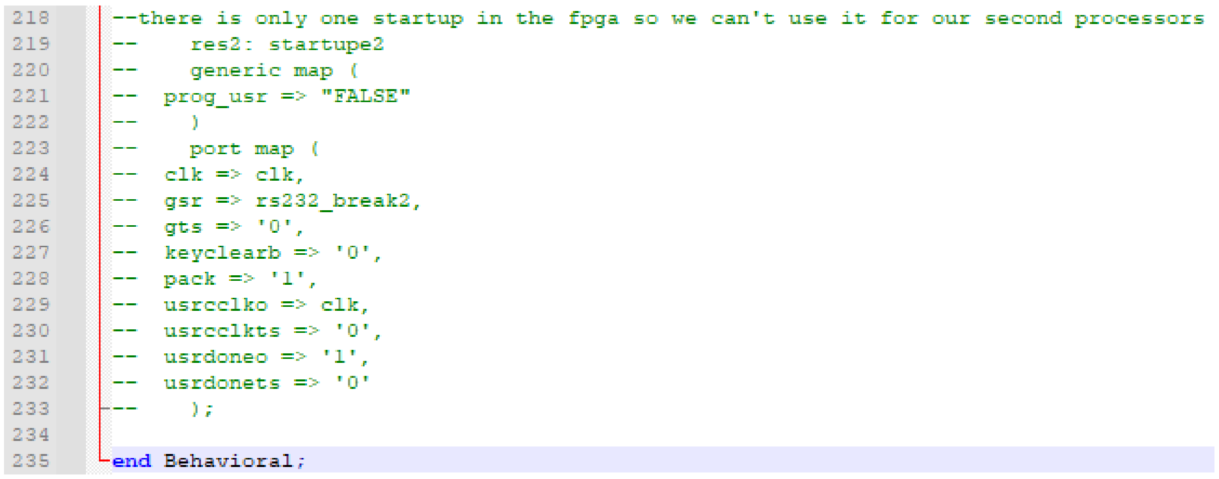
```vhdl
begin
    -- Assigning LEDs to two different processors
    led(15 downto 8) <= led1(7 downto 0);
    led(7 downto 0) <= led2(7 downto 0);
    process(clk) begin
    -- When clk rising edge check if the switch is 1
    -- If switch is open assign all signals to first processor
       if rising_edge (clk) then
           if sw(0) = '1' then
             RsTx <= RsTx1;
            RsRx1 <= RsRx;
            --JA1 <= "00000000";
            --JB1 <= "00000000";
            --JC1 <= "00000000";
            seg <= seg1;
            dp <= dp1;
            an <= an1;
--          led <= led1;
            sw1 <= sw;
            btnC1 <= btnC;
            btnU1 <= btnU;
            btnD1 <= btnD;
            btnL1 <= btnL;
            btnR1 <= btnR;

            RsRx2 <= '0';
            --JA2 <= "00000000";
            --JB2 <= "00000000";
            --JC2 <= "00000000";--(others => '0')
            --sw2 <= "0000000000000000";
            btnC2 <= '0';
            btnU2 <= '0';
            btnD2 <= '0';
            btnL2 <= '0';
            btnR2 <= '0';
    -- If switch is closed assign all signals to second processor
        else
            RsTx <= RsTx2;
            RsRx2 <= RsRx;
            --JA2 <= "00000000";
            --JB2 <= "00000000";
            --JC2 <= "00000000";
            seg <= seg2;
            dp <= dp2;
            an <= an2;
--          led <= led2;
            sw2 <= sw;
            btnC2 <= btnC;
            btnU2 <= btnU;
            btnD2 <= btnD;
            btnL2 <= btnL;
            btnR2 <= btnR;

            RsRx1 <= '0';
            --JA1 <= "00000000";
            --JB1 <= "00000000";
            --JC1 <= "00000000";
            --sw1 <= "0000000000000000";
            btnC1 <= '0';
            btnU1 <= '0';
            btnD1 <= '0';
            btnL1 <= '0';
            btnR1 <= '0';
```

***Appendix 8**: Top Module code part II*



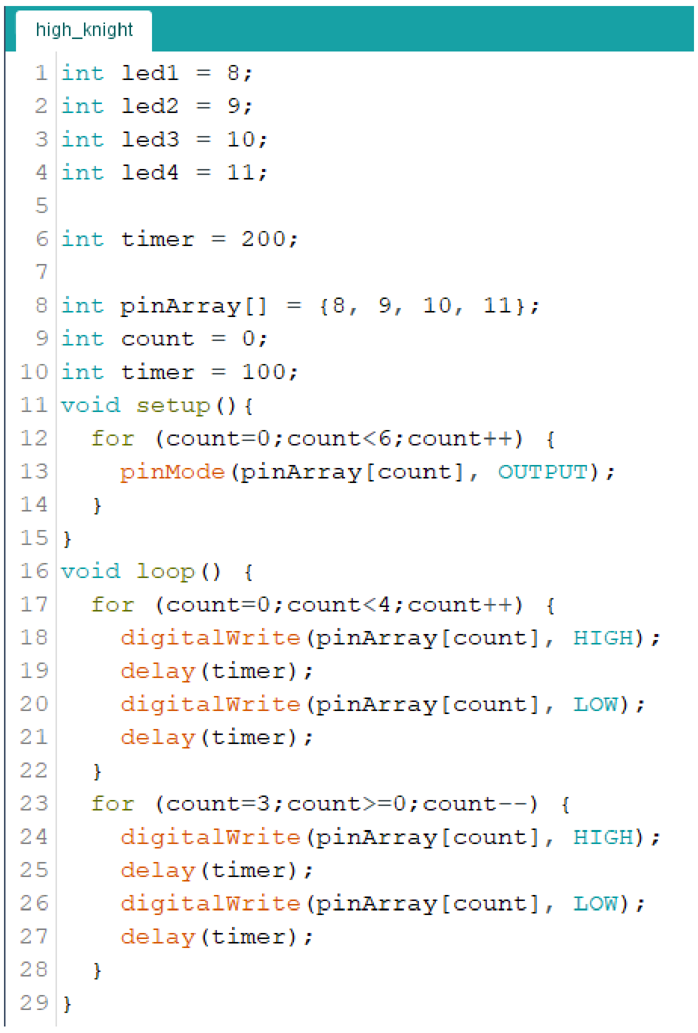
```
        end if;
        end if;
    end process;


--In this code, the previously determined signals of processor 1 get assigned
    -- generic BRAM glue 1
    glue_bram1: entity work.glue_bram
    generic map (
    C_clk_freq => C_clk_freq,
    C_arch => C_arch,
    C_bram_size => C_bram_size
    )
    port map (
    clk => clk,
    sio_txd(0) => RsTx1, sio_rxd(0) => RsRx1, sio_break(0) => rs232_break1,
    gpio(7 downto 0) => ja1, gpio(15 downto 8) => jb1,
    gpio(23 downto 16) => jc1, gpio(127 downto 24) => (gpio_s_104),
    simple_out(15 downto 0) => led1, simple_out(22 downto 16) => seg1,
    simple_out(23) => dp1, simple_out(27 downto 24) => an1,
    simple_out(31 downto 28) => (gpio_s_4),
    simple_in(15 downto 0) => btns1, simple_in(31 downto 16) => sw1,
    spi_miso => (others => '0')
    );
    btns1 <= x"00" & "000" & btnc1 & btnu1 & btnd1 & btnl & btnr;

    res1: startupe2
    generic map (
    prog_usr => "FALSE"
    )
    port map (
    clk => clk,
    gsr => rs232_break1,
    gts => '0',
    keyclearb => '0',
    pack => '1',
    usrcclko => clk,
    usrcclkts => '0',
    usrdoneo => '1',
    usrdonets => '0'
    );

--In this code, the previously determined signals of processor 2 get assigned
     -- generic BRAM glue 2
    glue_bram2: entity work.glue_bram
    generic map (
    C_clk_freq => C_clk_freq,
    C_arch => C_arch,
    C_bram_size => C_bram_size
    )
    port map (
    clk => clk,
    sio_txd(0) => rstx2, sio_rxd(0) => rsrx2, sio_break(0) => rs232_break2,
    gpio(7 downto 0) => ja2, gpio(15 downto 8) => jb2,
    gpio(23 downto 16) => jc2, gpio(127 downto 24) => (gpio_s_104),
    simple_out(15 downto 0) => led2, simple_out(22 downto 16) => seg2,
    simple_out(23) => dp2, simple_out(27 downto 24) => an2,
    simple_out(31 downto 28) => (gpio_s_4),
    simple_in(15 downto 0) => btns2, simple_in(31 downto 16) => sw2,
    spi_miso => (others => '0')
    );
    btns2 <= x"00" & "000" & btnc2 & btnu2 & btnd2 & btnl & btnr;
```

***Appendix 9**: Top Module code part III*



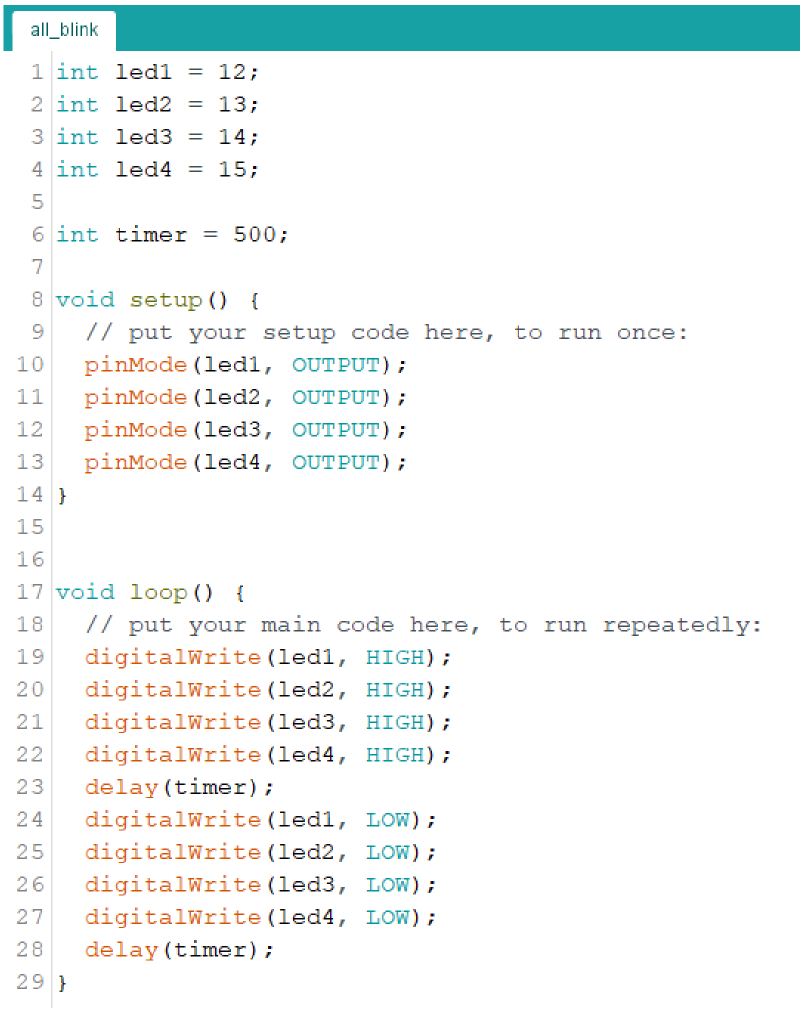
```vhdl
--there is only one startup in the fpga so we can't use it for our second processors
--    res2: startupe2
--    generic map (
--  prog_usr => "FALSE"
--    )
--    port map (
--  clk => clk,
--  gsr => rs232_break2,
--  gts => '0',
--  keyclearb => '0',
--  pack => '1',
--  usrcclko => clk,
--  usrcclkts => '0',
--  usrdoneo => '1',
--  usrdonets => '0'
--    );

end Behavioral;
```

Appendix 10*: Top Module code part IV*



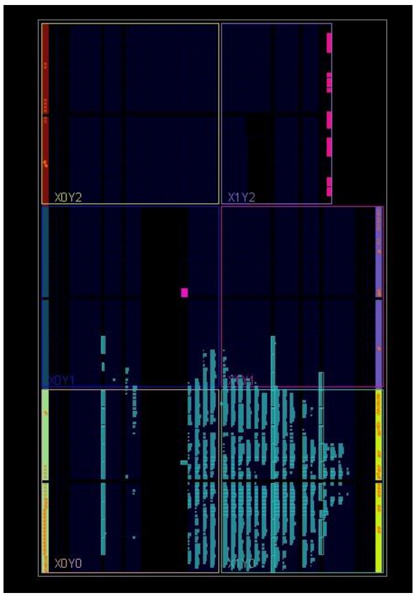
```c
int led1 = 8;
int led2 = 9;
int led3 = 10;
int led4 = 11;

int timer = 200;

int pinArray[] = {8, 9, 10, 11};
int count = 0;
int timer = 100;
void setup(){
  for (count=0;count<6;count++) {
    pinMode(pinArray[count], OUTPUT);
  }
}
void loop() {
  for (count=0;count<4;count++) {
    digitalWrite(pinArray[count], HIGH);
    delay(timer);
    digitalWrite(pinArray[count], LOW);
    delay(timer);
  }
  for (count=3;count>=0;count--) {
    digitalWrite(pinArray[count], HIGH);
    delay(timer);
    digitalWrite(pinArray[count], LOW);
    delay(timer);
  }
}
```

Appendix 11*: High Level Knight Rider Code (Arduino, C Language)*


all_blink

```
int led1 = 12;
int led2 = 13;
int led3 = 14;
int led4 = 15;

int timer = 500;

void setup() {
  // put your setup code here, to run once:
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
}


void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(led1, HIGH);
  digitalWrite(led2, HIGH);
  digitalWrite(led3, HIGH);
  digitalWrite(led4, HIGH);
  delay(timer);
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  delay(timer);
}
```

***Appendix 12**: Blink Code (Arduino, C Language)*



***Appendix 13**: Device After Implementation*