Assignment 1: Co-optimization of application and architecture for a coarse grained reconfigurable architecture

Mark Wijtvliet

2017

Organization

One of the big buzzwords of the year is 'brain inspired computing'. However, to do so we first need to know what our brain is doing. A good way to find out is to use electroencephalography[1] (EEG) to record brain activity. Using advanced algorithms these brain signals can be classified, e.g. they can be used to recognize a specific brain activity. Signals obtained from an EEG are very noisy and need to be filtered first. Figure 1 shows an example EEG signal, do you see what activity this is?[2].

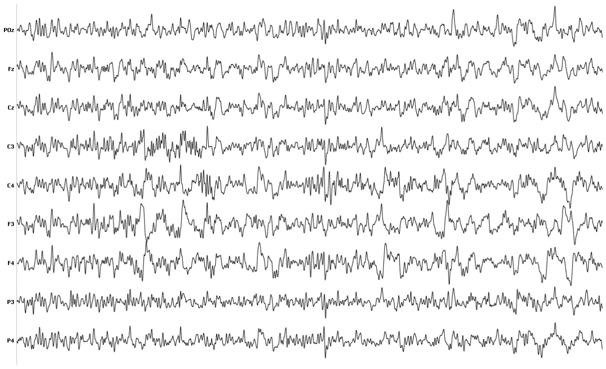


Figure 1: *Brain signals*

Your task will be to implement the first stage of the filtering. A popular filter for this is the Butterworth filter, which is an IIR type filter. An initial, quite naive implementation of a filter will be given to you, you will have to optimize it for energy, area and performance. We will also supply you with the a C-implementation of this filter to help you check if everything is working good. The processor architecture we will be using is our Coarse Grained Reconfigurable Architecture (CGRA[3]), which is, as the name implies, reconfigurable. More details about this architecture will follow in the next sections, the important aspect for this assignment is that you will be able to[4] modify the processor to allow a more efficient mapping of the application.

For this assignment you will work on one of our servers where all parts of the tool-flow are known to work. You can of course attempt to run the tools used for this assignment at your own computer but support will be limited.

This assignment consists of two parts, the first part is a cookbook exercise to guide you through using the tool-flow to modify the application and processor architecture to more efficiently support multiply

[1] Try saying *that* quickly 5 times ...

[2] Me etiher.

[3] www.cgra.nl

[4] And you really will have to.

accumulate operations. The second part is to change the application and architecture to your own insights to obtain a very low cycle-count while also having a decent resource utilization (e.g. adding lots of units and not really using them is not a good solution). Make sure to describe your reasoning behind the changes you make in the report, the reasoning is more important than the actual results[5].

1 The architecture

The architecture that we will use for this assignment is a research architecture from the Electronic Systems group. As we did not decide on a name for it yet we will just call it the CGRA for now. The next sections will describe the architecture, and how to use it, in a bit more detail.

1.1 The CGRA

As mentioned before CGRA means Coarse Grained Reconfigurable Architecture. This architecture differs from 'normal' CPUs by it's ability to reconfigure the processor data-path. This allows design a different processor for every application that you plan to run on it. When you start your application your processor design will be instantiated on the chip and execution of your application will start. Figure 2 shows what the CGRA architecture looks like and how a example data-path could be mapped onto it.

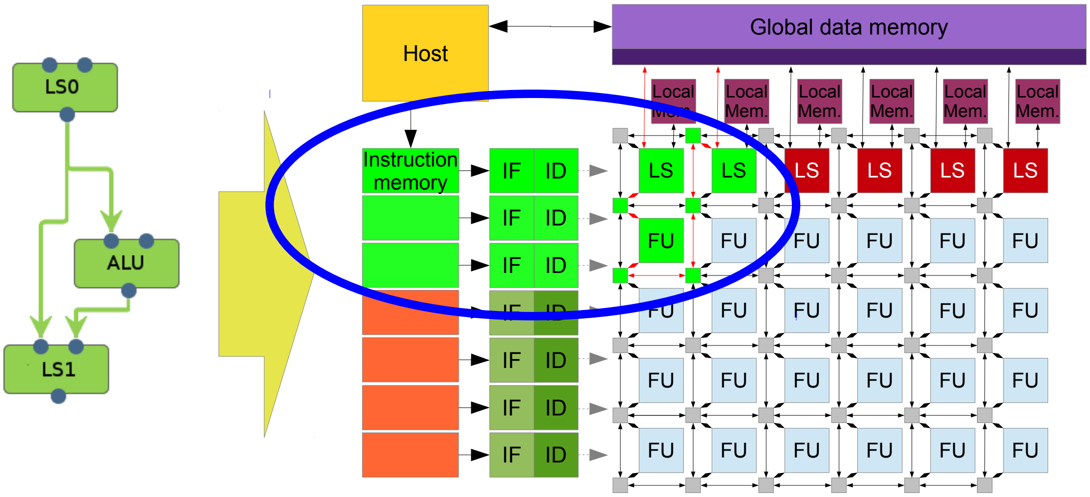


Figure 2: *CGRA architecture overview and example data-path mapping*

As you can see in figure 2 there is a grid of functional units (FU) that are connected to small grey boxes. These boxes are called switch-boxes and are used to route data from a specified input to a specified output to make connections between FUs. The routing is static for an application to reduce power consumption. In the architecture description, we will talk about this later, these connections can be specified. In order to reduce simulation time and make debugging a bit less difficult we will ignore the switch-boxes in this assignment and make direct connections instead[6].

We have several types of FUs:

- **ALU** - These perform typical operations such as addition, subtraction, shifting and comparison.
- **MUL** - These are multiplier units.

[5] But of course, they do count.
[6] Which is fine since we only want to see you optimize a single application anyway.

- **RF** - These are register files and contain 16 registers in which you can store variables. Register file operations in the CGRA are explicit[7].
- **LSU** - The Load Store Units are used for reading and writing data from/to the memories. Each LSU has its own local scratch-pad type memory and a connection to the global memory that all LSUs can access.
- **ABU** - This is the Accumulate and Branch Unit. Depending on its configuration it can be used as a multiple-register accumulator (very useful for filtering applications) or as the unit that generates the program counter (PC) and performs branch operations.
- **IMM** - The immediate unit can be used to generate values on the data network, these values typically are constants used in the application.

Connecting functional elements via switch-boxes is quite similar to how a FPGA works. The main difference with the CGRA is the much bigger building blocks, a FPGA typically uses building blocks of 1 (or sometimes a few) bits that implement simple logical operations (and, or, sometimes a 1-bit full-adder) while the CGRA uses much larger bit-widths[8] that perform more complex operations. This reduces the amount of configuration memory-bits requited, which is good for energy.

You might already have noticed that there are also some units called 'ID' and 'IF', these are Instruction Decode and Instruction Fetch units. These units can be connected to one or more FUs and control their operation, this allows us to execute programs on the CGRA (which is another difference to FPGAs[9]). By connecting multiple FUs to the same ID a vector processor can be constructed, since all FUs will now perform the same operation (DLP), see figure 3. If multiple IDs are used to control the FUs a VLIW style processor is constructed (ILP). Any mix between ILP and DLP can be made in order to allow for efficient application mapping. for example the reduction tree shown in figure 3.

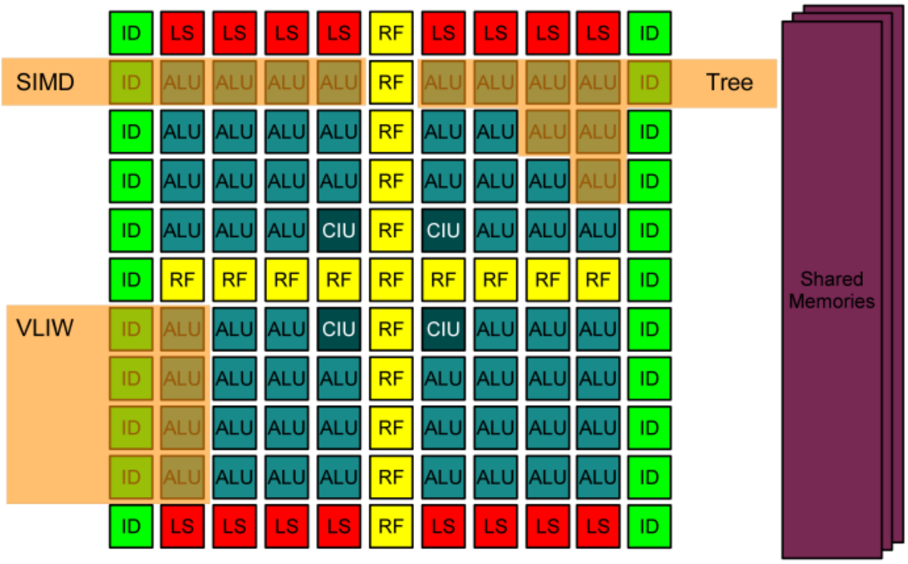

ID LS LS LS LS RF LS LS LS LS ID
SIMD
ID ALU ALU ALU ALU RF ALU ALU ALU ALU ID
Tree
ID ALU ALU ALU ALU RF ALU ALU ALU ALU ID
ID ALU ALU ALU ALU RF ALU ALU ALU ALU ID
ID ALU ALU ALU CIU RF CIU ALU ALU ALU ID
ID RF RF RF RF RF RF RF RF RF ID
VLIW
ID ALU ALU ALU CIU RF CIU ALU ALU ALU ID
ID ALU ALU ALU ALU RF ALU ALU ALU ALU ID
ID ALU ALU ALU ALU RF ALU ALU ALU ALU ID
ID ALU ALU ALU ALU RF ALU ALU ALU ALU ID
ID LS LS LS LS RF LS LS LS LS ID
Shared Memories

Figure 3: *SIMD, VLIW and mixed DLP+ILP configurations on the CGRA*

1.2 CGRA architecture description

To make your own CGRA instance you will have to change the supplied architecture description. This is an XML file containing a description of which FUs are present and how they are connected. The

[7] As opposed to a typical processor where register operations are implicit in the instruction.

[8] 8, 16 or 32-bit, depending on the version of the CGRA. For the assignment we will use the 32-bit version.

[9] Unless you implement a soft-core processor on the FPGA but this is not very efficient.

functional units in the supplied CGRA instance support up to 4 inputs and up to 2 outputs. The supplied architecture description is shown in listing 1. This file can be found in: `CGRA/platforms/butterworth/naive/architecture.xml`.

Line 11 in this listing shows an example instantiation of an instruction decoder (type="ID"). Each ID must have a unique name[10]. You will also notice that each of the IDs is connected to output 1 of a functional unit called ABU, this is a branch unit that calculates the program counter. In order for the program to advance this connection is required, all IDs that are connected to the same ABU work in lock step and form a processor. It is recommended that you keep this structure, since the assignment application does not require multiple independent processors.

Line 24 shows an instantiation for a LSU, with the name "lsu". The ID for this LSU is specified to be an instruction decoder with the name "id_lsu" (see line 11), this means that the FU "lsu" is controlled by "id_lsu". If we would like to make a vector unit we can connect multiple LSUs to "id_lsu"[11]. For each FU one or more inputs are specified, the inputs should have a unique index (port number that you want to connect them to) and a reference to an output of another FU. On line 25, for example, we can see that the LSU uses output 0 from the immediate unit.

Of course, humans like pictures much better than a plain text XML file so the CGRA toolchain produces a .dot file of the architecture description. This file can be viewed (in Linux) with xdot[12] and can be found in `benchmarks/<benchmark name>/report/architecture.dot` after building the benchmark[13]. Figure 4 shows the graphical representation of the default architecture description for the assignment.

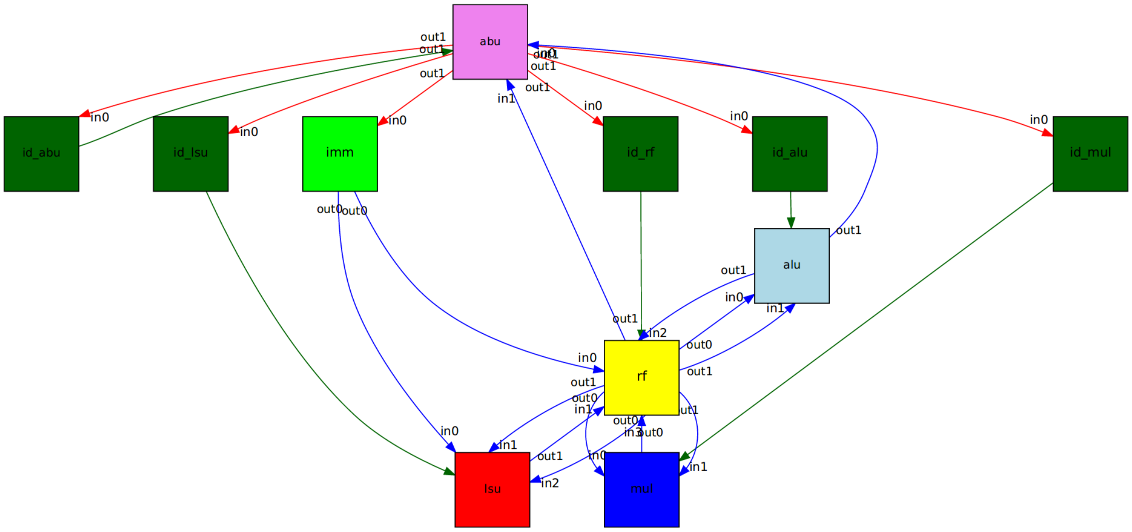


Figure 4: *Graphical representation of the default architecture description for this assignment.*

1.3 Programming the CGRA

Just specifying the architecture description that you want to use is of course not sufficient, some code has to be written as well. Since the compiler for this architecture is in very early stages of development programming this architecture[14] has to be done in our own assembly dialect, which we call PASM (Parallel Assembly), an example of this is shown in figure 5 and 6. You will notice that there are

[10] The same for FUs actually

[11] It is not possible to mix FU types and connect them to the same decoder

[12] If you do so on the server you should start your SSH session with X11 forwarding enabled.

[13] See further sections of this document on how to do so

[14] You might want to sit down before you continue reading this sentence.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<architecture>
    <Includes>
        <Base file="../../Common/32b.xml"/> <!-- Inherit from base configuration -->
    </Includes>
    <configuration>
        <functionalunits>
            <fu type="ID" name=’id_abu’> <!-- IDs -->
                <input index="0" source="abu.1"/>
            </fu>
            <fu type="ID" name=’id_lsu’>
                <input index="0" source="abu.1"/>
            </fu>
            <fu type="ID" name=’id_alu’>
                <input index="0" source="abu.1"/>
            </fu>
            <fu type="ID" name=’id_mul’>
                <input index="0" source="abu.1"/>
            </fu>
            <fu type="ID" name=’id_rf’>
                <input index="0" source="abu.1"/>
            </fu>

            <fu type="LSU" name=’lsu’ ID="id_lsu"> <!-- FUs -->
                <input index="0" source="imm.0"/>
                <input index="1" source="rf.1"/>
                <input index="2" source="rf.0"/>
            </fu>
            <fu type="ALU" name=’alu’ ID="id_alu" config="1">
                <input index="0" source="rf.0"/>
                <input index="1" source="rf.1"/>
            </fu>
            <fu type="MUL" name=’mul’ ID="id_mul" config="1">
                <input index="0" source="rf.0"/>
                <input index="1" source="rf.1"/>
            </fu>
            <fu type="RF" name=’rf’ ID="id_rf">
                <input index="0" source="imm.0"/>
                <input index="1" source="lsu.1"/>
                <input index="2" source="alu.1"/>
                <input index="3" source="mul.0"/>
            </fu>
            <fu type="ABU" name=’abu’ ID="id_abu" config="1">
                <input index="0" source="alu.1"/>
                <input index="1" source="rf.1"/>
            </fu>
            <fu type="IU" name=’imm’>
                <input index="0" source="abu.1"/>
            </fu>
        </functionalunits>
    </configuration>
    <Core>
        <Peripherals>
            <Peripheral type="Console" name="CON" addr_offset="32768"/>
        </Peripherals>
        <Memory type="">
            <GM width="32" depth="8192" addresswidth="32" interface="DTL"/>
        </Memory>
    </Core>
</architecture>
```

Listing 1: *The supplied architecture description*

several columns with assembly instructions. Each of these columns controls an ID, the name on the first line of each column must match one of the instruction decoders in the architecture description. E.g. if you have an ID called ”id_alu” in the architecture description then you should also have one in the PASM file. Important to note is that in the CGRA execution starts at instruction 1 (not at 0), for this reason the first line of code has perform a 'nop' (no operation) for all IDs. You will also notice a 'jai 0' instruction at the end of the most left column, this is a branch to instruction 0. Doing so will halt execution (ends your program).

The instruction set details can be found on Oncourse, in the ‘CGRA structure’ document. Most of the FUs and their instructions operate in a very similar way: after execution the results are available on the output ports in the next clock cyle. The RF is an exception to this, data is available for other units in the same cycle.

Note: all functional units except the immediate unit (IU) use NOP to specify a No-Operation. The immediate unit uses NOPI.

In order to make editing the PASM files a little easier we will supply a text highlighting package for two text editors: SublimeText and VIM. You can download these on the Oncourse website.

Trick question 1

How do you know what the source of the data is for a certain functional unit input? E.g. if we have the instruction `sga BYTE, in2, in1` for the LSU, where does the data on in1 come from?

1.4 Bugs

Since this architecture is a research project, and not a commercially available processor, you most likely will run into some bugs. These can be either in the tool-flow or the hardware. If you encounter such bugs, please report them via the ‘bugs’ section on the forum. For us to be able to solve a bug we always need a clear description of the problem and a method for reproducing the bug. Out of all reported bugs we will select the ‘best’ one and reward this with a beer card for the Walhalla.

2 Cookbook exercise

In order to work with the CGRA you will need to know something about the tool-flow. In the next sections we will explain the commands to use it[15].

2.1 Setup of the tool-flow

If you will be working on the university servers the tool-flow is already pre-installed and you can simply test it by logging onto the server with SSH[16], you can use any of the servers listed below[17]:

```
co2.ics.ele.tue.nl
co3.ics.ele.tue.nl
co4.ics.ele.tue.nl
co8.ics.ele.tue.nl
co9.ics.ele.tue.nl
co10.ics.ele.tue.nl
```

[15] In the command listings shown in this document you will see the $ symbol used to represent the command prompt, many of the online instructions that you will find will use the same convention. The advantage is that this allows you to distinguish the commands entered from the output which will be shown without the $ sign.

[16] make sure you have a Xserver installed on your laptop and X11 forwarding enabled, or use MobaXTerm.

[17] Your username and password can be found in the grade information on Oncourse.

```
| id_abu                     | id_rf                                            | id_lsu                                         | imm                        | id_alu                              | id_mul                                         |
|                            |                                                  |                                                |                            |                                     |                                                |
| .text                      | .text                                            | .text                                          | .text                      | .text                               | .text                                          |
|                            |                                                  |                                                |                            |                                     |                                                |
| nop                        | nop                                              | nop                                            | nopi                       | nop                                 | nop    ; always leave this line with only NOPs |
| ; LOAD COEFF INTO MEM (LM) |                                                  |                                                | imm 0 ; coeff write addr.  |                                     |                                                |
|                            | srm r0, in0 ; addr -> r0                         |                                                | imm 4 ; coeff addr incr.   |                                     |                                                |
|                            | srm r1, in0 ; incr -> r1                         |                                                | imm 65536 ; a[0]           |                                     |                                                |
|                            | srm r2, in0 ; a[0] -> r2                         |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; a[1]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[1] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm -64572 ; a[2]          |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[2] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; a[3]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[3] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 63818 ; a[4]           |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[4] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; a[5]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[5] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm -25323 ; a[6]          |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[6] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; a[7]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[7] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 7287 ; a[8]            |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[8] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; a[9]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[9] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm -740 ; a[10]           |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, a[10] -> r2     |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 64 ; coeff write addr. |                                     |                                                |
|                            | srm r0, in0 ; addr -> r0                         |                                                | imm 7104 ; b[0]            |                                     |                                                |
|                            | srm r2, in0 ; b[0] -> r2                         |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; b[1]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[1] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm -35513 ; b[2]          |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[2] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; b[3]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[3] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 71021 ; b[4]           |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[4] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; b[5]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[5] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm -71021 ; b[6]          |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[6] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; b[7]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[7] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 35513 ; b[8]           |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[8] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm 0 ; b[9]               |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[9] -> r2      |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | imm -7104 ; b[10]          |                                     |                                                |
|                            | lrm_srm r1, r2, in0 ; load incr, b[10] -> r2     |                                                | nopi                       | add out1, in0, in1 ; increment addr |                                                |
|                            | srm r0, in2 ;store addr+incr (from ALU)          |                                                | nopi                       |                                     |                                                |
|                            | lrm r2      ; load coeff                         | sla WORD, in2, in1 ; store value r2 on addr r0 | nopi                       |                                     |                                                |
|                            |                                                  |                                                | imm 4 ; incement           |                                     |                                                |
| ; init x and y delay lines | srm r0, in0 ; 4 -> r0                            |                                                | imm 44 ; i=11*4            |                                     |                                                |
|                            | srm r5, in0 ; i->r5                              |                                                | imm 128 ; addr x[0]        |                                     |                                                |
|                            | srm r3, in0 ; addr x[0] -> r3                    |                                                | imm 192 ; addr y[0]        |                                     |                                                |
|                            | srm r4, in0 ; addr y[0] -> r4                    |                                                | imm 0                      |                                     |                                                |
| ; branch here (i)          | lrm r3 ; load addr x[i]                          | sla WORD, in1, in0 ; write 0 in x[i]           | imm 0                      | add out1, in0, in1 ; addr x+4       |                                                |
|                            | lrm_srm r4, r3, in2 ; load addr y[i], store Ax-4 | sla WORD, in1, in0 ; write 0 in x[i]           | imm 0                      | add out1, in0, in1 ; addr y+4       |                                                |
|                            | lrm_srm r5, r4, in2 ; load i, store Ay-4         |                                                | imm 0                      | sub out1, in0, in1 ; i-1            |                                                |
| bcri -5, in0               | srm r5, in2                                      |                                                | imm 0                      |                                     |                                                |
```

Figure 5: *PASM code for the Butterworth filter 1/2*

```
| ; branch delay slot 1       |                                                        |                                              | imm 0                         |                                              |                                            |
| ; branch delay slot 2       |                                                        |                                              | imm 0                         |                                              |                                            |
|                             |                                                        |                                              | imm -72 ; np branch jump      |                                              |                                            |
|                             | srm r7, in0  ; branch jump -> r7                       |                                              | imm 0  ; data load addr       |                                              |                                            |
|                             | srm r6, in0 ; store data load addr -> r6               |                                              | imm 2000 ;np                  |                                              |                                            |
|                             | srm r2, in0 ; store np -> r2                           |                                              | nopi                          |                                              |                                            |
| ;DELAY LINE x + branch (np) |                                                        |                                              | imm 10 ; i=10                 |                                              |                                            |
|                             | srm r5, in0 ; i->r5                                    |                                              | imm 132 ;load address         |                                              |                                            |
|                             | srm r0, in0 ; load address -> r0                       |                                              | nopi                          |                                              |                                            |
| ; branch here (i)           | lrm r1 ; load incr, r0 is available implicit           | lla WORD, out1, in2 ; load x[i]              | nopi                          | sub out1, in1, in0 ; load_addr - 4           |                                            |
|                             | lrm_srm r1, r0, in2 ; load incr,  store load_addr-4    |                                              | nopi                          | add out1, in1, in0 ; load_addr + 4           |                                            |
|                             | srm r3, in2 ; store load_addr + 4 -> r3                |                                              | nopi                          |                                              |                                            |
|                             | lrm_srm r3, r4, in1 ; load addr+4, store x[i] -> r4    |                                              | imm -1 ; for i=i-1            | pass out1, in1 ; pass addr+4                 |                                            |
|                             | lrm_srm r4, r0, in0 ; load x[i-1] sore -1 -> r0        | sla WORD, in2, in1  ; store x[i] in x[i-1]   | nopi                          |                                              |                                            |
|                             | lrm_srm r5, r0, in2 ; load i store load_addr+4         |                                              | nopi                          | add out1, in1, in0 ; i-1                     |                                            |
| bcri -8, in0                | srm r5, in2 ; store i-1                                |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 1       |                                                        |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 2       |                                                        |                                              | nopi                          |                                              |                                            |
| ;DELAY LINE y               |                                                        |                                              | imm 10 ; i=10                 |                                              |                                            |
|                             | srm r5, in0 ; i->r5                                    |                                              | imm 196 ;load address         |                                              |                                            |
|                             | srm r0, in0 ; load address -> r0                       |                                              | nopi                          |                                              |                                            |
| ; branch here (i)           | lrm r1 ; load incr, r0 is available implicit           | lla WORD, out1, in2 ; load x[i]              | nopi                          | sub out1, in1, in0 ; load_addr - 4           |                                            |
|                             | lrm_srm r1, r0, in2 ; load incr,  store load_addr-4    |                                              | nopi                          | add out1, in1, in0 ; load_addr + 4           |                                            |
|                             | srm r3, in2 ; store load_addr + 4 -> r3                |                                              | nopi                          |                                              |                                            |
|                             | lrm_srm r3, r4, in1 ; load addr+4, store x[i] -> r4    |                                              | imm -1 ; for i=i-1            | pass out1, in1 ; pass addr+4                 |                                            |
|                             | lrm_srm r4, r0, in0 ; load x[i-1] sore -1 -> r0        | sla WORD, in2, in1  ; store x[i] in x[i-1]   | nopi                          |                                              |                                            |
|                             | lrm_srm r5, r0, in2 ; load i store load_addr+4         |                                              | nopi                          | add out1, in1, in0 ; i-1                     |                                            |
| bcri -8, in0                | srm r5, in2 ; store i-1                                |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 1       |                                                        |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 2       |                                                        |                                              | nopi                          |                                              |                                            |
|                             | lrm r6 ; load data address                             | lga WORD, out1, in1 ; load data              | nopi                          | pass out1, in1 ; pass data addr              |                                            |
|                             | srm r0, in2 ; data_addr -> r0                          |                                              | nopi                          |                                              |                                            |
|                             | lrm_srm r1, r3, in1 ; load incr, data -> r3            |                                              | nopi                          | add out1, in1, in0 ; data_addr + 4           |                                            |
|                             | srm r6, in2 ; data_addr+4 -> r6                        |                                              | imm 168 ; x[10] addr          |                                              |                                            |
|                             | srm r0, in0 ; x[10] addr -> r0                         |                                              | imm 232 ; y[10] addr          |                                              |                                            |
|                             | lrm_srm r3, r0, in0 ; load data, y[10] addr -> r0      | sla WORD, in2, in1 ; data -> x[10]           | imm 0 ; new y                 |                                              |                                            |
|                             | srm r3, in0 ; 0 -> r3                                  |                                              | nopi                          |                                              |                                            |
|                             | lrm r3 ; load r3                                       | sla WORD, in2, in1 ; 0 -> x[10]              | nopi                          |                                              |                                            |
|                             |                                                        |                                              | nopi                          |                                              |                                            |
|                             |                                                        |                                              | imm 4 ; incr                  |                                              |                                            |
|                             | srm r0, in0 ; incr -> r0                               |                                              | imm 0 ; set yy=0              |                                              |                                            |
|                             | srm r15, in0 ; yy -> r15                               |                                              | imm 44 ; set j=11*4           |                                              |                                            |
|                             | srm r10, in0 ; j -> r10                                |                                              | imm 64 ; set Addr_b =64       |                                              |                                            |
|                             | srm r11, in0 ; Addr_b -> r11                           |                                              | imm 168 ; set Addr_x=10*4+128 |                                              |                                            |
|                             | srm r12, in0 ;  Addr_x ->  r12                         |                                              | nopi                          |                                              |                                            |
| ;branch here (j)            | lrm r11 ; load Addr_b                                  | lla WORD, out1, in1 ; load b[j]              | nopi                          | add out1, in0, in1 ;addr_b+4                 |                                            |
|                             | srm r11, in2 ; store addr_b                            |                                              | nopi                          |                                              |                                            |
|                             | lrm_srm r12, r3, in1 ; load addr_x, store b[j] -> r3   | lla WORD, out1, in1 ; load x[i-j]            | nopi                          | sub out1, in0, in1 ;addr_x-4                 |                                            |
|                             | lrm_srm r10, r12, in2 ; load j, store addr_x           |                                              | nopi                          | sub out1, in0, in1 ;j-4                      |                                            |
|                             | lrm_srm r12, r0, in1 ; load addrx , store x[i-j]       |                                              | nopi                          |                                              |                                            |
|                             | lrm_srm r3, r10, in2 ; load b[j], store j-4 -> r10     |                                              | nopi                          |                                              | mulls_sh16 out0, in1, in0 ;b[j]*x[i-j]     |
|                             | srm r0, in3 ; store b[j]*x[i-j]                        |                                              | nopi                          |                                              |                                            |
|                             | lrm r15 ; load yy                                      |                                              | imm 4 ; incr                  | add out1, in0, in1                           |                                            |
| bcri -10, in1               | lrm_srm r10, r0, in0 ; ; load j-4 , incr -> r0         |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 1       | srm r15, in2 ;store yy+ b[j]*x[i-j]                    |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 1       |                                                        |                                              | imm 40 ; j=10*4               |                                              |                                            |
|                             | srm r10, in0 ; j -> r10                                |                                              | imm 4 ; set Addr_a =4         |                                              |                                            |
|                             | srm r11, in0 ; Addr_b -> r11                           |                                              | imm 228 ; set Addr_y=9*4+192  |                                              |                                            |
|                             | srm r12, in0 ;  Addr_x ->  r12                         |                                              | nopi                          |                                              |                                            |
| ;branch here (j)            | lrm r11 ; load Addr_a                                  | lla WORD, out1, in1 ; load a[j+1]            | nopi                          | add out1, in0, in1 ;addr_a+4                 |                                            |
|                             | srm r11, in2 ; store addr_a                            |                                              | nopi                          |                                              |                                            |
|                             | lrm_srm r12, r3, in1 ; load addr_x, store a[j+1] -> r3 | lla WORD, out1, in1 ; load y[i-j-1]          | nopi                          | sub out1, in0, in1 ;addr_y-4                 |                                            |
|                             | lrm_srm r10, r12, in2 ; load j, store addr_y           |                                              | nopi                          | sub out1, in0, in1 ;j-4                      |                                            |
|                             | lrm_srm r12, r0, in1 ; load addrx , store y[i-j-1]     |                                              | nopi                          |                                              |                                            |
|                             | lrm_srm r3, r10, in2 ; load b[j], store j-4 -> r10     |                                              | nopi                          |                                              | mulls_sh16 out0, in1, in0 ;a[j+1]*y[i-j-1] |
|                             | srm r0, in3 ; store a[j+1]*y[i-j-1]                    |                                              | nopi                          |                                              |                                            |
|                             | lrm r15 ; load yy                                      |                                              | imm 4 ; incr                  | sub out1, in0, in1                           |                                            |
| bcri -10, in1               | lrm_srm r10, r0, in0 ; ; load j-4 , incr -> r0         |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 1       | srm r15, in2 ;store yy+ a[j+1]*y[i-j-1]                |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 1       |                                                        |                                              | imm 232 ; y[10] addr          |                                              |                                            |
|                             | srm r0, in0 ; y[10] addr -> r0                         |                                              | imm 16380 ;write offset       |                                              |                                            |
|                             | lrm_srm r15, r0, in0 ; load yy, write_offset -> r0     | sla WORD, in2, in1 ;store yy in y[10]        | nopi                          |                                              |                                            |
|                             | lrm r6 ; load read data addr                           |                                              | nopi                          | add out1, in0, in1 ;read addr + write offset |                                            |
|                             | srm r0, in2 ;store write_addr -> r0                    |                                              | imm -1 ; for np-1             |                                              |                                            |
|                             | lrm_srm r15, r0, in0  ; -1 -> r0                       | sga WORD, in2, in1 ; store yy in data memory | nopi                          |                                              |                                            |
|                             | lrm r2 ; load np                                       |                                              | nopi                          | add out1, in1, in0; np-1                     |                                            |
| bcr in1, in0                | lrm_srm r7, r2, in2  ; load branch jump, np-1 -> r2    |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 1       |                                                        |                                              | nopi                          |                                              |                                            |
| ; branch delay slot 2       |                                                        |                                              | nopi                          |                                              |                                            |
| jai 0 ; terminate execution | nop                                                    | nop                                          | nopi                          | nop                                          | nop                                        |
```

Figure 6: *PASM code for the Butterworth filter 2/2*

```
co11.ics.ele.tue.nl
co13.ics.ele.tue.nl
co14.ics.ele.tue.nl
```

You can test the tool-flow by typing the following commands:

```
$ cd ~/CGRA/benchmarks/butterworth/naive
$ make compare
```

This should complete without any errors and tell you that the compared files are identical[18].

You can also install the tool-flow on your own laptop, please read the following sections for two options to get this working.

Note: it is much harder to provide quick bugfixes to everyone who is not working on the server. You will be responsible for reading the announcements and applying any bugfixes yourself.

2.1.1 If you have Linux

If you plan to run the tool-flow on your own laptop (Linux only) you will have to do the following:

- Download and extract the tool-flow that can be found on Oncourse.
- Install the python packages `xmltodict` and `svgwrite` using pip.
- Install modelsim. A linux version for Modelsim can be downloaded here: `http://download.altera.com/akdlm/software/acdsinst/16.1/196/ib_installers/ModelSimSetup-16.1.0.196-linux.run`
- Modify setup.sh in the root of the CGRA tool-flow to point to the correct executable locations.

When you want to use the tool-flow you need to source the setup script with the following commands:

```
$ cd CGRA
$ . setup.sh
```

You can also automate this such that whenever you open a new terminal session it automatically sets-up the tool-flow. You can do so by adding a line to `.bashrc` such that it automatically gets executed when you login. Use the following commands for this (assuming you put the tool-flow directory in your home directory):

```
$ cd ~
$ echo "cd ~/CGRA && . ./setup.sh && cd ~" >> .bashrc
```

If you want to test if the whole flow works you can run the following commands:

```
$ cd ~/CGRA/benchmarks/butterworth/naive
$ make compare
```

This should complete without any errors.

[18] If not, run: `make clean` and try again

2.1.2 Using a Virtual Machine

If you plan to run the tool-flow on your own laptop and you do not have Linux or do not want to risk de-configuring your laptop you can use our VirtualBox Virtual Machine. The VM can be downloaded at `http:/www.cgra.nl/CGRA_2017.ova` (5.3 GB). All required tools are already pre-installed.

- You will need to install VirtualBox in order to be able to run the Virtual Machine. You can download VirtualBox via `https://www.virtualbox.org/wiki/Downloads`.
- After downloading the VM import it into VirtualBox via the `File->Import Appliance` option.
- The default user for this VM is 'cgra', the password is also 'cgra'[19].

2.2 Capabilities of the makefiles

The makefile included with the Toolchain can be executed with several options, these are:

- **sim** : This will run Modelsim with a graphical interface, you can use this for debugging if things are not working as you expect.
- **run** : This will run Modelsim in the command line, this is mostly useful for generating a memory dump and performance estimate files.
- **compare** : This will run the simulation and compare files in the `compare` directory of the benchmark against the results generated by the simulation.
- **performance** : This will run the simulation show quick performance estimates. Although this method is fast, it is a bit less accurate than `make report`.
- **report** : This option will generate a report in `build/<benchmark name>/report` (and on the terminal) reporting the energy (pJ), area (um^2) and performance results. This method works on a trace of the simulation results and is slower than `make performance` but more accurate.
- **clean** : This will clean the build directory for the benchmark. Please do so after you gathered your results to limit disk space usage.

An example showing how to use this is shown in the listing below, you first go to the directory of the benchmark that you want to run and then run the makefile.

```
$ cd benchmarks/butterworth/naive
$ make compare
```

If make is run without any options, it will just build the hardware and software for the benchmark but it will not execute it. This is useful for a quick check if the syntax of all files is correct.

2.3 A first optimization

In order to help you understand the assembly code shown in figure 5 and 6 a bit easier, listing 2 shows the exact same algorithm in C. The parts where the data is loaded and the result stored to a file are left out for clarity, we just assume that the data is loaded already. The C-code will not actually be used but it could be useful to obtain some intermediate (reference) results for debugging purposes.

Trick question 2

How many multiplications are performed for computing one Butterworth result (the result for a

[19] Without the single quotes.

```
...

#define order 10
#define SCALE 16
#define NP 2000

static const int32_t a[order+1] = {65536, 0, -64572, 0, 63818, 0, -25323, 0, 7287, 0, -740};
static const int32_t b[order+1] = {7104, 0, -35513, 0, 71021, 0, -71021, 0, 35513, 0, -7104};

//Read input data from file
...

while(np--)
{
    for (i=1; i<=order; i++)
    {
            x[i-1]=x[i];
            y[i-1]=y[i];
    }

    i = order;
    x[i] = *xi++;
    yy = 0;

    for (j=0;j<order+1;j++)
        yy = yy + (uint32_t)(((int64_t)(b[j])*(int64_t)(x[i-j])) >> SCALE);
    for (j=0;j<order;j++)
        yy = yy - (int32_t)(((int64_t)(a[j+1])*(int64_t)(y[i-j-1])) >> SCALE);
    *yo++ = y[i] = yy;
}

//Write output data to file
```

Listing 2: *The Butterworth algorithm in C*

single output value) shown in listing 2?[a]

[a] Hint: Make sure to include computations needed for loading and storing the data. You might have a look at the assembly code as well, but make sure you understand why.

By inspecting the assembly code we can observe that most of the memory address calculations involve some additions and a multiplication. The intermediate results of each add and multiply are now stored in the register file, after which they have to be loaded from the register file again. This is quite similar to what a normal processor will do (unless it is lucky and can bypass something within its stages) but not very efficient. In this part of the assignment we will improve the processor by adding bypass options between the MUL and ALU units. By doing so we can prevent intermediate results from passing through the register file.

Exercise 1

Let's first check how much time it takes to execute the current implementation. You can run the original code by executing the following commands (in the Toolchain directory). If this is the first time you run anything with make for a certain benchmark (or you made changes to the source files) it can take some time to build and simulate the hardware.

```
$ cd benchmarks/butterworth/naive
$ make performance
```

The benchmark will now be executed on the CGRA platform. You will see some warnings appear during simulation, these can safely be ignored. Once simulation has finished the performance metrics for your application will be shown.

What is the utilization of the register file? Why do you think that it is so high?

In the example implementation it is clear that the register file operations are the bottleneck, both units responsible for computation (the ALU and MUL units) are not really doing all that much. If we have an addition followed by a multiplication then we see that the following pattern:

1. The ALU computes the result of the addition.
2. The addition result is stored in the register file.
3. The result from the addition is loaded from the register file.
4. The MUL computes the result of the multiplication.

Step 2 and 3 in this pattern do not contribute anything to the actual computation, it is just for moving data around. It would be much more efficient if we can directly use the output from the ALU as an input for the MUL. Which is what we are going to do in this cookbook.

The first thing to do is make a new benchmark, called '`bypass`' (instead of '`naive`'). We do this by simply copying the directory in the `benchmarks` folder.

```
$ cd ~/CGRA/benchmarks/butterworth/
$ cp -R naive bypass
```

Since we are also going to modify the processor architecture we should point the benchmark make file to another platform description. We can do this by editing the make file with your favourite editor, the make file is: `/CGRA/benchmarks/butterworth/bypass`

We will change line 3 from `PLATFORM=butterworth/naive` to `PLATFORM=butterworth/bypass`.

If we would try to run the make file now it would fail because the platform cannot be found. We will make this new platform with the following commands.

```
$ cd ~/CGRA/platforms/butterworth/
$ cp -R naive bypass
```

This is all that is required to make a new benchmark and a new platform. You can, of course, have multiple benchmarks pointing to the same platform.

We can now start modifying the new platform, and the code running on it, to implement our register file bypasses. We will start by modifying the architecture description. We will therefore edit the file /CGRA/platforms/butterworth/bypass/architecture.xml.

Find the following section in the architecture description file:

```
<fu type="ALU" name='alu' ID="id_alu" config="1">
    <input index="0" source="rf.0"/>
    <input index="1" source="rf.1"/>
</fu>
```

And change it to:

```
<fu type="ALU" name='alu' ID="id_alu" config="1">
    <input index="0" source="rf.0"/>
    <input index="1" source="rf.1"/>
    <input index="2" source="mul.0"/>
</fu>
```

Now find the section describing the MUL unit and add a new input with index 2 and source alu.1.

To see the changes in your architecture, execute the following commands:

```
$ cd ~/CGRA/benchmarks/butterworth/bypass
$ make
```

If you did everything write the last commands should fail without any errors. When the command is finished we can have a look at how the new architecture looks by opening the graphical representation, we can do that with the following command:

```
$ xdot ~/CGRA/build/butterworth/bypass/report/architecture.dot
```

You will then see[20] the image shown in figure 7. If you compare this with the original architecture description, you will notice the extra arrow, going from ALU to MUL. This new connection will allow us to bypass the register file and directly use the result from one functional unit by another functional unit.

Just modifying the architecture will not be sufficient, we also have to modify the assembly code to make use of this new functionality. As an example we will modify the multiplication of the weights with the input data, every multiplication result is now stored in the RF before being added to the variable ‘yy’. We will remove this using our new bypass.

If you look at lines 133 and 147 in the ‘id_mul’ column you can see the multiplication happening. The result of the multiplication is stored on out0. In the next line in the ‘id_rf’ column the result is stored in register r0[21]. In the cycle after that the ‘id_alu’ performs the add action on the multiplication result that was stored (and loaded from) register r0.

[20] If you did everything correct

[21] r0 is a special register, its value is always available on out0 without an explicit load action. This allows loading two values from the RF at once, without addressing two

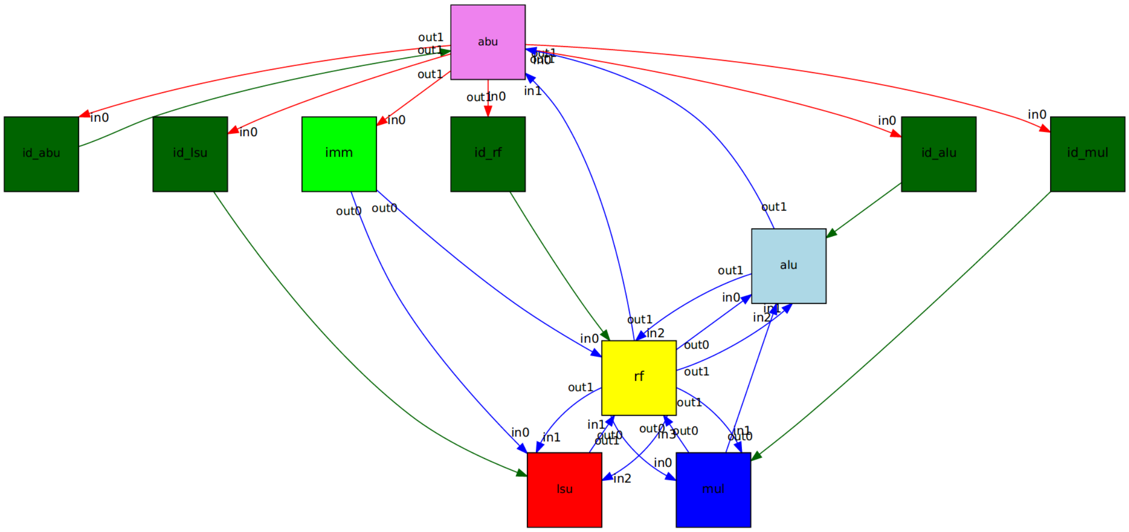


Figure 7: *Graphical representation of the modified architecture description.*

We can simply change the instructions on lines 135 and 149 for the ALU from:

```
add out1, in0, in1
```

to:

```
add out1, in2, in1
```

Storing the multiplication result in the RF is now no longer needed so lines 134 and 148 will be removed entirely. However, to be able to do so the branches (where the for-loops jump back) have to be modified. The instructions for the ABU on lines 136 and 150 have to be changed to jump back 9 cycles instead of 10:

```
bcri -9, in1
```

Additionally the outer loop is now two cycles shorter, this can be modified by changing the IMM instruction on line 85 to:

```
imm -70
```

Do you see why this line has to be modified[22]?

You can test your changes by running:

```
$ make compare
```

To check if the result is still correct, and:

```
$ make performance
```

In order to check what influence your changes made to the execution time.

[22] A nice trick to know how far to branch back is: click on the line after the branch instruction and select all lines up to the branch target, the number of selected lines is the number you need to use in the branch instruction.

A much more extensive report, including area and energy can be obtained by running the following commands:

```
$ make report
```

In a similar way, by editing the XML file, it is possible to add and remove functional units. In case you add an instruction decoder, make sure to add another column to your PASM file to control this ID. This concludes your first optimization on the CGRA architecture. You now have a basis to start working on the main assignment for this course

Exercise 2

How many cycles was the execution time reduced by implementing this optimization?

Exercise 3

What happened to the untilization of the ALU, MUL and RF? Can you explain why this is happening?

2.4 Debugging with Modelsim

During development of your optimizations you will very likely end up with some bugs in your program that you will need to resolve. The way to do this is by analysing the waveforms using Modelsim[23]. This section will give a short example of how to view the values that appear in the register file.

Exercise 4

Suppose that something is going wrong during calculation of the loop iterator of loop 'i'. By inspecting the assembly code we can see that the value for 'i' is stored in register r11. To view the contents of r11 we can run Modelsim with the graphical interface with the following commands:

```
$ cd ~/CGRA/benchmarks/butterworth/bypass
$ make sim
```

This will start a graphical interface as shown in figure 8[a]. After a while you will get the question 'Are you sure you want to finish?', click no to avoid Modelsim from closing.

We will now browse through the structure of the Verilog hardware description to find the functional that we are interested in (the register file). The document 'CGRA_structure.pdf' describes the code structure in more detail, including the naming conventions.

To add the register file to the wave viewer we browse to:
`TB_CGRA_Top.dut.CGRA_Core_inst.CGRA_Compute_Wrapper_inst.CGRA_Compute_inst.rf_inst`
in the instance viewer (the most left sub-window) and double click on rf_inst to open the Verilog description for the RF.

By inspecting the Verilog description we can see that the name of the 'variable' that stores the register values is 'rRegisters'. Knowing this we left-click on rf_inst in the instance viewer to show all its objects and right click on `rRegisters` to select `Add Wave`. You will observe that the signal is now added to the wave viewer (the black window). Right click the name of the wave in the wave viewer and select `Radix->Decimal` so you don't have to convert it from hexadecimal all the time.

We will now restart the simulation by typing `restart` in the command window in the bottom. When a dialog window appears click `OK`. You can now run the simulation for a specified amount

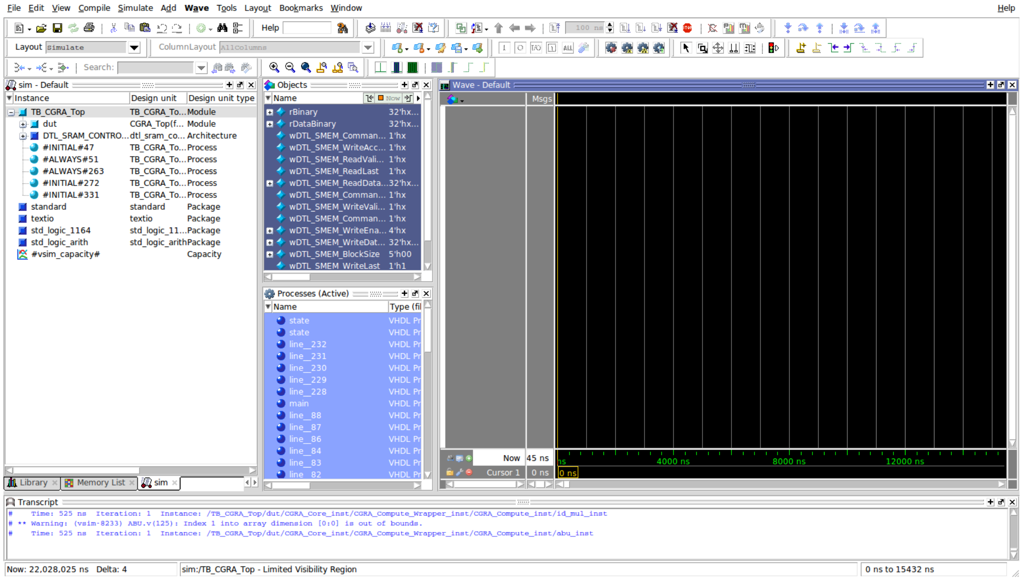
[23] A real hardware debugger is in development but not operational yet.

of time by typing `run 100ns`[b] or until the program finishes by typing `run -all`.

When simulation is complete you can click the small + before the name of the wave to expand it. You will now see all individual registers as shown in figure 9.

hint: in order to not add the waves every time you start the simulation, click somewhere in the wave viewer and press CTRL+S. If the name of the file ends on `wave.do` it will be automatically reloaded the next time you run the simulation.

[a] If you do not see the black part of the screen, click View→Wave.
[b] Or another time value of course.



Instance	Design unit	Design unit type
TB_CGRA_Top	TB_CGRA_To...	Module
dut	CGRA_Top(f...	Module
DTL_SRAM_CONTRO...	dtl_sram_co...	Architecture
#INITIAL#47	TB_CGRA_To...	Process
#ALWAYS#51	TB_CGRA_To...	Process
#ALWAYS#263	TB_CGRA_To...	Process
#INITIAL#272	TB_CGRA_To...	Process
#INITIAL#331	TB_CGRA_To...	Process
standard	standard	Package
textio	textio	Package
std_logic_1164	std_logic_11...	Package
std_logic_arith	std_logic_arith	Package
#vsim_capacity#		Capacity

Name	Now
rBinary	32'hx...
rDataBinary	32'hx...
wDTL_SMEM_Comman...	1'hx
wDTL_SMEM_WriteAcc...	1'hx
wDTL_SMEM_ReadVali...	1'hx
wDTL_SMEM_ReadLast	1'hx
wDTL_SMEM_ReadData...	32'hx...
wDTL_SMEM_Comman...	1'hx
wDTL_SMEM_WriteVali...	1'hx
wDTL_SMEM_Comman...	1'hx
wDTL_SMEM_WriteEna...	4'hx
wDTL_SMEM_WriteDat...	32'hx...
wDTL_SMEM_BlockSize	5'h00
wDTL_SMEM_WriteLast	1'h1

Name	Type (fil
state	VHDL Pr
state	VHDL Pr
line__232	VHDL Pr
line__231	VHDL Pr
line__230	VHDL Pr
line__229	VHDL Pr
line__228	VHDL Pr
main	VHDL Pr
line__88	VHDL Pr
line__87	VHDL Pr
line__86	VHDL Pr
line__84	VHDL Pr
line__83	VHDL Pr
line__82	VHDL Pr

```
#    Time: 525 ns  Iteration: 1  Instance: /TB_CGRA_Top/dut/CGRA_Core_inst/CGRA_Compute_Wrapper_inst/CGRA_Compute_inst/id_mul_inst
# ** Warning: (vsim-8233) ABU.v(125): Index 1 into array dimension [0:0] is out of bounds.
#    Time: 525 ns  Iteration: 1  Instance: /TB_CGRA_Top/dut/CGRA_Core_inst/CGRA_Compute_Wrapper_inst/CGRA_Compute_inst/abu_inst
```

Now: 22,028,025 ns Delta: 4 | sim:/TB_CGRA_Top - Limited Visibility Region | 0 ns to 15432 ns

Figure 8: *Startup screen for Modelsim*

3 The assignment

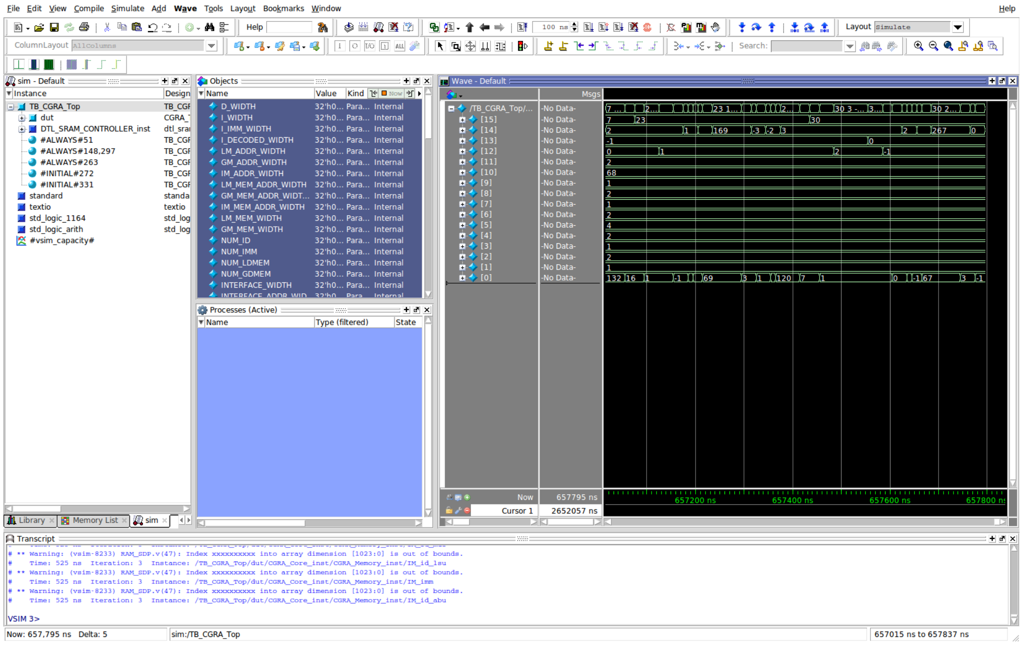
The final assignment is to modify the architecture and the assembly code in any way you want to achieve the following goal: best performance for minimal energy and area. You can do this by calculating the Energy-Delay-Area Product (EDAP). EDAP can be defined as:

$$EDAP = E \cdot D_{cycles} \cdot A$$

3.1 Optimization hints

There are many things you can do to improve the architecture for this application. Some of them are listed below but feel free to add your own ideas! After each hardware modification you will probably have to adapt the assembly code to actually use the optimization.

- **data parallelism**, you can connect multiple functional units to an instruction decoder in order to do SIMD (data parallel) processing. This gives you a lower instruction decoding overhead while increasing throughput.
- **instruction level parallelism**, you can exploit ILP by pipelining the instructions. Adding functional units do this this will increase area and energy, so you will have to make a trade-off between performance, area and energy.



Instance	Design
TB_CGRA_Top	TB_CGF
dut	CGRA_T
DTL_SRAM_CONTROLLER_inst	dtl_srai
#ALWAYS#51	TB_CGF
#ALWAYS#148,297	TB_CGF
#ALWAYS#263	TB_CGF
#INITIAL#272	TB_CGF
#INITIAL#331	TB_CGF
standard	standa
textio	textio
std_logic_1164	std_log
std_logic_arith	std_log
#vsim_capacity#	

Name	Value	Kind	Now
D_WIDTH	32'h0...	Para...	Internal
I_WIDTH	32'h0...	Para...	Internal
I_IMM_WIDTH	32'h0...	Para...	Internal
I_DECODED_WIDTH	32'h0...	Para...	Internal
LM_ADDR_WIDTH	32'h0...	Para...	Internal
GM_ADDR_WIDTH	32'h0...	Para...	Internal
IM_ADDR_WIDTH	32'h0...	Para...	Internal
LM_MEM_ADDR_WIDTH	32'h0...	Para...	Internal
GM_MEM_ADDR_WIDT...	32'h0...	Para...	Internal
IM_MEM_ADDR_WIDTH	32'h0...	Para...	Internal
LM_MEM_WIDTH	32'h0...	Para...	Internal
GM_MEM_WIDTH	32'h0...	Para...	Internal
NUM_ID	32'h0...	Para...	Internal
NUM_IMM	32'h0...	Para...	Internal
NUM_LDMEM	32'h0...	Para...	Internal
NUM_GDMEM	32'h0...	Para...	Internal
INTERFACE_WIDTH	32'h0...	Para...	Internal
INTERFACE_ADDR_WID...	32'h0...	Para...	Internal

Name	Type (filtered)	State

```
# ** Warning: (vsim-8233) RAM_SDP.v(47): Index xxxxxxxxxx into array dimension [1023:0] is out of bounds.
#    Time: 525 ns  Iteration: 3  Instance: /TB_CGRA_Top/dut/CGRA_Core_inst/CGRA_Memory_inst/IM_id_lsu
# ** Warning: (vsim-8233) RAM_SDP.v(47): Index xxxxxxxxxx into array dimension [1023:0] is out of bounds.
#    Time: 525 ns  Iteration: 3  Instance: /TB_CGRA_Top/dut/CGRA_Core_inst/CGRA_Memory_inst/IM_imm
# ** Warning: (vsim-8233) RAM_SDP.v(47): Index xxxxxxxxxx into array dimension [1023:0] is out of bounds.
#    Time: 525 ns  Iteration: 3  Instance: /TB_CGRA_Top/dut/CGRA_Core_inst/CGRA_Memory_inst/IM_id_abu

VSIM 3>
```

```
Now: 657,795 ns  Delta: 5    sim:/TB_CGRA_Top    657015 ns to 657837 ns
```

Figure 9: *Simulation of the register file*

- **data reuse**, the Butterworth kernel has quite some 'locality'. This means that many data values are used more than one time. Instead of loading them from the global memory every time you might consider storing them in the local memory or add a special register file to hold these values.
- **bypassing**, we already gave a small example of this in the cook book exercise but there are more opportunities for bypassing.
- **automatic address generation**, the load-store units now use an address that is explicitly calculated by the application. These units do however also have the option to automatically generate load and store addresses, this will save you the explicit calculation. Refer to the LSU documentation for more details.
- **Add another ABU**, the ABU cannot only be used for branches; it also supports accumulation. This can be very handy for filter type applications such as Butterworth.
- **Repeated instructions**, repeating the same instruction costs less energy than fetching a new one. The ideal situation will be a loop body with only one instruction that never changes until the loop finishes.

It is also possible to define your own functional units and/or instructions but this is a bit more advanced and not really required for the assignment, students that feel like they cannot optimize too much more without doing so can contact me (m.wijtvliet@tue.nl) for an explanation on how to do this.

3.2 The report

The report should consist of at most 4 pages (sides) of A4 paper in IEEE 2 column format, font size 10. Everything beyond page 4 **will not be read**. Only submissions in PDF will be accepted, people submitting in other formats will be requested to revise their submission. The assignment should be made **individually**.

Some other guidelines for the report are:

- Do not just present the results, but also explain the reasons. What do you expect? Do the results match what you expect? Why or why not?
- Explain why certain optimizations are performed. Are you optimizing the dominating parts? Are the chosen optimization techniques solving the bottleneck? If you do not manage to optimize the dominating parts, explain the reason.
- Explain results in a concise and clear way. Your understanding of the results is much more important than just getting the best result.
- It is important to understand the fundamental limits. E.g. when it does not make sense to optimize a certain aspect further.
- Tables and graphs will help you to explain results without long sections of text.
- If you use ideas or code from someone else (or work together) include the proper acknowledgements in your report. Failing to do so will be considered fraud.
- It is allowed to discuss with other students but the implementation and the report should be your own work.
- It is not required to give a long introduction into the assignment[24].

3.3 Contact hours

If you have a question or problem, first try to have it solved by using the forum that can be found on the Oncourse page for this assignment. If you really cannot get it solved via the forum, you can come to the contact hours on Wednesdays between 15:00 and 16:00 (ask for Sayandip De). If the problem is very urgent, post your problem on the forum and request an appointment. Depending on the problem and your description of what you already tried it might be possible to make an appointment outside of the normal contact hours

Trick question answers

Trick question 1: You can find this out by either looking at the graphical representation of the architecture description or by inspecting the `architecture.xml` file. If we look at the connections of the LSU we will see that the source of input 1 is specified to be `rf.1`. This refers to output 1 of the register file.

Trick question 2: Besides the multiplications required for the actual filter (21) we also need to calculate the addresses of the input data (21), and the weights (21).

[24] We know what it is about and you can better use the space for your own work.