# Digital Systems
# EEE4084F

## FINAL EXAM

***6 June 2011***

***3 hours***

*Examination Prepared by:*
*Simon Winberg*

*Last Modified: 31-May-2011*

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 100 marks. Time per mark = 1min 48sec



**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 1.1**

In the lectures, the design and implementation of parallel programs generally involved seven major steps. These steps are listed below, which we can informally refer to as the 'seven steps to parallelization model':

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

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.

Answer the following:

(a) 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]

(b) Explain how the seven steps to parallelization model can be positioned within the spiral model. Indicate for example where each of the seven steps would occur in the spiral model, and which of these steps might be revisited again and again as the project progresses. [6 marks]

(c) Name one commonly occurring problem in the design of large parallel computing systems. [1 mark]

[12 marks]

**Question 1.2**

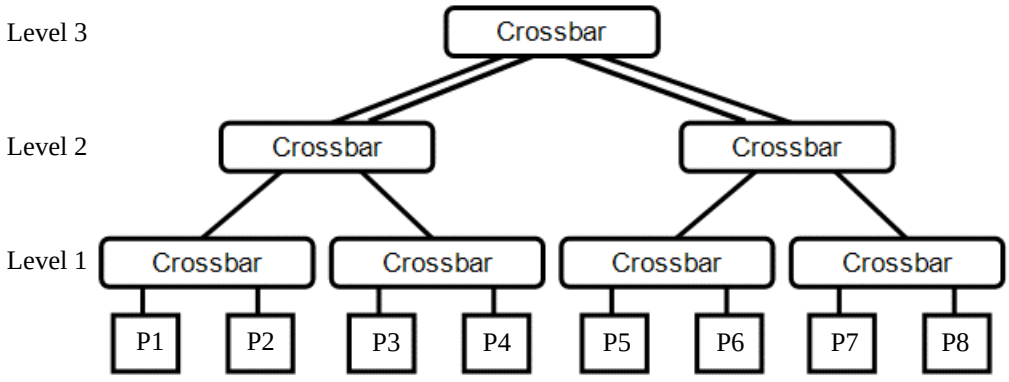
The concept of *bisection bandwidth* is useful in gauging the performance of networked systems.

(a) Briefly explain the concept of bisectional bandwidth and why it is useful in judging the network performance design of a high performance computer system. [3 marks]

(b) The three-level crossbar tree network below is a non-uniform configuration. Calculate the bisection bandwidth for the network, assuming each link is 1Gbps. [3 marks]. (*Hint:* Appendix B has formulas.)

(c) What change would be needed to make it into a three-level fat tree network? Provide a sketch if you find it difficult to explain only in text. [3 marks]

(d) If each link in the network is 1Gbps, explain the maximum speed P1 can send data to P8, and P5 can simultaneously send data to P3. Would additional downlinks in level 2 speed things up? [3 marks]




[12 marks]

*Figure 1 crossbar tree for Q2*

**Question 1.3**

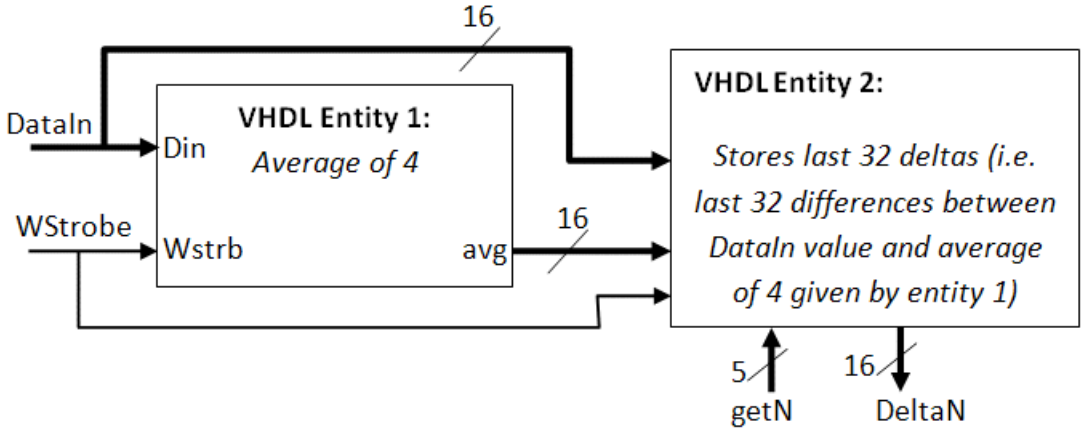
This question relates to FPGAs and RC platforms.

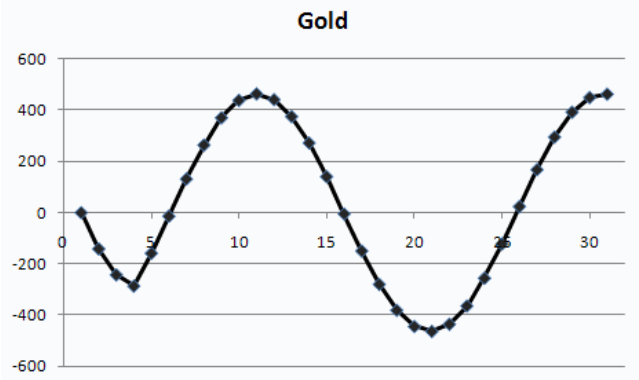
(a) Define what is understood to be a *reconfigurable computer*, including an explanation of how to discriminate between a reconfigurable computer and a non-reconfigurable / regular networked computer. [2 marks]

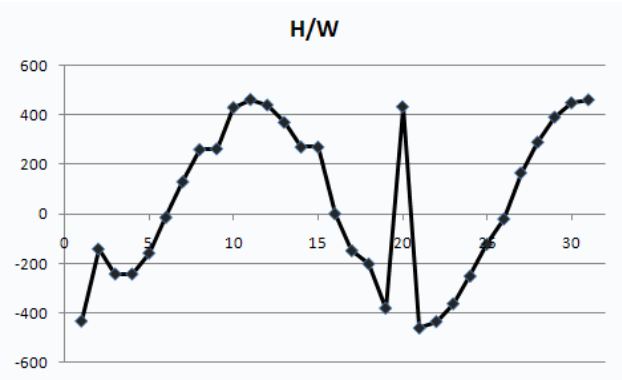
(b) What is a LUT in relation to FPGAs? Why do FPGAs usually have lots of LUTs instead of the more fundamental logic gates (e.g., AND, OR and NAND gates)? [2 marks]

(c) The Xilinx Virtex-6 comprises an architecture composed of two types of configuration logic blocks (CLBs), namely SLICEM and SLICEL blocks. Firstly, what exactly is a CLB – surely it is just the same as a LUT? [2] Discuss the differences between SLICEL and SLICEM blocks and how different slides can be beneficial in terms of FPGA manufacture and programming [2]. [4 marks]

(d) Consider the design below. The system comprises two VHDL entities, imported into a schematic editor (such as the Altera Quartus II block diagram editor). But when the design is synthesized and executed on hardware, unexpected results occur (see the graphs below) even though data always comes into the system at a much slower rate than the speed at which the entries operate. Indeed, much of the output from the hardware looks correct (see right hand graph below) yet doesn't match the golden measure. The correlation coefficient between the datasets is 0.834, verifying that something is wrong. Comment on what may be causing this problem and how it might be solved *without* having to change the VHDL code of either entity. [4 marks]




[12 marks]

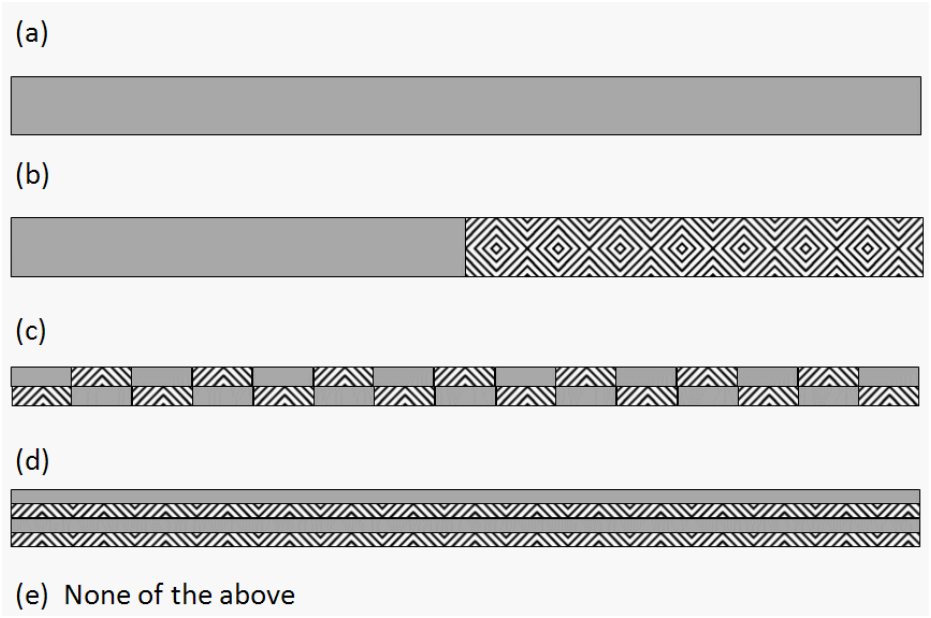
# Section 2: Multiple Choice [20 marks]

Q2.1 Select the most accurate description of a ‘configuration architecture’ for a reconfigurable computer:

(a) A software application used to program a FPGA or PLD on the reconfigurable computer.
(b) ‘Configuration architecture’ simply refers to the general architecture of a reconfigurable computer.
(c) The circuit components of a reconfigurable computer for communication between a PC or back-end system and the FPGA(s).
(d) Circuitry on a reconfigurable computer that stores configuration data and loads it into the relevant locations or devices (e.g. FPGAs) on the platform.
(e) The way in which a cluster of computers are networked.

[4 marks]

Q2.2 Which visualization below represents an *interleaved partitioning* considering that it is an image (e.g. a BMP file) that is being separated between two processors?




[4 marks]

Q2.3 In terms of processor architectures, what does the acronym SMP stand for?

(a) Synthetic Mico-Processor
(b) Symmetric Multi-Processor
(c) Systematic Microprocessor Production
(d) Silicon-based Multiple Processor
(e) Standard Microprocessor Platform

[4 marks]

Q2.4 How would you classify a typical nVidia CUDA-enabled GPU architecture on Flynn's taxonomy? Choose the most appropriate classification below.

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

[4 marks]

Q2.5 Answer **true** of **false** to each question below (each answer is 1 mark).

(i) The commonly used HPEC acronym “SWAP” means Stop Wait And Process.
(ii) VHDL can be used to program FPGAs, but it can’t be used to program CPLDs.
(iii) Front-end processors tend to be the highest-speed element of a HPEC system.
(iv) A correlation between two identical data sets returns the value 1.

[4 marks]

# Section 3: Long Answers [44 marks]

## Question 3.1 [16 marks]

Consider the template code for a pthread application shown in Appendix A. Your task for this question is to provide a multi-threaded parallel solution for the vector scalar product. The formula is shown on the right. Assume vectors *a* and *b* are defined as global arrays of doubles and *x*, the result, is also a double, i.e. using the code below:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

here: *n* = LENGTH of vector

```
double a[LENGTH];  // input 1  -- assume all element in arrays are assigned
double b[LENGTH];  // input 2 -- you can also assume LENGTH is a power of 2, say 16384
double x; // result
```

(a) Provide a quick golden measure in C or C++ for this problem, include printing of the result. Use the LENGTH constant for the vector lengths. The *n* parameter in the code (i.e., for num. threads) is obviously irrelevant for this part of the question. [4 marks].

(b) Rework the code of function *threadfn* in Appendix A to provide a parallel pthread implementation for this problem. Account for 1 to n partitions, i.e. one partition for each thread – so if four threads are specified, all four thread needs to do some of the work. You don't need to rewrite the whole program; use the line numbers to indicate what you would change or where you'd add code. [12 marks]

(*Hint:* remember to combine the results of the threads, and keep in mind that *writing* to the same global variable can cause synchronization problems.)

## Question 3.2 [17 marks]

This question concerns writing a hardware module using Handle-C style in order for the code to be converted directly to VHDL and then used in a digital accelerator to speed-up processing (Appendix C provides Handle-C syntax). The module you are to develop is called the 'Period Length Calculator' (or PLC). The PLC takes three inputs, namely: an eight-bit data input *x* that represents a sampled value; *newx* to indicate (via a positive edge) that the *x* input has changed, and *reset* to clear the device. The module has four outputs: *ready* indicates the module is ready to receive more data (i.e. ready for another *x* value), *valid* is set high when the other output lines are valid (i.e., when it has finished computing a result), the *period* output gives the pulse length of the sampled input which indicates the most recent number of samples calculated between the last two peaks found, and *peak* indicates the most recent peak value found (i.e. the most recent local maximum of *x*). A prototype for the Handle C function is provided below:

```
// C prototype for pulse length calculator (PLC)
void PLC ( _in  byte 8 x,   // input data byte
           _in  bit newx,  // positive edge when new data available
           _in  bit reset, // set high to clear the
           _out bit ready, // set high when device is ready for more input
           _out bit valid, // high when output is valid
           _out unsigned 16 period, // num samples from one peaks to the next
           _out unsigned 8 peak, // the most recent peak value
         );
```

Implementation notes:

The peak value needs to be calculated based on turning points of the sampled data that is coming in as *x* values. Take a look at the graph on the next page. As you may notice, the amplitude of the sampled signal is diminishing yet the PLC is able to account for this by sensing when the gradient on *x* values changes. The figure indicates a period of 21 was calculated, and that after the final x was sent, the peak output remained at the value 160 (i.e., the value of the last peak in the data). Assume a reset will always be given before *x* values start being sent. The *x* input will be valid once the *newx* input line goes from a 0 to and 1, and it will remain at 1 for a few nanoseconds before returning to 0 in good time before the next x value is sent.

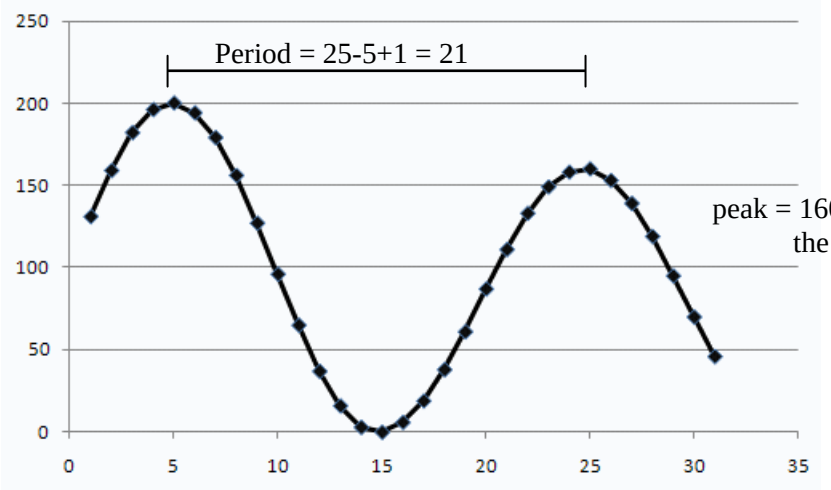



*Figure 2: Plot of sampled x values fed into the PLAC module.*

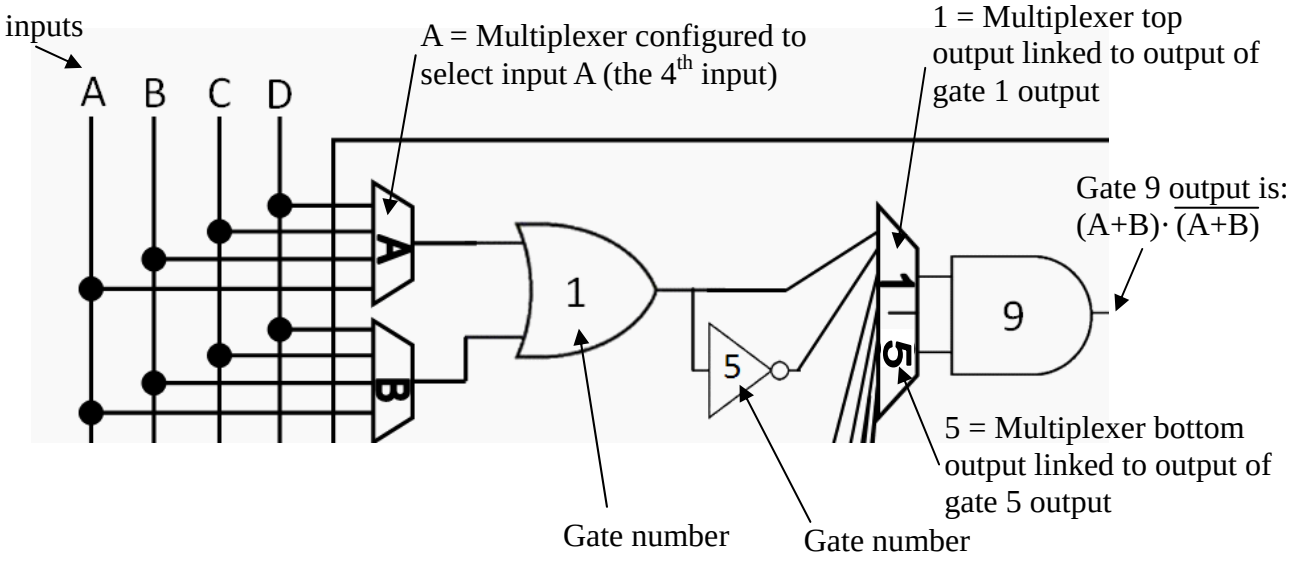
***What to do for this question (Question 3.2)…***

Provide the HandleC implementation for the PLAC function. Duplicate the function prototype in your answerbook, and if needed you may add additional parameters (and comments explaining why if you do so). Some of the marks are allocated to use of comments in your answer. *Hint: Make sure you use the right datasizes for datatypes.*

[17 marks]

## Question 3.3 [11 marks]

Consider a hypothetical FPGA with programmable blocks as shown on the next page. The notation used in the diagram to label gates and describe how multiplexors are configured is explained below.



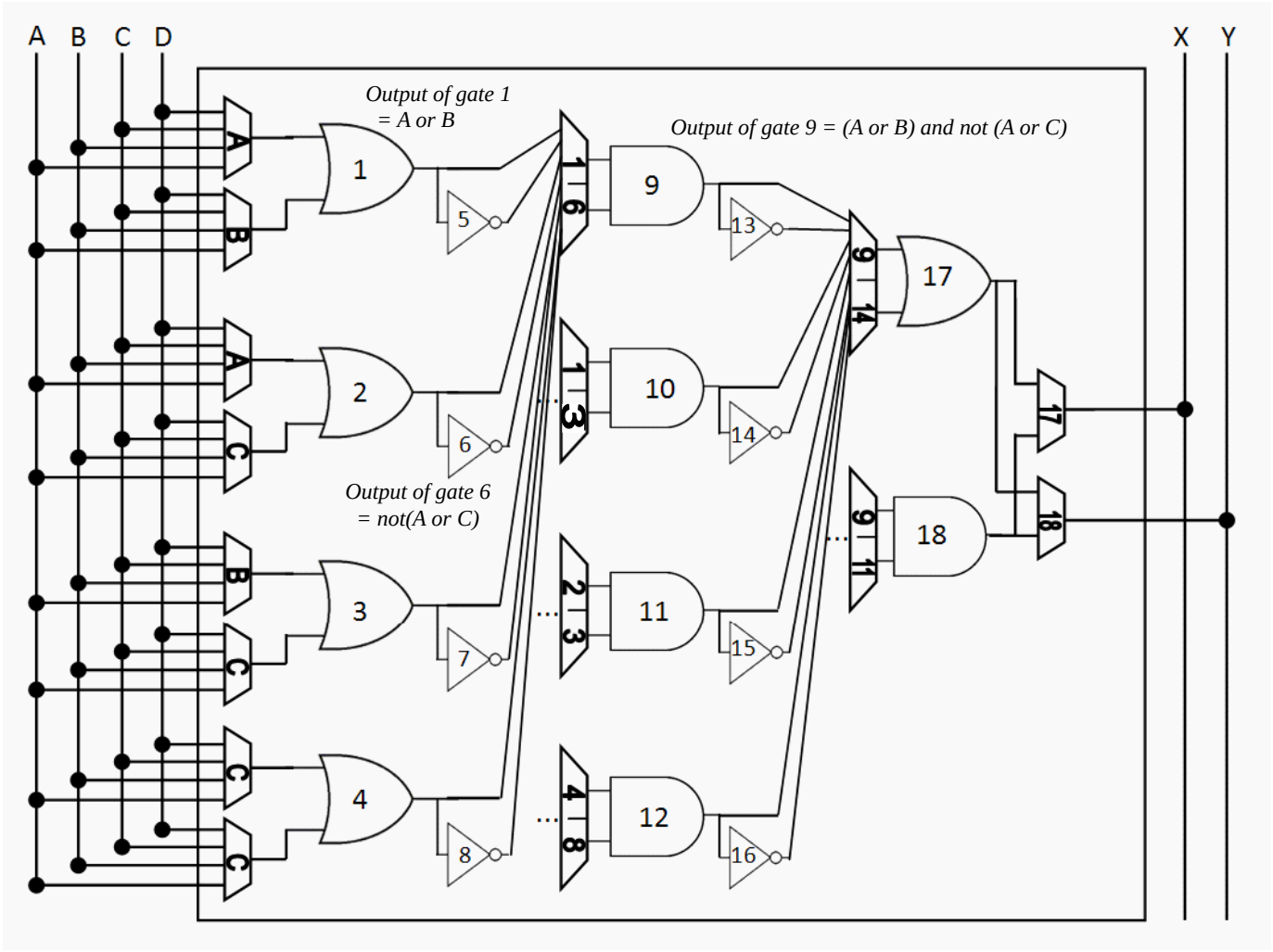

Essentially, the label in the multiplexor indicates which input is selected to be outputted by the multiplexor.

In the example above, the first multiplexor connects input A to the first OR gate, and similarly first second multiplexor connects input B to the second multiplexor. The multiplexor on the far right is configured to connect the or gate (gate 1) to the top output of the multiplexor that connects to the top input of gate 9, and similarly the bottom input of gate 9 connects to the inverter output. You can thus work out that the output of gate 9 will end up being the expression:

(A or B) and not (A or B)

In this trivial case the output value is obviously always going to be 0 (according to the axium X and not X = 0).

Now take a look at the configured FPGA on the next page… and answer the questions that follow.




*Notes re multiplexors:* the inputs of all multiplexors in the second column respectively connect to outputs of gates 1-8. Similarly, each multiplexor in the third column has eight inputs which respectively link to the outputs of gates 9-16.

**What to do for Question 3.3…**

(a) Provide a logical expression (or VHDL statement) that specifies the outputs of X and Y in terms of the inputs A, B, C and D. This is fairly straightforward: I've shown how you can determine the intermediate results of independent gates (i.e., gates 1, 6 and 9) according to how the multiplexors are configured. [6 marks] [1]

(b) If each multiplexor has a propagation delay of 8ns, and the AND, OR and NOT gates all have a 10ns propagation delay, what can you say about the overall propagation delay from a change in the input signals to a subsequent change of the X or Y output lines? [5 marks]

(*hint:* you can focus your insights on the minimum and maximum propagation delays.)

END OF EXAMINATION

[1] Bonus marks (up to 4 marks) may be awarded to you depending on the quality of your answer, and if you are able to simplify the Boolean logic formulae describing the X and Y outputs. No need to hand in this page as I don't need to see your working.

# **Appendix A:** Pthread C Code Template

```
01 /* pthreads_template.cpp ----  A simple program using pthreads.
02  * Compiles under Linux / Cygwin using gcc   */
03 #include <iostream>
04 #include <stdlib.h>
05 #include <sys/types.h>
06 #include <pthread.h>
07 #include <string>
08 using namespace std;
09 #define MAX_THREADS 1000   // For safety, define max num threads allowed
10
11 int n; // global holding num threads specified (see main function)
12
13 typedef struct {  // Define struct for passing parameters to the threadfn thread
14   int id;
15   string text;
16 } Params_threadfn;
17
18 // Define a thread function:
19 void *threadfn (void *arg) {
20   Params_threadfn& p= *((Params_threadfn *)arg);
21   cout << "Node " << p.id << " activated and says " << p.text << endl;
22   return 0; // The thread stops at this point
23 }
24
25 // Entry point to the application:
26 int main(int argc, char* argv[])  {
27  int i; // a counter
28  pthread_t *threads; // array of threads
29  pthread_attr_t pthread_custom_attr; // same attribute used for all threads
30  Params_threadfn *p; // the parameters to pass to the threads
31  // Check that command line parameter was given
32  if (argc < 2) {
33    cout << "Command syntax: \n";
34      cout << " " << argv[0] << "    where n = number threads to create\n";
35    return -1;
36    }
37  // read the first command line param and convert into an integer
38  n=atoi(argv[1]);
39  if ((n < 1) || (n > MAX_THREADS)) {
40    cout << "Please specify num threads between 1 and " << MAX_THREADS << endl;
41    return -1;
42    }
43
44  // create the threads and parameters
45  threads=new pthread_t[n];
46  pthread_attr_init(&pthread_custom_attr);
47  p = new Params_threadfn[n];
48
49  //start the threads
50  for (i=0; i<n; i++) {
51      char s[16];
52      p[i].id=i;
53      sprintf(s,"hello%d",i);
54      p[i].text = (string)s;
55    pthread_create(&threads[i], &pthread_custom_attr, threadfn, (void *)&p[i]);
56    }
57  // Synchronize completion of all thread.
58  for (i=0; i<n; i++) pthread_join(threads[i],NULL);
59  // free the allocated memory
60  delete p;
61  delete threads;
62  return 0;  // exit successfully
63 }
```

# Appendix B:

## 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 | $\frac{d^q - u^q}{d-u}$ | $\sum_{k=1}^{q} U_{q-k-1} D_k$ <br> $U_j = \frac{\prod_{i=1}^{j} u_i}{u_j} \quad D_j = \frac{\prod_{i=1}^{j} d_i}{d_j}$ |

Table from pg 291 of Martinez, Bond and Vai 2008

# Appendix C:

### 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 |