Digital Systems
EEE4084F

FINAL EXAM

3 June 2014

3 hours

Examination Prepared by:
Simon Winberg

Last Modified: 01-Jun-2014

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

Marked out of 120 marks / 180 minutes. Time per mark = 1min 30sec

Section 1	Section 2	Section 3	Appendices
Short Answers *(4 x 12 mark questions)* [48 marks] pg 2	Multiple choice *(6x4-mark questions + 3x2-mark true/false q's)* [30 marks] pg 4	Long Answers *(1 question)* [42 marks] pg 6	A: Formulae B: Verilog cheatsheet C: OIC Instructions D: detachable sheet pg 8

RULES

- You must write your name and student number on each answer book.
- Write the question numbers attempted on the cover of each book.

NB ⇨
- Start **each section on a new page**.
- 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 carefully constructed responses.
- Answer all questions, and note that the time for each question relates to the marks allocated.

Section 1: Short Answers [48 marks]

Question 1.1 [12 marks]

This question related to the grid network shown in Figure 1 below, which comprises 18 of the same type of nodes, where each node only connects to its nearest neighbors via a 1Gbps link. This is essentially a non-uniform configuration because it is not a square and some of the nodes (e.g. corner nodes) have less links than others. Answer the following…

(a) The concept of *bisection bandwidth* is useful in gauging the performance of networked systems. Briefly explain the concept of bisectional bandwidth and why it is useful in judging the network performance design of a high performance computer system. [4 marks]

(b) Discuss, and calculate a bandwidth value, for the bisection bandwidth of the network shown in Figure 1. Assume each link has bandwidth of 1Gbps. Explain (using a rough diagram if needed, or a listing links to show the) how you decided the 'bisection' would be done in this case. [5 marks]. (*Hint:* Appendix A has formulas that might, or might not, be useful.)

(c) If each link in the network is 1Gbps, explain the following:

i. What is the maximum speed P1 can continuously stream data to P18, and P18 stream back to P1, assuming a 0us (i.e. zero) time cost for each node to route data from one link to another[1] [2 marks].

ii. What is the maximum speed P1 can stream to P18 and P6 can simultaneously stream data to P13? Only indicate the bandwidth P1 will steam at… [1 marks]

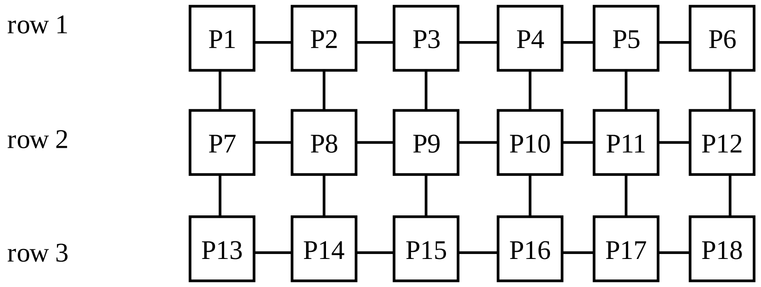


Figure 1 Cluster Topology for Question 1

Question 1.2 [12 marks] [12 marks]

Answer the following questions and sub-questions concerning computation approaches…

(a) What is the different difference between temporal and spatial computation? Explain briefly. [2 marks]

(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 an ABI is [2 marks], and why different operating systems that run on the same platform might have different ABIs? [2 marks]

(c) Many RC platforms are built using FPGAs; but these systems often also include a CPLD in addition to a FPGA in the platform design.
i. What is the main difference between a FPGA and a CPLD? [2 marks]
ii. Explain why the CPLD in a reconfigurable computing system is usually used as configuration / glue logic whereas FPGA(s) are more commonly used for the actual computation (e.g., digital filtering) operations. [4 marks]

[1] Consider that a connection from P1 to P3 routed via P2 still gives a 1Gbps performance; but if P1 sends two streams at the same time to both P2 and to P3 via P2, then there is only a 500MHz performance for both P1-P2 stream and for the P1-P2-P3 stream).

Question 1.3 [12 marks]

The design and implementation of parallel programs can be considered to involve seven major steps. These steps are listed below, refer to as the 'seven steps to parallelization model' in this question:

1. Understand the problem
2. Partitioning
3. Decomposition & Granularity
4. Communications
5. Identify data dependencies
6. Synchronization
7. Load balancing
8. Performance analysis and tuning

Answer the following:

(a) In terms of the step "Understanding the problem" the typical approach starts with establishing requirements for the system and later identifying what is termed critical "hotspots" where most of the parallelization work is done. Motivate why this identification of hotspots and parallelization of them is, all things considered, probably a more effective strategy than attempting to develop an entirely parallel solution for the problem. [3 marks]

(b) The spiral model provides a different perspective on the procedures, showing a more cyclic view. Yet both models are generally accurate, especially for large system development. Briefly describe the spiral model, indicating how it models the progression of development and mention (at least three) major activities that are commonly iterated in this model. Provide a rough figure to assist your explanation.[5 marks]

(c) The concept of granularity is sometimes misunderstood. Help clarify the term by answering the following: If the sequence of data {x1, x2, x3, … x*N*} has to be passed through each of *M* processors nodes in a cluster to compute the result (considering N large, M > 10, N >> M), would this computing problem be classified as *course grained* or *fine grained* granularity? (One word answer is fine.) [2 mark]

(d) In terms of load balancing, what is a typical case where the approach of "dynamic work assignment" would be applied rather than the approach to "partition work as a pre-process". [2 marks]

Question 1.4 [12 marks]

This question concerns programmable logics.

(a) Describe two advantages that FPGA-based development has over ASIC development. [2 marks]

(b) The time it takes to effectively learn how to use an HDL can be a significant constraining factor to using an FPGA in a design. Discuss two tools or type of HDL code generators that corporations such as Altera or Xilinx (and also other companies) have provided as a means to work around this difficulty. [4 marks]

(c) What does the acronym PLA stand for? Discuss (2) differences between a PLA and a FPGA? [3 marks]

(d) Altera and Xilinx are the Big 2 FPGA manufactures. Actel and Lattice, while also big companies, focs on more specialized FPGAs. Choose either Actel or Lattice and explain what sort of specialized FPGAs they focus on developing. [3 marks]

Section 2: Multiple Choice [30 marks]

NOTE: Choose only one option (i.e. either a, b, c, d or e) for each question in this section.
The first 6 questions are 4 marks each, question 7 has 3 true/false 2-mark each questions.

Q2.1 The IBM Cell Processor comprises multiple cores; collection of cores does it comprise? Chose one option below…

(a) The IBM Cell comprises 1 power processor and 7 Synergistic Processor Elements (SPEs).

(b) The IBM Cell comprises 7 power processors and 1 Synergistic Processor Element (SPE).

(c) The IBM Cell comprises 1 power processor and 8 Synergistic Processor Elements (SPEs).

(d) The IBM Cell comprises 8 power processors and 2 Synergistic Processor Element (SPEs).

(e) The IBM Cell comprises 1 power processors and 9 Synergistic Processor Element (SPEs).

[4 marks]

Q2.2 What is another term for synchronous communications?

(a) Untimely communications.

(b) Non-blocking communications.

(c) Handshaking communications.

(d) Serialized transfers.

(e) Blocking communications.

[4 marks]

Q2.3 Consider that the following variables are defined:

f = fraction of computation that can be parallelized n = number of processors for parallel case

Then, according to Amdahl's law, the maximum speed-up achievable for the parallel case over a sequential case is (chose one option below):

(a) Speedup = $(1 – f) / ((1 – n·f))$

(b) Speedup = $1 / ((1 – f) + f/n)$

(c) Speedup = $(1 – f) / (f/n - 1)$

(d) Speedup = $1 / (f – 1)(n+1)$

(e) Speedup = $(n + 1) / (1 – f)$

[4 marks]

Q2.4 Which statement below is accurate concerning CMOS?

(a) CMOS is found in only a few ICs nowadays.

(b) CMOS gates operate significantly faster than TTL.

(c) CMOS can only implement NAND and NOR operations.

(d) CMOS is more power hungry that TTL.

(e) All the above statements are wrong.

[4 marks]

Q2.5 What is the ‘configuration architecture’ of a FPGA system? (Select the most accurate answer below.)

(a) The structuring of the FPGA that specified how LUTs are arranged in the chip.

(b) The specification of how binary data is clocked into the FPGA during programming.

(c) The mapping of the FPGA internal chip circuitry to the pins that are soldered to the PCB.

(d) The underlying circuitry that loads configuration data and stores it at the correct locations.

(e) This refers to the layout of HDL modules in a design that the FPGA is programmed with.

[4 Marks]

Q2.6 Which description below summarizes the **work queues** design pattern? (Choose one option).

(a) This is a client/server structure where the server creates a client thread to do work.

(b) This has a producer/consumer structure where the producer adds a new job to the queue that is then worked on by the consumer at a later stage (usually in a first come first served basis).

(c) This is a structure where two paired workers, the consumer and producer, are created and that consumer works on data generated by that producer.

(d) This is a type of divide and conquer approach where one producer spawns multiple consumer threads that work on data generated by the producer.

(e) The ‘work queues’ pattern is simply a shared memory approach where a shared block of memory is set up and written to by the main thread and then read and processed by multiple worker threads.

[4 Marks]

Q2.7 Answer **true** or **false** to each question below (each answer is 2 mark).

(i) The Berkeley Landscape of Parallel Computing Research paper (seminar 1) motivated that a new conventional wisdom is that “Power is free, but transistors are expensive.”

(ii) The resolution of an ADC is specified using the index value n in the expression 2^n where 2^n is the total number of digital output codes that the ADC can produce.

(iii) A network arranged into a FAT tree structure has a bisection bandwidth that outperforms that of a binary tree structure with more than three nodes in the network.

[3 x 2 mark each = 6 marks]

Section 3: Long Answers [42 marks]

This section has only one rather lengthy question and not-so-lengthy question

Question 3.1 [36 marks]

Consider the HDL pseudo code below:

```
MyOp (Y, X   : 4-bit register output;
      A, B, C: 4-bit input; clk : input) {
     Do (whenever positive edge of clk) {
      IF GREATER_THAN(A,B) THEN X := AND(A,C);   /* i.e. if A>B then … */
               ELSE X := AND(B,C);
      Y := AND(ADD(A,B),C);
     }
} /* end of MyOp definition */

MyOp_Testbench () {
/* declare some registers: */
4-bit register: A, B, C;
4-bit register: X, Y;
1-bit register clk;
initial begin {
     /* Set the initial conditions */
     A := 1; B := 7; C := 15;
     clk := 0;
     /* wait a bit to send the low clk */
     clk := 1;
     /* create a positive edge for clk and wait a bit for MyOp to compelte */
     print("X = ",X); /* show the results */
     print("Y = ",Y);
     end simulation;
     }
   MyOp op1(A,B,C,X,Y,clk);  /* instantiate MyOp and link registers to it */
}; /* end of testbench */
```

See the next page for the questions… (you can detach this page from the question paper if it you like)

In the code above the first statement (i.e., the if statement and its else part) and the second statement (the assignment to Y) is done concurrently and continuously form when the system is turned on. Complete the following…

(a) Convert the above code, both the MyOp module and the test bench into Verilog. You do not need to provide any includes, but make sure that you use the syntax as correctly as you can to define the operation as a module and to instantiate the module appropriately. Appendix B contains a Verilog cheat sheet.
(3 marks will be given to use of comments.) [14 marks]

(b) Consider that you wanted to program the programmable logic device in Appendix D with the MyOp module shown in the HDL program above. Use Appendix D to show how MyOp could be implemented on the device. Detach Appendix D from the question pages, write your student number where indicated on the page and make sure you include it in your answer book. (Note that the inputs are on the left, marked A,B,C and there is also input lines linked to logic 0 on the left, which might be useful, and also input lines on the right linked to logic 1.) [7 marks]

(c) Appendix D shows in italic font delay latencies in ns for each logic element in the device. Determine what the latency of the implemented MyOp module that you did in answer to (b) above. [5 marks]

(d) Consider that you were to convert MyOp into CPU instructions to run on a simple PIC-like microcontroller, specifically the hypothetical OIC (One Instruction per Clock) processor that executes each instruction in just one clock that runs at 50MHz. See Appendix C shows the OIC CPU instruction set. You can assume A,B,C,X,Y are respectively assigned port addresses 1,2,3,4 and 5. Determine whether the OIC be faster or slower running the implementation for (b) running on the programmable device? To do this, complete the following steps:

i. Write a simple OIC assembly program that implements MyOp (NB: Only implement code for the MyOp module not for its testbed. Also you need to use IN and OUT to get the inputs for A,B,C and outputs for X and Y. You can be creative with assembly syntax but you can't add additional instructions). A mark for code comment(s) will be awarded. [6 marks]

ii. Based on each instruction to complete within one clock cycle, show your working to determine how long the OIC implementation of MyOp will take to run [3 marks].

iii. What is the speed-up (to two decimal points) of the faster of the two implementations over the slower of the two? (obviously it depends on your solution to ii) [1 marks]

[36 marks]

Question 3.2 [6 marks]

The figure on the right illustrates the Von Neumann architecture. Answer the following brief sub-questions:

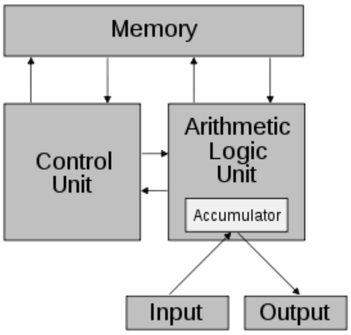


Illustration 1: Von Neumann Architecture

(a) Discuss one advantage and one disadvantage of the Von Neumann architecture, from a perspective of a CPU designer. [2 marks]

(b) Explain what is meant by the Von Neumann bottleneck in relation to data access and transfer. [2 marks]

(c) Provide an argument as to whether or not the Von Neumann architecture is still an appropriate strategy for today's highly parallel computer systems. (You can refer to your above answers). [2 marks]

END OF EXAMINATION

Appendix A:

Formulae

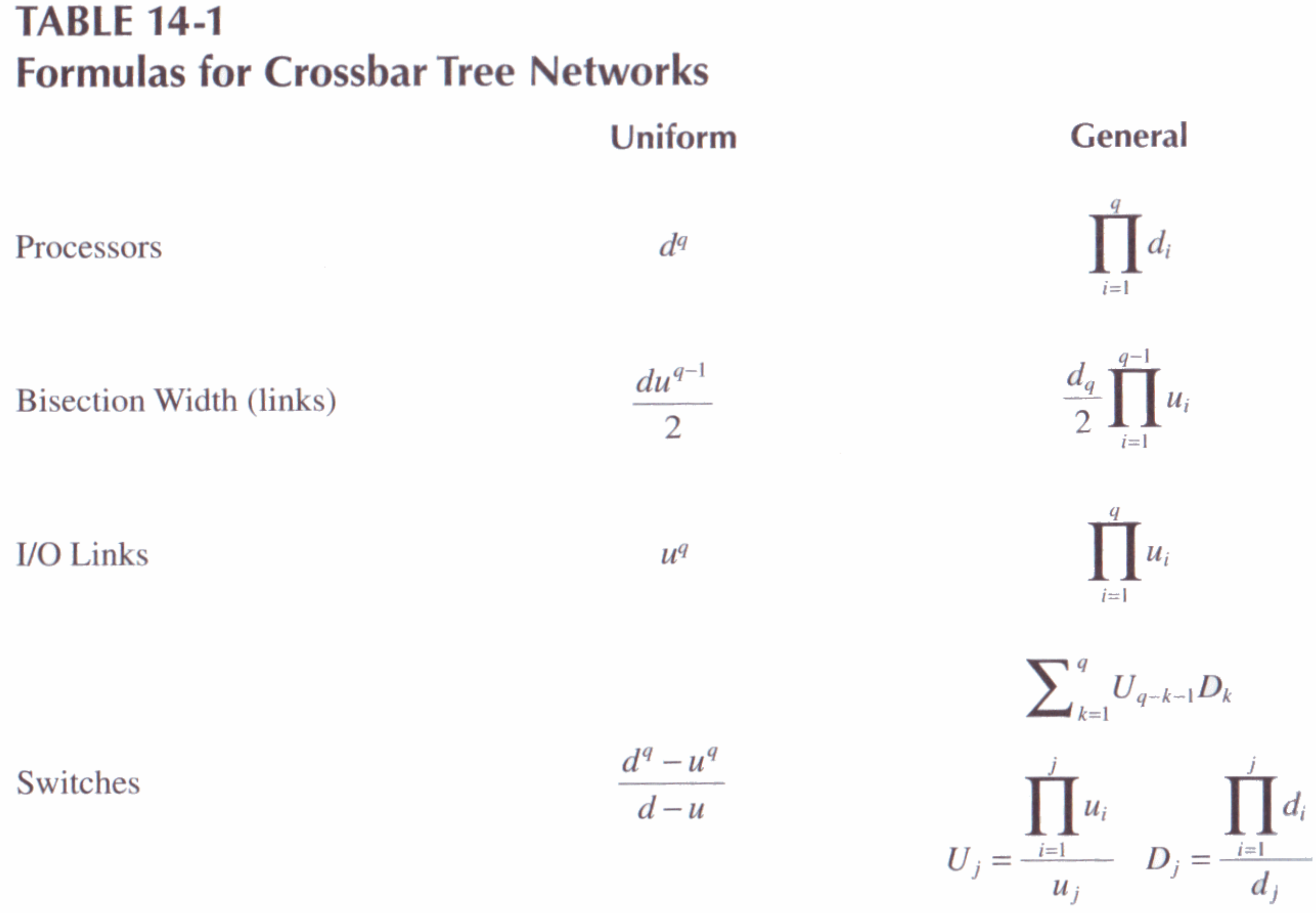


TABLE 14-1
Formulas for Crossbar Tree Networks

	Uniform	General
Processors	d^q	$\prod_{i=1}^{q} d_i$
Bisection Width (links)	$\frac{du^{q-1}}{2}$	$\frac{d_q}{2} \prod_{i=1}^{q-1} u_i$
I/O Links	u^q	$\prod_{i=1}^{q} u_i$
Switches	$\frac{d^q - u^q}{d - u}$	$\sum_{k=1}^{q} U_{q-k-1} D_k$ $U_j = \frac{\prod_{i=1}^{j} u_i}{u_j} \quad D_j = \frac{\prod_{i=1}^{j} d_i}{d_j}$

Table from pg 291 of Martinez, Bond and Vai 2008

Appendix B: Verilog Cheat sheet

Numbers and constants

Example: 4-bit constant 10 in binary, hex and in decimal: 4’b1010 == 4’ha -- 4’d10

(numbers are unsigned by default)

Concatenation of bits using {}

4’b1011 == {2’b10 , 2’b11}

Constants are declared using parameter:

parameter myparam = 51

Operators

Arithmetic: and (+), subtract (-), multiply (*), divide (/) and modulus (%) all provided.

Shift: left (<<), shift right (>>)

Relational ops: equal (==), not-equal (!=), less-than (<), less-than or equal (<=), greater-than (>), greater-than or equal (>=).

Bitwise ops: and (&), or (|), xor (ˆ), not (˜)

Logical operators: and (&&) or (||) not (!) note that these work as in C, e.g. (2 && 1) == 1

Bit reduction operators: [n] n=bit to extract

Conditional operator: ? to multiplex result

Example: (a==1)? funcif1 : funcif0

The above is equivalent to:

```
  ((a==1) && funcif1)
  || ((a!=1) && funcif0)
```

Registers and wires

Declaring a 4 bit wire with index starting at 0:

wire [3:0] w;

Declaring an 8 bit register:

reg [7:0] r;

Declaring a 32 element memory 8 bits wide:

reg [7:0] mem [0:31]

Bit extract example:

r[5:2]   returns 4 bits between pos 2 to 5 inclusive

Assignment

Assignment to wires uses the assign primitive outside an always block, e.g.:

assign mywire = a & b

Registers are assigned to inside an always block which specifies where the clock comes from, e.g.:

```
always@(posedge myclock)
    cnt = cnt + 1;
```

Blocking vs. unblocking assignment <= vs. =

The <= assignment operator is non-blocking (i.e. if use in an always@(posedge) it will be performed on every positive edge. If you have many non-blocking assignments they will all updated in parallel. The <= operator must be used inside an always block – you can’t use it in an assign statement.

The blocking assignment operator = can be used in either an assign block or an always block. But it causes assignments to be performed in sequential order. This tends to result in slower circuits, so avoid using it (especially for synthesized circuits) unless you have to.

Case and if statements

Case and if statements are used inside an always block to conditionally update state. e.g.:

```
always @(posedge clock)
  if (add1 && add2) r <= r+3;
  else if (add2) r <= r+2;
  else if(add1) r <= r+1;
```

Note that we don’t need to specify what happens when add1 and add2 are both false since the default behavior is that r will not be updated. Equivalent function using a case statement:

```
always @(posedge clock)
  case({add2,add1})
  2’b11  : r <= r+3;
  2’b10  : r <= r+2;
  2’b01  : r <= r+1;
  default: r <= r;
endcase
```

Module declarations

Modules pass inputs, outputs as wires by default.

```
module ModName (
  output reg [3:0] result,  // register output
  input [1:0] bitsin,  input clk  );
 … code …
endmodule
```

Verilog Simulation / ISIM commands

```
$display ("a string to display");
$monitor ("like printf. Vals: %d %b", decv,bitv);
#100   // wait 100ns or simulation moments
$finish  // end simulation
```

Appendix C:
One Instruction per Clock (OIC) Instruction Set

The OIC is an 8-bit processor that can address only 256 bytes of memory and get input/output via port access to 256 port addresses. Each register-only instruction takes one byte to store; each instruction using a constant takes two bytes, one for the instruction the other for the 256 byte constant value.

OIC Instructions

Instruction	***Description***
IN X, *reg / const*	Input value into X from port *reg* or from construct *const*
IN Y, *reg / const*	Input value into Y from port *reg* or from construct *const*
OUT X, *reg / const*	Output accumulator to port *reg* or constant *const*.
ADD X, *reg / const*	X = (X + *reg*) *or* X = (X + *const*)
ADD Y, *reg / const*	Y = (X + *reg*) *or* Y = (X + *const*)
AND X, *reg / const*	X = (X and *reg*) *or* X = (X and *const*)
AND Y, *reg / const*	Y = (X and *reg*) *or* Y = (X and *const*)
OR X, *reg / const*	X = (X or *reg*) *or* X = (X or *const*)
OR Y, *reg / const*	Y = (X or *reg*) *or* Y = (X or *const*)
SWP *reg1*, *reg2*	Swap values stored in reg1 and reg2
NOT X, *reg*	X = not *reg*
ROR reg, *n*	X = rotate right *reg* by *n* bits
ROL reg, *n*	X = rotate left *reg* by *n* bits
NEG X	X = -X
NEG Y	Y = -X
CMP X, *reg*	Compare X and *reg* *(reg can be any reg A-H or X or Y)*
CMP X, *const*	Compare X and *const*
JMP *reg*	Jump to address pointed to by *reg* (JMP X is the same as SI)
JMP *const*	Jump to address *const*, i.e. I = *const*
JMP*f* *reg*	Jump to address pointed to by *reg* if flag *f* set
JMP*f* *const*	Jump to address pointed to by *const* if flag *f* set
SKIP	Skip the next instruction unconditionally
SKIP*f*	Skip the next instruction only if conduction flag f is set…
LI	X = I (i.e. load instruction pointer into X)
SI	I = X (i.e. jump to address X)

OIC Registers

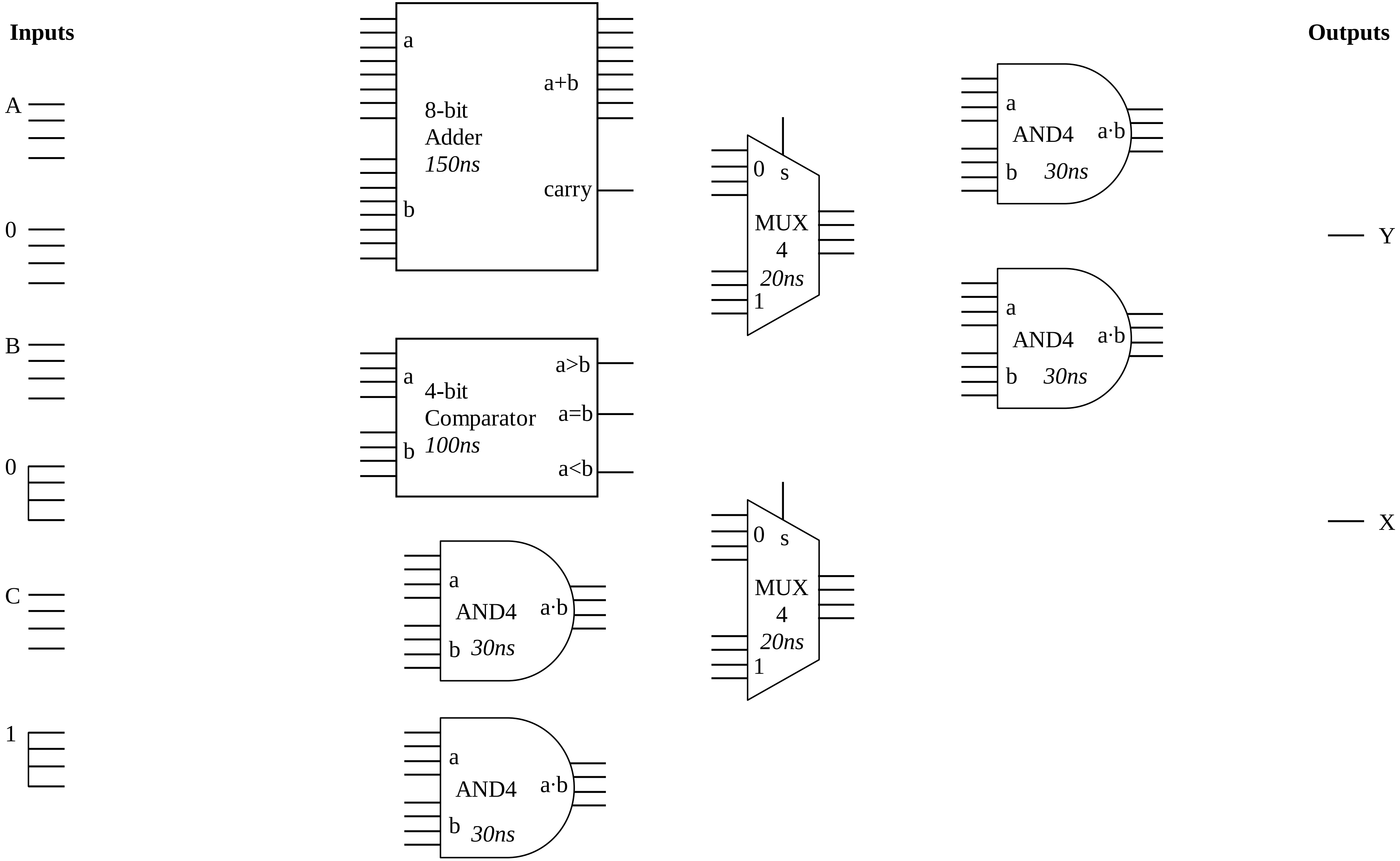
Register Name	***Register Description***
X	Accumulator
Y	General purpose 8-bit register, can also store results from some ALU operations
A – F	General purpose 8-bit register
H	Flag register (can also be used as input to ALU operations)
I	Instruction pointer – cannot be used with ALU operations beside LI or SI

OIC Flags

Flag Name	***Flag Description***
z	Zero / accumulator result was 0 or comparison was equal
eq	Comparison was equal or accumulator result was equal to 0
gt	Set true if *reg* > A for CMP A, *reg* or accumulator result was >0
lt	Set true if *reg* < A for CMP A, *reg* or accumulator result was <0
gte	Set true if *reg* >= A for CMP A, *reg* or accumulator result was >=0
lte	Set true if *reg* <= A for CMP A, *reg* or accumulator result was <=0
c	Set true if carry occurred in accumulator

Appendix D: Detachable Appendix – Programmable Logic Device for Question 3 (b)

Name: ________________________ Student Number: ________________________ Please detach and put in your answer book!!



Inputs
A
0
B
0
C
1
a
8-bit
Adder
150ns
a+b
carry
b
a
4-bit
Comparator
100ns
a>b
a=b
a<b
b
a
AND4
a·b
b
30ns
a
AND4
a·b
b
30ns
0
s
MUX
4
20ns
1
0
s
MUX
4
20ns
1
a
AND4
a·b
b
30ns
a
AND4
a·b
b
30ns
Outputs
Y
X