02131 Assignment 3: Implementation of an ECG co-processor

Michael Reibel Boesen, Jan Madsen, Linas Kaminskas,
Karsten Juul Frederiksen and Thomas K. Malowanczyk

November 10, 2015

1 Introduction

At Medembed the product management of the wearable ECG scanner wants to investigate the benefits of implementing parts of the ECG scanner as a dedicated hardware implementation as a so called co-processor. Theoretically, this should give the device better performance. In this project you will implement a dedicated version of the Moving Window Integration filter to test this hypothesis.

We will divide the project into the following subtasks

- **A3.1:** Identify suitable algorithm to be implemented in hardware
- **A3.2:** Execute implementation plan
- **A3.3:** System integration
- **A3.4:** Performance analysis

1.1 Deadline

By **December 8, 23:59** you have to deliver an electronic version of the report and source files as an archive uploaded on Campusnet at the location:

`Campusnet/Assignments/Assignment 3`.

1.2 Expectations

The recent years we have experienced that the students have had a hard time of determining whether they are behind or on track, time-wise. Therefore this plan should guide you into checking that you're on track. You will have 4 TA supervised labs to complete this exercise. Please note that we **also** expect that you work on this as part of your preparation for the course! You will probably not be able to finish the assignment if you only use labs to work on it.

Date	Complete
12/11	A3.0, A3.1
19/11	A3.2
26/11	A3.2, A3.3
3/12	A3.3, A3.4

Table 1: Table of expectations

1.3 Notes & Hints

- To help you get an idea about how good your following the schedule we have added an "expected schedule" metric on the individual sub-tasks. This metric tells you in the terms of "labs" how much time you should expect to use on the individual tasks. An example for how to use it: Let's say it is now Nov 28 17:30, i.e. lab nr. 3 just finished and you're still in task A3.2. To keep on schedule you should start on A3.3 next time. So it might be a good idea to spend some time on A3.2 until the next lab. We hope this might guide you in understanding where we expect you to use the most time. If you get stuck somewhere and see that you are falling behind, don't hesitate to contact us.

2 Project overview

A co-processor is a hardware implementation, which is dedicated to a specific purpose. It's "co-" because it usually performs only a part of an application for which for instance a general purpose processor performs the rest. In your case, the processor you built in A2 will control the co-processor - the co-processors job is to do a certain task as fast (or as low power or with the smallest amount of area) as possible.

In order to implement something as dedicated hardware it is important that you understand what it is that you will be implementing. Therefore, it is necessary to understand the mathematical operation required to implement the Moving Window Integration (MWI) filter. The mathematical representation of the MWI filter is seen in equation 1.

$$y_n = \frac{1}{N}(x_{n-(N-1)} + x_{n-(N-2)} + \cdots + x_n) \tag{1}$$

As seen the operations involved in this filter is a division and a summation. You will need to decide on how to implement each of these operations.

Once you have decided what you want to implement you should plan what to implement by drawing a block diagram of your solution. This will help you get an overview of what should be implemented.

When you have the co-processor working as a separate entity you should connect it to the rest of the platform, i.e. the same platform as you connected you processor to. This means that you should write a program for your processor which controls the co-processor as well as connect your co-processor to a slave interface on the bus.

Finally, you should perform a performance analysis very similar as the one you performed in A2. The next sections will go into detail about each of the parts of the assignment.

3 A3.0: Resolve issues from A2

In case you did not manage to finish some parts of A2 or had technical problems, such as the controller in A2.3 or the system integration in A2.4, then you need to do solve the issues before continuing with A3.1.

4 A3.1: Identify suitable algorithm to be implemented in hardware

The main objective of this project is to implement a dedicated co-processor to perform the operation seen in equation 1 with the best "performance" as possible. In the first task you need to decide what performance means to your group - is it:

- High speed
- Low area
- Low power

4.1 Functionality

With this decision in mind you should then identify a suitable way to implement the operation in equation 1. It is totally up to you to decide how. One way to do it could be to divide x_n by 30 and then sum it with the next division. Another way to do it could be to sum all the x_n values first and then divide. Think about the following:

- What are the implications of using one method as opposed to the other method in MWI?
- What effects does it have on speed, area and resource usage, power?
- Is parallelization of some operations in the co-processor possible?
- Is it possible to implement a co-processor with minimal area?

After thinking, taking notes of your ideas, draw a block diagram that illustrates your solution based on your design parameter (high speed, low power or low area). Discuss with your group members and with the TAs.

4.2 Communication interface

Given the block diagram, which implements the basic functionality of this equation, think about how to implement the communication with the processor. In other words:

- How should the filter transfer data from the memory to the co-processor?

- How will the processor tell the co-processor to begin execution?
- How will the processor tell the co-processor how to stop executing?

A simple idea of performing such communication between the processor and the co-processor, is that the processor will send a `cmd` signal and possibly `data` signal to the co-processor via the bus (in a similar fashion as when passing data between the processor and memory). The co-processor needs to be able to parse the `cmd` signal, and based on it figure out what to do. This is very similar in nature of how the instruction decode works in a processor.An example could be that the processor sends a `START` command (encoded for instance as `0x00000001` on the `cmd` bus) to the co-processor and this signal will cause the co-processor to start the execution.

In order to get data from the memory to the co-processor your processor will first need to get the data (using your load instruction) and then in a succeeding clock cycle send this data to the co-processor. Therefore you will need a `DATALOAD` command as well. The processor will send a `DATALOAD` command on the `cmd` bus and should include some data which the processor has just fetched from the memory on the `data` bus. When the co-processor receives this `DATALOAD` command it might store the data in some registers, RAM block or something similar.

For the communication part you need to:

1. Prepare a list of commands needed to carry out the calculation of the equation.
2. Augment your block diagram from section 4.1 with features about how to implement the communication interface.
3. Discuss the above with the TAs.

NB! Later in A3.3, the processor needs to be augmented to be able to send these commands to the co-processor. For now assume that your co-processor has the interface as seen on the `slave` side of the slave interface from figure 1, i.e. it will receive data from the `S_dataout` and a command from `S_cmdout` and that these are valid once `S_dataoutrdy` is 1. It can then pass data back to the bus by setting `S_datainrdy` to 1 and providing the data on `S_datain`.

5 A3.2: Execute implementation plan

Start by implementing the co-processor, without connecting it to the bus. Proceed in a similar fashion as when you built the processor in A2. Build the components necessary for the datapath - if possible, try to reuse the components given in A2 or other datapaths used in A2. Further, connect the components for the functionality part (as given section 4.1) and make it works. Continue by adding a communication interface part (as given in section 4.2), including the necessary control signals. For testing, use test benches for each step to test and verify that your components work as expected. When implementing, make sure to handle special cases - i.e. what happens when you divide by zero? How can you handle this case? Before you proceed further with the assignment, it is highly recommended that you build a testbench to test the co-processor. This

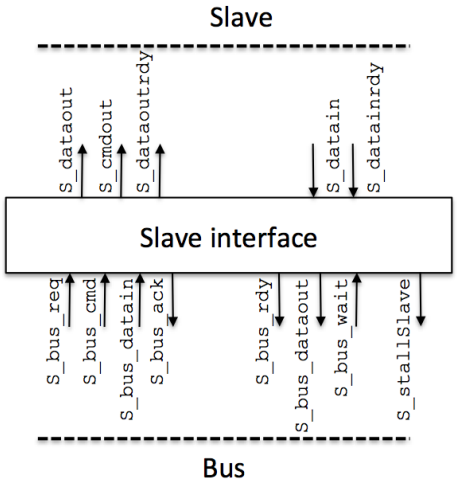

Figure 1: Slave interface

will make the integration of the co-processor to the rest of the system transit more smoothly.

6 A3.3: System integration

Now that your co-processor works, integrate it to the rest of the system built in A2. This involves several sub-tasks:

1. Connect the co-processor to the bus using a slave interface.
2. Add a `SCMD` (a send command) instruction to the processor, such that the communication with the co-processor is possible via assembly code.
3. Write a small program for the processor, which can communicate with the co-processor using the `SCMD` instruction.

The following subsection will explain each of these in detail.

6.1 Connect the co-processor

The co-processor will act as a slave to the processor, and should consequently be connected to a slave interface. The master-slave interface of the bus was described in great detail in A2. Connect the co-processor in a similar way, but using the slave interface. Observe how the slave interface is connected to the memory and try to mimic this.

6.2 Augment processor with additional instruction(s)

We recommend that you at least add a `SCMD` instruction to the processor. The `SCMD` instruction could have the following format `SCMD cmd, $data`, where `cmd` is an x bit vector telling the processor which command to put on the `cmd` bus

and `$data` is a register from which the processor should take data and put on the `data` bus. In this way you can for instance load data from the memory to the processor using a `LOAD` instruction and then in the next clock cycle send the data to the co-processor using a `SCMD` instruction.

6.3 Write code for communicating with co-processor

Using the instructions you added earlier write a small assembler program that can interact with the co-processor and execute the MWI filter.

7 A3.4: Performance analysis

Finally, analyze the performance of your ECG scanner in terms of speed, area and power and compare it to your analysis in Assignment 2. Consider the following:

- **Analyzing speed:** Think about, how can you say something about the speed of your implementation. What can you measure? What can you use this for? And how can you compare it to your implementation in A2.
- **Analyzing area:** What can be used in the Gezel programming language as a measure for area?
- **Analyzing power:** Gezel can count the number of toggles (i.e. signals switching from '0' to '1') using the `$option "profile_toggle_alledge"`, `$option "profile_toggle_upedge"`, `$option "profile_display_operations"` and `$option "profile_display_cycles"`. Read more about it here[1] and figure out for yourself how to do it.

[1] `http://rijndael.ece.vt.edu/gezel2/tools.html`