Digital Systems
EEE4084F

JUNE EXAM

03 June 2009

2 hours

Examination Prepared by:
Simon Winberg

Last Modified: 27-May-2009

SOLUTIONS

Answers are shown in blue Arial font

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]

Answer: a reconfigurable computing platform is a computer system that can change part of its hardware according to the computation it is to performed.

(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.

Answer: .

[6 marks]

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

Answer: Temporal computation is sequential, doing one thing at a time. Spatial computation is expressed in as a space, dependencies and linkages between parts of the computation, which is not related to a time-based sequence of steps. Spatial computation algorithms are well suited for implementation as on parallel processor or as hardware because multiple tasks could overlap in time.

(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.

Answer: (only one of the differences listed below is needed):

- A CPLD contains considerably less programmable logic elements (Les) in comparison to an FPGA. CPLDs contain a few tens of 1000s LEs; whereas FGPAs typically comprise millions of LEs.
- CPLDs are cheaper than FPGAs, and in terms of configuration arch there is not as much need for so many LEs as per used in application-specific hardware the FPGAs are programmed with (i.e. no need to use an expensive FPGA for config arch where a cheaper CPLD would do).

CPLDs are generally used as configuration glue logic instead of FPGAs because the configuration architecture is generally much simpler, taking considerably less LEs than does the application-specific hardware implemented in the FPGA(s) (which the configuration architecture programs). Accordingly, to save cost and complexity of the RC platform, CPLDs are used for the configuration architecture.

[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]

Answer: ABI stands for Application Binary Interface, and is a means by which two applications (or software programs) connect with one another – usually ABI refers to a connection between an application program and an operating system running on a computer platform. ABI are generally formulated according to a specification, rather than purely hardware constraints, for specified methods of passing information from one program to another. Thus a platform may support operating system *A* which has an ABI that explains that parameters are passed via the stack; and the same platform may be supported by operating system *B* that passes parameters via a particular CPU register.

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. **<== the correct answer is (d)**

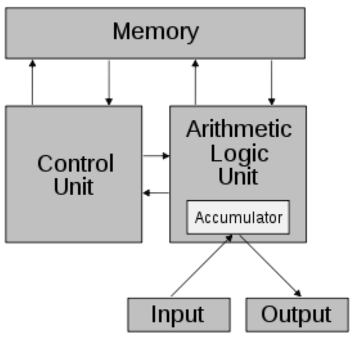

Illustration 1: Von Neumann Architecture

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.

[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. **<== (b)**

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. **<== (d)**

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.

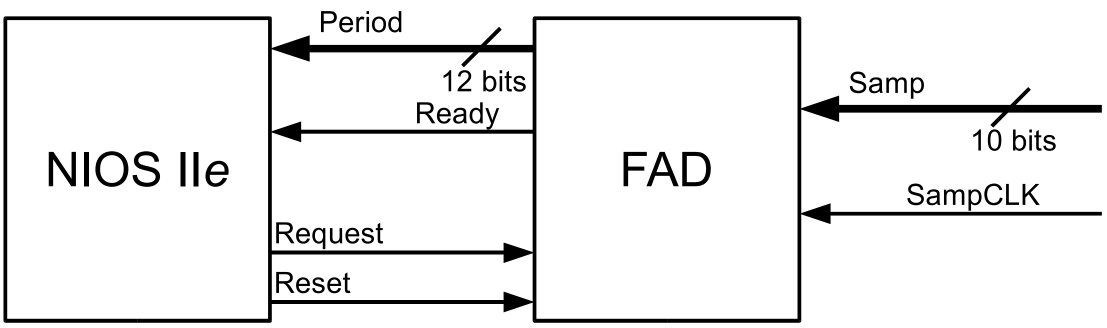

Illustration 2: High level block diagram showing the FAD within a FPGA

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).

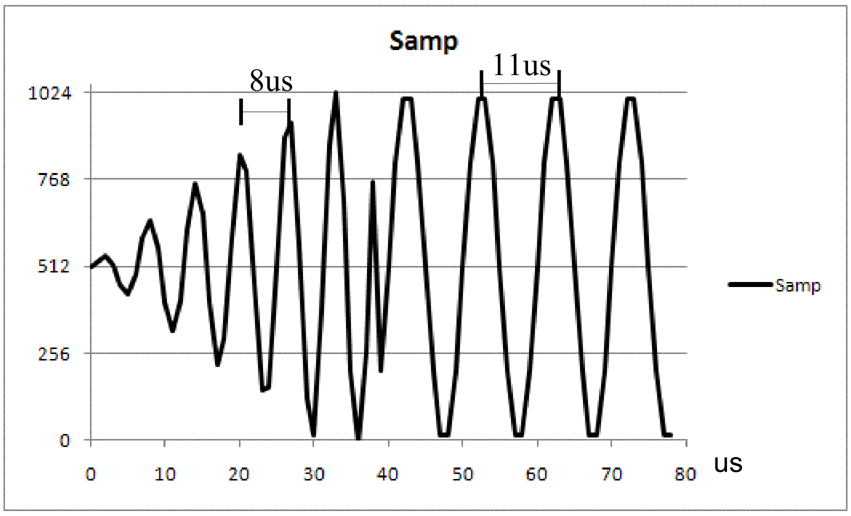

Illustration 3: Capture of sampled data showing pediod calculations

An example signal is shown in Illustration 3. The signal shows a certain amount of startup instability, and then stabalizes. 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.
Answer: sample code provided on page 7.

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]

Answer: The translator is likely to use latching of data between two meta-logic flip-flop elements (or latch elements), say between FF1 and FF2. Such an arrangement would need to have a multiplexer on the data in line of FF1 to choose whether FF1 is to be assigned the output of FF2 or to values on some other bus. An OR gate could be used on the input to the FF1 *set* input to tell FF1 when it needs to be set to FF2.

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]

Answer: see addition to code below.

CODE LISTING FOR QUESTION 2.3

SOLUTION

```
/* 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 = 2; /* reset the FAD */          ✓ 2 mark
  *request = 1; /* request new period */
    while (1) {
        /* wait for ready bit to be set */
        if ( (*ready) & 0x1 ) {   ✓2 mark
            frequency =  1000 / (*period & 0x3FF);   ✓   2 marks       ✓ 2 mark for a comment
            *request  =  1; /* request new period – although the request line should still stay high */
          }
        }
}
```

No mark allocated for this line because it's not really needed considering the NIOS II PIO operation

Sample code for 2.1 (out of 20 marks)

```
/** Declare the Interface and implementation of the FAD
    Assume there is a bit datatype defined. */
void FAD (
 /* Define the inputs first */
 _in bit request,    // 1st bit
 _in bit reset,      // 2nd bit
 _in bit sampCLK, // 3nd bit
 _in unsigned 10 samp, // 3nd bit

 /* Define the outputs last */
 _out bit ready,    // 1st bit
 _out unsigned 12 period // 12 bits for period
 )
{
     static unsigned B(10) max  = 0; // initializations often ignored
     static unsigned B(10) prev = 0;
     static bit going_up = 0;
     static unsigned num = 0;
     static unsigned first_max = 0;
     static unsigned second_max = 0;
     static bit prev_clk;
     // first check if there was a reset
     if (reset) {
     // RESET MODE...
        going_up = 0;
        first_max = 0;
        second_max = 0;
        prev = samp;
        ready = 0;
        num = 0;
     } else {
     // RUNNING....
        if (prev_clk == sampCLK) return; // take account of clocking
        if (sampCLK == 0) return; // only do stuff on positive edge

        if (samp > prev) {
           // going up
           if (!going_up) first_max = second_max; // remember time of last max
           second_max = num;
           going_up = 1;
           } else {
           // going down
           if (going_up) {
              period = second_max - first_max + 1;
              // note the +1 because we are one after the max!!
              ready = 1;
           }
           going_up = 0;
           }
     prev = samp;
     num++;
     }
}
```

✓ 3 marks for correct declaration of parameters – should be close to what is shown!!

✓ 2 marks for suitable use of comments

✓ 3 marks for declaration of suitable static or global variables

✓ 2 mark for handling asynchronous reset mode

✓ 2 marks for synchronous sampling mode, getting samp on upgoing sampCLK edge

✓ 6 mark for handling the detection of a max and calculation of period

✓ 2 mark for somehow tracking the number of pulses (i.e. time measure)

See code file FAD.cpp provided with the sample solutions.

See output from sample on next page (using data as used in the scenario for the question).

Question 2.1 SAMPLE OUTPUT (running on DEV C++)

```
clk=03 NEW PERIOD: 3
clk=09 NEW PERIOD: 7
clk=15 NEW PERIOD: 7
clk=21 NEW PERIOD: 7
clk=28 NEW PERIOD: 8   // case 1 in scenario
clk=34 NEW PERIOD: 7
clk=39 NEW PERIOD: 6
clk=43 NEW PERIOD: 5
clk=53 NEW PERIOD: 11
clk=63 NEW PERIOD: 11  // case 2 in scenario
clk=73 NEW PERIOD: 11
DONE.
Press any key to continue . . .
```

END OF PART A

Solutions to Part B questions are given after the Part B questions...

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 a 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).

Solutions

Question One

Bookwork. **1 mark** for each correct definition, **2 marks** for good examples.

Question Two

- Good definitions **5 marks**
- Difference to GRID computing **2 marks**
- Advantages of Cloud Computing: about 1 mark per point, up to **5 marks**
- Valid future predictions, **2 marks**

Question Three

5 marks for the task list:

1. Preheat (Oven)
2. Grease and Flour (pans)
3. Beat (in bowl and with mixer, medium speed) butter and sugar
4. Add eggs etc., beat (with mixer at low speed, in bowl)
5. Fold in blueberries (in bowl)
6. Spread (from bowl) to (pan)
7. Bake 60 minutes (in pan in oven)

Process 1

7 marks for a diagram of some sort, pointing out that only oven heating can occur in parallel until pan is loaded, and baking starts, giving a time value to each (as a symbol). The bowl, mixer, pan, oven are bottlenecks for the tasks associated with them. The single operator is also a bottleneck for the mixing etc.

Process 2

Here we have 3 bowls, 3 pans, 3 mixers. However, we have one operator, so this has to be brought into the equation. It is not completely clear from the question, but it might be possible to leave the mixers running at the appropriate speeds, so much of the mixing can move to parallel. However, someone that interprets the mixer as a hand mixer, requiring the operator, might not see this and must not be penalised.

7 marks for the discussion.

Process 3

Addition of 2 operators (bringing total to three), large oven, now opens up for parallelism. Note oven heating must still run in parallel during the mixing.

7 marks for the flow charts / discussion.

Process 4

4 marks for catergorisation of tasks, and objects such as mixers, ovens, pans as data.

Question Four

Bookwork. See page 635 of Patterson and Hennessey, numbers slightly altered. 15 marks, spread evenly over the steps. 7 marks for setting up the problem and identifying the parallel and serial portions.