Digital Systems
EEE4084F

FINAL EXAM

18 May 2010

3 hours

Examination Prepared by:
Simon Winberg

Last Modified: 11-May-2010

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 **Final 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 relates to the marks allocated.

Section 1 : Short Answers [36 marks]

Question 1

There are three general classes of system for carrying out electronic computation: 1) purely hardware (e.g., the computation is fixed); 2) reconfigurable computer systems; and 3) software processor systems.

(a) Contrast the drawbacks and benefits of these three approaches mentioned above. Ensure your answer provide one advantages and one disadvantage for *each* class of system [2x3 makrs = 6 marks].

(b) Explain the main determining characteristic that differentiates between computer platform being considered "non-reconfigurable" and "reconfigurable". [2 marks]

(c) Communication is a fundamental aspect in the design of parallel computers. Explain what is meant by the terms "communication latency" and "effective bandwidth" [2 marks]. Briefly describe how this effective bandwidth can be can be calculated (e.g., in the context of two computers linked via a Gigabit Ethernet wired network) [2 marks].

[12 marks]

Question 2

Large scale microprocessor-based computer cluster may incorporate hundreds of computers, often comprising networked computers that have either SMP processors architectures or Cell processors architectures (e.g., IBM cell-based blade servers or PlayStation3 computers). Often, these cluster systems are programmed using a combination of Pthreads and the Message Passing Interface (MPI).

(a) Provide two common examples of applications that run on these large-scale computer clusters. [2 marks]

(b) Briefly discuss the main differences between the design of an SMP processor and that of the Cell processor [3 marks].

(c) When building a cluster for general use, a decision is often made to build the cluster using SMP processors rather than Cell processors. Give three reasons why SMP processors are such a popular choice. [3 marks]

(d) Programming in MPI, and programming in Pthreads follow different approaches. Yet, to develop optimized solutions for clusters of computers with SMP processors, programmers often end up using a combination of both MPI and Pthreads. Surely it makes more sense, and gives better performance, if just one or the other approach is used. Explain why in this kind of situation programmers often use a combination of MPI and Pthreads. [4 marks]

[12 marks]

Question 3

(a) The increase in GFLOPS has shown exponential growth, from the humble beginnings of computing. For example, before the 1950s even 1MFLOP was unobtainable. Around what decade (1950s, 1960, 1970, 1980 or 1990) was an *excess* of 0.5GLOPS (i.e., 500MFLOPS) achieved for a processor? [1 mark]

(b) Reproduce the table below in your answer book, and use it to indicate what if the types of processing listed below are either done typically on a front-end processor or on a back-end processor (place a cross in the relevant column to show this). To save time, you can substitute the letter in brackets for the name of the processing routine (e.g., 'D' for 'database processing (D)').

Processing type	**Front-end**	**Back-end**
Database processing (D)		
Sampling ADCs (S)		
Pulse compression (P)		
Convolvers or FIR filtering (C)		
Clutter mitigation (M)		

[5 marks]

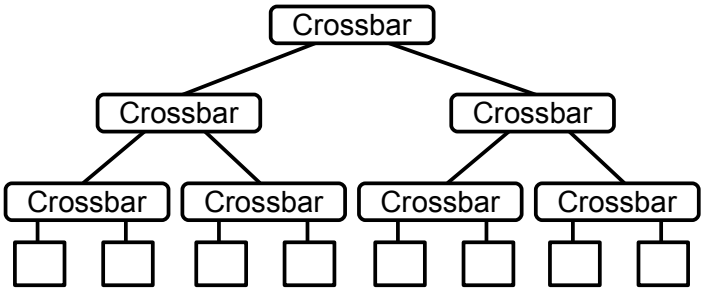
(c) Describe one way you can quantify the *complexity* of a high performance embedded computer (HPEC) system, and one way you can measure the *performance* of a HPEC system. Mention units of quantification (for example, the power of a system can be measured by how quickly the system converts energy into work; the unit for power is the WATT). [4 marks].

(d) GPUs, for example the nVIDIA cards that support CUDA, are not appropriate for all types of application. Briefly describe the nature of applications that GPUs are well suited to handle. [2 marks]

[12 marks]

Section 2: Multiple Choice [20 marks]

Q1. What is the *bisection bandwidth* of a three-level tree network of crossbars shown below? (Note that each crossbar's port has a bandwidth equal to 1).



Bisectional bandwidth = (a) 1 (b) 2 (c) 3 (d) 4 (e) 8

[4 marks]

Questions Q2 and Q3 refers to the Pthread code given in Appendix B

The Pthread program in Appendix B simply counts the number of times the value *P* comes up as a random number.

Q2. Consider the program in Appendix B. The program is run 11 times on a Intel Core2 Quad computer (which has four processors), using the command line parameter 1 (i.e., so that only one thread is spawned). The first run is ignored, and the timing of the other ten runs are averaged, giving an average execution time per run of precisely 1000ms (for 1 thread). It is calculated that the initialization of each thread attribute and creation of each threads (i.e., lines 43, 44) take together 30ms to run for each thread created. Similarly, the *join* operation takes 18ms per thread. Assume it takes 0ms for each thread to activate, and the *rand()* function does not do any synchronization. If the program is run 11 times (ignoring the first run) with the parameter 4 (i.e., four threads are spawned) determine the expected **speed-up** of the program (choose the closest answer below).

(a) 0.4 (b) 1.8 (c) 2.3 (d) 2.8 (e) 3.5

[4 marks]

Q3. Considering that the code in Appendix B is fairly representative of the type of program that would run on a Intel Core2 Quad processor, how would you classify the Intel Core2 Quad according to Flynn's taxonomy? Choose the most appropriate classification:

(a) SISD (b) SIMD (c) MISD (d) MIMD (e) None of Flynn's classifications fit.

[4 marks]

Q4. Choose the description that best describes the term “beamforming” in relation to HPEC applications.

(a) Beamforming is any kind of signal processing method that involves a sequence of stages in which directional information is maintained through the entire process.

(b) Beamforming involves combining signals from multiple sensors to simulate one sensor with enhanced directionality.

(c) Beamforming is simply a technique that allows you to focus data at a particular processing element; it is equivalent to the way you can use a magnifying lens to focus sun rays.

(d) Beamforming is a signal processing model commonly used in HPEC design, it is a sequence of stages where each stage inputs data, processes the data, and passes the data to the next stage.

(e) Beamforming is the process of converting analogue inputs (e.g., radar signals) into digital data captured and filtered by the front-end which then relays relevant results to the back-end.

[4 marks]

Q5. Answer true of false to each question below (each answer is 1 mark).

(i) A correlation between two identical data sets returns the value 0.
(ii) The 'golden measure' refers to a trusted baseline implementation of an algorithm to parallelize.
(iii) The modern trend in HPEC design is moving away from using COTS solutions.
(iv) The commonly used HPEC acronym “SWAP” means Stop Wait And Process.

[4 marks]

Section 3: Long Answers [44 marks]

Question 1

This question relates to FPGA-based RC platforms. Figure 1 shows a block diagram for a LUT used as a programmable block (PB) in a FPGA.

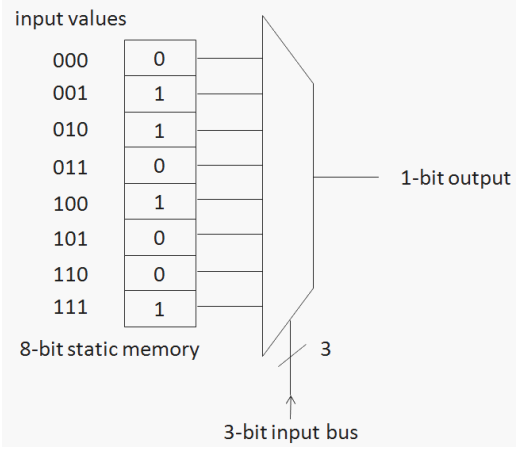


Figure 1: a LUT within a PB of a FPGA.

(a) What is meant by the term "configuration architecture" in relation to an FPGA-based RC platform [2 marks].

(b) What is a LUT? And why do FPGAs usually have lots of LUTs instead of the more fundamental logic gates (e.g., NAND gates)? [2 marks]

(c) The Xilinx Virtex-6 comprises an architecture composed of two types of configuration logic blocks (CLBs), namely SLICEM and SLICEL blocks. Provide motivation for why the Xilinx designers chose this approach instead of simplifying the design by having one type of slice. You can comment on the difference between SLICEL and SLICEM blocks if you think it aids your reasoning. [5 marks]

(d) Usually a FPGA has many pins that are hard-wired to off-chip hardware (e.g., the PCB might wire a FPGA pin directly to a pin of an RS232 M-type connector as shown in Figure 2). However, each PB might only have a few inputs as Figure 1 shows. In addition, individual PBs can usually be configured to be *synchronous* or *asynchronous*. Clearly, Figure 1 is lacking some important details. If you have not already done so, detach the answer sheet (the last page of the question paper). The answer sheet shows a FPGA that has only two PBs. Use the answer sheet to visually describe how the 6 inputs and 2 outputs of the FPGA could be directed to PBs, and how each PB can individually be set as asynchronous or synchronous. Show interns of PB#1 – the other PB will have the same implementation. (Note you don't need to show how the configuration flip flops C1-C6, CS1, CS2 and CO are set/reset.) [13 marks]

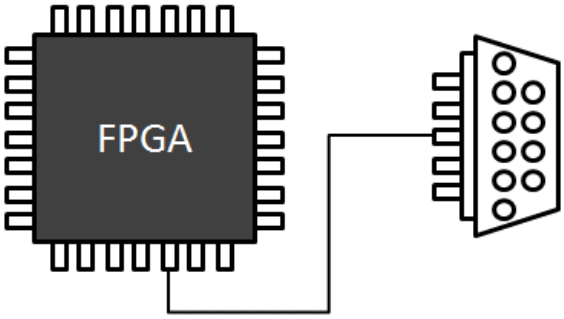


Figure 2: FPGA pin hard-wired to RS232 connector

[22 marks]

Question 2

This question concerns porting a program from standard C to the Handle-C style syntax that can be converted directly to VHDL and used as a digital accelerator to speed-up processing. Imagine you are developing a parallel application that runs on a FPGA platform. Your application has two parts: 1) a digital accelerator device called PPG, and 2) a soft processor that does other work and sends parameters to the PPG. The digital accelerator is called the Parallel Pulse Generator (or PPG); all it does is generate a digital binary waveform, which oscillates between high for a certain time and low for a certain time as shown in Figure 3. The C code listing below explains how the signal could be generated if it was running on a processor. Figure 3 shows a block diagram indicating how the PPG would connect up to a soft processor on a FPGA.

```
// C function to generate a digital pulse
void PPG ( int L, int H )
{
  while (1) {
    out = 0; // set the out line low
    delay_microseconds(L);
    out = 1; // set the out line high
    delay_microseconds(H);
   }
}
...
PPG(100,200); // generate signal shown in Figure 3
```

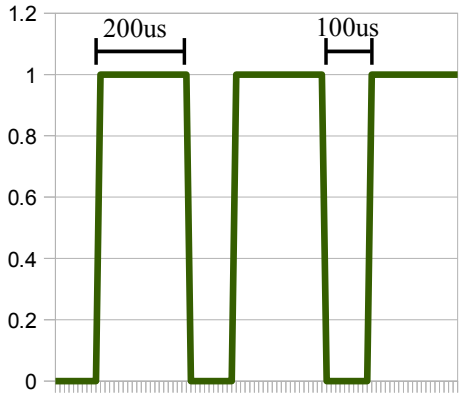


Figure 3: Signal L=100us H=200us

Figure 4 shows the block diagram layout that you are planning to use. As the figure shows, a soft processor connects to the PPG via two 16-bit PIO lines and two single bit control lines (i.e., start and enable). If *enable* is low, the PPG does nothing but wait for a *start* pulse. When a positive edge is sent over *start* (i.e., a start pulse) the PPG immediately reads and stores the L and H inputs and then starts generating the signal sent on the *out* output. As soon as *enable* is set back to low, the PPG stops producing an output signal (it can leave the *out* pin at either low or high) and returns to waiting for a start pulse.

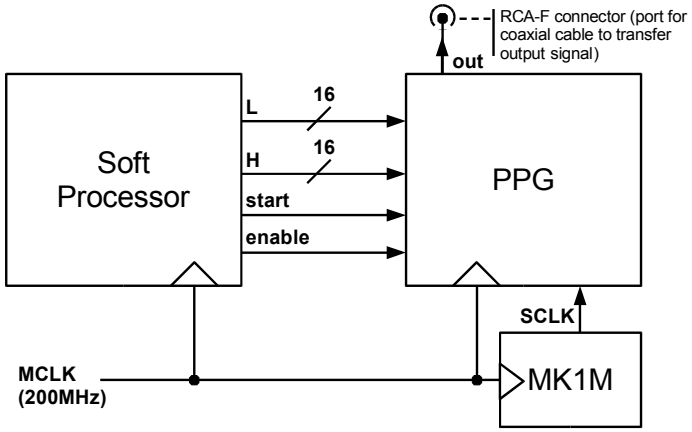


Figure 4: Block diagram showing the PPG and interconnects

Complete the following steps:

(a) The MK1M block shown in Figure 4 converts the 200MHz master clock into a slower 1MHz *SCLK* clock to make it easier to generate output pulses that are multiples of microseconds. Explain how the MK1M component could be implemented to produce this *SCLK* clock. Your answer needs some explanatory text; you can include a circuit/block diagram if you think it would aid your explanation. [6 marks]

(b) Appendix A provides Handle-C style syntax, a C syntax that similar to the syntax used by many C to HDL automatic conversion programs. Develop a C function, using the provided Handle-C syntax, to implement the PPG entity. That the inputs and outputs to PPG entity need correspond to parameters of the C function (the parameters obviously need to be declared using appropriate datatypes listed in Appendix A). Note *SCLK* is a single bit input, and *out* a single bit output, of the PPG. [12 marks]

(c) Having two input 16-bit lines, L and H seems a bit wasteful in terms of interconnects used on the FPGA; surely the PPG entity could be modified to have just one 16-bit input port (e.g., called X). Describe how you could rework the implementation of the PPG so it still provides the same functionality, but you get by with just one 16-bit input. [4 marks]

[22 marks]

END OF EXAMINATION

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 significant 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

Appendix B:

Pthread Code

Line	Code
1	`/* @file countrand.c`
2	`   A pthread program to count the number of times a certain number`
3	`   comes up using rand.`
4	`   Compile on Linx/Cygwin using: gcc -o countrand countrand.c`
5	`*/`
6	`/* Library includes */`
7	`#include <stdio.h>`
8	`#include <time.h>`
9	`#include <sys/types.h>`
10	`#include <pthread.h>`
11	
12	`/* Global constants and variables */`
13	`const int RAND_RANGE = 1000;`
14	`int numthreads;`
15	`int N = 5000000; // number of random numbers to process`
16	`int found[4] = {0,0,0,0};`
17	`int P = 101; // number to search for`
18	
19	`/** The thread function for parallel computation */`
20	`void* mythread (void* arg)`
21	`{`
22	`  int id = (int)arg;  // determine the ID for this thread`
23	`  int num = N / numthreads;  // see how many times this thread must loop`
24	`  int i,r;`
25	`  for (i=0; i<num; i++) {`
26	`       r = rand()%RAND_RANGE;    // generate a random number`
27	`       if (r == P) found[id]++;  // see if pattern is found`
28	`       }`
29	`}`
30	
31	`/** Entry point to the program. */`
32	`int main ( int argc, char** args )`
33	`{`
34	`  int i;`
35	`  pthread_t th[4]; // thread variables`
36	`  pthread_attr_t attr[4]; // thread attributes`
37	`  // initialize the random number generator`
38	`  srand(time(0));`
39	`  // Indicate number of threads that are going to be used`
40	`  if (argc<=1) return printf("Please give number of threads.\n");`
41	`     else numthreads = atoi(args[1]);`
42	`  // Initialize the thread attributes and spawn the threads`
43	`  for (i=0; i< numthreads; i++) {`
44	`       pthread_attr_init(&attr[i]);`
45	`       pthread_create(&th[i], &attr[i], mythread, (void *)i);`
46	`       }`
47	`  // Join all the threads`
48	`  for (i=0; i< numthreads; i++) pthread_join(th[i],NULL);`
49	`  // print out the result`
50	`  printf("Found matches (using %d threads) = %d\n", numthreads,`
51	`      found[0] + found[1] + found[2] + found[3]);`
	`}`

ANSWER SHEET 1

Your student number:

ANSWER TO SECTION 3 Q1(d)

Please fill in your student number above in case this pages falls out your answer book.