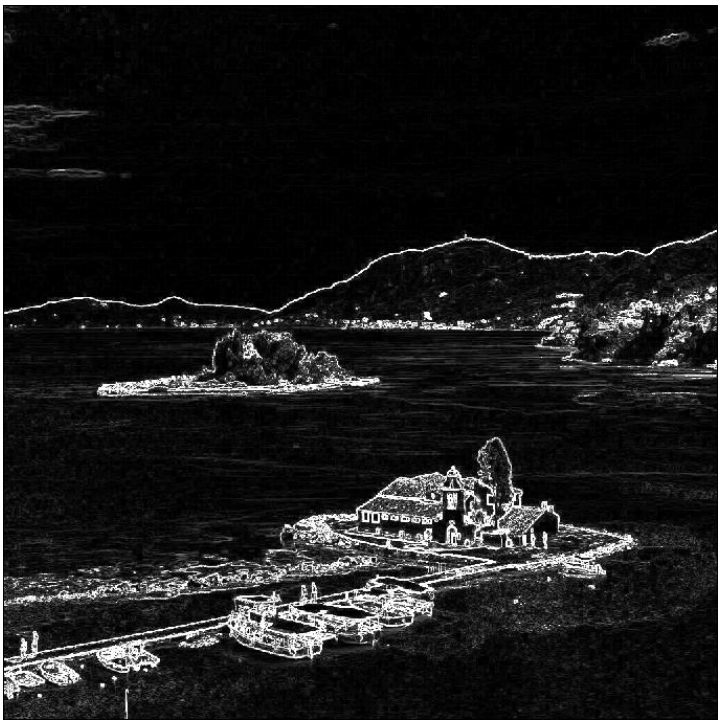
Παναγιώτης Κρεμμύδας AEM:1435
Χρητος Κοντομητρος AEM:1798
6/19/2017

Final Project: Hardware Optimized Sobel Filter

CE435: Embedded Systems

Electrical& Computer Engineering

University of Thessaly



Contents

Introduction:

The goal is to successfully offlead a computationally demanding process (Sobel filter) from the CPU to an external module implemented in the FPGA fabric.

Software running on a processor, no matter how well optimized it is, is slower than a hardware accelerator implementing the same functionality. Translating the functionality of a computationally intensive function to a HW Description language using all the available advantages HW parallelism and flexibility has to offer can be a really time consuming and often repetitive feat.

Based on a SW version of a Sobel filter written in C, using the Vivado HLS (High Level Synthesis) tool a hardware accelerator is build reducing the function's execution time on an ARM processor to the minimum using various optimizations through code directives. The optimizations are tested individually for their additional performance gains. Various HLS directives and code restructuring is used to increase performance on the output Sobel accelerator IP .

The following methods are tested for their effectiveness: Loop unrolling and pipeling, avoiding conditionals in the source code (if-then-else) and replacing short conditionals, Determining which operations are in the critical path and try to circumvent their effect by running them in parallel or even replacing them with cheaper (even approximate!) operations. Re-writing source code to improve bandwidth and solve reduntant memory accesses.

1: Default Implemetation

Code reduced and shaped to its bare minimum functionallity as given (found in a generic Sobel filter implementation). The communication inteface used follows theAXI4 protocol supported by ARM processors. Data burst functionallity to greatly increase bandwidth is not found compatible automatically on the code supplied..

Experiment/Resulting implementation 1.50320 sec

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	458	963
FIFO	-	-	-	-
Instance	4	12	4113	4845
Memory	0	-	128	10
Multiplexer	-	-	-	413
Register	-	-	575	-
Total	4	12	5274	6231
Available	280	220	106400	53200
Utilization (%)	1	5	4	11

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
334236925	334236925	334236926	334236926	none

2: Convolution Algorithm Decomposition

Convolution algorithm rewritten from adhering to a generic NxN matrix implemntation to the prespesified requisites of the current project, targeted to serve the supplied vertical and horizontal kernels passing through a 1024x1024 8bit pixel array

Experiment/Resulting implementation 0.35094 sec

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	2	-	-
Expression	-	-	458	1653
FIFO	-	-	-	-
Instance	4	0	3452	4393
Memory	-	-	-	-
Multiplexer	-	-	-	501
Register	-	-	888	-
Total	4	2	4798	6547
Available	280	220	106400	53200
Utilization (%)	1	~0	4	12

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
69982473	69982473	69982474	69982474	none

(As expected by retargeting the source code to the specified parameters, computation time is greatly reduced.)

3: Arbitrary Precision

Registers used now match the exact size necessary for the given implementation, slightly reducing overall board usage and rooting path lenth.

Experiment/Resulting implementation 0.34625 sec

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	2	-	-
Expression	-	-	420	1442
FIFO	-	-	-	-
Instance	4	0	3452	4393
Memory	-	-	-	-
Multiplexer	-	-	-	478
Register	-	-	762	-
Total	4	2	4634	6313
Available	280	220	106400	53200
Utilization (%)	1	~0	4	11

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
67893505	67893505	67893506	67893506	none

4: Removing Conditional

Due to the tool’s inabillity to detect simple multiplexor functionallity of an “IF/ELSE” statement, if such a thing is required, an “?/ :” should be used instead. For the cropping function of the filter, to avoid delay on a register value retention implementation used for “IF/ELSE” statements the above operation is also making the source code synthesis fit for data bursting on the output AXI4 stream

Experiment/Resulting implementation 0.32024 sec

```
INFO: [XFORM 203-811] Inferring bus burst write of length 1022 on port
'OUTPUT_BUNDLE' (IP_2.2_removing_conditional.c:38:5).
```

As mentioned, not being in an IF/ELSE conditional also results into the above positibe effect. Without directly inferring a burst operation by using the C function memcopy the HLS compiler detects the compatibility of such operation automatically. (bursting data at the end of every row loop)

On the other hand, the HLS tool is still unable to detect a patern in the input DDR memory access and does not implement it utilizing a burst operation.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	2	-	-
Expression	-	-	420	1448
FIFO	-	-	-	-
Instance	4	0	3452	4393
Memory	-	-	-	-
Multiplexer	-	-	-	439
Register	-	-	764	-
Total	4	2	4636	6280
Available	280	220	106400	53200
Utilization (%)	1	~0	4	11

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
62676195	62676195	62676196	62676196	none

(The output operation is synthesized into a multiplexor used with output masking.)

5: Forced Input data Burst into 3 row buffer

In order to force the HLS tool to use the burst function of the AXI4 protocol for all the transactions with the DDR memory, the memcopy command has to be implemented together with associated input and output buffers. The input buffer stores exactly the amount of data necessary for the Sobel parsing of a pixel row (3 rows storage) and with every loop iteration one of its rows has its data renewed with a data burst rotationally (row_number%3)

Experiment/Resulting implementation : 0.24266 sec

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	2	-	-
Expression	-	-	420	1095
FIFO	-	-	-	-
Instance	4	0	3842	4600
Memory	3	-	0	0
Multiplexer	-	-	-	731
Register	-	-	676	-
Total	7	2	4938	6426
Available	280	220	106400	53200
Utilization (%)	2	~0	4	12

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
51220622	51220622	51220623	51220623	none

6: Pipelining row iterations

Having costly and slow computational processes inside the main function loop can be further optimized by braking down the execution into independent components that evenly share processing so that every new loop can be run concurently with its previous iterations

Experiment/Resulting implementation 0.19170 sec

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	4	-	-
Expression	-	-	840	2255
FIFO	-	-	-	-
Instance	4	0	4304	4861
Memory	10	-	0	0
Multiplexer	-	-	-	933
Register	-	-	1382	128
Total	14	4	6526	8177
Available	280	220	106400	53200
Utilization (%)	5	1	6	15

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
6357877	6357877	6357878	6357878	none

(Pipeline Iteration Interval: 8 cycles.)

Loop Forwarding: In every iteration there are altered arguments which issue pipeline flashing , there is no improvement in execution time using Loop Forwarding.

Loop unrolling: As long as there no bandwidth bottleneck and there is free space on the FPGA it can be utillized to divide the computation workload into indentical processing logic that is

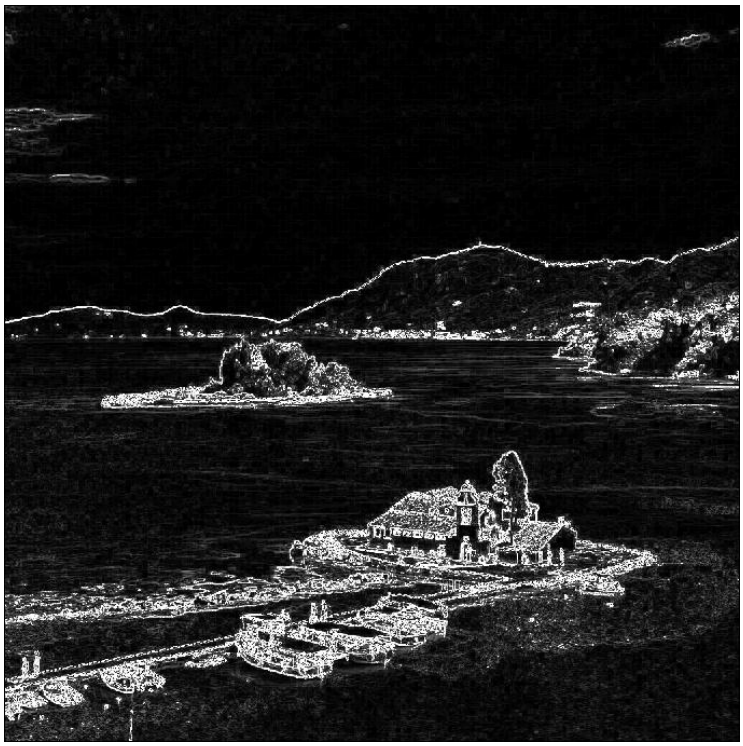
executing parallely. From a series of tests run, loop unrolling was ineffective in every way to decrease computation time due to the already miniscule computational delay produced by the pipelined main computational workload (bandwidth bottleneck).

7: Approximate Solution:

Implementation using an approximate solution to the sobel filter output. Absolute values for the vertical and horizontal convolution iterations instead of their square values under a square root sum greatly increases computation speed while producing a highly usable output for any possible image recognition application on it.

As noticed above, dastrically improving the computation speed does not yield a considerable performance gain due to the memory bound communication delay (bandwidth bottleneck)

Experiment/Resulting implementation 0.177332 sec (PSNR=57.732288)



Accurate Solution

Approximate Solution

Utilization Estimates

Summary

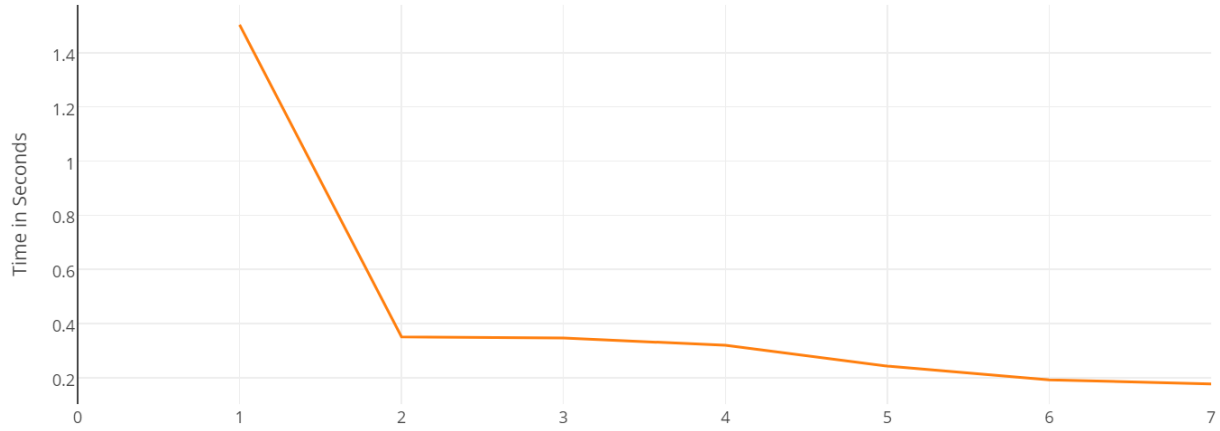
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	568
FIFO	-	-	-	-
Instance	4	-	1598	1775
Memory	3	-	0	0
Multiplexer	-	-	-	581
Register	-	-	508	-
Total	7	0	2106	2924
Available	280	220	106400	53200
Utilization (%)	2	0	1	5

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
6310876	6310876	6310877	6310877	none

8: Comparison Graph



1: Default implementation

2: Convolution algorithm Decomposition:

3: Arbitrary Precision

4: Removing Conditional

5:Forced Input data Burst

6: Pipelining

7: Approximate Solution

9: ARM Software execution:

Software performance of the Sobel Filter source code measured on a single-core ARM Cortex-A9 general purpose CPU implemented onboard the Xilinx Zynq-7000.

Accurate Solution (-O0):

Experiment/Resulting implementation 1.9019 sec

We tested the performance of the filter used on the arm processor using the -O0 compile option observing the given performance wihout any optimizations.

Accurate Solution (-O3):

Experiment/Resulting implementation 0.1604 sec

Using the -O3 optimization we observed the best combination of performance and precision in terms of software and also better performance than any of our HLS solutions.

Approximate Solution (-O3):

Experiment/Resulting implementation 0.1526 sec

Using the abs function instead of sqrt function we observed the best performance partly sacrificing some accuracy.

Conclusions:

Currently, a naive hardware implementation can hardly put up against the processing power of a modern consumer general purpose CPU and the various compiler improvements over the source code (gcc -O3). A targeted and well planned approach on the other hand can greatly decrease the execution time and energy cost of a computation subroutine such as the application of the Sobel filter on an given image.

While the whole process of creating such an IP is greatly improved in terms of time consumption, there is still lots of underlying hardware knowledge that is necessary in order to take advantage everything that the PL fabric has to offer.