IL2212 EMBEDDED SOFTWARE

Technical Report
Multicore Digital Signal Processing Application

JUN ZHANG JAAKKO LAIHO
junzha | jaakkol @kth.se

February 26, 2019

1 Introduction

1.1 Scope

The project is the laboratory assignment of the course IL2212 Embedded Software[1] at KTH, Sweden in January-February of 2019. The workload of the laboratory part of the course is reported to be 3 ECTS, which equates to 80 hours of work.[2]

1.2 Purpose

The purpose of this project is to enable our learning in terms of the established learning goals of the course. That goal is expressed followingly: "Construction flow for software for embedded systems: System modeling, systems analysis and system synthesis."[1]

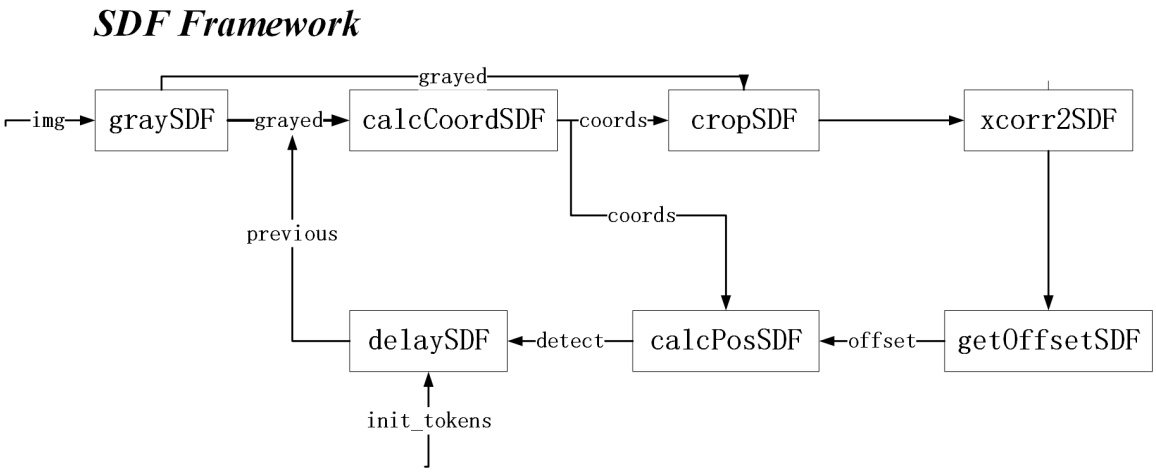
1.3 Application

This project is a practice in implementing a digital signal processing application on a multi-core hardware platform. More specifically, the task of the project is "to implement an image tracking algorithm which tracks a (given) moving 'circle' pattern in a series of image frames for the purpose of further processing." [3] The application is implemented in three ways on the custom multi-processor hardware platform hosted on a DE2 FPGA: On a single core using the MicroC/OS-II, on a single core without OS, and on multiple cores without OS. The following assumptions are considered valid: [3]

- In any given input frame there can be at most one 'circle' object.
- Between any two consecutive frames the 'circle' object can move at most 15 pixels in any direction.
- There is an acceptable detection error of +/- 2 pixels.
- The first frame in a set of images is known to contain the object in the vicinity of a fixed position.

2 Application Model

Figure 1: SDF Framework.



2.1 Algorithm Descriptions

Grayscaling transforms the input RGB matrix to a grayscale matrix by performs so-called convolution, where "the input samples are combined using a weighted sum with specific fixed weights associated with each offset input." [4] It uses a Map[1]pattern to compute the output, where for each input sample and the next 2 input samples the output sample calculated as a weighted average. For a matrix input the processing time is $O(n^2)$, where n is the dimension of the image. *Cropping* cuts the image to a smaller image of the given size. Its processing time is also $O(n^2)$, however in this case n denotes the dimension of the cropped image. The function *xcorr2* reduces[2]the cropped image by adding together the greyscale values of a series of stencils[3], where the stencils are "sub-images" based on the size of the pattern being searched. The summary value is then paired with the position of that value. The execution time depends on the dimensions of the cropped image and the size of the pattern to be detected, and its processing time is approximately $O(n^2)$. The actor *getOffsetSDF* gets the position of the image with the greatest grayscale sum by reducing the paired datapoints with a comparer function, which returns the pair with the greater grayscale value. It has an execution time of $O(n)$. *CalcCoord* calculates the coordinates of the next cropping point according to the previous known object position, unless it is the first time the function is being called, in which case it returns a default value. Its execution time is $O(1)$. *CalcPos* calculates the position of the detected image by adding together the cropping point as well as the found offset and executes in $O(1)$.

2.2 Speculation

Mapping and reducing apply a function to all elements in a collection, where the ordering of the execution of the functions do not need to necessarily be sequential. In other words, there is reason to believe that significant performance increase can be achieved by parallelising the execution of grayscaling and xcorr2 onto multiple processors.

[1]**Map**: "In the map pattern, an elemental function is applied to all elements of a collection, producing a new collection with the same shape as the input. In serial execution, a map is often implemented as a loop (with each instance of the loop body consisting of one invocation of the elemental function), but because there are no dependencies between loop iterations all instances can execute at the same time, given enough processors"[4]

[2]**Reduce**: "In the reduce pattern, a combiner function is used to combine all the elements of a collection pairwise and create a summary value. It is assumed that pairs of elements can be combined in different orders, so multiple implementations are possible."[4]

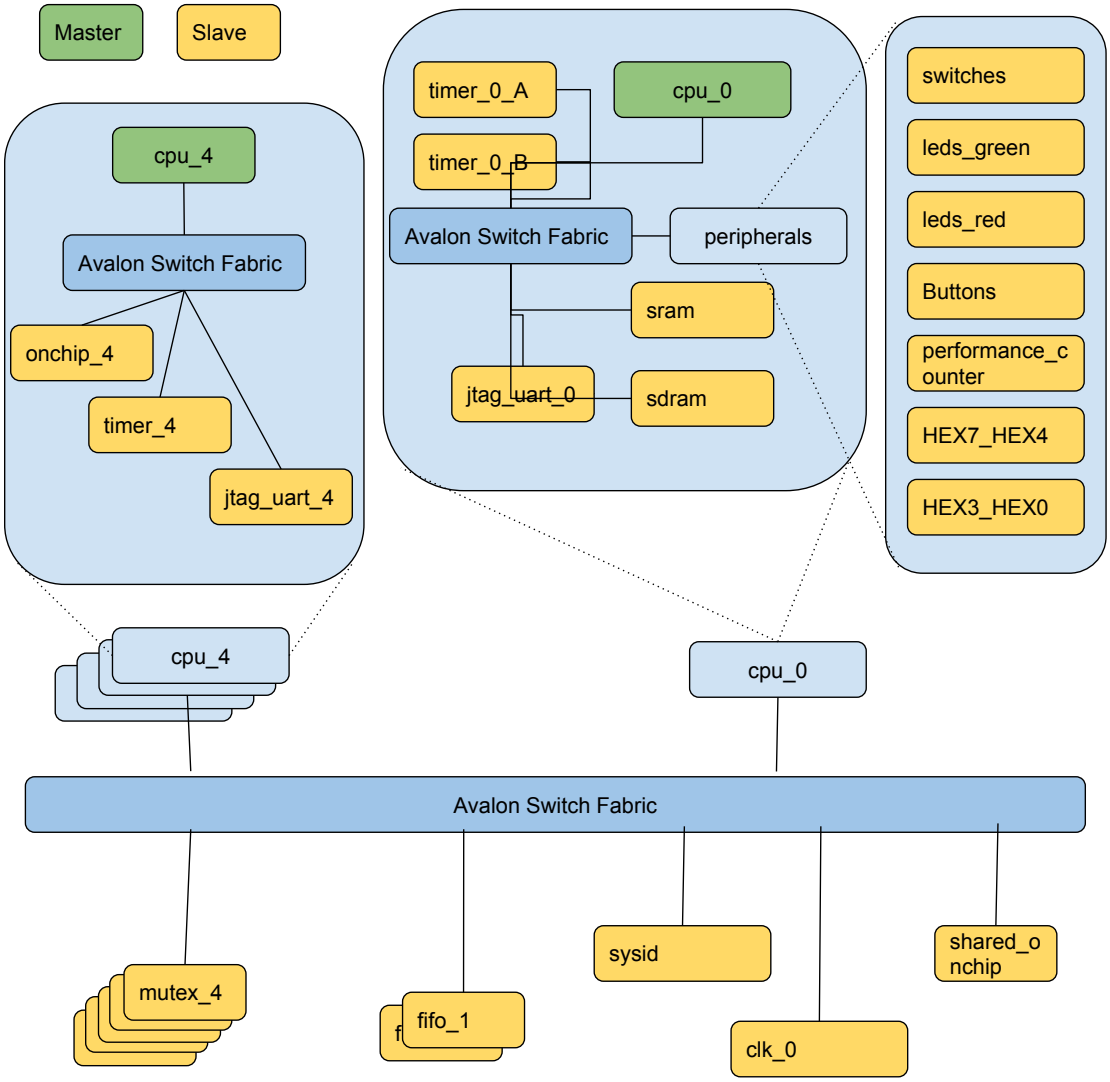
[3]**Stencil**: "A stencil is a map in which each output depends on a 'neighborhood' of inputs specified using a set of fixed offsets relative to the output position."[4]

3 Hardware Multi-processor Platform Specification

3.1 Platform Architecture graph

The hardware platform features multiple cores and peripherals a network-on-chip architecture that communicate via the Avalon Switch Fabric. There are 5 Nios II Processor cores in the system, and a wide selection of peripherals. Most peripherals are connected to the data master bus of processor core CPU-0. These peripherals include the following: switches, LEDs (both green and red), buttons, a hex display, and notably a performance counter. CPU-0 also has 2 timers, an sram and 8MB sdram and jtag uart component connected to its data bus, while the other 4 CPUs (cpu-1 to cpu-4) each have only an 8kB onchip memory, one timer, and a jtag uart component connected to their data master. Furthermore, the system also features 5 hardware mutexes, 2 on-chip fifo memories, 1 shared 8kB on-chip RAM memory, all of which connect to the data master bus of each CPU.

Figure 2: Network-on-Chip Architecture.[5]



3.2 Interconnection Network

Each component in the system communicates via encapsulated packet transactions, where the network-on-chip (NoC) interconnect transports the transactions between nodes in the system. The NoC treats the interconnect as a protocol stack where each layer implements the different functions of the interconnect. Each master or slave node is connected to the network through a network interface component. This interface receives the transaction or response and sends it to the network as a properly formatted packet. The packet is then delivered to the appropriate endpoint by the packet network for that packet, where it is passed to other network interfaces. The packet is then terminated by the network interfaces which also sends the command or response using the transaction layer protocol to the recipient.[6]

In other words, communication between the components happens through using the transaction interface of the Avalon Memory-Mapped interface in this case. Simply put, the network interfaces are the ones that communicate with each other, which in turn rely on the command and response networks provided by the network. This in turn shapes the question the software developer must answer into the following: ”How do I best map transactions to packets?” With the transaction layers being separated, the developer may optimize each layer independently without needing to consider the other layers.

4 Single Core+RTOS, Single Core+Bare-metal

4.1 Description

The application modeled as an SDF graph has a good mapping to the target architecture. this is because an actor in an SDF graph can be translated into a task (in case of a RTOS) or a function (in case of no RTOS). The signals in the SDF graph describe data dependencies of the actors - in an RTOS implementation using OS kernel objects such as message queues (mailboxes / semaphores in case each signal only has at maximum one token) will enable synchronisation of the tasks. Alternatively, in the bare metal implementation the beauty of a SDF graph shines through - a static schedule can be computed, in which case the main loop of the program will simply call the actors mapped as functions according to that schedule.

Table 1: Single Core Implementations.

	Single Core (RTOS)	Single Core (Bare Metal)
Throughput [s-1]	2.41	1.26
SRAM [bytes]	220000	145000

4.2 Schedule Graphs

Figure 3: Single-core RTOS schedule

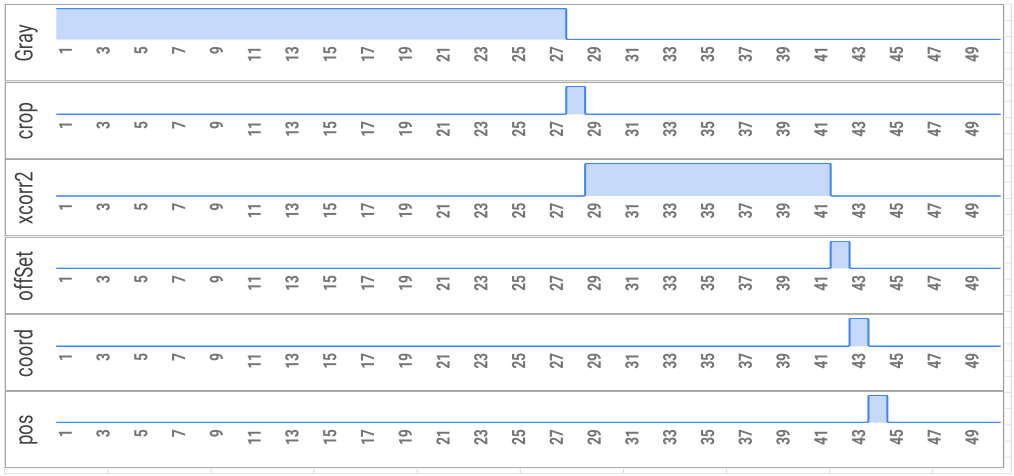
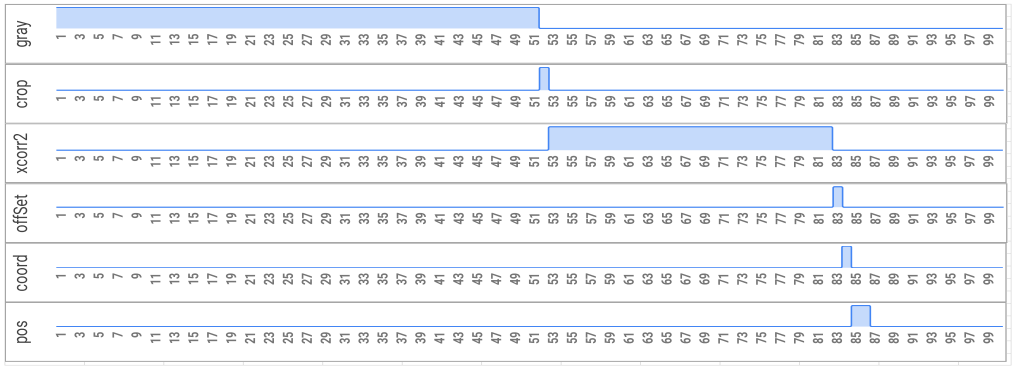


Figure 4: Single-core bare schedule



5 Initial Multi-core Implementation

Table 2: First Multi-Core Implementation.

Throughput [s-1]	SRAM [bytes]	OnChip CPU 1 [bytes]	OnChip CPU 2 [bytes]	OnChip CPU 3 [bytes]	OnChip CPU 4 [bytes]	OnChip Shared [bytes]	Total Memory [bytes]

5.1 rough time schedule

5.2 Mapping Strategies

starts presenting strategies for optimized mapping in order to fulfill the design constraints. Support your proposals with graphs, measurements, and analyses on the correctness of the transformations (i.e. does the application still fulfills the specifications). These strategies need to be discussed with the TAs during the lab session.

6 Optimized Multi-core Implementation

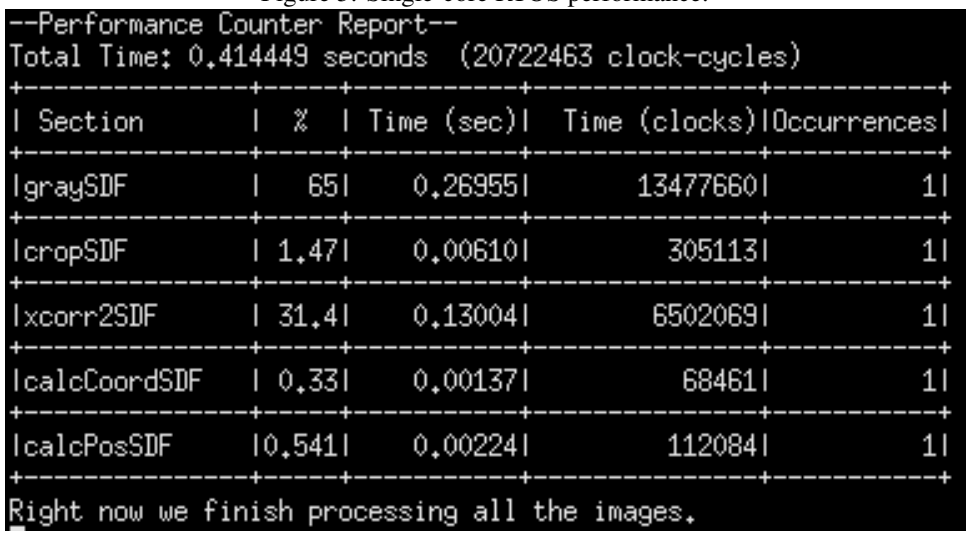
presents the which satisfies the design constraints. Include measurements, schedule, and a mapping (actor-to-processor) graph. Support your implementation with a discussion on the optimization methods employed, and the transformations which the original model had to go through. Fill in an updated version of the table below.

Table 3: Final Multi-Core Implementation.

Throughput [s-1]	SRAM [bytes]	OnChip CPU 1 [bytes]	OnChip CPU 2 [bytes]	OnChip CPU 3 [bytes]	OnChip CPU 4 [bytes]	OnChip Shared [bytes]	Total Memory [bytes]

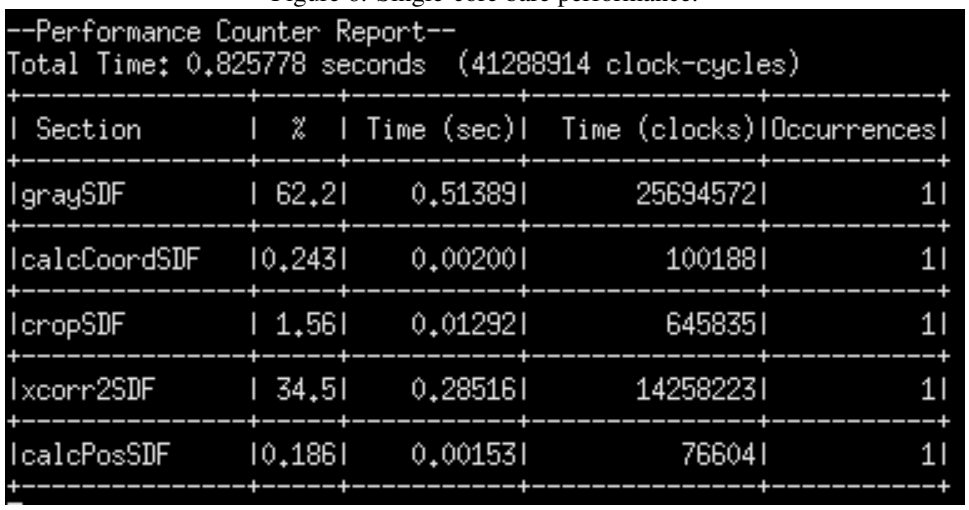
7 Appendix

Figure 5: Single-core RTOS performance.



```
--Performance Counter Report--
Total Time: 0.414449 seconds  (20722463 clock-cycles)
+---------------+-----+----------+--------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+----------+--------------+-----------+
|graySDF        |   65|    0.26955|       13477660|          1|
+---------------+-----+----------+--------------+-----------+
|cropSDF        |  1.47|    0.00610|         305113|          1|
+---------------+-----+----------+--------------+-----------+
|xcorr2SDF      |  31.4|    0.13004|        6502069|          1|
+---------------+-----+----------+--------------+-----------+
|calcCoordSDF   |  0.33|    0.00137|          68461|          1|
+---------------+-----+----------+--------------+-----------+
|calcPosSDF     | 0.541|    0.00224|         112084|          1|
+---------------+-----+----------+--------------+-----------+
Right now we finish processing all the images.
```

Figure 6: Single-core bare performance.



```
--Performance Counter Report--
Total Time: 0.825778 seconds  (41288914 clock-cycles)
+---------------+-----+----------+--------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+----------+--------------+-----------+
|graySDF        |  62.2|    0.51389|       25694572|          1|
+---------------+-----+----------+--------------+-----------+
|calcCoordSDF   | 0.243|    0.00200|         100188|          1|
+---------------+-----+----------+--------------+-----------+
|cropSDF        |  1.56|    0.01292|         645835|          1|
+---------------+-----+----------+--------------+-----------+
|xcorr2SDF      |  34.5|    0.28516|       14258223|          1|
+---------------+-----+----------+--------------+-----------+
|calcPosSDF     | 0.186|    0.00153|          76604|          1|
+---------------+-----+----------+--------------+-----------+
```

References

[1] I. Sander, "Embedded software." https://www.kth.se/student/kurser/kurs/IL2212?l=en. Visited on 02/22/2019.

[2] studentdata@kth.se, "Student at kth." https://www.kth.se/en/student/kurs/studieresultat-1.373671. Last changed: Jun 21, 2018.

[3] G. Ungureanu, "Laboratory." https://kth.instructure.com/courses/7693/pages/laboratory-information-about-the-project. Visited on 02/22/2019.

[4] A. D. R. Michael McCool and J. Reinders., *Structured parallel programming patterns for efficient computation*. Amsterdam ; Boston, Mass.: Morgan Kaufmann, 2012.

[5] J. Laiho, "Network-on-chip architecture graph." https://docs.google.com/drawings/d/1F3l7kJZKiIVBYh8ljDyU77SW0-kB9Sm1T0EeRqUxI50/edit?usp=sharing. Generated on 02/25/2019.

[6] K. Orthner, "Applying the benefits of network on a chip architecture to fpga system design." https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01149-noc-qsys.pdf. Visited on 02/22/2019.