Digital Systems
EEE4084F

FINAL EXAM – MEMORANDUM

3 June 2016

3 hours

Section 1: Short Answers [48 marks]

Question 1.1 [12 marks]

The following questions concern computation approaches.

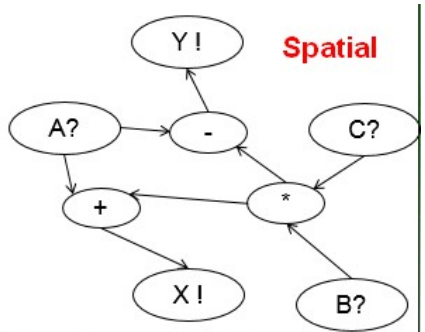
1.1 (a) Briefly explain the difference between the concepts of temporal and spatial computation? You can use a diagram to elaborate if you want. **[4]**

Temporal Computation: This is the traditional paradigm where a programmer typically describes tasks to be done over a sequence of steps that are run in time, one after the other.

Spatial Computation: Describes computation as spaces and relationships. This is closer to a typical hardware description of circuits where data is explicitly generated in one space or sent from one space to another. The aspect of time is more implicit in this paradigm, being an implication of dependencies and when computing spaces are active. The figure below illustrates this...

Temporal

```
A = input("A= ? ");
B = input("B =? ");
C = input("B multiplier ?");
X = A + B * C
Y = A – B * C
```



Can provide a clearer indication of relative dependencies.

1.1 (b) 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. Explain what the acronym ABI stands for [1 mark] and what an ABI provides. In particular, how is an ABI is different from an API [3 marks]? **[4]**

ABI stands for "Application Binary Interface (ABI)". It is a specification that defines data types, register usage, calling conventions, and object formats to ensure compatibility of code generators and portability of code. Examples include: IBM SPE (Strategic Processor Elements) ABI, and Linux Cell ABI.

1.1 (c) The processing done on many reconfigurable computing platforms are centred around high-performance FPGAs. Provide a brief argument motivating why a CPLD is sometimes included as a means for configuration and additional I/O routing, with the CPLD sometimes replaced or supplemented by a micro-controller to provide further facilities. [4 marks] **[4]**

A Complex PLA (CPLA) is a lower cost programmable logic chip than an FPGA.

They are often used on an FPGA platform in order to implement a bridge between the development PC communication interface (typically USB) to the FPGA configuration and debugging interface (typically JTAG). The same CPLD can further provide a bridge between the development PC and the FPGA configuration flash memory, among other peripherals that might be present on the board. This ensures that the FPGA platform remains under the development PC control, even when the FPGA is not in a valid configured state.

Regarding routing logic through the CPLD: the CPLD does not have a "configuration" state. It is instantly on, without pulling any of the I/O pins into an undesirable "weak pull-up" state during configuration. The CPLD can therefore be used to "condition" certain FPGA signals during configuration. This is especially useful for peripheral reset and MOSFET driver signals, among others.

A micro-controller is sometimes used in addition, or as a replacement of, a CPLA in order to provide a more sophisticated, software-based method for programming the FPGA. It is often used when control over Ethernet, or WiFi, is desired instead of a USB interface. The micro-controller further functions as a boot monitor and provide features to integrate with the FPGA via a firmware-software interface that links the processor to the FPGA. This can be particularly useful for implementing an interface to the FPGA firmware control registers and system bus.

Question 1.2 [12 marks]

This question concerns parallel and high performance computing.

1.2 (a) In an important business meeting with clients and your senior engineering colleagues, you bravely suggest, despite your inner voice berating you for being a show-off in your new job, that a parallel language should be considered as a way for quickly building, or at least piloting, a parallel computing solution. A colleague responds to you saying: "No way, that is too risky! Parallel computing languages don't really exist and automatic parallelisation is still useless". Present an argument to stand by your statement and illustrate examples confirming that parallel languages indeed exists – also (to be more of a smart-ass) clarify the current situation of automatic parallelisation techniques. **[6]**

Parallel programs do exist: these are essentially programming languages that can be used to implement parallel systems. The definition is quite vague: standard C could be classified as a parallel language (e.g. C using OpenMP). Another example is OpenCL platforms for GPU and FPGA.

In terms of a spatial language, or visual models, to represent parallel operations, there are many. Simulink, for example, effectively describes parallel operations without the user having to explain how to go from the parallel model into a parallel implementation. In terms of automatic parallelism, there are systems that work, like Simulink and SystemC compilers, but the problem is that the generated code is often difficult to read and to debug.

1.2 (b) When porting a legacy sequential program into a parallel version, why is it that much of the code tends to remain sequential, with only a small part of the program actually made into a parallel form, and this part typically being meticulously hand-crafted instead of largely automatically generated (e.g. by using pragmas understood by the compiler)? **[4]**

Parallel code is often hand-crafted for two reasons: typically it is just the critical section of a program that is made parallel (to save development costs and because significant performance boosts are unlikely from parallelising infrequently used parts of code). In addition, there tends to be much complication (especially for fine-grained problems) in terms of getting the synchronisation and partial results to be done correctly.

Frequently, automated solvers are more used in a dual programming approach, used as a safety check to verify that the hand-crafted parallelised code is producing the same results as the original non-parallel code.

1.2 (c) Nowadays, server class machines tend to use SMP architectures. Briefly explain what an SMP is and what the acronym SMP stands for. **[2]**

SMP stands for Symmetric Multi-processor. It is a processor chip that combines more than one (usually a power of 2) processor cores for which each core follows the same design (for example, a dual core Pentium chip that is made up of two Pentium processor cores).

Question 1.3 [12 marks]

The question concerns parallel computing architectures and networking/interfacing associated with their use for high performance embedded system processing.

1.3 (a) What does the Von Neumann bottleneck refer to? **[3]**

The von Neumann bottleneck refers to a throughput limitation whereby instruction fetch and a data operation cannot occur at the same time, it can generally be considered the delay in transfer of data between the CPU and main memory (while caching greatly reduces these delays, they can add significant overhead when memory, external to the CPU, happens routinely, such as read/write to distantly located memory addresses that occur close to the same time).

1.3 (b) What is meant by synchronous communications? Does the concept 'blocking communications' refer to the same issue? **[3]**

1. Synchronous communications is a method by which the sender and receiver thread arrive at a point in computation whereby both the sender and receiver must interchange information. Data or signals sent by the sender thread will only be sent by the sender when the receiver thread is at a point ready to receiver this data. Such communication tends to need synchronisation whether sending or receiving, for example, if the receiver is ready to receive

data it waits (or blocks) until such time as the sender is ready to send data.

2. Blocking communications is another term used to refer to this.

3. The student might also answer the question in relation to hardware protocols, such as RS232 (asynchronous) vs. SPI or I^2C (synchronous). Synchronous communication in this sense refers to a communication protocol that explicitly sends a clock signal from the "master" to the "slave" device.

1.3 (c) What does the acronym ENOB stand for in relation to ADCs? If two 12-bit ADCs have the same speed, but the one has higher ENOB and the other lower ENOB, which one would likely be the more accurate and also more expensive one to purchase – provide a brief argument to support your answer. **[6]**

1. ENOB = Effective Number Of Bits

2. The higher ENOB will probably be more expensive because it is more accurate, requires more precision manufacture, will have fewer of the LSBs fluctuate while sampling the same voltage.

Question 1.4 [12 marks]

This question relates to distributed memory and Message Passing. Please see Appendix B.

1.4 (a) What are some of the advantages of distributed memory over shared memory? Name at least two benefits. **[2]**

Distributed memory systems are more scalable. They are less dependent on a single architecture, for example a cluster of PCs that are not all of the same architecture is possible, where a new PC added doesn't necessarily have to have the same exact specifications as the other machines in the cluster.

1.4 (b) A starting point MPI program code is given in Appendix C together with a list of MPI commands in Appendix B for sending messages. Please complete the program given in order to satisfy the following requirements: **[10]**

i The master needs to ask the user (in the console) if it must be in mode A or B. In mode A, it prints out computed x:$y(x)$ pairs. In mode B, it prints each x value paired with '+', '0' or '-' to indicate if the $y(x)$ values returned is positive, zero or negative. The master must determine how many processes (`world_size`) are active. If `world_size` is less than two, the program exits.

ii Each process will calculate the result $y(x)$, where x is the node rank number and sends the x value and $y(x)$ result back to the master (node 0). The master does not compute a value. The equation for $y(x)$ is given below:

$$y(x) = x^2 - 12x + 20$$

iii The master must print the x:$y(x)$ pairs sent to it from the other nodes in the manner described above, depending if it is in mode A or B. (Hint: the x values don't need to be in order.)

Example: If the world size is 8 (i.e. 7 threads are spawned), these x values will be used by threads for $y(x)$:

```
x: 1,2,3,4,5,6,7
```

In mode A the following will be displayed (not necessarily in this sequence):

```
1:9 2:0 3:-7 4:-12 5:-15 6:-16 7:-15
```

In mode B the following characters will be displayed:

```
1:+ 2:0 3:- 4:- 5:- 6:- 7:-
```

Besides indicating local or global variables needed, you only need to write the lines of text you would add at the point indicated as:

```
if (world_rank==0) {
  // In Master node
  char mode = 'A';
  cout << "Please select mode... a: Mode A or b: Mode B\n";
  cin >> mode;
  if ((mode == 'a') || (mode == 'b')) {
    if (world_size>1) {
      int xy[2];
      for (int i=1;i<world_size) {
        MPI_Recv  (xy,sizeof(int)*2,MPI_BYTE,i,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
        cout << xy[0] << ":";
        if (mode == 'A') cout << xy[1];
        else if (xy[2]<0) cout << "-";
        else if (xy[2]>0) cout << "+";
        else cout << "0";
        cout << " ";
      }
      cout << endl;
    }else cerr << "Invalid number of processes, need at least 2\n";
  }else{
    // Report invalid mode given.
    cerr << "Invalid mode\n";
  }
}else{
  // In Slave node
  int xy[2];
  xy[0] = world_rank; // this is the x value
  xy[1] = x*x - 12*x + 20; // a float isn't needed
  MPI_Send (&y,sizeof(int)*2,MPI_BYTE,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
}
```

Section 2: Multiple Choice [30 marks]

Choose one option for each of the questions 2.1 to 2.4 in this section.

2.1 It is important to design OpenCL programs around the notion of workers and groups. Which option below accurately describes a difference between workers and groups in OpenCL? **[5]**

(a) In OpenCL, all the workers can run different program instructions, but they are all restricted to accessing the same "work-group" of memory – i.e. the 'group' refers to the specific group of memory blocks accessible to the workers.

(b) In OpenCL, all the workers run the same code, but they have different IDs (accessible by means of a special function for each worker); each group is an array of workers, a subset of the total workers on the GPU chip. ⇐ This one

(c) In OpenCL, workers run one at a time, and each one can access the same or different group of memory blocks but cannot access memory outside that group.

(d) In OpenCL, 'workers' refer to the processes that run in lock-step, and 'group' simply refers to the group of workers that are involved in the same mutex-synchronisation operation.

(e) None of the above.

2.2 Consider the Verilog code for moduleX (see next page). Select the option that best describes what this module does. **[5]**

(a) This module returns active 1 or active 0 respectively on whether the module has been tasked to count up or to count down.

(b) This is an up/down counter with reset, it counts up on every positive clock when active is high or counts down when active is low.

(c) This is an up counter with reset, it continuously counts up on every positive clock when rst is low. ⇐ This one

(d) This is a down counter, on reset high the initial count is set and held; when reset is low it starts counting down with every clock until zero is reached at which point it sets active high.

(e) This module initially sets active high, then counts up from 0 and when the maximum value is reached it sets active to low.

```
// Code for Question 2.2
module moduleX  (
  output reg [7:0]out,    // an output
  output          active, // another output
  input           clk,    // clock input
  input           rst     // reset input
);

// ------ Implementation -------
always @(posedge clk)
  if (rst) begin
    out    <= 8'b0;
    active <= 1'b0;
  end else begin
    out    <= out + 1;
    active <= 1'b1;
  end
endmodule
```

2.3 An SMP computer would, according to Flynn's taxonomy, be classified as which one of these computer architecture types? **[5]**

(a) SISD

(b) SIMD

(c) MISD

(d) DIMS

(e) MIMD ⇐ This one

2.4 What is meant by handshaking in a communications system? **[5]**

(a) This is a type of error detection scheme used in high-speed networks.

(b) This refers to initiating an asynchronous data transfer agreement between machine and the user.

(c) This refers to how two devices initiate communication and ensure successful exchange of desired information. ⇐ This one

(d) This refers to the parts of a computer design where there are bidirectional data connections.

(e) It merely refers to part of the boot sequence of an operating system whereby the CPU greets and introduces itself to each peripheral in the computer.

2.5 Answer **true** or **false** to each question below (each answer is 2 marks). **[10]**

(a) DeepQA is a natural language processing system. True

(b) A GPU is only able to process graphics data, in particular two dimensional matrices, vectors and pixels. False

(c) The real-world performance measure is a major telling factor in determining whether or not it was worth the effort to develop a parallel solution. True

(d) A GPU tends to support more threads than a CPU. True

(e) Amdahl's law states that processing speed always doubles when you double the number of processors. False

Section 3: Long Answers [42 marks]

You are tasked to implement a system with the following characteristics:

1. There are 7 input channels of fast analogue to digital (ADC) data, which are handled separately at this point. This data comes from a data-acquisition card that slots into the PCI bus of the computer. This data must be split into frames of constant length (512 samples, in this case). There are some component of house-keeping that must be performed to control the ADC hardware.
2. For each of the input channels, each frame must pass through an FFT.
3. The results from these FFTs are analysed by comparing the relative phases (i.e. subtract the phases element-wise) and comparing the amplitudes (also element-wise) with some constant threshold.
4. The results are displayed to the user by means of a GUI.
5. The GUI contains an interactive graph.

Question 3.1 [22 marks]

Speed is of utmost importance, but your hardware is limited to an 8-core Intel CPU and a single nVidia GPU. For each of the questions below, explain your reasoning.

3.1 (a) How would you split the work across the hardware in order to obtain the optimal performance? **[10]**

The CPU should handle the GUI on one core and the ADC data aquisision in another. It could be argued that there could be 1 core per ADC channel (7 cores total, but this depends largely on the hardware implementation of the data-acquisition card, which is not provided, so a single core for data aquisision is good.

A third core can be used to administer the general-purpose GPU (calculation of FFT's, data transfer, etc.), and a fourth can be used to administer the graphical aspects of the interactive graph. This results in 4 cores being used for 4 very different tasks, which a SMP, such as a multi-core Intel, is designed for.

The GPU should be used to calculate FFTs and perform element-wise vector analysis (by means of OpenCL or CUDA). Both of these algorithms will benefit from the SIMD architecture of GPUs.

The GPU should also be used to accelerate the graphics of the interactive graph (by means of OpenGL or DirectDraw, or similar).

A clever implementation will keep the OpenCL result in the GPU memory and display it directly, without any further data-transfer to the CPU RAM. This scheme further

keeps data transfer between the GPU and CPU (over the relatively slow PCIe bus) to a minimum.

3.1 (b) For each of the processing stages, which synchronisation method would you use (for example 'barrier', 'mutex', etc.)? **[8]**

This question is open to discussion, as it depends on the work-load spread suggested in the answer above. However, I would expect something along the lines of:

1. The ADC's are read in a single thread. The synchronisation between the ADC hardware and the CPU is most likely hard-ware implemented by means of an interrupt scheme, but an answer of "no synchronisation" would be acceptable.
2. The OpenCL controller runs in a different thread, so I would typically expect mutually exclusive access to the data buffers of the ADC thread. Once the data has been transferred to the GPU, and the GPU is told what to do, the OpenCL programming model makes use of barrier synchronisation automatically.
3. The FFTs and vector analysis runs on the GPU, by means of the OpenCL programming model. This model makes use of barrier synchronisation.
4. Even though the GUI runs in a single thread, it requires data from the other threads, and it needs to set parameters and switches in other threads. These would use mutually exclusive access.
5. The interactive graph thread is controlled by the GUI thread, which would use mutually exclusive access.

The synchronisation can also be implemented by means of semaphores, of which mutual exclusion is a special case.

3.1 (c) What programming model (for example 'message-passing', 'shared memory', etc.) would you use for each of the processing stages? **[4]**

This depends on the previous answers. The implementation in the memo thus far makes use of shared memory, but one can just as easilly argue in favour or message-passing.

Question 3.2 [20 marks]

Say, for argument sake, that the initial prototype (on the hardware above) proves the principle well enough that you are provided with a much larger budget (money, time and man-power) in order to implement the next generation of the system. You want to maximise profits for your company, so you'll need to make some engineering compromises (between performance, ease-of-maintenance and flexibility of the hardware, for instance). In each of the following questions, provide an argument for your design decisions.

3.2 (a) What hardware would you use? Would you consider an FPGA or ASIC, for instance, or even a Xeon Phi? Or would you stick to the CPU / GPU above? Would you make the system free-standing and feed results to a PC over the network, or keep everything in the PC (as above). **[10]**

This is also open to debate and depends on the arguments presented by the student. I would expect something along the lines of:

The system looks like a special-purpose device that will not have a mass-production market, in which case an ASIC is not appropriate. The student may also argue that the device is useful and will sell lots of copies, in which case an ASIC would be appropriate.

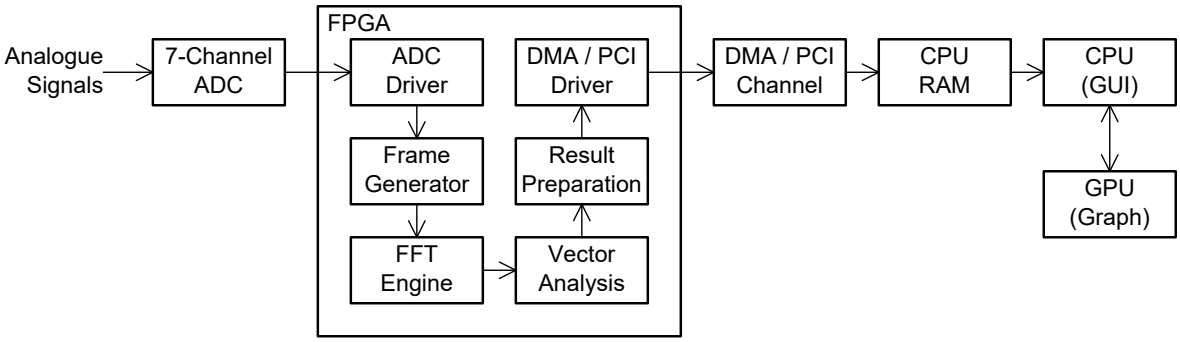
The use of an FPGA is promising. The ADCs can be connected directly to the FPGA instead of the PCI bus. This provides more flexibility in design. FPGA firmware is relatively easy to modify, making the hardware usable in other applications as well.

I would typically replace the ADC data-acquisition card with a custom-designed ADCs + FPGA card, also in the PC (one could make good arguments for using an external unit connected via Ethernet / USB / Blue-Tooth instead, but I would use a PCI card). I would keep the GPU.

This way, the FPGA can perform the data-acquisition, FFTs and vector analysis (or whatever algorithm the application dictates). The CPU can handle the GUI, and the GPU can handle the interactive graph.

3.2 (b) Provide a rough overview of how you would implement the system on your choice of hardware. A block diagram would help. **[10]**

A simple block diagram along the lines of:



is good, but a design with a soft-core micro-controller and system bus within the FPGA is also appropriate.

END OF EXAMINATION