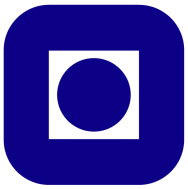


NTNU

TDT4295 - Computer Design Project

Project Report

Convolution team

Peter Aaser
Mattis Spieler Asp
Truls Fossum
Fredrik Haave
Øyvind Kjerland
Arnstein Kleven
Mathias Ose

October 26, 2015

Abstract

Contents

1 | Introduction

2 | Description & Methodology

2.1 Convolution

Some words on convolution. Background-ish

2.2 FPGA

The very heart of our computer is our custom made architecture implemented on our FPGA. In this section we will first describe the overall architecture of our convolution engine, and then we will drill down and examine the core modules. Modularity and extendibility is a core principle in our design, however in this section we will describe the processor as if only being able to do 3x3 convolutions.

2.2.1 Overall design

Convolution is a very regular task where data flows only forward. This means that our processor can be simplified, removing the need for a central control module. Instead each module simply operates under the assumption that all data inputs are correctly formatted and ordered and does its operations accordingly. In the figure we see this design principle

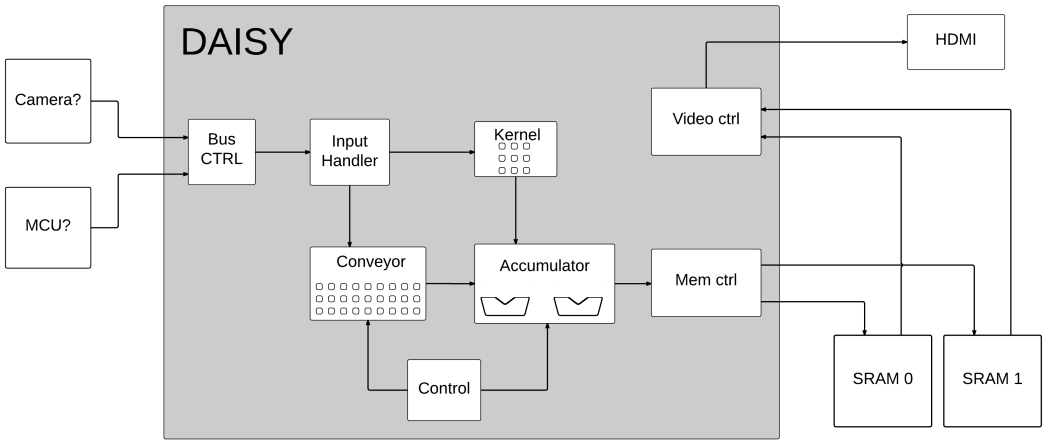


Figure 2.1: The data path for our processor, named daisy after the way it daisy chains control

reflected, no data ever flows backwards, only forwards. Breaking up the individual steps we get the typical flow of data.

1. Data from the bus is read by the bus controller. This controller serves as a clock domain crossover and is responsible for delivering data to the input handler in a clock domain crossing FIFO queue.

2. The input handler recieves data from the queue and does magic

3. Pixel data is held in a special buffer with rows representing the current sweep window.

4. The accumulator recieves kernel data from the kernel buffer and performs its mapping function on the kernel value and the pixel from the conveyor belt. When an accumulator has accumulated all its pixels it flushes its value, resetting the accumulator register and writing the old data to the memory control unit.

5. Accumulated pixels are reassembled in the mem ctrl unit and written to one of the two SRAM banks, allowing us to double buffer.

6. After being buffered in SRAM pixel data is read by the video ctrl module which outputs video to an HDMI cable, ensuring crisp image quality served in a modern fashion.

2.3 Data in

Fig:Sweeps shows the pattern in which data is being fed into the processor. A good analogy is how a squeegee (nal) is used to clean a window, by doing either horizontal or vertical sweeps. In fig:SweepFrontier we show the area of the image we are currently working on. Our workarea, or sweep window, is nine pixels wide and three deep, and for each row the convolutions that have the middle row as center pixels are calculated.

2.4 Convolver

This section encompasses the four modules forming the heart of the convolution engine as depicted in fig:DaisyView. In the order they are covered, they are:

the conveyor belt
: A buffer responsible for maintaining the three rows of the sweep window and to feed the accumulator with three reads each cycle, one read from each row every cycle. The three rows must be read in such a way that each accumulator may collect the nine subpixels it needs per convolution by reading each conveyor output at the correct time.

the kernel buffer
: The kernel buffer in the figure is an abstraction, in the implementation the kernel buffer resides much closer to the accumulators, but we separate it in our overview to show how each ALU in the accumulator reads kernel values.

the accumulator
: The accumulator is being fed data over three wires, and by choosing which wires to read from ensures that each accumulator works on the correct subpixels.

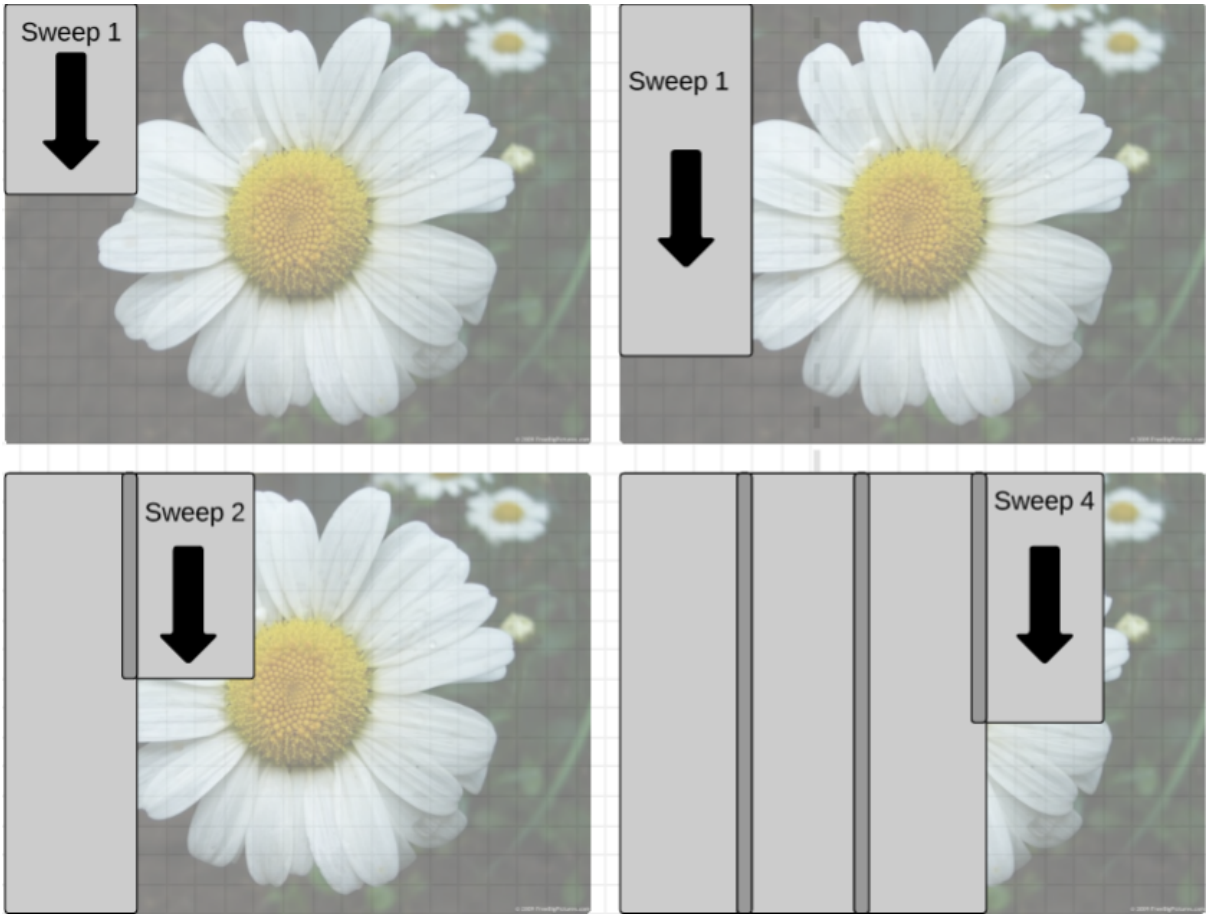


Figure 2.2: The sweep pattern used to collect data for convolution. The deep grey region indicates overlap, pixels which will be collected twice

the control unit

In order for the accumulators and pixel conveyor to be in sync a control unit sends control signals periodically, which are then propogated throughout the system.

Fig:DaisyView shows an overview of the four components in the heart of daisy. Lastly,

2.4.1 The accumulator

Although the accumulator unit is the last unit in the datapath of daisy we will cover it first since it helps motivate the designs further up in the chain. The accumulator actually consists of seven accumulators, hereby referred to as pixel accumulators. Each pixel accumulator corresponds to a pixel in the middle row. The leftmost and the rightmost pixel in the middle row has no accumulator associated with it because it lacks three pixels necessary for a full convolution, necessetating the slight overlap in our feed pattern. We will first focus on the leftmost accumulator starting at time T0 doing the following reads. It first reads three values from DATA_1 which corresponds to its southwestern subpixel at T0, its southern subpixel at T1 and its southeastern subpixel at T2. At time T3 it starts reading from DATA_2, reading its western, middle and eastern subpixels at respectively T3, T4 and T5. Finally it reads its northwestern, northern and northwestern subpixel at time T6 to T8 from DATA_3. What about the second leftmost accumulator then? By following the exact same read pattern, but waiting one cycle allows it to read

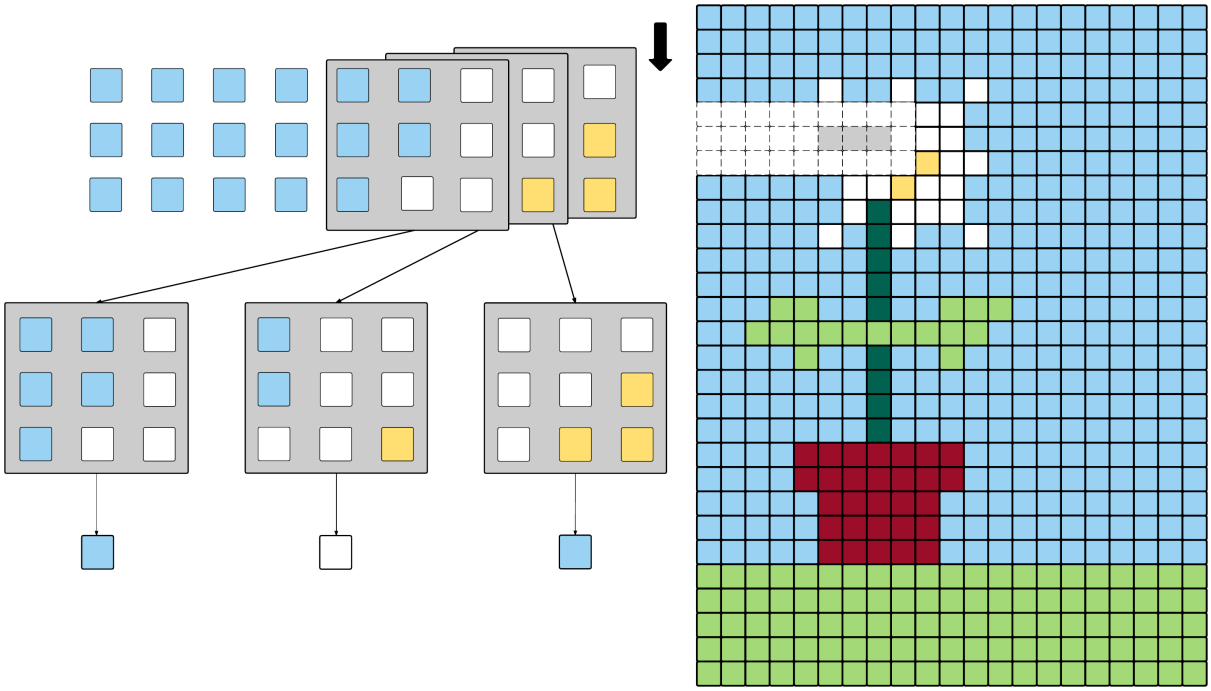


Figure 2.3: The image three greyed out pixels in the source image represent the three pixels which the regions in grey boxes help calculate. The window is three pixels deep and nine pixels wide

all its values in the same manner as the leftmost accumulator! The second accumulator reads DATA_1 at time T1, switches to DATA_2 at T4 and reads from DATA_3 from T6 to T9. In fact, every accumulator starts one cycle later than its left neighbour, meaning every accumulator has accumulated its necessary pixels at different times.

2.4.2 The conveyor belt

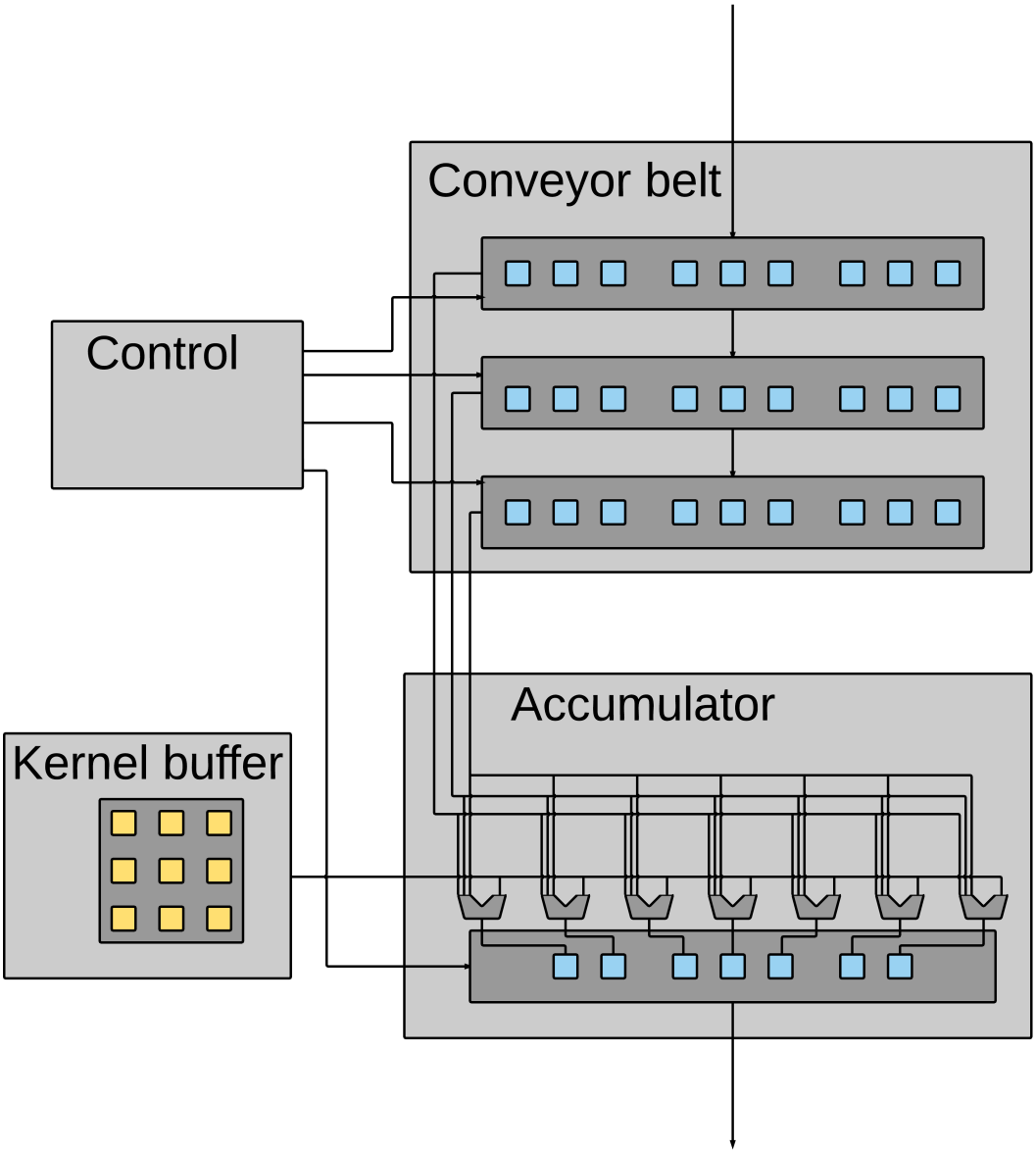
In fig:FeedTime we show the time each register is read, giving a clearer view of how the accumulators can obtain the pixels it needs by offsetting its read time. In context of an accumulator, as soon as the reads from one row goes out of range, the reads from the row above comes in range. In a conveyor working on 5 by 5 kernels each read would have to be spaced five registers rather than three. Of course, simply feeding data to the accumulator would leave us with a pretty uninteresting final picture. The other job of the conveyor is to feed new data onto the belt, and to propogate that data downward, giving rise to its name. In order to propagate data downward, whenever a register is read the register directly below it will also read, allowing data to transefer laterally. We have seen that both the accumulator and the conveyor works in a wave pattern, which has a very powerful implication. Since every accumulator and register will perform the same operation as its left neighbour offset by one cycle we can let each element be responsible for controlling the element to its right! This gives rise to the name Daisy, as we daisy chain the instructions rather than using a central control module. An analogy to the read signals is a key. At time T0 the control element gives the leftmost register the read key. The register does as it is told and reads its new value from the register above, and hands the key over to its neighbour. At T1 the neighbour recieves the key, and reads from the register above it and sends the key further on.

2.4.3 The controller

We have established how control flows throughout the system, but we still need a module to give out keys to the leftmost accumulators and registers. The controller is responsible for sending out the keys in the correct order and at the correct time. Timing is everything, if we misalign read signals and write signals we may end up in a situation where a register fails to read when the register above it writes. For example, if the controller sends each read signal one cycle too late each register will read when the register to the right side of the one directly above it, causing the data to be sheared as it moves laterally! Other than issuing read and write keys to the registers, the controller also issues a flush key for the accumulators which tells accumulators to drive DATA_OUT with its contents, and to reset.

2.4.4 The kernel buffer

In our schematic we present the kernel buffer as a separate submodule. However, having already introduced how instructions are daisy chained it makes sense to do the same with kernels since seven different kernel values are in use at all times. Thus the kernel values are kept in a shift register queue living as close to the accumulators as possible, needing only two registers to hold the currently unused kernel values. The responsibilities of the kernel unit is thus to collect the first data values into the kernel buffer chain and to buffer the two kernel values which are not currently in use.


Conveyor belt
Control
Accumulator
Kernel buffer

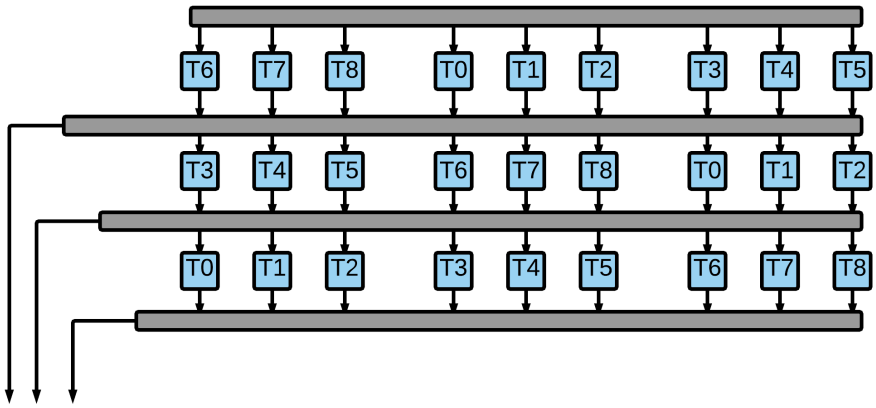


Figure 2.5: The timestep in which each register is read.

3 Results

3.1 Discussion

4 | Evaluation of assigment

5 | Conclusion

References

[1] Edgar Xavier Ample. Foo. Technical report, Foxford University, 3000.

[2] Mathias Ose. ma.thiaso.se. `http://ma.thiaso.se/`, 2014.