Digital Systems Design

Report 1

By Group 9:
Johan Jino (johan.jino21@imperial.ac.uk)
Sohailul Alvi (sohailul.alvi@imperial.ac.uk)

Abstract

The need for specialized computer architectures and accelerators are growing over time due to the demise of Moore's Law and Denard Scaling. And to suffice this need, the use of FPGAs in prototyping custom RTL designs has become prominent over time. Hence, this coursework aims to give a detailed tour of the fundamentals involved in the design and development of custom digital systems on FPGAs, in order to accelerate specific computations.

Objective Overview

This first report involves the discussion around Task 1 and Task 2 from the Coursework Instructions Manual, where a digital system is designed via setting up a Nios II processor with other components. Later, after bursting the designed hardware into the DE1-SoC FPGA, the software to run on the soft-core processor is also written and uploaded onto the system.

This system designed aims to support the execution of the function (1) specified below which is written in the software uploaded to the Nios II processor.

$$y = \sum_{i=1}^{N} x_i + x_i^2 \tag{1}$$

Design Brief

Schematic in Fig 1 highlights the top-level view of the digital system designed in Quartus.

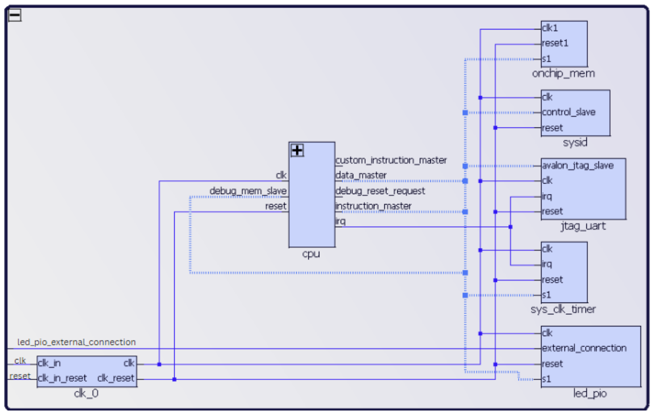


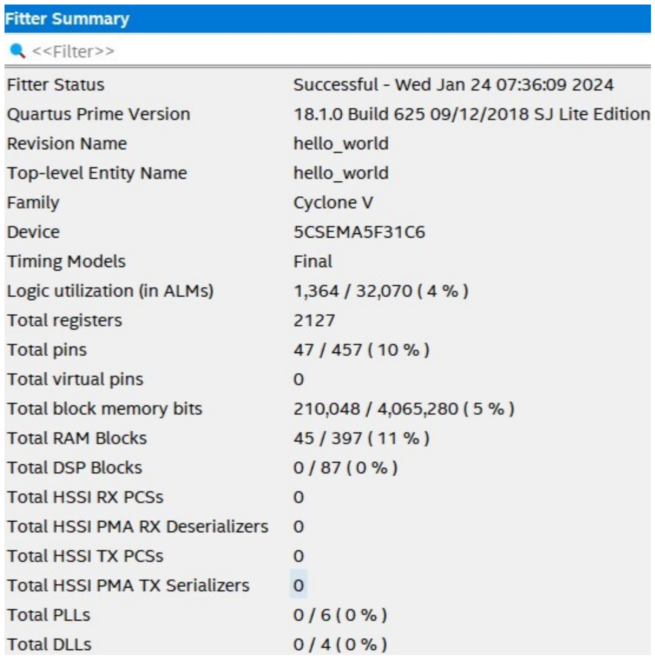
Figure 1: Quartus Generated System Schematic

The internal configurations of the components shown in the above schematic is detailed below.

- Nios II Processor: We use the Nios II/f core which is performance optimised 32 bit RISC. We instantiate it without floating point ALUs nor Multiply/Divide units.
- On-Chip Memory: We set the On-Chip Memory to RAM(Writable) type so that it can store the instructions to run the processor as well runtime data such as Global/Static Variables, Stack, Heap, Memory Mapped I/O, etc.
- JTAG/UART IP: This module is necessary to be able to program the FPGA as well as communicate with it and send data to the console.
- System Clock Timer: This timer is necessary to perform either synchronisation or act as a Real Time Clock. In the coursework this is used to count the time taken to run the designed function.
- Clk: This is the system clock which is used to control the clk input for all other modules. This sets the frequency of the entire system.
- LED PIO: This allows the NIOS II to control the LEDs in the FPGA using memory mapped IO controls. In the tasks covered in this report, we do not program to use these LEDs.

Resource Utilization

The list of FPGA resources used by the designed system shown in Figure 1 is found from the Fitter Summary within the Compilation Report generated by Quartus. The detailed resource utilization summary is shown in Figure 2 below.


Fitter Summary

<<Filter>>

Fitter Status	Successful - Wed Jan 24 07:36:09 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	hello_world
Top-level Entity Name	hello_world
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	1,364 / 32,070 (4 %)
Total registers	2127
Total pins	47 / 457 (10 %)
Total virtual pins	0
Total block memory bits	210,048 / 4,065,280 (5 %)
Total RAM Blocks	45 / 397 (11 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 2: Resource Utilization Summary

Using the proposed metric for FPGA resource utilization measurement, that is the weighted sum of the utilized resources, it can be written for the system:

$$\text{FPGA Resources} = \frac{1}{4}\left\{\frac{\text{ALMs Used}}{\text{Total ALMs}} + \frac{\text{Pins Used}}{\text{Total Pins}} + \frac{\text{Memory Bits Used}}{\text{Total Memory Bits}} + \frac{\text{RAM Blocks Used}}{\text{Total RAM Blocks}}\right\} \tag{2}$$

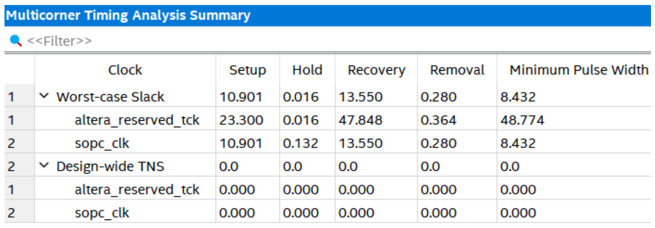
Following the above equation (2), and picking the values from the resource utilization summary above in Figure 2, it is found:

$$\text{FPGA Resources} = \frac{1}{4}\left\{\frac{1,364}{32,070} + \frac{47}{457} + \frac{210,048}{4,065,280} + \frac{45}{397}\right\} = 0.078 \tag{3}$$

So, the FPGA resource utilization for the digital system designed as in Figure 1 is approximately 7.8%.

Timing Analysis

In the compilation process of the RTL design, it is crucial to assess the Timing Analyzer on Quartus for insights into the design's timing characteristics. Within this, the Multicorner Timing Analysis Summary table as in the figure 3 below, provides a detailed overview of the timing analysis resukst across different corners.


Multicorner Timing Analysis Summary

<<Filter>>

	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	10.901	0.016	13.550	0.280	8.432
1	altera_reserved_tck	23.300	0.016	47.848	0.364	48.774
2	sopc_clk	10.901	0.132	13.550	0.280	8.432
2	Design-wide TNS	0.0	0.0	0.0	0.0	0.0
1	altera_reserved_tck	0.000	0.000	0.000	0.000	0.000
2	sopc_clk	0.000	0.000	0.000	0.000	0.000

Figure 3: Multicorner Timing Analysis Summary

The key aspect to verify is the Worst-case Slack values, for Setup, Hold, Recovery and Reomval. Positive values for these parameters indicate that the design adheres to the required timing constraints, whereas negative values signal potential timing violations that may compromise hardware functionality.

As in Figure 3, the digital system designed has all those parameters positive, indicating there is no need for any further actions and that the design meets its timing constraints.

Program Size and Memory Footprint

The size of the Nios II application is the size of the code written, plus the size of all the initialized data in the code file, generated via Eclipse. Generating the program for all the 3 given test cases, it is found that the application sizes do not vary with the increasing size of the vectors defined in the code. The application file sizes for the test cases were found to be:

Test Case 1:	14 KB
Test Case 2:	14 KB
Test Case 3:	14 KB

The reason why the program sizes do not change with varying test cases is that the only change we have among all these tests is the defined values of **N** and **Step** which remain of the same data type. Also, for each case, there was a consistent addition of 4540 bytes of memory space for stack and heap allocations.

Then, to check for the maximum memory required for each test case, the on-chip memory size has been increased in some arbitrary amounts. The tests were found to running successfully at the below-specified memory sizes:

Test Case 1:	20 KB
Test Case 2:	60 KB
Test Case 3:	.....

The test case 3 could not run in any of the increasing memory configurations, reaching the maximum available on-chip memory on the FPGA. It indicates that, even the maximum available on-chip memory is not enough to store and use the vector formed in test case 3. Hence, it demands the need to add external memory to the system to hold this large vector.

Test Case Analysis

We were provided with 3 test cases. For all the test-cases we run multiple iterations of the same calculation and then observe the average number of ticks taken to calculate them.

1. $Step = 5, N = 52$
   This test case passed with around 0-1 ticks on average. Since these only included ints, it was easier for processor to calculate them since inbuilt instruction support is available for these calculations. There was no deviation from the answers calculated from Python 3.12.

2. $Step = 1/8.0, N = 2041$
   This test case passed with around 25-27 ticks on average. These calculations involved. There was significant deviation from the answers calculated from Python. This is because our Nios II processor has be made to not support floats. Hence, the float arithmetic is emulated as well as stored only in single precision, hence a lot a precision is lost here.

3. $Step = 1/1024.0, N = 261121$
   This test case was never able to run due to the large memory requirement required by the vector for size 261121. Since each floats are stored in single precision, this equates to $4bytes * 261121 \approx 1MB$ of memory which is unavailable in the on-chip memory. Therefore, this task cannot be performed without larger off-chip memory.

It was also noted that average over multiple iterations increased the memory requirements for the program to execute, hence we also made sure to run a single iteration calculation to ensure that our average calculation overhead was not a limiting factor.

System Performance

1 tick from the system clk refers to 1ms time. This was setup in platform designer.

1. $Step = 5, N = 52$
   This test case passed with around 0-1 ticks on average. Therefore the latency on average was between 0 to 1 millisecond.

2. $Step = 1/8.0, N = 2041$
   This test case passed with around 25-27 ticks on average. Therefore the latency on average was between 25-27 millisecond.

3. $Step = 1/1024.0, N = 261121$
   This test case unsuccessful since there was insufficient memory to run this program.

Discussion on Cache and Compiler Flags

Cache is not beneficial for the designed system. Since only on-chip memory is used, adding cache is only going to provide little to no benefit. Although, it was observed instruction cache was reducing the latency of the program. This is likely because as the cache size increases, it is likely to be more pipelined. This added to the fact that compiler (with optimisation flag) will unroll small loops to reduce control overheads, hence making code more serialized and giving good i-cache performance. This is a possible reason for the unexpected performance gain.

Conclusion

In this phase of the coursework, involving tasks 1 and 2, we have learned to design and test custom RTL IPs on Intel FPGAs. Further, we learnt to investigate the underlying architecture, and profile systems to calculate the expected performance and memory utilization of any software being run on the designed digital system.