Digital Systems
EEE4084F

JUNE EXAM

03 June 2009

2 hours (1 hour for each part)

Examination Prepared by:
Simon Winberg

Last Modified: 27-May-2009

REGULATIONS

This is a closed-book exam. Scan through the questions quickly before starting, so that you can plan your strategy for answering the questions. If you are caught cheating, you will be referred to University Court for expulsion procedures. Answer on the answer sheets provided. Make sure that you **put** your **student name and student number,** the course code **EEE4084F** and a title **June Exam** on your answer sheet(s). Answer each section on a separate page.

DO NOT TURN OVER UNTIL YOU ARE TOLD TO

Exam Structure

RULES

- You must write your name and student number on each answer book.
- Write the question numbers attempted on the cover of each book.
- Make sure that you cross out material you do not want marked. Your first attempt at any question will be marked if two answers are found.
- Use a part of your script to plan the facts for your written replies to questions, so that you produce a carefully constructed responses.
- Answer all questions, and note that the time for each question is the same as the marks allocated.

PART A

Reconfigurable Computing (RC) [60 marks]

Section 1 : Short Answers & Multiple Choice [26 marks]

Question 1.1 and 1.3 are worth 9 marks, 1.2 is worth 8 marks (

1.1 (a) Briefly explain what is meant by a "reconfigurable computing platform". [3 marks]

(b) Computation is generally performed using one of the following methods:

1. A specialized hardware platform running dedicated software (e.g. an embedded system);
2. Application software running on a general purpose platform (e.g., a PC or supercomputer); or
3. Using a reconfigurable computing platform.

Briefly contrast these three approaches, highlighting types of applications that are well suited to certain certain approaches and why they may or not be suitable to the others methods. Ensure your answer is articulate and includes examples. Figures are welcome but not a requirement.

[6 marks]

1.2 (a) Explain the difference between temporal computation and spatial computation. [3 marks]

(b) RC platforms are typically build around the use of FPGAs. These systems often include a CPLD as well. Describe the main differences between a FPGA and a CPLD, and discuss why the CPLD is usually used as configuration and glue logic whereas the FPGA(s) are used for computation.

[2 marks]

(c) A particular platform can be supported by multiple ABIs. For example, the IBM Cell processor is supported by both the commercial IBM SPE ABI and the open-source Linux Cell ABI; yet the two ABIs are not identical.
What is an ABI, and why might different operating systems that can run on the same platform have different ABIs? [3 marks]

1.3 Multiple choice questions. Select *one* of the letter options as an answer for the questions below.

(question i is worth 3 marks; questions ii and iii are work 2 marks)

i. What is meant by the Von Neumann bottleneck?

a) The delay in waiting for the ALU to complete lengthy commands such as MULtiply.

b) The delays in swapping data between registers due to I/O access having to go through the accumulator.

c) The delays cause by jumping backwards in a loop, thus flushing the pipeline.

d) The delays in transferral of data between the CPU and main memory.

e) The tapering-off of Moore's law, in which the number of transistors in an integrated circuit is no longer doubling every two years.

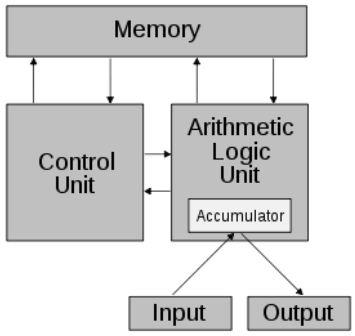

Illustration 1: Von Neumann Architecture

[3 marks]

ii. Which statement below accurately explains what OpenCores is?

a) OpenCores is a company specializing in the design of FPGA-based accelerator hardware.

b) OpenCores is a loose community interested in using programmable logic components.

c) OpenCores is a widely used FPGA interconnect fabric.

d) OpenCores is an open-source organization dedicated to improving design automation tools.

e) OpenCores is similar to the Open System movement in software development, in which the system's architectural is built around standard interfaces to improve interoperability with other systems.

[3 marks]

iii. According to your knowledge of CMOS, which statement below is accurate?

a) CMOS is more power hungry that TTL.

b) CMOS is often used with NMOS to improve the speed of circuits.

c) Microprocessors generally use CMOS gates because they are much faster than TTL.

d) CMOS gates generally takes less area on chip compared to gates using TTL or NMOS.

e) CMOS gate designs usually use more resisters than TTL designs for the same gate.

[3 marks]

Section 2: Long Questions [34 marks]

This section concerns the design and implementation of a Frequency Analysis Device (or FAD). The FAD is to be in the form of HDL code that can be slotted into a design, for example connected to a NIOS II softcore processor as shown in Illustration 2. First read through the text below that explains the operation of the FAD and then completed the THREE questions that follow. You may use pencil for design drawings.

Operation of the FAD

The FAD component has four inputs and two outputs, described the table below and shown in Illustraiton 2.

Connection Name	Direction	Description
Period	Output	A 12-bit output (unsigned integer value) that indicates the period for the incoming sampled data. The units of time is based on the speed at which the FAD is clocked by the SampCLK input line.
Ready	Output	Set to high (1) when period output is stable; low (0) if there is either no period value calculated of the period output is being updated.
Samp	Input	A 10-bit input indicating a sampled value (e.g., sent by an ADC).
SampCLK	Input	A positive edge (0 to 1) indicates a new sample is available on the samp line. In the design below the FAD is clocked at 1us, the FAD must read and process the samp input within half that time (i.e. ½ us).
Request	Input	Instructs the FAD to generate a period calculation (i.e. that the device it is connected to wants to read the period). Note that this line can be ignored depending on your FAD design.
Reset	Input	Reset the FAD. This line is set high (1) to tell the FAD to reset. This is done at system start up, or whenever the CPU wants to reset the FAD.

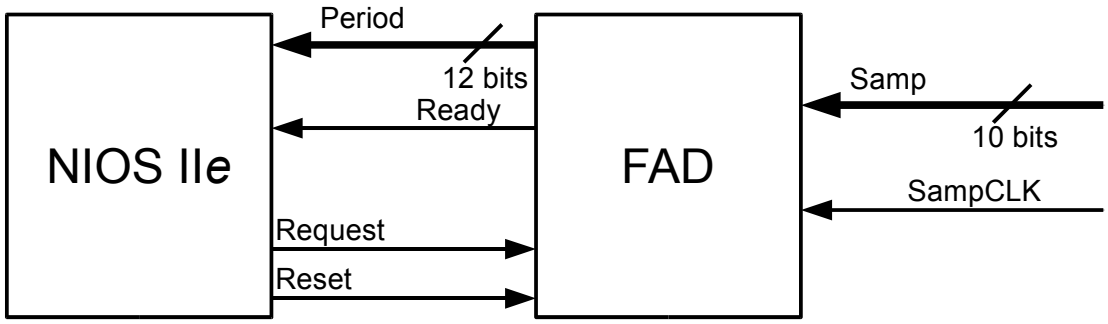

Illustration 2: High level block diagram showing the FAD within a FPGA

Scenario of operation

In this scenario we will consider, the FAD is connected to a NIOS II and an ADC (see Illustration 2). The ADC clocks the FAD (via SampCLK) whenever it has completed a new sample. The ADC clocks the FAD at 1us, i.e. sending a new sample every 1us. The FAD keeps track of the peaks and calculates a period. Whenever a new period is being calculated (e.g. at the detection of a peak) the *ready* line is set low (to indicate busy), then the *period* lines are written (and latched) with the newly calculated period, and then the *ready* line is set high (1) again to indicate the processor can now read a valid value. (A possible refinement is for an acknowledgement by way of setting the *request* low once period is read from the NIOS and then to set *request* high again to ask for a new value; but you *don't* need to worry about doing that).

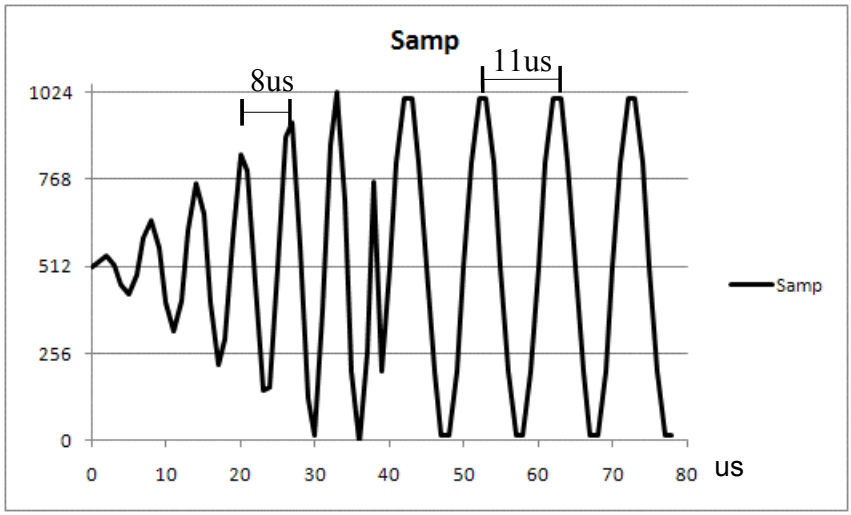

Illustration 3: Capture of sampled data showing pediod calculations

An example signal is shown in Illustration 3. The signal shows a certain amount of startup instability, and then stabilizes. The FAD should be able to determine the period of the signal in various stages, see case 1 and case 2 below...

CASE 1: When asked for the *period* at t=29us (measuring from the number of clock pulses given by the ADC) the FAD should respond with the value 8. The NIOS II could then convert this into a frequency value.

Looking back from before t=29... 1st max at 28us; 2nd max at 20us thus =>

T = 8 f = 125000 Hz = 125.0 KHz

CASE 2: When asked for the *period* at t=70us the FAD should respond with 10.

Looking back from t=70us... 1st max at 63us; 2nd max at 52us thus =>

T = 11 f = 90909 Hz = 90.9 KHz

Question 2.1

Assume you want to use a C to VHDL conversion tool for implementing the FAD.

Write a C code module that is likely to be translatable into VHDL using the conversion tool (remember the limitations and interfacing techniques presented in the lectures concerning such code). Your C module should contain ***one*** C function called FAD that does not call any other functions *within* your module. You can include declarations (e.g. typedefs) before the function implementation if needed. You may assume a **bit** datatype is available. Use the **_in** and **_out** modifiers to indicate which parameters are inputs / outputs. Use the **int *size name*** variable declaration syntax for creating arbitrary sized variables/parameters -- see **Appendix A** for details of the C syntax to use. Two of the marks for comments. [20 marks]
Hint: you can assume the signal is smooth, noise-free.

Question 2.2

Your C code probably has some form of assignment statement which is not simple a bit assignment, such as `int variable1 = variable2`. Discuss how such a statement is likely to be translated into VHDL by a C to VHDL translation tool so that your program can be executed as hardware in the FPGA. Use a diagram (e.g. a block diagram) if you think it needed to assist your explanation. [6 marks]

Question 2.3

Assume that all you want to do on the NIOS II, within the while(1) loop, is read the period when ready and calculate the frequency in KHz (saved to a global variable *frequency*) – see code listing on next page. The connection to the FAD is via PIO and the relevant memory-mapped registers (*period*, *ready*, etc) are already in the code. Complete the sections of code indicated to show how this can be done. Assume 1us sampCLK clock as in the scenario above. (PS: you may assume that multiply * and divide / can be used in the NIOS code) [8 marks]

CODE LISTING FOR QUESTION 2.3

```
/* Code for the NIOS IIe compiled with NIOSII IDE */
unsigned char* ready  = 0xFF00;/* address of ready bit (on the first bit) */
unsigned char* request= 0xFF01;/* address of request (on the first bit) */
unsigned char* reset  = 0xFF01;/* address of reset (on the second bit) */
unsigned* period = 0xFF02; /* address of period (first twelve bits) */
unsigned int frequency; /* frequency in KHz  (e.g. if period=8; then
                           frequency==125 as in case 1 above) */
/*************** Program entry point ******************/
int main ( void )
{
  /* Reset FAD and request new period */
  ?? fill in code here ??

   while (1) {
      /* wait for ready bit to be set */
      if (  ?? fill in code here ??  ) {
            ?? fill in code here ??
         }
      }
}
```

END OF PART A

Appendix A:

C to VHDL translation tool language specification

(loosely based on the Handle-C syntax)

The C to VHDL translation tool supports a large portion of the ANSI C syntax standard. The supported datatypes and modifiers are listed in the table below. The **bit**, **byte**, and **short** datatypes are commonly used, together with the **in** and **out** modifiers. As in ANSI C, the unsigned, short and long keywords can be used as datatypes if used alone (e.g. int ix) or as a modifier if used with another dataype (e.g. unsigned int ux). **Floating point** values (e.g. float, double) are not supported.

Support for sized arrays but not for pointers or unsized arrays

Please note pointers are not supported as either parameters or as variable declarations. Arrays are however supported.

Examples:	SUPPORTED	NOT SUPPORTED
	void test (int p [10]);	void test (int* p); OR void test (int p[]);
	int array1[10];	int* array1; OR int array1[];

Datatype sizing

Int and **unsigned** (or unsigned int) datatypes can have their their size (in number of bits) modified. All other datatypes (bit, bool, nibble, byte, char, unsigned char, llint, ulint, etc) cannot have their size modified.

The syntax for arbitrary sized declaration is as follows:

type size name;

Type: the datatype, namely: int, unsigned or unsigned int
Size: a positive integer (between 1 and 128)
Name: name of the variable

Examples:	SUPPORTED	NOT SUPPORTED
	int 8 signedbyte;	char 8 signedbyte;
	int 8 signedbyte;	char 8 signedbyte;
	int 6 intarray[10];	int* 6 intarray;
	unsigned 11 xu;	unsigned char 11 xu;
	int 7 xu;	byte 7 xu;

Arbitrary sized integers/unsigned variables can be used as are normal integers / unsigned values. Only the least significany bits are processed/copied; for example (int 2 x = 4; // x is set to 0 as the two least significant bits of integer 4 are both 0.)
Example:

```
unsigned int 7 x = 20;
int y = 10;
x += 1; // x changed to 21
x = x + y; // x changed to 31
y = x;  // y set to 31
```

See the next page for list of datatypes.

Datatypes

C -> VHDL Translator datatype/modifier	Default size	ANSI-C Standard equivalent keyword	Comments
_in		N/A	Indicate input parameter (use only with function parameters)
_out		N/A	Indicate output parameter (use only with function parameters)
enum	1 to 4 bits	enum	Translator limited enums limited to sets of 16 items
bram		N/A	Use as datatype or as modifier (e.g. bram int x = 10;) Forces data into block RAM. “bram” without type => “bram int”
sram		N/A	Use as datatype or as modifier (e.g. sram int x = 10;) Forces data into SRAM if available; otherwise into BRAM. “sram” without type equates to “sram int”
dram		N/A	Use as datatype or as modifier (e.g. dram int x = 10;) Forces data into DRAM or external ram if available; else into BRAM. “dram” without type equates to “dram int”
ext		N/A	Use as datatype or as modifier (e.g. ext int x = 10;) Forces data into external memory if available; otherwise into BRAM. “ext” without type equates to “ext int”
rom		N/A	Use as datatype or as modifier (e.g. rom int x = 10;) Forces data into read only memory if available; otherwise into BRAM. “rom” without type equates to “rom int”. Do not confuse keyword “rom” with ASNI keyword “const” -- const a variable cannot be changed (e.g., “const bram y = 5;” means *y* is located in BRAM but cannot be changed by the C program)
int	32-bit	int	Signed 32-bit value
short	16-bit	short	Signed 16-bit value
unsigned	32-bit	unsigned	Unsigned 32-bit value
unsigned short	16-bit	unsigned short	Unsigned 16-bit value
char	8-bit	char	Signed 8-bit value (-128 to +127)
unsigned char	8-bit	unsigned char	Unsigned 8-bit value (0 to 255)
byte	8-bit	unsigned char	Unsigned 8-bit value (0 to 255)
nibble	4-bit	N/A	Unsigned 4-bit value (0 to 15)
bit	1-bits	N/A	Single bit (0 to 1)
bool	1-bit	N/A	Single bit (0 to 1) equivalent to bit
long	32-bit	long	Signed 32-bit value
unsigned long	32-bit	unsigned long	Unsigned 32-bit value
long long	64-bit	long long	Signed 64-bit value
unsigned long long	64-bit	unsigned long long	Unsigned 64-bit value
llint	64-bit	long long	Signed 64-bit value
ulint	64-bit	Unsigned long long	Unsigned 64-bit value

University of Cape Town
Department of Electrical Engineering
June Examination 2009
EEE4084F
Digital Systems Part B

June 6th, 2009

TIME ALLOCATED 60 MINUTES

- You must write your name and student number on each answer book.
- Write the question numbers attempted on the cover of each book.
- Make sure that you cross out material you do not want marked. Your first attempt at any question will be marked if two answers are found.
- Use a part of your script to plan the facts for your written replies to questions, so that you produce a carefully constructed responses.
- Answer all questions, and note that the time for each question is the same as the marks allocated.
- DO NOT OVER-RUN YOUR TIME ON ANY QUESTION.

Question 1

In the course, we covered, in Patterson and Hennesey's book, the topics of *Multicores, Multiprocessors and Clusters.* Carefully explain the differences between these terms, with examples if possible. (**5 marks**)

Question 2

Cloud Computing has been discussed quite heatedly in the IT world. Define *Cloud Computing* very carefully, and give your considered view of how you believe the next 5 years will treat *Cloud Computing*. Make sure that you clarify the advantages and disadvantages of Cloud Computing. (**15 marks**)

Question 3

You are trying to bake three bluebury cakes. The ingredients are as follows:

- 1 cup butter, softened
- 1 cup sugar
- 4 large eggs
- 1 teaspoon vanilla extract
- 1/2 teaspoon salt
- 1/4 teaspoon nutmeg
- 1 1/2 cups flour
- 1 cup blueberries

The recipe for a single cake is as follows:

Preheat the oven to 160C. Grease and flour your baking pan.

In a large bowl, beat together with a mixer, butter and sugar at medium speed until light and fluffy. Add eggs, vanilla, salt and nutmeg. Beat until thoroughly blended. Reduce mixer speed to low, add flour, 1/2 cup at a time, beating until just blended.

Gently fold in the blueberries. Spread evenly in prepared baking pan. Bake for 60 minutes.

1. You are required to cook 3 cakes as efficiently as possible. Assuming that you have only one oven large enough to to hold one cake, 1 large bowl , 1 cake pan, and 1 mixer, come up with a schedule to makes these cakes as quickly as possible. Identify the bottlenecks in completing the task.

2. Assume you manage to obtain 3 bowls, 3 cake pans and 3 mixers, how much faster is the process of making the 3 cakes, given the extra resources?

3. Assume now that you have two friends that will assist you with the cooking (and not eat the raw ingredients), and you have a large oven that can accommodate all 3 cakes. How much faster will this be than the time taken for task (1) above.

4. Compare the cake-making task to comuting three iterations of a loop on a parallel computer. Identify data-level parallelism and task level parallelism in the cake-making loop.

You will need to assign times for each part of the process. Use relative units and do not try and tie it up to real units, since we are just trying to compare methods of parallel execution. (**25 marks**)

Question 4

A form of Amdahl's law states:

$$T_i = \frac{T_a}{I} + T_u$$

T_i is the time after improvement by parallelisation

T_a is the time affected by speedup.

I is the improvement factor.

T_u is the part that cannot be improved.

What percentage of the original time can be sequential if we wish to speed up the computation by a factor of 50, give that we have 64 processors available? (**15 marks**).