Embedded Computer Architecture Laboratory: A Hands-on Experience Programming Embedded Systems with Resource and Energy Constraints

Andrew Nelson[1], Anca Molnos[1], Ashkan Beyranvand Nejad[1], Davit Mirzoyan[1], Sorin Cotofana[1], Kees Goossens[2]
[1]Delft University of Technology, The Netherlands
[2]Eindhoven University of Technology, The Netherlands
{A.T.Nelson, A.M.Molnos, A.BeyranvandNejad, D.Mirzoyan, S.D.Cotofana}@tudelft.nl, k.g.w.goossens@tue.nl

ABSTRACT

Embedded systems are complex, requiring multi-disciplinary skills for their design. Developing appropriate educational curricula is a non trivial problem. Embedded system design requires both theoretical and practical understanding. It is common in embedded system education to provide practical laboratory sessions to put into practice what is learnt from lectures and textbooks.

In this paper, we present our embedded systems laboratory that is given as part of the Embedded Computer Architecture (ECA) module at Delft University of Technology. Our laboratory provides practical, hands-on experience of programming a multiprocessor embedded system, that is prototyped on an FPGA. We provide details of the hardware platform and software APIs that are provided to the students, along with the laboratory assignment that was given to the ECA students in the 2011-2012 academic year. We present example results that were submitted by groups taking part in the laboratory, and describe the lessons we learned from our own practical experience of giving the laboratory.

Categories and Subject Descriptors: K.3.2 [Computers and Education]: Computer and Information Science Education

General Terms: Algorithms, Design, Experimentation

Keywords: Embedded System Design, Education, Multi-Processor System on Chip

1. INTRODUCTION

Modern embedded systems are widely used in various industries, e.g. automotive and consumer electronics, to execute complex applications. These systems are typically resource constrained, and thus it is desirable that applications share resources, such as processors, interconnect, and memory blocks, in order to reduce cost. To fully exploit parallel resources and improve performance, applications are required to be parallelised. Parallelisation of applications poses many challenges, not least of which increasing the complexity of understanding the available trade-offs, e.g. for performance and energy. Our laboratory assignment enables students to gain hands-on experience of these challenges, and allows them to experimentally explore the trade-off design space for a MultiProcessor System-on-Chip (MPSOC). The laboratory exercises guide the students through programming and optimising a fractal application for an MPSOC that is prototyped on an FPGA board (Xilinx ML605).

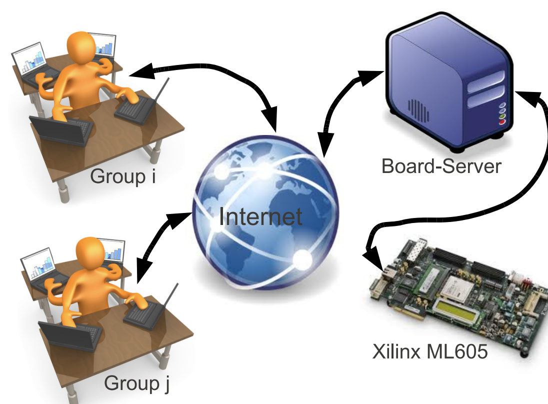


Figure 1: Remote access to FPGA prototype

Like other embedded MPSOCs, resources, such as memory, are limited and as a result deep software stacks that abstract from the details of the architecture are not available. Students have to get to grips with understanding the low-level workings of the MPSOC, giving them a chance to observe first hand all the hazards and pitfalls that come with programming an embedded multi-core system. Due to this, it is common to find laboratory work as part of embedded systems curricula, as is described in more detail in Section 2. After the students have finished our laboratory exercises, they will have experience in programming a resource constrained multi-core system, with low debugging visibility, while trying to maximise application performance through parallelism at the same time as trying to minimise energy consumption.

On top of the technical engineering aspects, we recognise that undertaking projects in international companies and academic institutes involves communicating with people from many different nationalities and backgrounds [8,9]. Not only should the students be able to overcome the technical aspect of the assignments, but they must also be able to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$10.00.

work effectively in a mixed nationality group with diverse technical backgrounds and experience. To achieve this the students are assigned to a group of four people, with the group mix selected to achieve as much intra group diversity as possible while trying to ensure an even inter group spread of experience. At the Delft University of Technology, our laboratory assignment is given to first year Master's students concurrently with theoretical lectures on Embedded Computer Architecture (ECA). From experience, first year Master's students have completed their Bachelor's studies in various similar topics and at different universities. Our laboratory assignments and group organisation encourages the exchange of this varied knowledge base, through group problem solving.

Our laboratory assignment consists of five exercises for the groups to complete. The exercises are structured to expose the students gradually to the complexity of programming an embedded multi-core platform. The CompSOC multi-core platform [1], that is developed as part of a combined effort between Delft and Eindhoven Universities of Technology, is used as the target platform for the assignment. The CompSOC platform is used as part of various embedded multi-core research objectives, including real-time and low-power research [13, 14], making it applicable to the subject matter of the assignments. This platform was also chosen due to its flexibility and automated design flow, enabling a custom platform to be created that is suitable for the educational needs of the laboratory assignment. The CompSOC platform is prototyped on an FPGA board that is accessible to the students through a server that arbitrates board usage, as shown in Figure 1. During the assignments the groups take a sequentially executing fractal program written in C and finish the assignments with a parallel executing fractal program that executes on the prototyped CompSOC platform.

Assessment of the assignment completion is carried out in multiple parts. Before students receive a final grade, for the laboratory assignment, they will have been assessed in mid-term group meetings, given a group presentation, submitted a written report and provided the C code that was used for the benchmarks of the report.

In the rest of this paper, we continue by presenting the educational context in which our laboratory is set, in Section 2. We follow that, in Section 3 with a detailed description of the ECA-CompSOC embedded multiprocessor platform, including its associated software API, that is used in the assignment. We present details of the assignment as given in the 2011-2012 academic year in Section 4. We finish by detailing what we learnt from our experience of giving the laboratory assignment in Section 5 and making concluding statements in Section 6.

2. ECA LABORATORY CONTEXT

The field of Embedded Systems is rapidly evolving, with Embedded Systems education evolving to match [2, 15, 19]. Practical experience, with relevance to industry, forms an important part of many Embedded Systems curricula [3, 4, 9, 17]. Our laboratory provides the students with an environment in which they can put theory into practice using the same tools and prototyping methods that can be found in industry [18].

Embedded systems are a combination of hardware and software, with many Embedded Systems courses teaching hardware and software co-design [11, 16]. This style is closer to what was referred to in [7] as the "something of everything" approach, due to the breadth of the topic covered. Our laboratory focusses on software development for multi-core embedded systems, which is closer to the "everything of something" approach, from [7]. We do not claim to have exhaustively covered "everything" on the topic of software design for embedded multi-core systems, but by narrowing the scope of our laboratory we enable students to explore the topic in depth. While there are merits to both approaches, we chose this approach as being more suited to first year Master's students, to whom the laboratory is given at Delft University of Technology.

Our laboratory assignment, as described in this paper, formed part of the Embedded Computer Architectures (ECA) module, given at Delft University of Technology, in the 2011-2012 academic year. The ECA module is compulsory for first year Master's students studying the Embedded Systems Master's programme, and may be followed as an optional module for students following related programmes. As such, it is important that the contents of the ECA module, including the laboratory assignment, provides a relevant and up-to-date educational experience. In this paper, we focus solely on the ECA laboratory assignment that gives students hands-on experience of the following educational goals of the ECA module as a whole:

1. Students can operate with concepts and notions related to:
   (a) Instruction sets: characteristics, functions, formats, addressing modes;
   (b) Processor structure, functions, and pipelining;
   (c) Distributed memory hierarchy;
   (d) Multiprocessors;
   (e) Interprocessor communication.
2. Students can optimise code for a particular processor using, e.g. code scheduling and loop unrolling.
3. Students can perform design space exploration and quantify design decisions in terms of performance, energy consumption, cost, flexibility, programmability, predictability for various processor and multiprocessors building blocks and architecture features, e.g. instruction set, message passing vs. shared memory, etc.

The ECA module is worth five ECTS credit points. The ECA laboratory assignment is worth 25% of the final grade for the module, meaning that students should expect to spend around 35 hours working on the assignment. In order to make sure that the students use this relatively short amount of time effectively, we decided that using a simple application, that was reasonably easy to parallelise was important. We also decided that an application with visual output would be more interesting to the students than one that produced text on a command line. As such, we chose a fractal generator application for the ECA laboratory in the 2011-2012 academic year, which is described in more detail in Section 4.1.

3. MULTIPROCESSOR PLATFORM

We continue by describing the multiprocessor platform that is used for the ECA laboratory assignment. The

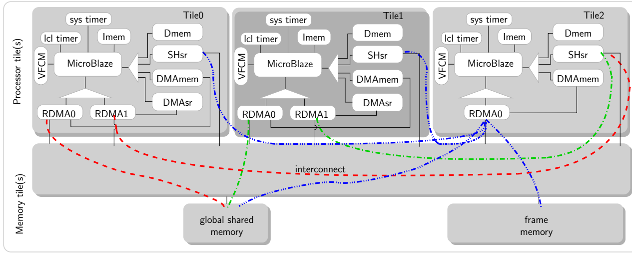


Figure 2: ECA-CompSOC hardware platform

CompSOC platform template is a tile-based, Network-on-Chip (NoC) centric, multiprocessor architecture with a distributed memory architecture. For the purpose of the laboratory assignment an instance of the CompSOC platform is used, which we refer to from this point as the ECA-CompSOC platform. The ECA-CompSOC platform instance consists of three processor tiles, a NoC interconnect, and two memory tiles, as presented in Figure 2. In what follows, we describe each, in turn. The theoretical motivation and principles behind the platform, its architecture, its software platform, and the research topics that it serves, is available in the literature [6, 12–14].

3.1 Processor tiles

The ECA-CompSOC platform consists of heterogeneous processing tiles, i.e. Tile0 and Tile1 are different from Tile2, as illustrated in Figure 2. While the tiles are architecturally different, each processor tile comprises a MicroBlaze processing core, a Voltage-Frequency Control Module (VFCM), a local instruction memory (Imem), a local data memory (Dmem), and a set of Remote Direct Memory Access (RDMA) modules with an associated set of local memory blocks for inter-tile communication.

The 32bit Xilinx MicroBlaze processor is a common core used in both industry and academic research. It is a soft core making it suitable for FPGA prototyping. A gnu compiler tool-chain for the Miroblaze is provided as part of the Xilinx FPGA prototyping tool set. In the processor tile, instructions and data for the processor are stored in the tile's local Imem and Dmem respectively. All of the tile's local memories are accessible via a single-cycle latency bus, as shown in Figure 2 with buses represented as triangles.

Each processor tile has an independent clock domain, allowing the voltage and frequency of each tile to be independently managed. The Voltage and Frequency Control Module (VFCM) has the primary purpose of providing the tile's clock signal. By default it provides a clock frequency of 100MHz, and it can be programmed to clock gate or to scale the voltage and frequency via a set of APIs described in Section 3.5. The VFCM module scales the frequency through clock division. The voltage is not actually scaled on the FPGA prototype, instead a power model, similar to an instance from the paramatrised model from [10], is used to calculate the power consumption at each frequency given a minimum voltage necessary to support the frequency. This calculation is carried out automatically based on the frequency scaling information. The VFCM also has an embedded timer that can be read by the application using the API defined in Section 3.5.3.

Inter-tile communication is carried out using RDMAs. The RDMA's allow parallelisation of computation and communication, and are used via an API. There are two RDMA API, defined in Section 3.5.1, commands that are used to send and receive data relative to the tile initiating the communication.

Each RDMA is connected to a local scratch pad memory, from which data is sent and received via the NoC. Tiles 0 and 1 have two RDMAs while Tile 2 has 1. On Tiles 0,1 and 2, RDMA0 is connected to a local memory called DMAmem. On Tiles 0 and 1, RDMA1 is connected to a local memory called DMAsr. Embedded systems are typically resource constrained, and the amount of memory available is not an exception. As such, the memory available on the ECA-CompSOC platform reflects this. The processor tile memory sizes are presented in Table 1.

Table 1: Sizes of processor tile local memories

Memory block	Size
Imem - all tiles	8 KBytes
Dmem - all tiles	8 KBytes
DMAmem - all tiles	4 KBytes
DMAsr - all tiles	32 Bytes
SHsr - all tiles	32 Bytes

With memory in such short supply the first thing the students need to do is modify their program code so that it can even fit into the tile's instruction and data memory.

3.2 Memory tiles

In addition to the local memories available on each processor tile, the ECA-CompSOC platform has two memory tiles that are accessible via the NoC, as illustrated in Figure 2. The global shared memory tile, is a relatively large memory, compared to the local memory available on the tiles. Due to its remote location from the tiles, writing and reading data, to and from this memory, is slow in comparison to the single-cycle needed to access a processor tile's local memories. The frame memory tile acts as a frame buffer, where visual data can be written for display. Unlike a frame buffer where the image would be displayed on a screen, an API call from the application signals that the frame is ready, initiating its retrieval to the student's computer. The sizes of both the global shared memory and the frame memory are presented in Table 2.

Table 2: Sizes of memory tiles

Memory tile	Size
Global shared memory	32 KBytes
Frame memory	256 KBytes

The ECA-CompSOC platform provides the students with a variety of memories of different capacities to chose from. This leaves the students with some interesting design choices as not all memories are accessible from every tile, due to point-to-point connection availability on the NoC, and not every memory can be accessed with the same bandwidth.

3.3 The NoC Interconnect

Once the students start working on a multi-core implementation of their programme code, an understanding of the platform's interconnect becomes important. The ECA-CompSOC platform uses the Æthereal NoC [6] intercon-

nect that provides point-to-point virtual channel connections, with per connection bandwidth guarantees. The NoC as configured for the ECA-CompSOC platform does not provide a fully connected interconnect, i.e. there are a limited number of point-to-point connections. The connections that are available do not have uniform bandwidth capabilities. Figure 2 presents the point-to-point connections that available are available across the NoC on the ECA-CompSOC platform. On Tiles 0 and 1, RDMA0 has a connection to the global memory tile. On Tile 0, RDMA1 has a connection with SHsr memories that are local to Tile 1 and 2. Similarly, on Tile 1, RDMA1 has a connection with SHsr memories that are local to Tile 0 and 2. On Tile 2, RDMA0 has a connection to the global memory tile, the frame memory, and the SHsr memories that are local to Tile 0 and 1. The qualitative bandwidth that is available on these connections is listed in Table 3.

Table 3: Qualitative connection bandwidths

Tile	Connection	Bandwidth
Tile0	RDMA0 → global shared mem.	slow
	RDMA1 → SHsr	medium
Tile1	RDMA0 → global shared mem.	slow
	RDMA1 → SHsr	medium
Tile2	RDMA0 → global shared mem.	fast
	RDMA0 → frame mem.	fast
	RDMA0 → SHsr	medium

In general, the larger the memory the lower the access bandwidth. This is in keeping with what is seen in other embedded systems, where larger memories are less likely to be in close proximity to the processors, or may even be located off-chip. Given the ECA-CompSOC platform's variation in memory sizes and access bandwidths, the students have plenty of scope for design space exploration. It also allows the students to run into common hazards, such as memory consistency problems, as the variation in access bandwidths helps to exacerbate memory transaction race conditions.

3.4 ECA-CompSOC memory map

All data communication on the ECA-CompSOC platform is carried out using Memory Mapped I/O (MMIO). From the processor tile's perspective, the ECA-CompSOC platform has two address spaces. Each processor tile has a local address space, that includes the addresses of the tile's local memories and registers for programming the RDMA modules, that are accessible via the tile's local buses. In the ECA-CompSOC platform, no memory is present in more than one local address space. The ECA-CompSOC platform's global address space is relative to the NoC, and includes addresses of memories that are connected to the NoC. The same memory may be present in both local and global address spaces, meaning that it is accessible locally via one of the processor tile's local buses and remotely via the NoC. As an example, the memory map of Tile 2 is presented in Table 4.

The local and global address maps of all of the ECA-CompSOC platform's memory locations is provided in a header file for the students to use in their programmes.

Table 4: Example of memory map adresses of Tile2

Memory mapped block	Local base address	Global base address
mb2_DMAmem	0x00005000	-
mb2_SHsr	0x00004000	0x03000000
mb2_rdma0	0x000F0000	-

3.5 Software API

The ECA-CompSOC platform is a complex multiprocessor embedded system. Programming the platform at a low abstraction level is great for students to gain practical knowledge about these sorts of system. While we do not want to hide this complexity from the students, there are some common tasks that risk students focussing too much on the details, and not seeing the wood for the trees. For this purpose we provide an API for common tasks, such as, remote memory access, measuring time, Dynamic Voltage and Frequency Scaling (DVFS), and debugging the applications. We proceed to explain each API command in more detail.

3.5.1 Remote Memory Access API

Remote memory access from processor tiles is carried out using RDMAs. In order to use an RDMA they must first be initialised. This is achieved using the following two API commands:

```
void hw_declare_dmas(int num_dmas);
DMA * hw_dma_add(int id, void * base_addr);
```

The first API is used to declare the number of RDMAs that are available during the execution of the application, i.e. two for Tile0 and Tile1, and one for Tile2. The second API is used to instantiate an RDMA and give it a unique `id`, with `0` < `id` < `num_dmas`. This function returns a pointer to the DMA object that can be used later in the program to transfer data using the APIs below. The RDMA base address, `base_addr` is specified in the global address map header file, along with the all the memory addresses in the ECA-CompSOC system.

In order for a processor tile to transfer data to and from remote memories, the following API commands are provided.

```
void hw_dma_receive(void * dst, void * src, int block_size,
DMA * dma);
void hw_dma_send(void * dst, void * src, int block_size,
DMA * dma);
int hw_dma_busy(DMA * dma);
```

The first API is used to program the RDMA, `dma`, to transfer `block_size` words of data from a remote address, `src`, (e.g. the global shared memory) to a local address, `dst` (e.g. DMAmem of a processor tile).

The second API is used to program the RDMA, `dma`, to transfer `block_size` words of data from a local address, `src` (e.g. DMAsr of a processor tile), to remote address `dst`, (e.g. SHsr of another processor tile). The MicroBlaze is a 32bit processor, hence each word equals 4 Bytes.

The third API tests the status of the given `dma`. It returns 1 when the dma is busy and returns 0 when the dma is ready for use.

3.5.2 Timer API

Each processor has access to a *system timer*, and a *local timer*. These timers can be read using the following API commands:

```
unsigned int get_system_time();
unsigned int get_local_time();
```

The system timer **always** runs at maximum frequency and it starts after the initial system's reset. The local timer runs on the tiles scaled clock frequency, e.g. after performing DVFS by calling the API commands described in Section 3.5.3, the local timer runs at the scaled frequency.

3.5.3 VFCM API

The following VFCM APIs are used to perform DVFS for power management purposes:

```
void hw_vfcm_clk_gate(unsigned int t);
void hw_vfcm_set_freq(unsigned int freq_level);
```

The first API command clock gates the processor, and hence halts the processor for a duration of `t` cycles in the system time domain. The second API command switches the processor frequency to the `freq_level` immediately, where `freq_level` is the frequency level in MHz. As explained in Section 3, the FPGA prototyped ECA-CompSOC platform doesn't actually perform voltage scaling, instead a power model is used, that assumes the minimum voltage level necessary to support the selected frequency, in order to calculate energy energy consumption at different frequency levels.

3.5.4 Debug API

Debugging applications running on an FPGA prototype is more complicated than on a desktop computer. The Xilinx tool suite contains debugging tools that provide some of the debugging functionality that programmers are used to. These debugging tools would be great if every group in the laboratory had their own FPGA board, but in practice this is not always going to be possible. For the laboratory given at the Delft University of Technology, all the groups share a single FPGA board. As such, it is not possible to allow groups to block the board for debugging purposes. In order to debug their platforms the students use the following API command:

```
void print_debug(int value);
```

This command prints an integer value (e.g. program line number) via the FPGA board's serial connection with the board server, which in turn relays the value to the student's computer. With this simple debug output, the students are able to monitor their application's progress and check data values. This gives the students experience in debugging platforms where it is not possible to use the well known graphical debugging tools.

3.5.5 A Frame Output API

The visual output of the application is written into the frame memory. Once the application has finished writing its visual output into the frame memory, the following API command can be called to return the image to the student's computer:

```
void print_framebuffer();
```

The image is retrieved in raw RGB format, before automatically being converted into the Bit Map Picture (BMP) format.

3.6 Platform Output

After each run on the FPGA board, the contents of the frame memory along with per-core debug and energy information is returned to the student's computer. The contents of the frame memory is formatted as a BMP file, as it is a common non-compressed graphical format. The system debug and energy information is output as a single text file per processor, an example of which looks as follows:

```
cycles|  energy nJ| information
------+-----------+-------------------------------
     0:          0 nJ: frequency -> 100 MHz
277629: 8606499 nJ: frequency ->   6 MHz
 DEBUG:            : hex(FFFFFFFF) int(-0000000001)
290450: 8683425 nJ: frequency ->  56 MHz
292094: 8711373 nJ: frequency -> 100 MHz
293092: 8742311 nJ: clock gated
294091: 8743310 nJ: clock ungated
294115: 8744054 nJ: execution finished

System energy consumed: 26490484 nJ
```

All power management changes are noted in the output file, along with the time, in cycles, when they were performed. From the example shown, it can be seen that the processor started with a default frequency of 100 MHz before changing frequency 3 times. First it changes to 6 MHz, then 56 MHz and finally 100 MHz where it remains for the rest of the run, except when the processor was clock gated. Similarly for clock gating, the time is noted in clock cycles whenever the processor is gated or ungated. The processor's energy consumption is calculated from the power management changes using a power model. All power management changes are accompanied by a running total of the processor's energy consumption for the current run. Once the execution has finished, the finishing time in cycles and the processors total energy consumption is noted. A final note in the file states the total system energy consumption for the run.

The debug API described in Section 3.5.4 provides the ability to return integer values from the FPGA board. The single value is presented as both a hexadecimal and signed integer value, for ease of use. The time at which the debug command was called is not noted by the API, with `DEBUG` printed in the cycle column instead of the time. While the time at which the debug occurred is not noted, events in the output file are noted in the order they occurred.

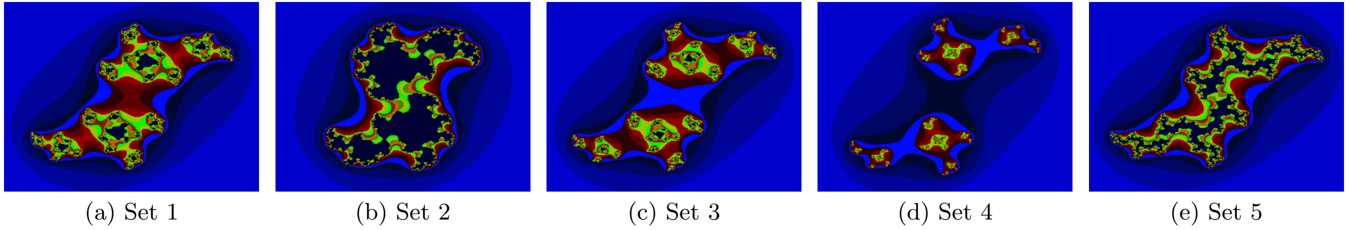
The output information described here is the only insight the students have into the functionality and performance of their solutions. We do not specify how the data returned should be visualised or interpreted, but we do state that we expect design choices to be motivated in the final report.

4. ASSIGNMENT OVERVIEW

In this section, we present and overview of the laboratory assignment. We describe the fractal application and the five exercises that the students complete in order to parallelise, map and optimise the application for the ECA-CompSOC platform. The students are expected to optimise the application for the commonly conflicting constraints of performance and energy. Following this, we propose our recommended approach for the students to start solving the exercises. Finally, we explain what deliverables we expect from the students, and how the assignment is evaluated.

4.1 Application

In the 2011-2012 academic year, the ECA students were given sequential C code for a fractal application. The fractal was chosen as the algorithm is quite compact, is quite easily parallelised, and provides a visual output, which is more interesting for students than reading text off of a command



(a) Set 1 (b) Set 2 (c) Set 3 (d) Set 4 (e) Set 5

Figure 3: Output references for the five different set of parameters of Fractal application.

line. For compactness, we show a grayscale version of the fractal algorithm below:

```
double x_min = -1.5;
double x_max = 1.5;
double x_step = (x_max-x_min)/x_size;
double y_min = -1.5;
double y_max = 1.5;
double y_step = (y_max-y_min)/y_size;
double x,y;
double new_x,new_y;
int m,n,num;
unsigned char R,G,B;
double real = 0.123;
double imaginary = 0.745;

for(n=0; n<y_size; n++){
  for(m=0; m<x_size; m++){
    x = x_min + x_step * m;
    y = y_min + y_step * n;
    for(num=0; ((pow(x,2) + pow(y,2)) <= 4)
        && (num < 0xFF); num++){
      new_x = pow(x,2) - pow(y,2) + real;
      new_y = 2*x*y + imaginary;
      x = new_x;
      y = new_y;
    }
    FrameMemory.R = num;
    FrameMemory.G = num;
    FrameMemory.B = num;
  }
}
```

Where the integers `x_size` and `y_size` are the pixel width and height of the fractal being produced. It is up to the students to analyse and understand the code provided. The `real` and `imaginary` variables control the shape of the fractal produced. We specify that all solutions to the exercises, created by the students, should remain functional for all possible `real` and `imaginary` inputs. So that the groups' solutions may be compared, we specify a set of inputs, presented in Table 5, for use with benchmarking their solutions.

Table 5: Input parameter sets

Input set	real	imaginary
1	0.123	0.745
2	0.4	0.3
3	0.15	0.8
4	0.4	0.8
5	0.01	0.8

4.2 Exercises

The laboratory assignment consists of five exercises, that when completed in order takes the groups from having a sequential C code representation of the fractal application to a parallelised and mapped version, for the ECA-CompSOC platform. In every exercise we clearly introduce the problem to be solved, and the concept that we expect the students will learn. To visualise the concepts, we present example results of actual group solutions in each exercise. The functional correctness of the student's solutions are checked, using a bit-wise comparison between the output file of their solution and the five output references provided in Figure 3. In the case where the output set does not match the reference set, the student's fractal application is deemed to be incorrect, and the solution is not acceptable. The set of fractal input parameter pairs that generate the references are presented in Table 5.

While the fractal application should remain working for all parameters, the students should use the five pairs when benchmarking their solutions.

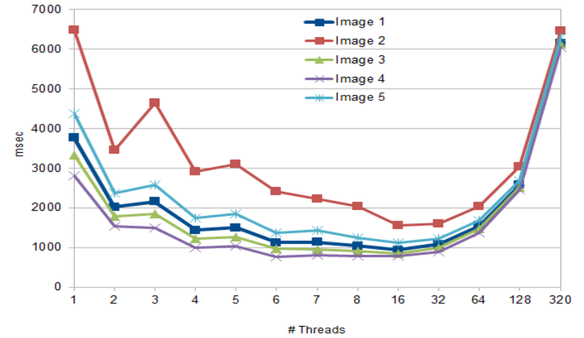


Figure 4: Actual results from a group for exercise 1

Exercise 1

In this exercise, each group has to parallelise the fractal application on a desktop computer utilizing the pthreads library. Furthermore, they should provide a `Makefile` [5] with a target "`run-fractal-pthreads`" and define a parameter X to specify the number of pthreads used. "`make run-fractal-pthreads THREAD=X`" should execute the fractal application with X threads on a PC.

This exercise is meant to get the students acquainted to the fractal application and to parallel programming on a desktop environment. An example of the graph presents the results of a solution for this exercise is presented in Figure 4. The graph shows the performance in milliseconds for the five fractal images, using different amounts of threads. The students are required to analyse and explain performance gain after parallelising the application.

Exercise 2

In this exercise, the groups have to map and execute the sequential fractal application on one core of the ECA-

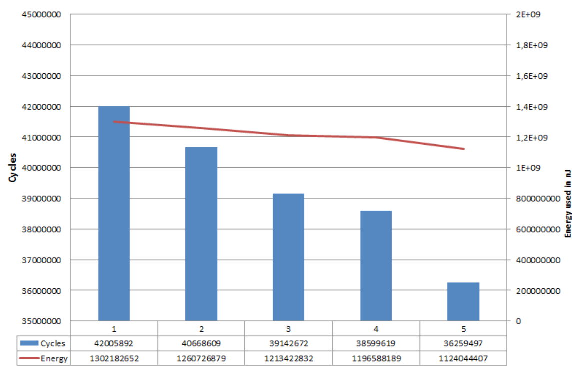


Figure 5: Actual results from a group for exercise 2

CompSOC platform. Their technical note should report the performance, i.e, execution time, and the energy consumed by the application.

This exercise is meant to familiarise the students to the ECA-CompSOC platform and to performance and energy consumption estimations. In Section 3 we explain how the performance and the energy consumption are estimated. The student should provide a graph like the example one presented in Figure 5 to compare the performance and energy consumption of application when executing with each set of parameters.

Exercise 3

In this exercise, the groups have to parallelise the application such that it runs on at least 2 of the ECA-CompSOC processor-tiles. The performance, i.e. execution time, and the energy consumed by their solution have to be evaluated and reported in the technical note. This exercise is meant to familiarise the students to parallelising an application on an embedded platform.

Exercise 4

In this exercise, every group has to optimise the performance of the parallel application executing on the ECA-CompSOC platform. For the optimisation, we recommend the students to focus on multi-core strategies, e.g. computation versus communication ratio. The quality of the solutions is graded according to the following list for the performance:

1. execution time > 35000000 cycles
2. execution time $\in [30000000\ 35000000]$ cycles
3. execution time < 30000000 cycles

This exercise aims at gaining experience with performance optimisation on an embedded platform and the trade-offs involved.

Exercise 5

In this exercise, the groups have to minimise the energy consumed by their application such that the execution finishes before a deadline, D (in cycles). Each of the groups receives a different value for the deadline D. Energy reduction is achieved by 1) clock gating a core for a period of time, and 2) scaling down the voltage and frequency of the core for a period of time. The available voltage-frequency management APIs are described in Section 3.5. The groups may chose to combine both of these techniques.

This exercise is meant to introduce the students to performance constrained energy optimisation on an embedded platform. Each group has to explain their general energy

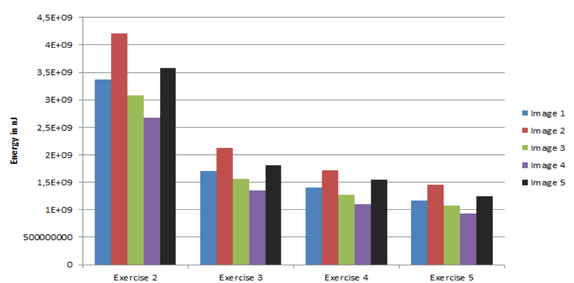


Figure 6: Actual results from a group for exercise 5

minimisation approach, and mention the exact points where they called voltage-frequency management APIs and motivate their choices.

Finally, the groups should provide an energy graph comparing their solutions for all exercise except the first one. An example of such a graph is presented in Figure 6.

4.3 Recommended approach

Before the students start solving the exercises of the assignment, the following steps are recommended:

- Take the time to become familiar with the fractal application, the tool-set, and the remote FPGA board environment. Read and understand the provided documents.
- Read description of the ECA-CompSOC, and consult the table of contents of the provided MicroBlaze processor reference guide.
- Execute the provided example on the FPGA prototype, and inspect the source code. Take the time to understand the ECA-CompSOC distributed memory map, and how to access memory blocks remotely.

Only when the students have completed and are comfortable with these steps, should they start to solve the exercises.

4.4 Assignment Deliverable

Each group submits the C code of their solution for the five assignment exercises and a final report. The final report should not exceed 4 pages as there is always a short page limit for scientific publications in conferences. The final report should include the following:

- Descriptions of their solution for each of the 5 exercises, giving the general idea behind their implementation/parallelisation and application mapping. The report should discuss explicitly the problems they encountered when parallelising, executing, and optimizing the application on the ECA-CompSOC platform, and their solutions.
- When applicable, details of all the applied performance optimisation strategies, and explanations about why those optimisations deliver an increase in performance.
- For exercise 4, the performance results (in execution time) and a comparison with the performance of the solutions of the sequential application baseline in exercise 2, as well as the first parallel implementation in exercise 3 and 5.
- For exercise 5, the energy consumption results and a comparison with the energy consumption of the exercises 2 to 4, without clock gating.

4.5 Evaluation Criteria

The ECA laboratory assignment is graded with 10 points,

accounting for 25% of the final grade for the ECA module. The 5 exercises account for a maximum of 6 points, the final report and the presentation account for a maximum of 2 points each. At the initiative of the instructors, bonus points are granted for original, exceptionally good solutions or observations. To get a final grade of 6, each group needs to solve exercise 1 to 3, to provide the report, and to give the final presentation. The final score for the project is determined based on the following criteria:

- Functional correctness. If, for an exercise, the output picture does not bit-wise match the reference picture provided, the group will not get any points for that exercise.
- The technical merit of the approach. Aspects as innovation level and implementation quality are considered.
- Code readability. The maximum number of points can be obtained only if the code is clear and commented. Unreadable, un-understandable code will be penalised with up to half of the maximum points awarded for that exercise.
- Clarity and conciseness of the final report. Report organisation, content, and language are important aspects at this point.
- The presentation. Here we look at the clarity, conciseness of the slides and talk, and at the students' capability to ask questions and to answer questions from the auditorium.

5. LESSONS LEARNT

The laboratory as described here was successfully given in the 2011-2012 academic year at Delft University of Technology. From conversing with the students, their feedback was in general positive. The students enjoyed the laboratory and found the level of difficulty to be appropriate. We found that in practice the students were spending more time than scheduled working on the laboratory project. This seems to indicate that even though the level of difficulty of the laboratory assignment is correct, the quantity of work was possibly a bit much.

Parts of our laboratory, were reappropriated for use with the Embedded Systems Laboratory (an evolution of the laboratory from [9]) given at Eindhoven University of Technology. Their laboratory course is a full module, which coulde be a better format for our laboratory course given at Delft University of Technology. While our laboratory remains part of the Embedded Computer Architecture (ECA) module, we will need to reconsider the workload to ensure a balance with the rest of the ECA module.

6. CONCLUSION

Students that follow our laboratory will have gained hands-on experience programming a multi-core embedded system. They will have overcome the difficulties of programming for a resource constrained platform with limited debug visibility. They will have investigated their solution's design space for both performance and energy consumption, learning the trade-offs that exist on such a platform. Aside from the technical achievements, the students will have done this while working in multi-cultural groups, with diverse backgrounds and experience, similar to what is found in international companies and academia.

7. REFERENCES

[1] B. Akesson, A. Molnos, A. Hansson, A. Ambrose, and K. Goossens. Composability and predictability for independent application development, verification, and execution. In *MPSoC: Hardware Design and Tool Integration*. 2010.

[2] P. Bertels, M. D'Haene, T. Degryse, and D. Stroobandt. Gathering skills for embedded systems design. In *WESE*, 2007.

[3] H. Dai, Z. Jia, X. Li, and Y. Guo. Practical training in the embedded system education: A new way to narrow the gap with industry. In *ICYCS*, 2008.

[4] A. G. Dean. Teaching optimization of time and energy in embedded systems. In *WESE*, 2010.

[5] GNU Org. Make, 2012. `http://www.gnu.org/software/make/`.

[6] K. Goossens and A. Hansson. The Æthereal network on chip after ten years: Goals, evolution, lessons, and future. In *DAC*, 2010.

[7] M. Grimheden and M. Törngren. How should embedded systems be taught?: experiences and snapshots from swedish higher engineering education. *SIGBED Rev.*, 2(4), 2005.

[8] B. Haetzer, G. Schley, R. S. Khaligh, and M. Radetzki. Practical embedded systems engineering syllabus for graduate students with multidisciplinary backgrounds. In *WESE*, 2011.

[9] A. Hansson, B. Akesson, and J. van Meerbergen. Multi-processor programming in the embedded system curriculum. *SIGBED Rev.*, 6(1), 2009.

[10] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *DAC*, 2004.

[11] H. Mitsui, H. Kambe, and H. Koizumi. Use of student experiments for teaching embedded software development including HW/SW co-design. *Education, IEEE Trans.*, 52(3), 2009.

[12] A. Molnos, A. Ambrose, A. Nelson, R. Stefan, K. Goossens, and S. D. Cotofana. A composable, energy-managed, real-time MPSOC platform. In *OPTIM*, 2010.

[13] A. B. Nejad, A. Molnos, and K. Goossens. A unified execution model for data-driven applications on a composable MPSoC. In *DSD*, 2011.

[14] A. Nelson, A. Molnos, and K. Goossens. Composable power management with energy and power budgets per application. In *SAMOS*, 2011.

[15] A. L. Sangiovanni-Vincentelli and A. Pinto. Embedded system education: a new paradigm for engineering schools? *SIGBED Rev.*, 2(4), 2005.

[16] P. Schaumant. Hardware/software co-design is a starting point in embedded systems architecture education. In *WESE*, 2008.

[17] Q. Shi, L. Xiang, T. Chen, and W. Hu. FPGA-Based embedded system education. In *ETCS*, 2009.

[18] Xilinx, 2012. `http://www.xilinx.com`.

[19] Y. Zhang, Z. Wang, and L. Xu. A global curriculum design framework for embedded system education. In *MESA*, 2010.