Digital Systems
EEE4084F

FINAL EXAM

11 June 2015

3 hours

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 = 1 min 30 sec

Section 1	Section 2	Section 3	Appendices
Short Answers (4 questions) [46 marks]	Multiple Choice (10×3 mark questions + 4×1 mark true/false q's) [34 marks]	Long Answers (3 questions) [40 marks]	A: Formulae
pg. 2	pg 6	pg 9	pg 13

RULES

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

Question 1.1 [10 marks]

Regular PCs, such as would run an Intel SMP processor and a Windows operating system, has the advantage of a long history and well understood specifications, as well as a range of thoroughly tested programming tools. The Cell processor, while being a popular processor, is not as commonplace.

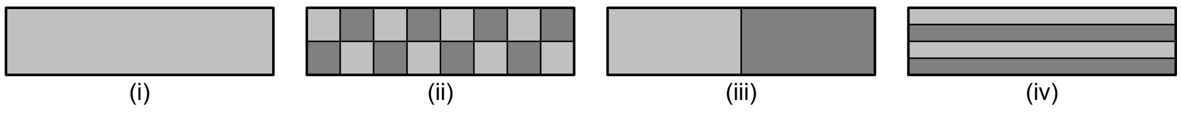
1.1 (a) Briefly explain fundamental architectural characteristics of the Cell processor, highlighting elements that are likely to differ to more standard processors, and what could make them more challenging to program. You can utilise a diagram to aid your answer. **[6]**

1.1 (b) An ABI is an important means to integrate executable code that can run together on one platform, for example allowing C programs compiled by different compilers to call each others' functions (if they are linked together in shared memory). But what is an ABI? **[2]**

1.1 (c) Why can you not say that an ABI and an API are one and the same thing? **[2]**

Question 1.2 [12 marks]

The figure below indicates various ways in which a two-dimensional array (or image) of `N` columns and `M` rows can be partitioned so that the data can be worked on by multiple processors.



1.2 (a) Give the official name of each partitioning method (i) to (iv) in the illustration above. **[4]**

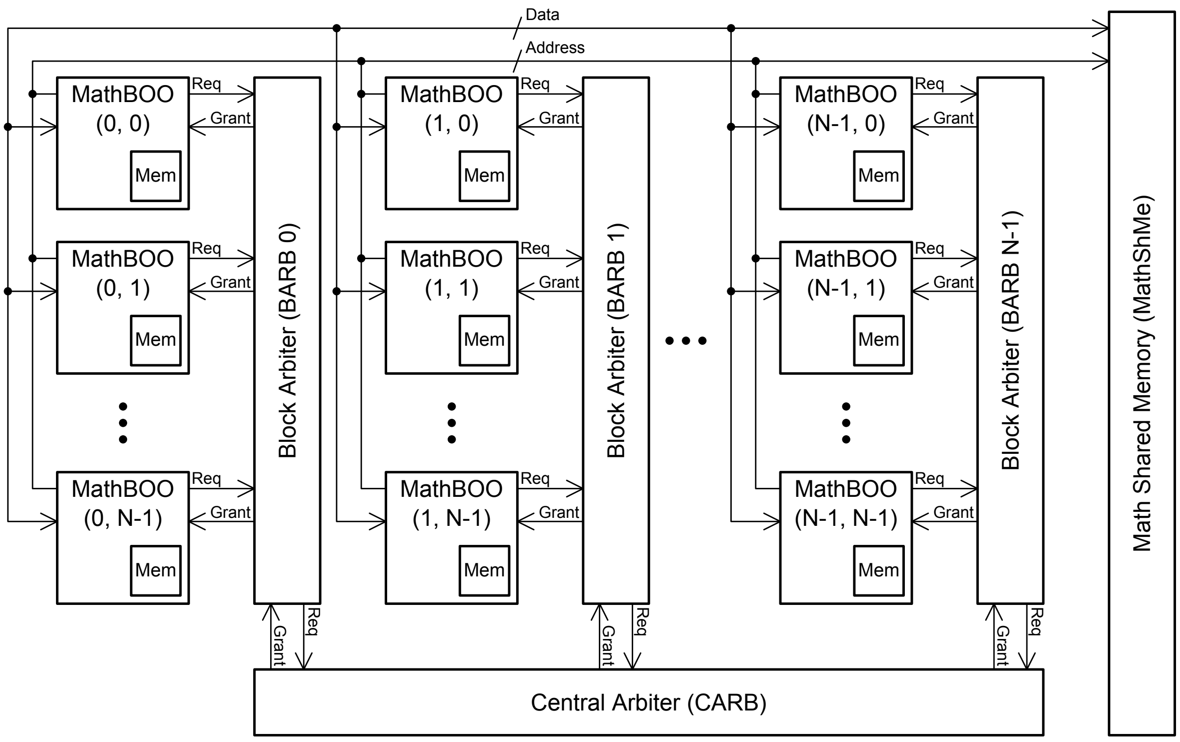
1.2 (b) Consider the case where the image is $N \times M$ pixels. You have a GPU that has $Bx \times By$ blocks, each with $Cx \times Cy$ cores (i.e. a total of $Bx \cdot By \cdot Cx \cdot Cy$ cores on the GPU).

Motivate which partitioning approach you would choose for a simple blur filter (i.e. `X[i,j] = average(X[i-1,j-1], X[i-1,j], X[i,j], X[i,j-1], ...)` and to what extent, if any, this may be influenced by the size of the image. The input and output memory are separate blocks. **[5]**

1.2 (c) Would you consider the blur mentioned in (b) above as more of a coarse-grained or find-grained operation? Briefly motivate your answer by emphasising the difference between these types of granularity. **[3]**

Question 1.3 [12 marks]

The (hypothetical) MultiMath processor architecture is designed to be a scalable array of processors. It allows the manufacturer, MultiMathMachines, to provide a variety of ASIC processor packages that offer a trade-off in processing power vs. cost. The architecture is illustrated below.



Processing cores, each called a "Math Block Ordered Operator" (or MathBOO), are arranged in an $N \times N$ matrix. Each MathBOO can do single-precision (32-bit) floating point arithmetic, including add, subtract, multiply and divide.

Each column of N cores connects to its Block Arbiter (BARB). The BARB in turn connects to the central arbiter (CARB). The BARB and CARB arbiters are used to grant mutually exclusive access to the Math Shared Memory (MathShMe). Both arbiters have much the same design.

Essentially, for a MathBOO to perform a memory read (or write), it has to ask its neighbouring BARB to grant access to the bus. The BARB in turn asks the CARB for access to the bus. Once access has been granted, the MathBOO connects to the bus and has exclusive access to the shared memory for as long as its "Request" line is held high. The BARB and CARB internal circuits make sure that access remains fair by means of a FIFO grant scheme.

Each MathBOO has its own generous amount (2 Mib (mebibit)) of private memory. When the MultiMath is in "hold" mode, all memory (including the shared memory and the private MathBOO blocks) maps to a single address space that can be accessed directly from the host CPU. The CPU can transact with this virtual block at about the same speed as it does to its own memory. You don't need to worry about how I/O, programming of the system, etc. is performed.

1.3 (a) The MathBOO is well-suited to calculate the dot-product (scalar-product) of two vectors. Discuss how scalar products could be implemented on this architecture and what performance bottlenecks exist in this case. Assume that many such scalar products are performed, so that programming time can be considered insignificant. **[5]**

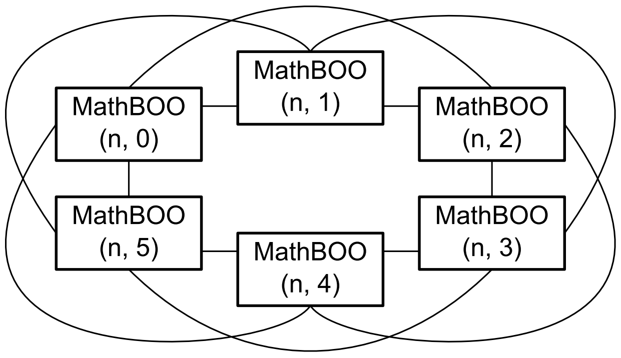
1.3 (b) If the MultiMath were to be used to implement matrix multiplication of two reasonably sized matrices, where the row and column dimensions are both smaller than `N`, discuss how this could be achieved. Again assume that programming time is insignificant due to the number of matrices that must be multiplied. **[5]**

1.3 (c) If the MathBOO cores support synchronous inter-process communications (IPC), whereby a service ID and word of data could be sent between nodes, could this provide a further boost to matrix multiplication performance? Give a short motivation for / against IPC in this case. **[2]**

Question 1.4 [12 marks]

Consider the case where the MultiMath architecture was changed such that any MathBOO core could transfer a 32-bit word of data to nearby cores. The links are half-duplex and only within the same Math Column. The connection schema is as follows:

MathBOO B_i connects (bidirectionally) to: B_{i-1}, B_{i+1}, B_{i-M} & B_{i+M}. All index additions and subtractions are performed modulo N. Below is a figure depicting the case where $N = 6$ and $M = 2$.



1.4 (a) What is be the bisection width, for each Math Column in a MultiMath processor, for the case where $M = 2$? **[1]**

1.4 (b) If the transfer of data sent from one MathBOO to another takes 2 ns, what would be the bisection bandwidth, for each Math Column in a MultiMath processor, for the general case where $M \in \left[2, \frac{N}{2}\right]$? Explain your answer clearly by using relevant illustrations and / or formulae where applicable. **[4]**

1.4 (c) If, instead, the Math Column was redesigned so that MathBOO cores were connected following a Binary Fat Tree schema, provide an illustration suggesting how this could be achieved. **[3]**

1.4 (d) What would be the bisection width for the Math Column in this case? **[1]**

1.4 (e) If a Math Column was storing a vector V of N elements, where each MathBOO Mem contained only one element of this vector (i.e. each MathBOO Mem = V_i), how much time would it take to flip the vector (i.e. $V_i \longleftrightarrow V_{N-i}$)? Assume that any operation, such as `SWAP(A, B)`, takes 2 ns (excluding the inter-core data transfer). Further assume that the Fat Tree design of (c) is used. **[3]**

Section 2: Multiple Choice [34 marks]

Choose one option for each of the questions 2.1 to 2.10 in this section.

2.1 Which one of the following is a new conventional wisdom, according to the paper: “The Landscape of Parallel Computing Research: A view from Berkeley” (presented in Seminar 1)? **[3]**

(a) The Power Boost – Power is free and but transistors are expensive.

(b) The Multiply Smackdown – Multiply is slow but access to memory is fast.

(c) The Instruction-Level Trampoline – Branch prediction, speculation, and very long wordy instructions result in superior performance.

(d) The Memory Wall – Load & store is slow but multiply is fast.

(e) The Bulking Monoliths – Monolithic uniprocessors in silicon are reliable internally and only likely to have errors at the pins.

2.2 In the textbook, the authors highlight that HPEC development teams often produce a significant amount of testing software and test vectors, which can complicate the project. Their advice is that: **[3]**

(a) The testing software should be considered throwaway code to keep things neat and manageable.

(b) There should be a strict classification maintained between testing resources of one version of the program and next.

(c) The test software and vectors should be reused to establish a “golden standard”, for measuring the correctness of system results, as development proceeds.

(d) Testing resources should all be kept to demonstrate the amount of progress achieved.

(e) The software should be thrown away, but the test vectors archived for possible use as evidence to show that the performance of the system is improving as the team works on it.

2.3 Which of the following is a characteristic of the classic Von Neumann architecture? **[3]**

(a) Seperate data and program memory interfaces

(b) CISC instruction set

(c) RISC instruction set

(d) Memory bottleneck

(e) Micro-operations

2.4 Which of the following is generally associated with lock-step execution? **[3]**

(a) SIMD architectures
(b) MISD architectures
(c) Parallel programs
(d) Gaming platforms
(e) Gravity calculation algorithms

2.5 Which of the following is not a computing device? **[3]**

(a) GPU
(b) DMA
(c) ASIC
(d) FPGA
(e) SPARC

2.6 What does HDL stand for? **[3]**

(a) High-level Digital Language
(b) Hardware Description Language
(c) High Density Language
(d) High Density Logic
(e) Hardware Design Language

2.7 With reference to HDLs, what does RTL stand for? **[3]**

(a) Real Time Logic
(b) Real Time Language
(c) Register-Transistor Logic
(d) Register Transfer Level
(e) Resistor Tunable Logic

2.8 If there are missing codes that occur in the operation of an ADC, for example the ADC tends to jump from 010 to 100 leaving out 011, this problem is called: **[3]**

(a) Gain error.
(b) Differential nonlinearity.
(c) Integration error.
(d) Dynamic range problem.
(e) Monotonicity.

2.9 Consider a crossbar tree network where each crossbar has 5 downlinks and 3 uplinks. There are 4 levels of switches. Which of the following is true? **[3]**

(a) This network supports 125 processing nodes.

(b) This network supports 625 processing nodes with a bisection width of 81 links.

(c) This network supports 625 processing nodes by means of 272 switches.

(d) This network supports 81 processing nodes.

(e) This network forms a binary fat-tree

2.10 Which of the following is not a commonly used design pattern? **[3]**

(a) Pipeline

(b) Divide-and-conquer

(c) Work queues

(d) Trial-and-error

(e) Producer / consumer flows

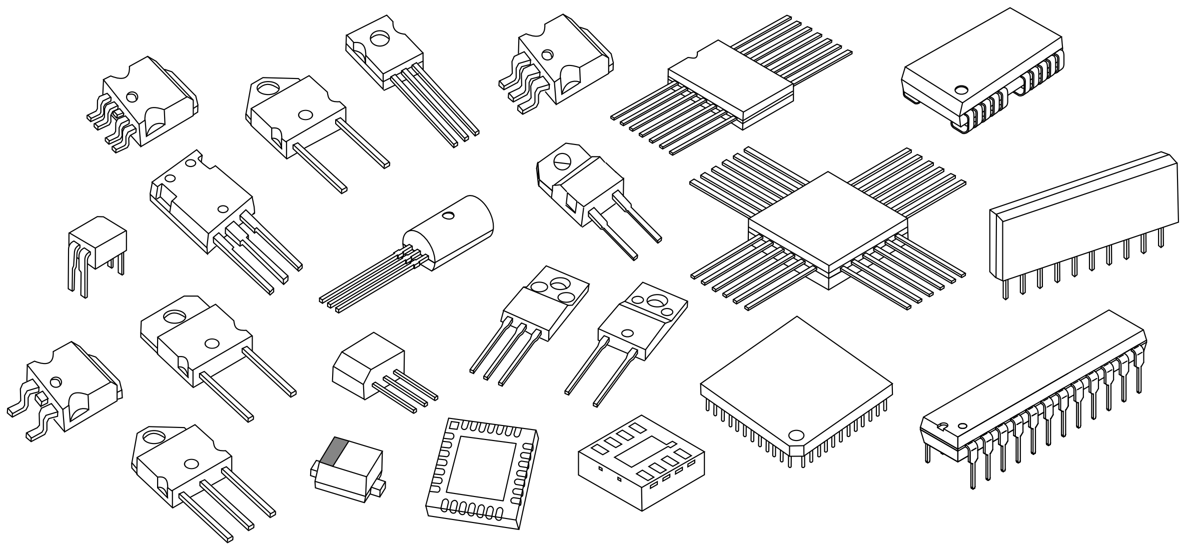
2.11 Answer “True” or “False” for each of the following statements: **[4]**

(a) Flash memory is usually programmed one byte at a time.

(b) Fly-by transfer is a DMA data transfer between different address busses.

(c) Shadow ROM is actually RAM.

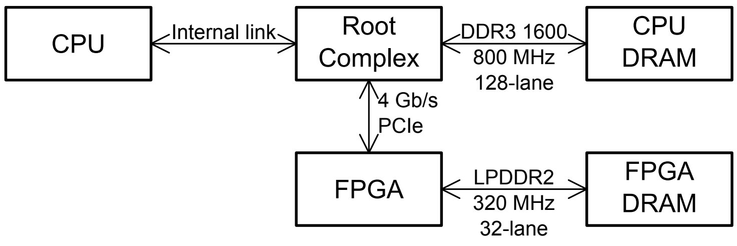
(d) The figure below includes a drawing of a BGA package.



Section 3: Long Answers [40 marks]

You are tasked to design a coprocessor to accelerate RSA encryption and decryption. It is a PCI-Express plug-in card that companies can install in their web servers in order to speed up secure connection services.

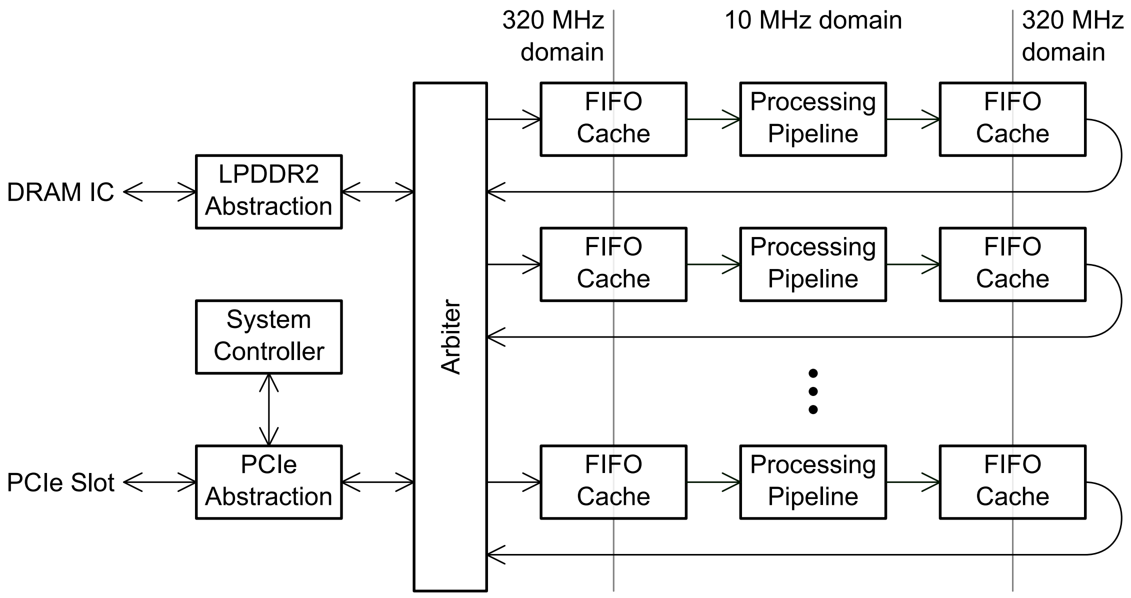
Consider the system block diagram below. The Root Complex is a component of modern computer processors that contain DMA channels between the CPU cores, CPU DRAM and other system peripherals. In this case, it provides DMA transfer between CPU DRAM and the FPGA, which is connected through a single-lane PCIe 1.0 port.



The system has the following characteristics:

- The CPU cache is large enough (and well enough implemented) that the CPU DRAM latency is negligible (for this application).
- The PCIe interface has 10% overhead (packet headers and such).
- DRAM spends 10% of the time in a “refresh” state.
- DRAM has a 10 clock cycle “setup” overhead for each burst read or write transaction.
- Both DRAM modules transmit (and receive) data at double the clock frequency (data is latched on both up and down edges of the clock).
- The CPU DRAM has a 128-bit wide data bus, whereas the FPGA DRAM has a 32-bit wide data bus.
- RSA encryption (or decryption) are performed on independent 4 096 bit blocks. In real RSA encryption the input and output block sizes differ, but for the purposes of these preliminary calculations you may assume that the input and output block sizes are both 4 096 bit.
- Data size is always a multiple of 4 096 bit.
- The CPU is a quad-core processor clocked at 3 GHz. It has a Dhrystone benchmark score of 25 instructions per clock per core.

Now consider the FPGA internal block diagram below. The LPDDR2 and PCIe abstractions run at the DRAM clock frequency of 320 MHz. Each processing pipeline runs at 10 MHz. There is one memory chip that is shared between all the processing pipelines. The FIFO cache modules make sure that each processing pipeline always have data to work on and a place to store the result. You may assume that the processing pipeline never needs to wait for the FIFO cache.



The PCIe abstraction determines whether the incoming PCIe packet is intended for the system control registers or the DRAM. The system controller controls the functions of the processing pipelines by means of control lines (not shown in the figure). The arbiter makes sure that only one module is connected to the DRAM at a time, by means of request and grant control lines.

The following steps are followed when the CPU wants data encrypted (or decrypted):

1. The CPU spends 10 ns to set up the DMA controller and other parameters.
2. The CPU hands over control to the DMA controller, which copies all the data to be encrypted from the CPU DRAM to the FPGA DRAM.
3. The DMA controller interrupts the CPU to signal that it is finished copying (assume 1 ns overhead).
4. The CPU resumes control of the PCIe bus and sends the key, as well as a control word, to the FPGA system controller. Assume that the total length of the data in this transaction is 8 192 bits.
5. The FPGA performs the encryption and signals (by means of an interrupt) that it is finished (assume 1 ns overhead for the interrupt).
6. The CPU spends 10 ns to set up the DMA controller and other parameters.
7. The CPU hands over control to the DMA controller, which copies the resulting data from the FPGA DRAM to the CPU DRAM.
8. The DMA controller interrupts the CPU to signal that it is finished copying (assume 1 ns overhead).

Question 3.1 [8 marks]

3.1 (a) When copying data from CPU DRAM to FPGA DRAM, what is the bottleneck in the transfer? Show calculations to support your answer. **[3]**

3.1 (b) Assume that the CPU has nothing else to do and can spend all its effort (multi-threaded) on encryption. The algorithm involves 128^3 iterations, of 5 instructions each, in order to encrypt one block. Use the Dhrystone benchmark to estimate the encryption rate (in blocks per second) that the CPU can achieve. Assume that the problem is computationally bound and therefore has no data overhead. **[5]**

Question 3.2 [10 marks]

3.2 (a) Assume that, while encrypting, the FPGA DRAM is read (or written) one 4 096 bit block at a time. How long does it take to read a block from one place in memory and immediately write it back somewhere else? Take the “setup” overhead into account, but ignore the “refresh” state. **[3]**

3.2 (b) Now take the DDR “refresh” state into account. What is the maximum long-term average rate at which blocks can be copied (in blocks per second), as described in the previous question? **[3]**

3.2 (c) Assume that the FPGA implementation of the algorithm is highly optimised so that each processing pipeline has a throughput of one block every 4 096 clock cycles (at 10 MHz). What is the maximum number of pipelines that can be used before the FPGA DRAM is not fast enough any more? **[4]**

Question 3.3 [22 marks]

Assume that the FPGA is large enough to accommodate only 100 processing pipelines. Each pipeline has a latency of 5 blocks (20 480 clock cycles).

3.3 (a) Calculate, for a data size of N blocks, the time it takes to complete steps 1, 2, 3, 4, 6, 7 and 8, as described in the question introduction. This is the setup and data transfer overhead. **[6]**

3.3 (b) Calculate, for a data size of N blocks, the time it takes to complete step 5, as described in the question introduction. Give two answers: one for $N \leq 100$ and another for $N \gg 100$. You may neglect any overhead that is less than 1% of the total time. **[8]**

3.3 (c) For what size of N does it become worth while to use the FPGA, rather than the CPU? You may neglect any overhead that is less than 1% of the total time. **[4]**

3.3 (d) What is the expected speedup factor for encrypting 100 MiB of data? **[4]**

END OF EXAMINATION

Appendix A: Formulae

Formulae 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	$\frac{d^q - u^q}{d-u}$	$\sum_{k=1}^{q} U_{q-k-1} D_k,$ $U_j = \frac{\prod_{i=1}^{j} u_i}{u_j},\quad D_j = \frac{\prod_{i=1}^{j} d_i}{u_j}$

Adapted from page 291 of Martinez, Bond and Vai 2008