Implementation of the Keccak Hash Function in FPGA Devices

Joachim Strömbergson – InformAsic AB[1]

Abstract
This paper describes some results and notes from implementing the reference design of the SHA-3 hash function candidate Keccak in FPGA devices

Introduction

The Keccak hash function [1] is one of more than fifty candidates [5] accepted by NIST for the SHA-3 hash function competition [4]. One important aspect to consider when evaluating the candidates is how well, that is how efficiently, implementations of will work in different types of architectures such as micro-controllers, multi-core processors, ASICs and FPGAs.

There are currently several independent attempts at evaluating candidates on aspects such as source and object code size/complexity [15], [16], software performance, [7], [17] and hardware performance [8]. The SHA-3 Zoo web page [6] contains a good list of evaluation activities and results related to the competition.

The authors of Keccak function has developed two different hardware reference implementations [3] which has been made available on the Keccak web page [2]. The Keccak authors have implemented the designs in a 130 nm CMOS-process from ST. What is not clear however is how well Keccak would work when implemented in a more register-rich and routing constrained environment such as an SRAM based FPGA.

In this paper we describe a simple experiment at implementing the Keccak reference implementations in FPGAs.

Implementation setup

The Keccak hardware reference designs consists of two rather different implementations. the *high_speed_core* design is, as the name suggest, a self contained high performance design. The *low_area_copro* on the other hand, is a small design where the state memory is external from the rest of the design. This allows the Keccak coprocessor to share memory with other design entities within a device.

We choose to try to implement the Keccak reference designs on four different families of SRAM based FPGAs. The FPGA families used are:

- Altera Cyclone III [9].
- Altera Stratix III [10]
- Xilinx Spartan-3A [12]
- Xilinx Virtex-5 [13]

The Altera Cyclone III and the Xilinx Spartan-3A devices are aimed at consumer and cost constrained embedded systems. The Altera Stratix III and Xilinx Virtex-5 devices on the other hand are high performance devices suitable for network equipment etc.

The tool we used to implement the designs on the Altera FPGA devices is the Altera Quartus II Web Edition, version 8.1 [11]. The tool we used to implement the designs on the Xilinx FPGA devices is the Xilinx ISE WebPACK, version 10.1.03 [14].

To achieve as good performance as possible we chose for the Xilinx devices the biggest device supported by the tool for the device families. For the Altera devices we chose to allow the tool itself to select the device within each family.

1 The author can be contacted at joachim at strombergson.com

Results

Table 1 shows the implementation results for the *high_speed_core* design.

Device	Logic resources	Registers	Clock frequency	Throughput
Cyclone III EP3C10 F256C2	5,842 (10,320) LEs	2,670 (10,320)	123 MHz	7,0 Gbit/s
Stratix III EP3SE50 F484C2	4,550 (38,000) ALUTs	2,670 (38,000)	176 MHz	10,0 Gbit/s
Spartan-3A XC3S1400AN FGG676-5	3,393 (11,264) slices	2780 (22,528)	85 MHz	4,8 Gbit/s
Virtex-5 XC5VLX50 FF324-3	1,483 (7,200) slices	2,669 (28,800)	118 MHz	6,7 Gbit/s

Table 1: Results for the high_speed_design implementation.

The throughput has been calculated based on the published results [8] for the 130 nm ASIC implementation of the high_speed_core design that achieves 28,4 Gbit/s at 500 MHz. This means that we assume that the same number of rounds and cycles are used, and that the same bits/round are produced.

The Xilinx tools replicates registers as part of the mapping and routing which explains the different number of registers, especially for the Spartan-3A implementation.

Table 2 shows the implementation results for the *low_area_copro* design.

Device	Logic resources	Registers	Clock frequency	Throughput
Cyclone III EP3C5 F256C26	1,769 (5,136) LEs Modified: 1,750	242 (5,136)	85 MHz	22,1 Mbit/s
Stratix III EP3SE50 F484C2	1,026 (38,000) ALUTs	242 (38,000)	133 MHz	34,57 Mbit/s
Spartan-3A	Failed	Failed	Failed	Failed
Virtex-5	Failed	Failed	Failed	Failed

Table 2: Results for the low_area_copro implementation.

The throughput has been calculated based on the calculations done for the 130 nm ASIC implementation with external memory [8] that achieves 26 Mbit/s at 100 MHz.

Design Notes

The VHDL code for the Keccak reference designs is written in somewhat unusual style which makes it harder to implement the design and both tools struggled with the code. There are also some bugs in the code. Some of the problems encountered are described here.

The *zero_state* signal in *keccak.vhd* in *the high_speed_core* design is missing from the sensitivity list of the *p_main* process causing latches to be inferred by both tools.

The design files for the *low_area_copro* design includes the *ieee.std_logic_textio* library, which is not a synthesizable library. Since no functions declared in the library are actually used in the RTL design, it can safely be removed. ISE recognizes that the library is not used and will just issue a warning. Quartus II however is more picky, and building the design in Quartus II with the library declaration fails.

The *low_area_copro* design uses a sub-typed integer for the *nr_rounds* port in *pe.vhd*. This is not a recommended port type and not well supported by tool vendors. Both tools ended up implementing this port as a five bit port, leaving 13 states uncovered. The result is that latches are inferred for the *iota* signal. When we added a final *else*-clause to the *iota* assignment statement, the design decreased in size from 1769 LEs to 1750 LEs. We recommend that the RTL is changed to use std_logic-type ports.

The FSM in *low_area_copro* design use a *mod* operation with an operator that is not an even power of two. VHDL does only support this operator usage for synthesis when both operands are constants. It is therefore up to tools vendors to decide if they want to support this usage of the operator for synthesis. Implementation of the *low_area_copro* design in the tool from Xilinx fails due to the usage of the mod operator:

```
ERROR:Xst:1775 - Unsupported modulo value 5 found in expression at
line 162. The modulo should be a power of 2.
```

The tool from Altera seems to support the operator usage by instantiating (allocating) a number of macros, but it is unclear if the result is functionally correct. For better portability we recommend that the specific mod operator is replaced with a function/component that conforms to VHDL synthesis rules.

There are a number of other, smaller details such as signals being assigned, but never used. We recommend that all warnings and notes generated by implementation tools are read through and handled.

Conclusions

The *high_speed_design* works well in all four FPGAs tested and it seems likely to assume that high performance is quite easy to achieve in all FPGA devices. The *low_power_copro* is a really tiny design, requiring very few resources, and is therefore possible to implement even in such a small device such as the Cyclone III C5. The VHDL code however should probably be reworked a bit to ensure that the tools will generate a proper and clean implementation.

The software reference implementations are equipped with a common interface. This makes it easier to test and evaluate the candidates. The implementation experiment described in this document leads us to suggest that evaluation of candidate hardware reference implementations would be easier if a common, well defined hardware interface existed. We propose that such an interface is to be developed.

Thanks to Jonas Thörnblad, Karin Tidlund and Fredrik Ljunggren for reviewing this paper.

References

[1] G. Bertoni, J. Daemen, M. Peters, G. Van Assche. *Keccak specifications*.
http://keccak.noekeon.org/Keccak-specifications.pdf

[2] G. Bertoni, J. Daemen, M. Peters, G. Van Assche. *The Keccak sponge function family*.
Web page. December 2008.
http://keccak.noekeon.org/

[3] G. Bertoni, J. Daemen, M. Peters, G. Van Assche. *Keccak Hardware implementation in VHDL*.
File archive. December 2008.
http://keccak.noekeon.org/KeccakVHDL-1.0.zip

[4] National Institute of Standards and Technology (NIST). *CRYPTOGRAPHIC HASH PROJECT*.
Web page. December 2008.
http://csrc.nist.gov/groups/ST/hash/index.html

[5] National Institute of Standards and Technology (NIST). *FIRST ROUND CANDIDATES*. Web page.

December 2008.
http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html

[6] ECRYPT. *The SHA-3 Zoo*. Web page. December 2008.
http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

[7] ECRYPT. *eBASH: ECRYPT Benchmarking of All Submitted Hashes*. Web page. December 2008.
http://bench.cr.yp.to/ebash.html

[8] ECRYPT. *SHA-3 Hardware Implementations*. Web page. December 2008
http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations

[9] Altera corporation. *Cyclone III FPGAs: Low Cost and Unlimited Possibilities*. Web page. December 2008.
http://www.altera.com/products/devices/cyclone3/cy3-index.jsp

[10] Altera corporation. *Stratix III FPGAs: Lowest Power, Highest Performance 65-nm FPGAs*. Web page. December 2008.
http://www.altera.com/products/devices/stratix-fpgas/stratix-iii/st3-index.jsp

[11] Altera corporation. *Quartus II Web Edition Software*. Web page. December 2008.
http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html

[12] Xilinx corporation. *Extended Spartan-3A FPGAs*. Web page. December 2008.
http://www.xilinx.com/products/spartan3a/

[13] Xilinx corporation. *Virtex-5 Multi-Platform FPGAs*. Web page. December 2008.
http://www.xilinx.com/products/virtex5/

[14] Xilinx corporation. *ISE WebPACK Software*. Web page. December 2008.
http://www.xilinx.com/ise/logic_design_prod/webpack.htm

[15] S. O'Neil. *Stripped down reference C source size*. Web page. December 2008.
http://www.sha3.org/source_size.html

[16] S. O'Neil. *Reference code without debug/intermediate print-outs*. Web page. December 2008.
http://www.sha3.org/binary_size.html

[17] N. Ferguson et al. *Engineering comparison of SHA-3 candidates*. Web page. December 2008.
http://www.skein-hash.info/sha3-engineering