EE3-05 Digital System Design

Coursework Report 1: Task 1-2

Jeremy Chan*
Department of Electrical and Electronic Engineering
Imperial College
London, United Kingdom
jc4913@ic.ac.uk

Chak Yeung Dominic Kwok*
Department of Electrical and Electronic Engineering
Imperial College
London, United Kingdom
cyk113@ic.ac.uk

*These authors contributed equally to this work

***Abstract*— This is the first report out of four for EE3-05 Digital System Design's (2015-2016) coursework of implementing, then accelerating a mathematical algorithm on an Altera Cyclone III FPGA. This first report will cover the first two tasks out of eight; the creation of the soft core NIOS II processor system to be synthesized using Altera's Quartus tool. A short section on benchmarking the performance of our system will be included.**

I. Introduction

The first two tasks revolved around familiarising ourselves with the toolchain, and executing a simple addition on a vector, using the NIOS II soft processor. Tests were done to benchmark the code in terms of time required to sum the vector, and the precision at which it did so, when compared to MATLAB.

II. System Design

A. *Generating the QSys system*

The QSys system was generated as specified, with a minor change in the on-chip memory, where the memory size was set to the maximum of 49152 bytes.

B. *Compiling the Quartus system*

The Quartus system was generated as specified; the block diagram is shown in Fig.1.

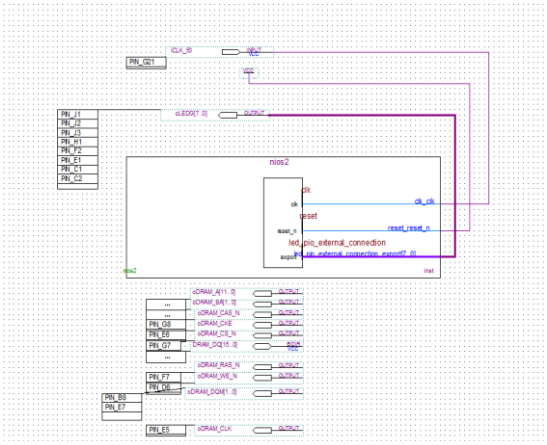

Fig. 1. Quartus .bdf configuration

C. *Creating the Eclipse project*

The Eclipse project was generated as specified, with a minor change in the BSP settings (see section IV.A).

III. Results

A. *FPGA Resource Usage*

Table I denotes our resource usage on the Cyclone III FPGA, after compilation and fitting using Quartus.

TABLE I.

	Logic Elements	Embedded Multipliers	Memory Bits
Used	2350	0	421824
Total	15408	112	516096
Ratio	0.15	0	0.82

The equation to quantitate these measurements is

$$\frac{1}{3}\left(\frac{LE}{Total\ LE}+\frac{EM}{Total\ EM}+\frac{MB}{Total\ MB}\right)=0.323$$

B. *Latency*

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.

TABLE II.

	Case 1	Case 2	Case 3
Ticks	3	165	N/A

Case 3 failed to compile as an array of 255001 floats did not fit in our design. This will be discussed in section IV.A.

C. *NIOS II .elf size*

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.

TABLE IV shows the size of the initialized array (consisting of 32 bit floats) for different test cases. This size limited the ability to compile, discussed in section IV.A.

TABLE III.

	Case 1	Case 2	Case 3
Program Size	32KB	42KB	N/A
Free Size for stack+heap	14KB	4692B	N/A

TABLE IV.

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

D. *Program Output*

The output of our summation function was printed using gcvt as opposed to printf to save space. Our compiled program used floats; a discussion on the accuracy of this when compared to a double precision implementation using MATLAB will be provided in section IV.C.

TABLE V.

	Case 1	Case 2	Case 3
FPGA Output (float)	1117.949219	54326.503910	N/A
MATLAB Output (double)	1117.949219	54325.214111	N/A
Absolute Error	0	1.289799	N/A

E. *Throughput*

The throughput was measured in terms of results per second, then converted to bytes per second. The array consists of 32-bit floats.

TABLE VI.

	Case 1	Case 2	Case 3
Array Size	52	2551	N/A
Latency (ms)	3	165	N/A
Throughput (Bps)	69333.33	61842.42	N/A

F. *Cache Size*

The instruction cache size of the NIOS II CPU was also changed to see the effect of the cache size (top left cell in TABLE VII, VIII, IX) on the operating time. See section IV.C for the Kahan summation.

TABLE VII.

512B	Case 1	Case 2	Case 3
Normal Summation (ms)	4	189	N/A
Kahan Summation (ms)	5	304	N/A

TABLE VIII.

1KB	Case 1	Case 2	Case 3
Normal Summation (ms)	3	174	N/A
Kahan Summation (ms)	5	276	N/A

TABLE IX.

2KB (default)	Case 1	Case 2	Case 3
Normal Summation (ms)	3	165	N/A
Kahan Summation (ms)	5	265	N/A

IV. Discussion of Results

A. *Ability to compile for different test cases*

The array of floats, previously defined inside the testbench of `main()`, was moved to outside main() to allow for the NIOS II compiler to take the size of the array into account when compiling. This move allowed us to see an direct relationship between array and program code size in the `.elf`.

Since our QSys system instantiated an on-chip memory of size 49152 bytes, the absolute maximum size of the 32-bit floats array we can fit, ignoring size of all other code, is 12288. Immediately, case 3 was ruled out of all measurements. When taking into account size of the BSP and surrounding test bench code, Eclipse reported 38KB in code (BSP + test bench) size when no array was defined. This meant the max array size was 10KB – 2500 elements – test 2 initially failed to compile.

As the test bench was executing simple functions, the BSP option of `enable_lightweight_device_device_driver_api` was turned on. This allowed Eclipse to reduce code size by around 6KB, and hence test 2 was able to compile.

B. *Instruction Cache*

Reducing the size of the instruction cache size was shown to affect the latency. However, the NIOS II's instruction cache master port can issue one read per clock cycle [1], adding a cache layer between the core and the SRAM memory will add further delay.

A test for latency with no instruction cache would be ideal to test this hypothesis, however, QSys fails to compile with no instruction cache.

Furthermore, the data cache is only effective if all data and instructions are in off-chip memory, along with satisfying the following criteria [2]:

- Off-chip memory access times are long compared to on-chip memory
- The largest bottleneck (instructions or data) fits inside the cache in its entirety

C. *Comparison with MATLAB implementation*

Most decimal numbers cannot be implemented exactly in binary [3]; using float or double in C will use the IEEE-754 method of implementing decimal numbers. For example, the step size of case 2 cannot expressed exactly. Table VI shows this phenomenon. Therefore, when generating and summing the array, there will be cumulative rounding errors.

TABLE X.

	Case 1 (5)	Case 2 (0.1)	Case 3 (0.001)
Float decimal	5.0000000	9.9999994e-2	9.9999993e-4
Float	0x40A00000	0x3DCCCCCC	0x3A83126E
Double decimal	5.0000000000000000	1.0000000000000000e-1	1.0000000000000000e-3
Double	0x4014000000000000	0x3FB999999999999A	0x3F50624DD2F1A9FC
Error (float)	0	6e-9	7e-11

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.

Alternatively, there is a method of summing a series of floating point numbers called the Kahan Summation to reduce error. This algorithm keeps track of lost significant digits when summing together a large and small floating point number [4]. However, since this task's summation is in order, the Kahan Summation only gave slightly better accuracy.

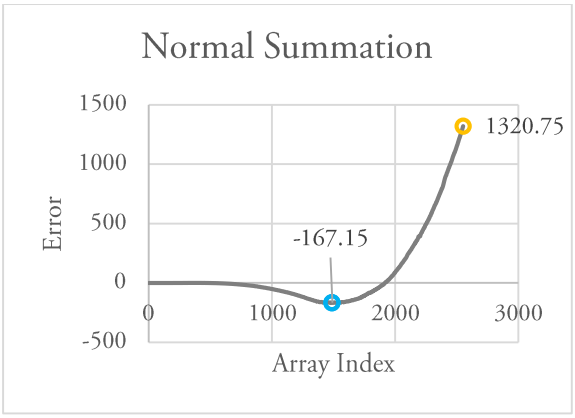

Fig. 2. Error of normal summation, compared w/ MATLAB double precision

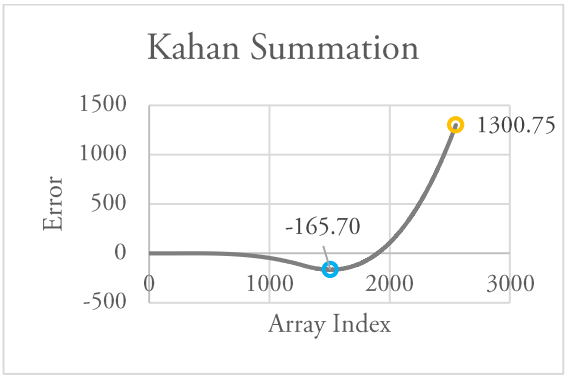

Fig. 3. Error of Kahan summation, compared w/ MATLAB double precision

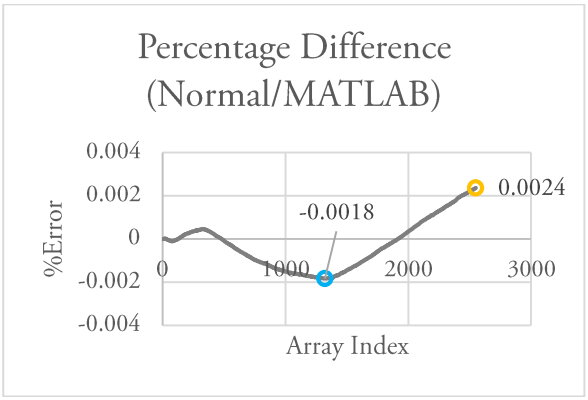

Fig. 4. %Error of normal summation, compared w/ MATLAB double precision

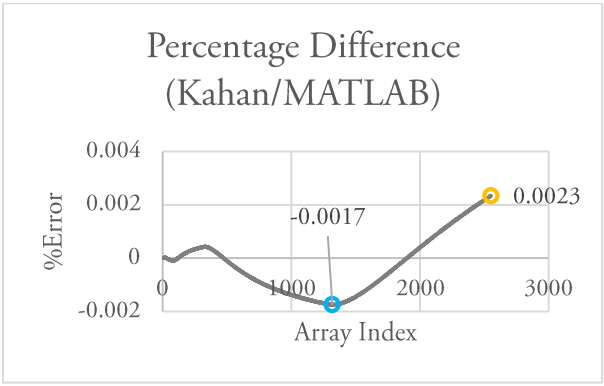

Fig. 5. %Error of Kahan summation, compared w/ MATLAB double precision

V. 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	Size = 49152 bytes	To fit larger arrays
NIOS II	Instruction cache = 2KB	Reduces latency
JTAG	As specified	N/A
Timer	As specified	N/A
System ID	As specified	N/A
LED PI/O	As specified	N/A
NIOS II BSP	`enable_lightweight_device_device_driver_api = 1`	Reduced program size, allows test 2 to compile

References

[1] Altera, "Nios II Core Implementation Details," 2 4 2015. [Online]. Available: https://www.altera.com/content/dam/altera-

www/global/en_US/pdfs/literature/hb/nios2/n2cpu_nii51015.pdf. [Accessed 28 1 2016].

[2] Altera, "Nios II Classic Processor Reference Guide," 2 4 2015. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf. [Accessed 28 1 2016].

[3] Python, "Floating Point Arithmetic: Issues and Limitations¶," [Online]. Available: https://docs.python.org/3.2/tutorial/floatingpoint.html. [Accessed 28 1 2016].

[4] N. Higham, "Department of Mathematics, University of Manchester," 5 1 1995. [Online]. Available: http://www.phys.uconn.edu/~rozman/Courses/P2200_11F/downloads/sum-howto.pdf. [Accessed 28 1 2016].

APPENDIX

A. *Source Code Functions*

```
void generateVector(float* x, unsigned
int n, double 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_kahan(float x[],
unsigned int M)
{
	float c=0, sum=0;
	float y,t;
	int i=0;
	for (i=0; i<M; i++){
		y = (x[i]+x[i]*x[i]) - c;
		t = sum + y;
		c = (t - sum) - y;
		sum = t;
	}
	return sum;
}
```