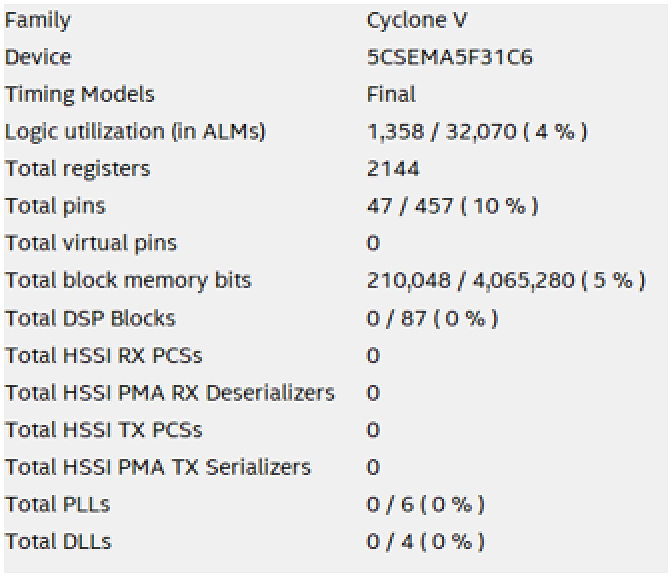
Digital Systems Design Report 1

Yeong Kyun Han
Benjamin Tan Jun Yi
Group number: 26

Initial Configuration

The initial configuration uses a 20KB on chip memory, with a 2KB instruction cache and a 2KB data cache. The resources used are shown in *Figure 1.*



Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	1,358 / 32,070 (4 %)
Total registers	2144
Total pins	47 / 457 (10 %)
Total virtual pins	0
Total block memory bits	210,048 / 4,065,280 (5 %)
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 1

This configuration was able to compile all the tests, but could not correctly execute test 2 or 3. This is because the on chip memory is not enough to store the arrays needed to execute test 2 and 3. It is also noted that the jtag uart and nios2 processor utilize 5776 bytes of memory blocks.

Memory Requirements

The next part investigated is the size of program so the usage of on chip memory can be optimized.

Application	size(bytes)	
	Base Program	With timer/prints
Test1	11076	14356
Test2	11076	14356
Test3	11112	14392

Figure 2

-The information in *Figure 2* is found using the nios2-elf-size command.

It is shown that the static memory requirements of test 1 and test 2 are the same which is expected because the program ran is the same, but test 3 shows a slight increase in the file size, this seems to be due to the increase in constant N which is the size of the array

The dynamic memory requirements are then found as in *Figure 3.*

	Test1	Test2	Test3
Array Size(bytes)	208	8164	1044484

Figure 3

It is shown that test 3 requires 1.04MB of dynamic memory, whereas the FPGA only has 570KB of block memory, hence only test 1 and 2 will be considered in the next FPGA configurations since it is impossible to run test 3 using only on chip memory.

-24200bytes on chip memory used for all proceeding tests

Floating point vs double precision

	result		time(ticks)		Static Memory used	
	Floats	Doubles	Floats	Doubles	floats	doubles
test1	1117	1117	0.73	1.24	15308	19000
test2	43466	43466	30	35	15308	19000

Figure 4

Test1 was ran a 100 times and averaged because the execution time of an individual run was too small to measure.

Figure 4 shows the results and execution time of test 1 and 2 when using floats or doubles. There is no loss in the result precision, but the execution time and static memory both increase, showing that there is no benefit to using doubles for the program executed. The increase in execution time and static memory utilized is expected since doubles results in increased memory accesses and larger initialized data, and also because hardware does not have a floating point implementation. It is worth noting that the dynamic memory required also doubles.

The results shown are consistent with the results when the program is tested using python.

Cache configurations

The instructions and data cache sizes were then tuned to investigate its effect on the system

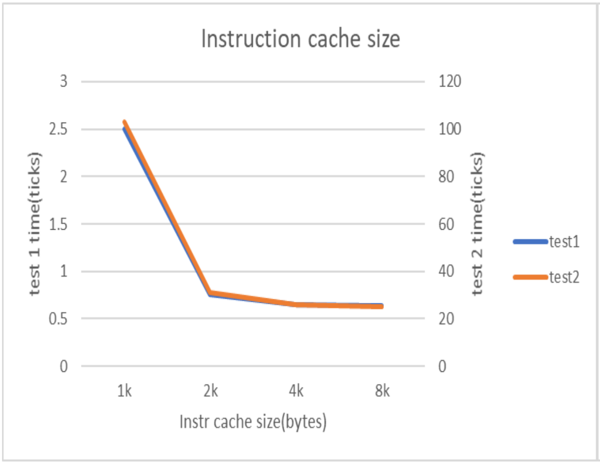


Figure 5

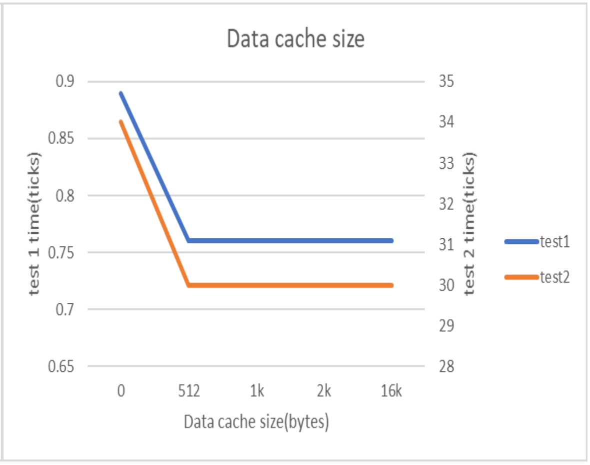


Figure 6

Figure 5 shows that increasing the instruction cache size from 1k to 2k byte results in a 70% decrease in execution time, but there is no further improvement when increasing the cache size, this makes sense since the *sumvector* function which is the critical loop in the program uses 1232 bytes of static memory as shown in the table below. There also seems to be a small decrease in execution time when further increasing the instruction cache size to 4k bytes, this is likely due to timing functions, but is difficult to investigate further.

	size(bytes)
With *sumvector* function	14312
Without *sumvector* function	13080

Figure 7

Figure 6 shows that the configuration without a data cache has a 10% increase execution time relative to any of the other data cache sizes. This makes sense because the *generateVector* function is the only section of the code in which memory accesses are repeated, and each data address only needs to be accessed once before it can be flushed from the cache, meaning that only 2 floats or 64bits need to be stored in the data cache at any point in time.

```
for (i=1; i<N; i++)
    x[i] = x[i-1] + step;
```

The effect of the data cache is small because the data cache and on chip memory utilize the same underlying hardware blocks, there is no difference in physical access latency, the effect on execution time is only due to architectural differences where on chip memory might require extra overheads to access the data.

Optimal/Final Configuration

A metric for the total FPGA resources used was found using the following formula:

$$\text{FPGA resources} = \frac{1}{3}\left(\frac{LE}{Total\ LE\ in\ the\ device} + \frac{EM}{Total\ EM\ in\ the\ device} + \frac{MB}{Total\ MB\ in\ the\ device}\right)$$

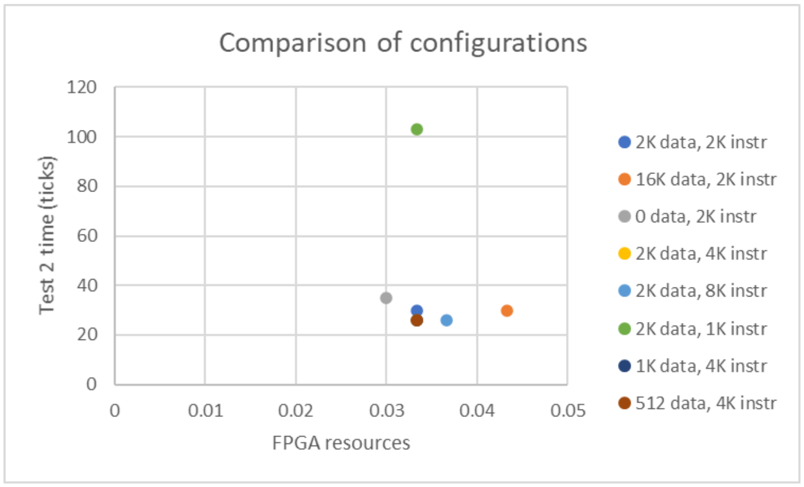


Figure 8

Figure 8 plots the execution time against the FPGA resources utilized for all the tested configurations. The optimal configuration uses 24200 bytes of on chip memory, a 512 byte data cache and a 4k byte instruction cache, utilizing the least resources for the best achieved performance.

test1(ticks)	test2(ticks)
0.65	27

Figure 9

Figure 9 shows the optimal execution times found from the above tests.

Conclusion

It is found that the optimizations to the design should consider the specific program and that increasing the resources utilized does not necessarily result in an increase in performance. The configuration should also consider the trade-off between resources used and performance, it might be better to pick a worse performing configuration which allows the use of resources for different purposes.