EE3-05 Digital System Design

Coursework Report 2: Task 3-5

*Abstract*— This is the second report out of three for EE3-05 Digital System Design’s (2015-2016) coursework of implementing, then accelerating a mathematical algorithm on an Altera Cyclone III FPGA. This second report will cover implementing an off-chip SDRAM to increase the amount of data that our test bench can process. A short section on benchmarking the performance of our system will be included

|  |  |
| --- | --- |
| Jeremy Chan\*  Department of Electrical and Electronic Engineering  Imperial College  London, United Kingdom  [jc4913@ic.ac.uk](mailto:jc4913@ic.ac.uk) | 00818433 | Chak Yeung Dominic Kwok\*  Department of Electrical and Electronic Engineering  Imperial College  London, United Kingdom  [cyk113@ic.ac.uk](mailto:cyk113@ic.ac.uk) | 00827832 |

\*These authors contributed equally to this work

Table of Contents

[I. Introduction 3](#_Toc443052132)

[II. System Design 3](#_Toc443052133)

[A. Generating the QSys system 3](#_Toc443052134)

[B. Compiling the Quartus system 3](#_Toc443052135)

[C. Creating the Eclipse project 3](#_Toc443052136)

[D. Removing the On Chip Memory 3](#_Toc443052137)

[III. Hardware Configuration for Testing 4](#_Toc443052138)

[A. Task 3 4](#_Toc443052139)

[B. Task 4 4](#_Toc443052140)

[C. Task 5 4](#_Toc443052141)

[IV. Results 4](#_Toc443052142)

[A. Latency, with variable cache size 4](#_Toc443052143)

[B. FPGA Resource Usage 5](#_Toc443052144)

[C. NIOS II .elf size 6](#_Toc443052145)

[D. Program Output 6](#_Toc443052146)

[E. Throughput 7](#_Toc443052147)

[V. Discussion of Results 7](#_Toc443052148)

[A. Ability to compile for different test cases 7](#_Toc443052149)

[B. Instruction Cache 8](#_Toc443052150)

[C. Comparison between single precision and double presicion 11](#_Toc443052151)

[VI. Conclusion 11](#_Toc443052152)

[VII. References 12](#_Toc443052153)

[VIII. Appendix 12](#_Toc443052154)

[A. Quartus Block Design File 12](#_Toc443052155)

[B. QSys System Connections 12](#_Toc443052156)

[C. Testbench Functions 12](#_Toc443052157)

[D. Testbench main() Exercept 13](#_Toc443052158)

# Introduction

To be written.

1. Remove on chip mem to test task 3, justify the decision
2. Reconfirm driver setting on task 3,4,5
3. Record the FPGA resource on task 3 design without on chip mem
4. Check PLL clk0/1 reverse?
5. Test all case on task 3 without on ship memory
6. Perform cache size test on task 3,4,5 from 0 to 32KB
7. Table for cache test and plot graph(time vs cache in range case 1-3)
8. Throughput -> conclusion

# System Design

## Generating the QSys system

The QSys system was generated as specified (8MB SDRAM), with a minor change in the clock arrangement of the PLL – instead of connecting the delayed clock to the SDRAM Controller, it was connected to the SDRAM to ensure the NIOS II system clock led the SDRAM clock by 3ns.

From the discussion of results in section XXXX, the on-chip memory was removed. The LED PI/O was also not used by our design and hence removed as well. The QSys system is shown in Appendix.B.

## Compiling the Quartus system

The Quartus system was generated as specified from the QSys system in section II.A. The PLL was moved to within the QSys project to reduce clutter in the board design file. The SDRAM components were also connected to the SDRAM Controller. As the LED PI/O was unused, it was removed and hence left unconnected; Quartus will automatically remove all unconnected wires during compilation anyway.

## Creating the Eclipse project

The Eclipse project was generated as specified, with the small C library switched off. From the previous report, enable\_lightweight\_device\_device\_driver\_api was also turned back on as there was now enough memory. C++ support was not turned on as there C supported our testbench capably.

## Removing the On Chip Memory

After the discussion of results in report 1, the on-chip memory was removed to test if this affected latency.

Table II shows the latency before and after removing the on-chip memory. The configuration used for the test case was as follows:



|  |  |
| --- | --- |
| Configuration | Value |
| Algorithm Used |  |
| Hardware Multiplication | None |
| Instruction Cache Size | 2KB |
| Test Case | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| Ticks | Case 1 | Case 2 | Case 3 |
| Before Removal | 6 | 282 | 28221 |
| After Removal | 5 | 281 | 28148 |

Since the latency improves after removing the on-chip memory, all further testing was done using this configuration. This also saves on system resources, as seen in the Table III.

|  |  |  |
| --- | --- | --- |
| Resource | With on-chip memory | Without on-chip memory |
| Logic Elements | 3207 | 2999 |
| Embedded Multiplier | 0 | 0 |
| Memory Bits | 422272 | 29056 |

# Hardware Configuration for Testing

## Task 3

|  |  |
| --- | --- |
| Configuration | Value |
| Algorithm Used |  |
| Hardware Multiplication | None |
| Instruction Cache Size | Variable |
| On-Chip Memory | None |

## Task 4

|  |  |  |
| --- | --- | --- |
| Configuration | 1 | 2 |
| Algorithm Used |  |  |
| Hardware Multiplication | None | |
| Instruction Cache Size | Variable | |
| On-Chip Memory | None | |

## Task 5

|  |  |  |
| --- | --- | --- |
| Configuration | 1 | 2 |
| Algorithm Used |  |  |
| Hardware Multiplication | Embedded Multiplier, Logic Elements | |
| Instruction Cache Size | Variable | |
| On-Chip Memory | None | |

# Results

## Latency, with variable cache size

Latency was measured using the system clock. The output is in ticks, defined in system.h.

From system.h, ticks are defined as

#define TIMER\_TICKS\_PER\_SEC 1000.0

Hence, 1 tick = 1 ms.

The cache size was also varied to investigate the result of changing the instruction cache size on latency. The lowest amount of cache tested was 2KB; the max 32KB. 64KB was not tested as there was not enough memory bits on the board to accommodate 64KB of instruction cache.

#### Task 3



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cache Size | 2KB | 4KB | 8KB | 16KB | 32KB |
| Case 1 | 5 | 4 | 4 | 4 | 4 |
| Case 2 | 254 | 215 | 213 | 213 | 213 |
| Case 3 | 26808 | 21552 | 21329 | 21329 | 21330 |

#### Task 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cache Size | 2KB | 4KB | 8KB | 16KB | 32KB |
| Case 1 | 78 | 74 | 59 | 57 | 56 |
| Case 2 | 3823 | 3636 | 2963 | 2825 | 2800 |
| Case 3 | 382038 | 362601 | 296649 | 282920 | 288024 |

#### Task 5

Task 5 required the testing of three different configurations of hardware multiplication.

Table VI uses no hardware multiplication. Therefore, it must be identical to Task 4 as they use the same hardware.

Table VII uses Embedded Multipliers.

Table VIII uses Logic Elements.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cache Size | 2KB | 4KB | 8KB | 16KB | 32KB |
| Case 1 | 78 | 74 | 59 | 57 | 56 |
| Case 2 | 3823 | 3636 | 2963 | 2825 | 2800 |
| Case 3 | 382038 | 362601 | 296649 | 282920 | 288024 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cache Size | 2KB | 4KB | 8KB | 16KB | 32KB |
| Case 1 | 55 | 41 | 30 | 27 | 27 |
| Case 2 | 2272 | 2067 | 1482 | 1366 | 1356 |
| Case 3 | 227074 | 206562 | 148078 | 136506 | 136496 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cache Size | 2KB | 4KB | 8KB | 16KB | 32KB |
| Case 1 | 46 | 42 | 30 | 28 | 28 |
| Case 2 | 2299 | 2094 | 1509 | 1393 | 1384 |
| Case 3 | 229874 | 209334 | 150830 | 139247 | 138379 |

## FPGA Resource Usage

Table IX denotes our resource usage on the Cyclone III FPGA, after the addition of the SDRAM, and after compilation and fitting using Quartus.

Table X changes the Hardware Multiplication from None to Embedded Multipliers, otherwise identical to Table IX.

Table XI changes the Hardware Multiplication from None to Logic Elements, otherwise identical to Table IX.

Resource usage was extracted from the Quartus compilation report – expressed in terms of Logic Elements (LE), Embedded Multipliers (EM), and Memory Bits (MB). The DE0 board consists of a total of 15408 LE’s, 112 EM’s, and 516096 MB’s.

The equation to quantitate these measurements is expressed in terms of resource usage (RU) (lower means less resources used):



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cache Size | 2KB | 4KB | 8KB | 16KB | 32KB |
| LE | 2994 | 2991 | 2993 | 2991 | 2989 |
| EM | 0 | 0 | 0 | 0 | 0 |
| MB | 29056 | 46720 | 81920 | 152064 | 291840 |
| RU | 0.084 | 0.095 | 0.118 | 0.163 | 0.253 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cache Size | 2KB | 4KB | 8KB | 16KB | 32KB |
| LE | 3214 | 3218 | 3218 | 3216 | 3218 |
| EM | 4 | 4 | 4 | 4 | 4 |
| MB | 29056 | 46720 | 81960 | 152064 | 291840 |
| RU | 0.100 | 0.112 | 0.134 | 0.180 | 0.270 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cache Size | 2KB | 4KB | 8KB | 16KB | 32KB |
| LE | 3345 | 3348 | 3343 | 3343 | 3353 |
| EM | 0 | 0 | 0 | 0 | 0 |
| MB | 29056 | 46720 | 81960 | 152064 | 291840 |
| RU | 0.091 | 0.103 | 0.125 | 0.171 | 0.261 |

## NIOS II .elf size

TABLE XII shows the size of the initialized array (consisting of 32 bit floats) for different test cases.

The size of the compiled .elf was obtained using the command ‘nios2-stackreport’ using the NIOS II Command Shell. ‘nios2-stackreport’ is advantageous to ‘nios2-elf-size’ as it provides a breakdown of the total size.

Since tasks 4 and 5 used the same algorithm, their .elf size was the same. Hence, only tasks 3 and 4 will be shown – the differences get negligible at large array sizes.



|  |  |  |  |
| --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 |
| Array Size | 52 | 2551 | 255001 |
| Compiled Size | 208B | 9.96KB | 996KB |

#### Task 3



|  |  |  |  |
| --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 |
| Program Size | 38KB | 48KB | 1035KB |
| Free Size for stack+heap | 8151KB | 8141KB | 7155KB |

#### Task 4/5

|  |  |  |  |
| --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 |
| Program Size | 66KB | 76KB | 1063KB |
| Free Size for stack+heap | 8122KB | 8113KB | 7126KB |

## Program Output

Since Task 4 and 5 only differ by hardware implementation, their results are the same.

#### Task 3

#### Task 4/5



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Implementation (type in/out) | MATLAB (double/double) | NIOS II (double/double) | NIOS II (double/float) | NIOS II (float/double) | NIOS II (float/float) | NIOS II (original) |
| Case 1 | 57879.9 | 57879.9 | 57879.9 | 57879.9 | 57879.9 | 57879.9 |
| Case 2 | -126818.14 | -126818.14 | -126819.41 | -76972.44 | -76973.14 | -76973.39 |
| Case 3 | -12774366.3 | -12774366.29 | 37022500 | -12774366.38 | 37022500 | 37022532 |

## Throughput

The throughput was measured in terms of results per second, then converted to bytes per second. The array consists of 32-bit floats. As previous results showed that using an instruction cache size of 32KB, the following throughput results use an instruction cache size of 32KB.

#### Task 3

|  |  |  |  |
| --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 |
| Array Size | 52 | 2551 | 255001 |
| Latency (ms) | 4 | 213 | 21330 |
| Throughput (Bps) | 3328.00 | 3065.99 | 3060.49 |

#### Task 4

|  |  |  |  |
| --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 |
| Array Size | 52 | 2551 | 255001 |
| Latency (ms) | 56 | 2800 | 288024 |
| Throughput (Bps) | 237.71 | 233.23 | 226.65 |

#### Task 5

|  |  |  |  |
| --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 |
| Array Size | 52 | 2551 | 255001 |
| Latency (ms) | 27 | 1356 | 136496 |
| Throughput (Bps) | 493.04 | 481.60 | 478.26 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 |
| Array Size | 52 | 2551 | 255001 |
| Latency (ms) | 28 | 1384 | 138379 |
| Throughput (Bps) | 475.43 | 471.86 | 471.75 |

# Discussion of Results

## Ability to compile for different test cases

As there is now an 8MB SDRAM instead of a 48KB on-chip memory, all 3 test cases compile when the array is defined as a global variable – the check for memory is done at compile time.

## Instruction Cache

Increasing the size of the instruction cache size was shown to affect the latency. All graph results are normalized – the longest latency is defined to have a normalized latency of 1. Hence, a smaller normalized latency is better. All data labels have been normalized to 3d.p.

#### Task 3

Figure 1: Effect of instruction cache size on latency

In Figure 1, the minimum latency achieved comes from using an instruction cache size of 8KB or more. There is no improvement from using a cache size larger than 8KB. Hence, the compiled instructions of task 3 should be between 4KB and 8KB in size.

#### Task 4

Figure 2: Effect of instruction cache size on latency

In Figure 2, the minimum latency comes from using an instruction cache size of 32KB. Hence, the compiled instruction size is larger than 24KB. Further testing is needed to determine an upper bound. Using 64KB of instruction cache is not possible due to memory bit restrictions of the DE0 board. The DE0 has 63KB of memory.

#### Task 5 w/ Embedded Multipliers

Figure 3: Effect of instruction cache size on latency

In Figure 3, there are several points to note. Since the software implementation has not changed, using a cache size of 32KB is expected to have the minimum normalized latency. However, the latency of case 1 when using 16KB and 32KB of instruction cache size was 27ms and 27ms respectively. Case 1’s test case is too small – our latency measurement’s resolution is 1ms. Case 2 and 3 also have very tiny changes between 16KB and 32KB. Full results can be found in the appendix; the difference between 16KB and 32KB are too small to be readily visible.

#### Task 5 w/ Logic Elements

Figure 4: Effect of instruction cache size on latency

In Figure 4, the change is even less apparent. However, albeit small, there is a decrease in latency between 16KB and 32KB for all 3 test cases.

#### Conclusion of instruction cache size testing

Figure 5: Different Hardware Configuration’s Effect on Latency

Figure 5 shows all these results put together. The minimum normalized latencies all come from the diamond marker – Embedded Multipliers, for all test cases. Hence, if performance is prioritized, the hardware configuration to be used should be Embedded Multipliers.

## Comparison between single precision and double presicion

#### Task 3 Summation

#### Task 4-5 Summation

From the last report, a difference between the precision of floats and doubles was shown.

These errors can be minimized by using higher precision types. For example, using double will allow for a 64-bit representation of the decimal number as opposed to the 32-bit float.

# Conclusion

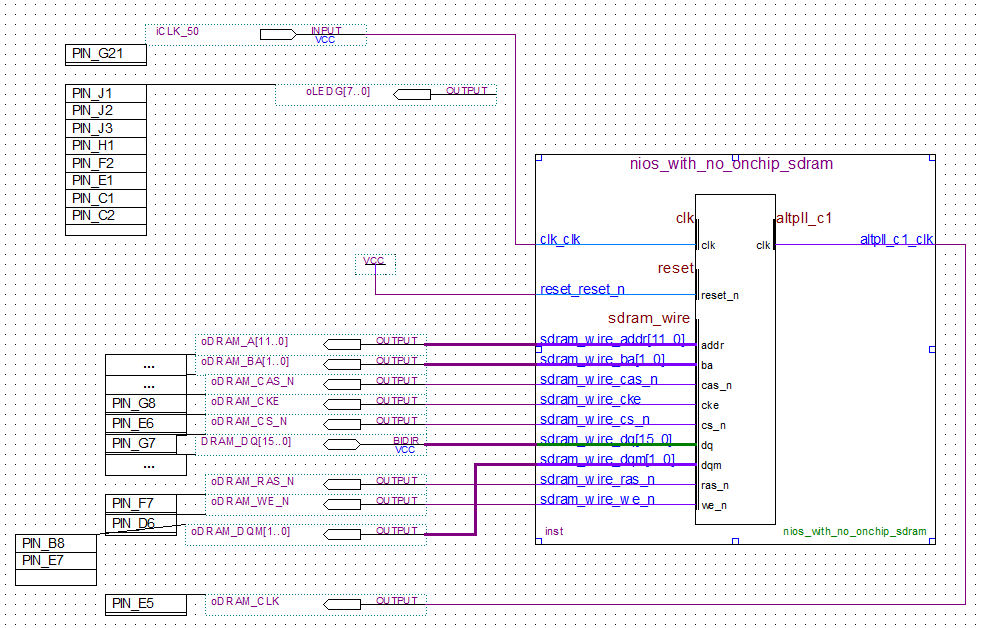
From the results and the discussions about the results, our conclusion is that the optimal configuration for our NIOS II project is the following:

|  |  |  |
| --- | --- | --- |
| Setting | Value | Reason |
| On Chip Memory | Removed | Reduces latency, reduces resources |
| NIOS II | Hardware Multiplier = Embedded Multipliers, Instruction cache = 32KB | Reduces latency at the expense of resources |
| PLL | clk0 to SDRAM controller, clk1 to SDRAM | SDRAM controller should have the earlier clock |
| JTAG | As specified | N/A |
| Timer | As specified | N/A |
| System ID | As specified | N/A |
| LED PI/O | Removed | Unused |
| NIOS II BSP | Reduced device drivers = 1, else 0 | C++ support is not needed, small C library turned off for printf() |

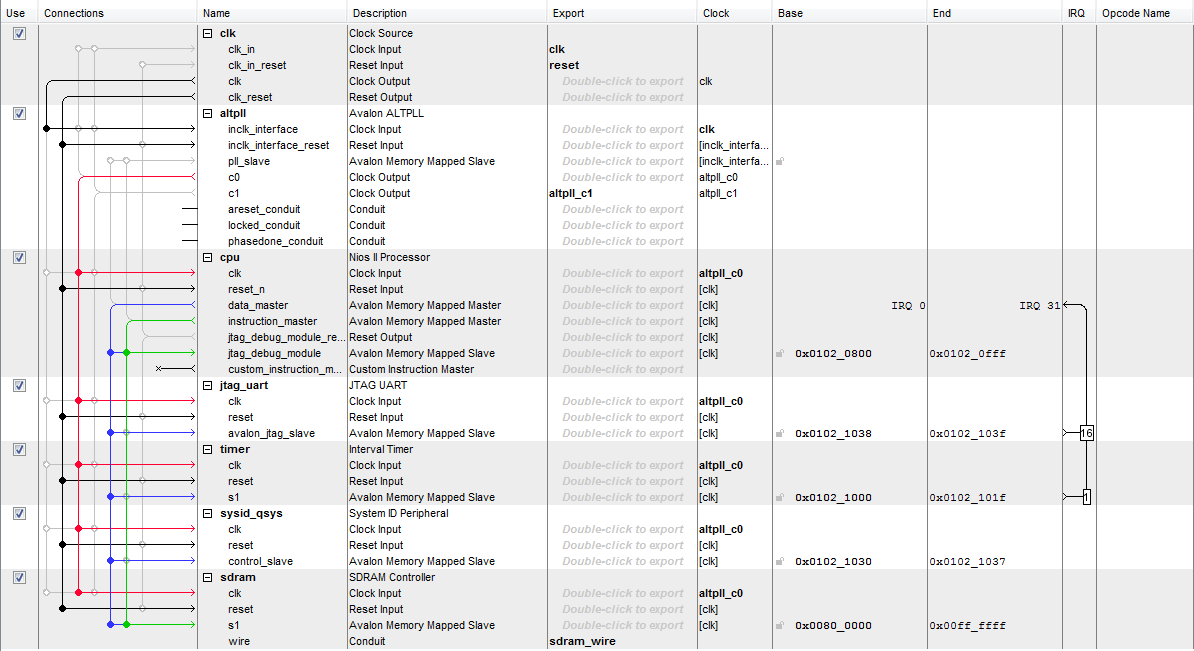
# References

# Appendix

## Quartus Block Design File



## QSys System Connections



## Testbench Functions

void generateVector(float\* x, unsigned int n, float s)

{

int i;

x[0] = 0;

**for** (i=1; i<n; i++){

x[i] = x[i-1] + s;

}

}

float sumVector(float x[], unsigned int M)

{

float result=0;

int i=0;

**for** (i=0; i<M; i++){

result += (x[i] + x[i]\*x[i]);

}

**return** result;

}

float sumVector\_cos(float x[], unsigned int M)

{

float result=0;

int i=0;

**for** (i=0; i<M; i++){

result += (x[i]/2) + (x[i]\*x[i]\*cosf(floor(x[i]/4)-32));

}

**return** result;

}

## Testbench main() Exercept

#define task switches between task 1-3’s and 4+’s algorithm. #define div\_1024 switches between whether the end result is divided by 1024.

printf("Running algorithm for task: ");

printf((task)?"4+**\n**":"1-3**\n**");

printf("Dividing result by 1024 is: ");

printf((div\_1024)?"ON**\n**":"OFF**\n**");

*////////////////*

printf("gen vector 1, ");

generateVector(x1, N1, S1);

printf("sum vector 1**\n**");

float y1=0.0;

**if** (task) {t1 = times(NULL); y1 = sumVector\_cos(x1, N1); t2 = times(NULL);}

**else** {t1 = times(NULL); y1 = sumVector(x1, N1); t2 = times(NULL);}

gcvt(t2-t1, 10, a);

alt\_putstr("Time = "); alt\_putstr(a); alt\_putstr("; ");

printf("Result = %f**\n**", ((div\_1024)?y1/1024:y1));