Performance Analysis of C/C++ on FPGA vs Microcontroller

Rainey Anson
d57w762

Skylar Tamke
s96s544

Hongchuan Wang
q14m192

Editor:

Keywords: Linux, FPGA, Microcontroller, semaphore

1. Introduction

The advent of field programmable gate array (FPGA) devices has changed the world of computing, allowing developers to run C/C++ code on FPGA devices to interact with custom hardware built into the fabric. The fabric is the field of transistors, gates, multiplexers available to a developer to setup whatever hardware they can imagine in VHDL or Verilog. The purpose of this report is to investigate how efficient this is compared to running C/C++ code on a microcontroller connected to custom hardware over conventional methods. Before using FPGA, microcontrollers were often used to do quick computations between operations of data processing electronics. Originally the microcontroller would do the computations on board, but later on some microcontrollers would incorporate digital signal processing (DSP) blocks. These DSP blocks can process math operations usually anywhere from tens to hundreds of times faster than the microcontroller. In this project the goal was to run C/C++ code on the FPGA versus a few microcontrollers to see how efficient, or how much overhead, each device had to each other.

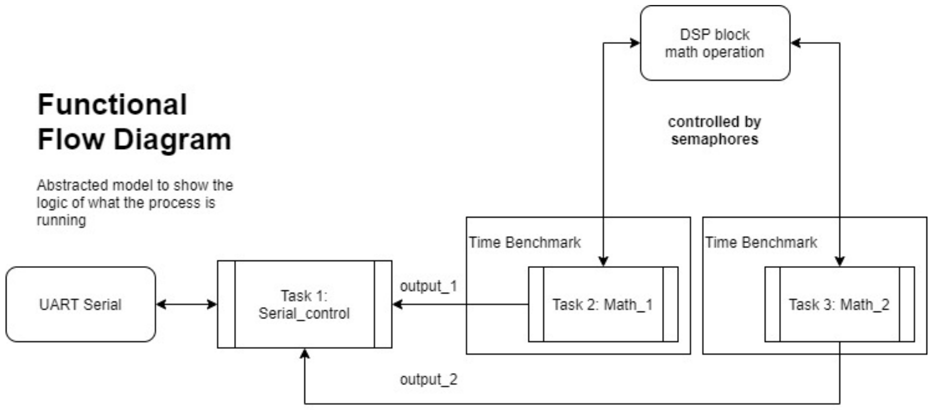


Figure 1: High level block diagram of the c code operation

2. Background and related work

2.1 Digital Signal Processing (DSP) Blocks

DSP blocks have been used in electronics requiring fast computations in a range of applications. In such microcontroller brands as Microchip, ARM, and AVR DSP blocks are usually reserved for the higher end processors compared to the cheap common parts hobbyists can pick up easily. These DSP blocks can be used in many situations like robotics, shaping of analog signals, and high speed encryption/decryption processes. Being able to adapt the algorithms that these devices implement to an FPGA greatly increases the scale of applications a single device can apply to.

2.2 Running C/C++ code on a FPGA

There exists two general kinds of FPGA devices that a developer can aquire for development. One is the pure FPGA, where 90% of the fabric is available for custom configuration by the developer, the last 10% is used for vital functions. Second is the system on chip (SoC) FPGA, these devices have a microcontroller that has access to FPGA fabric. Which eases the process of running applications on the device. For this project we are using a pure FPGA since most SoC devices are a pretty penny and the resources available to the team were only a pure FPGA. The FPGA that is being used for this project is the Digilent Basys 3 Artix7 board, this board comes with the smaller variant of the Artix7 FPGA chip (35T). There is enough space on the fabric for a soft-core processor, but not much else. A soft-core processor is a processor implemented on the fabric of the FPGA, meaning the processor can be completely erased or reset by the developer whenever.

2.3 Micro controllers

For this project a few microcontrollers were used as references against the FPGA. The chosen microcontrollers are the atmega2560, atmega328p, and atmega32u4. These were microcontrollers available to the group members for this project. To allow for multiple references against the FPGA's capabilities of running the C code. The IDEs that were used to program the atmega chips were a mix between the Arduino and a Visual Studio Code extension called PlatformIO. As the project neared the conclusion, the Arduino IDE was all but abandoned due to limited capability of

3. Approach

Overall the approach to this project was a multi-forked effort. The FPGA side of things required usage of the Xilinx software Vivado/VITIS to be able to run the C/C++ code. While the microcontroller required usage of the avr tool-chain to program the microcontrollers.

3.1 FPGA Board

The FPGA design started in Vivado to implement the soft-core processor in the fabric of the FPGA. The Xilinx development tools make it possible to prototype designs quicker than hand coding the VHDL for the whole design. Instead using block design methodology allows for rapid prototyping. While this simplifies the task of creating the Xilinx FPGA hardware design, knowledge of key aspects of an microcontroller are important to being able to run C code on the FPGA. In particular the micro blaze doesn't come setup with memory space for the instructions and data memory, same with setting up peripherals on the device. By default the soft-core processor is just a VHDL block that has no external connections. Reading the documentation that Xilinx puts out, while tedious, is very key to understanding how to setup the block design in Vivado.

Once the soft-core processor is complete, exporting the hardware design to VITIS is required to be able to use the FPGA in a similar way of using a microcontroller. VITIS allows programming of the FPGA's fabric as well as programming the soft-core processors program memory.

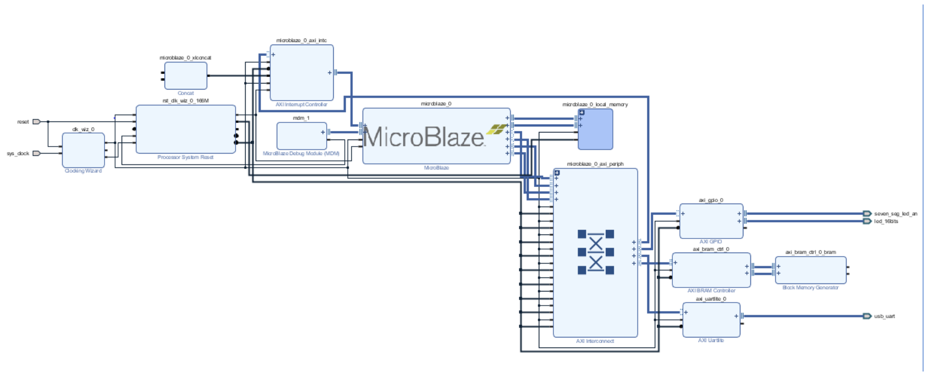


Figure 2: High level block diagram of the c code operation

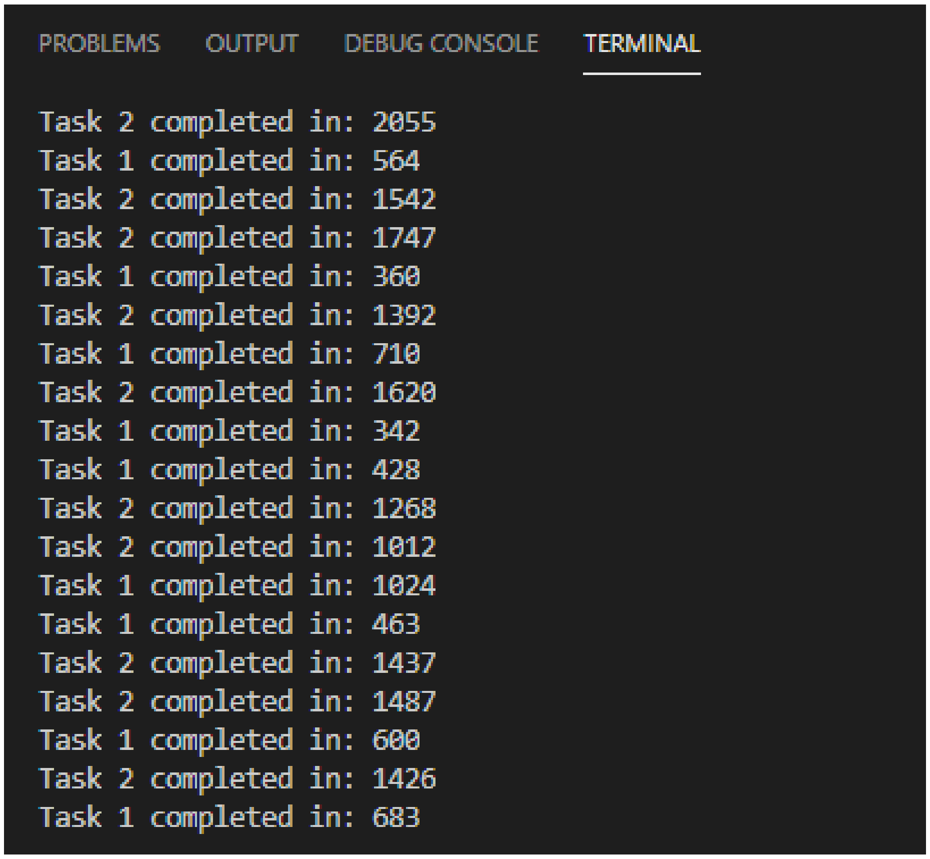
3.2 Microcontroller

When writing code for both the FPGA and the microcontrollers our group had to make sure to include some aspect of class into what we were running. On most microcontrollers the processor is a single threaded based system. So we decided to use FreeRTOS to allow scheduling of multiple tasks to be ran at once. To follow along with the design of the block diagram in Figure 2 mutexes were used to make sure only once task could access the math block at a time.

4. Results

This section discusses the results from running the test bench C/C++ code on each of the microcontrollers. As seen in Table 1 The microcontrollers did better in execution time compared to the soft-core processor on the FPGA. The first set of results were gathered by observing the terminal as seen in Figure 3. After finding that the code was running successfully a python script was ran to automatically record all data that came over the serial port for later processing in excel(google sheets). Each of the data sets was averaged with it's perspective task to get an average completion time, once this was done a metric could be established on each of the processors using their respective clock speeds. AtMega processors each had a clock speed of 16MHz so the

$$Performance\_metric = Completion\_Time/Clock\_Speed$$



```
Task 2 completed in: 2055
Task 1 completed in: 564
Task 2 completed in: 1542
Task 2 completed in: 1747
Task 1 completed in: 360
Task 2 completed in: 1392
Task 1 completed in: 710
Task 2 completed in: 1620
Task 1 completed in: 342
Task 1 completed in: 428
Task 2 completed in: 1268
Task 2 completed in: 1012
Task 1 completed in: 1024
Task 1 completed in: 463
Task 2 completed in: 1437
Task 2 completed in: 1487
Task 1 completed in: 600
Task 2 completed in: 1426
Task 1 completed in: 683
```

Figure 3: Output of the testbench data from each of the microcontrollers

Table 1: Results from a modest sample set of 200 output results on each board. The metric was found using the function earlier in this section.

Board	Clock Speed	Performance metric
Atmega2560	16	0.03304002809
Atmega328p	16	0.03220196661
Atmega23u4	16	0.03318903832
MicroBlaze	16	0.04075842763

5. Discussion

For this project a major allocation of time went to the FPGA development to ensure proper running of C/C++ on the FPGA, While the group members had a little experience with FPGA this task sat outside of previous experiences related to work/hobby/research. The C/C++ code for the microcontrollers, while the environment was different took substantially less time to complete. If we did this project again with a similar amount of time there are a few things we would approach a little differently to ease the process of development. Team 9 considers this project a excellent learning project for skill that we might be using in industry after departing MSU.

5.1 Conclusion

From the results that we gathered, it seems like choosing an microcontroller for most situations would be the proper choice. Especially when the project has a quick deadline, getting a microcontroller project together takes significantly less time due to the extra steps developing on an FPGA requires. With the microcontroller all the developer needs to know is what environment, what kind of peripherals it has and a working knowledge of programming in C. While on a FPGA, the developer needs to know all of that plus what kind of hardware to design, setup, and interface. Because the developer has such variability of design any number of things can be setup incorrect or in a flawed method.

In situations where the hardware needs to change to fit different needs it seems like FPGA performance is still within an acceptable scale that using one wouldn't hurt most project. While researching this project it seems that industry is moving towards using the best of both worlds. A system on chip (SoC) device, where there is a microcontroller that has access to FPGA fabric, allowing for a stable C/C++ hardware base (processor, memory, periphials) while being able to create and access custom hardware in the Fabric (greatly expanding the capabilities of the device.

5.1.1 Important Links

Github: https://github.com/rocky1991/CSCI460$_{final_{p}roject}$