,,,

Jacob Herbst (mwr148), Jonas Flach Jensen ()

Crypto in SME for FPGA

Advisor: Kenneth Skovhede

February 25, 2021

Contents

1 Introduction

2 Background

2.1 Field Programmable Gate Arrays

Why use FPGA's? To get a speed increase by moving frequently used functions and calculations from the CPU to a specialized chip (the FPGA). This will result less time "wasted" on the CPU, and the specialized chip will ideally also be able to do the computations faster than a generic CPU.

2.2 Synchronous Message Exchange

Synchronous Message Exchange (SME) is a programming model to enable FPGA development using high-level languages. SME is based on Communicating Sequential Processes (CSP) and at its core constructs as a strict subset of said process calculi, making use of the elements which has proven useful in hardware design[**?**]. Using the following concepts from the CSP model, SME can be derived:

- A program consists of a set of named processes.
- Each process runs on its own processor with no form of sharing with other processes.
- Concurrent processes can communicate using message passing with a `send(!)` and a `receive(?)` command. This message passing is Blocking and Non-buffered.

Without going into too much detail about the syntax and semantics of CSP[1], we can use the following syntax to describe a program.

~x PROCESS~, which assigns the PROCESS to the name x.

`x.in` is a compound name similar to an object field `in` of the object. Notice this abstraction makes the connection to SME and its C# implementation more obvious.

`x.out!y.in` This is the sending message passing. It will send y.in to x.out.

`x.out?y.in`. This is the receiving message passing. It will read x.out to y.in.

`x || y` will denote two concurrent processes, x and y.

Later we will show this can easily show abstractions of algorithms when using SME. SME has a similar notion of processes. There exist two types of SME processes, `simple process` and a `simulation process`. Of these, the simple process corresponds to a process in CSP as described above. Each simple process in SME will only share communication channels and constants with the other processes. For the communications channels, SME extends the concepts from CSP by using buses. Instead of using explicit naming for sources and destinations, each process will consist of a set of input and output busses that it can read and write to, respectively. Furthermore, these buses use broadcasting as means of synchronization instead of the blocking non-buffered approach. The broadcasting happens every clock-cycle on the internal clock. A bus is essentially just a collection of fields that can be read and written to depending on the process's access, merely a data transfer object. Here the syntax described above comes in hand. `message.text` would thus be the text field of the bus `message`. Corollary, we could define a very minimal process as such `[messageIn.text?messageOut.text]`, which would read the text field from the `messageIn` input bus and write it to the `messageOut` output bus. From these abstractions, one might be able to see how this effortlessly coincides with the hardware model.

2.3 A crypto library

Cryptografic functions are used by developers of across most branches, whether it'll be communicating securely over a network, or hashing programs to do version control. So there is a motive for having a crypto library for FPGA's. Infact such a processor has been made before. IBM created their own "IBM 4758 Secure Coprocessor"(ref. https://web.archive.org/web/

20170808032012/http://www.research.ibm.com/people/s/sailer/publications/2001/ibm4758.pdf).% Another point is modern Hardware security modules (HSM) which also does this. However the problem with the exsisting solutions is that many of them require setting up a royalty-based licensing deal, which makes it difficult to use for experimental development, small projects, and in reseach and academics. So we set out to create an opensource crypto library.

The crypto library consists of an implementation of various cryptografic functions, such as AES and SHA256. It should also have an API allowing users to utilize these functions in their projects, as they would with any other library. These implementations should also be optimized in terms of speed so that they are competitive with the exsisting software solutions. %Creating a cypto library for FPGA's ...

2.3.1 Hashing

Hashing is a mathematical concept refering to using a hash function to map some data of arbitrary size to a value of a fixed size. Cryptografic hash functions is a subset of all hash functions. The reason for this is that for a hash function to be a cryptografic hash function it needs to uphold several properties to ensure it is secure, such as ensuring that it is hard to find collisions. Computers also have limited space in memory which limits the implementaion of hash functions. Lastly and most importantly, computers can't do randomness. If a hash function can be implemented in with a limited input space, is pseudo random, and upholds certain properties it can be categorized as a "Cryptografic Hash Function". One such example is the outdated MD5 algorithm.

2.3.2 MD5

Merkle-Damgård construction

The Message-Digest algorithm MD5 is a reasonably simple one-way hashing function that produces a 128-bit digest specified in 1992 in RFC 1321[]. MD5 uses a Merkle-Damgård construction. The MD5 algorithm work by partition the input message into blocks of 512 bits. It is done by always padding the message with a single set bit followed by a series of 0's until the message = 448 mod 512. That is, even when the original message has a length of 448 mod 512, a 1' followed by 511 bits of 0's. Next, a 64-bit representation of the message length mod 2^{64} is appended to the padded message. The digest will be calculated in a 32-bit 4-word buffer (A, B, C, D), with the initial value:

`A: 0x67542301`
`B: 0xefcdab89`
`C: 0x98badcfe`
`D: 0x10325476`

and we use the following functions corresponding to each of the four rounds:

F(X, Y, Z) = XY $\vee$ $\neg$ X Z
G(X, Y, Z) = XZ $\vee$ Y $\neg$ Z
H(X, Y, Z) = X $\oplus$ Y $\oplus$ Z
I(X, Y, Z) = Y $\oplus$ (X $\vee$ $\neg$ Z)

These are defined as such to in "bitwise parallel" produce independent and unbiased bits in each of the rounds.

Process each 16-word block (512 bits) by copying it into a buffer X, save the current digest buffer and perform the following rounds: For each round a function [abcd k s i] denoting `a = b + (( a + round(b,c,d) + X[k] + T[i]) «< s)`, where round denotes the function corresponding to that round.

round 1 :: F [ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
ABCD 4 7 5
[DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
ABCD 8 7 9
[DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
ABCD 12 7 13
[DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]
Round 2 :: G
ABCD 1 5 17
[DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
ABCD 5 5 21

[DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
ABCD 9 5 25
[DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
ABCD 13 5 29
[DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]
Round 3 :: H
ABCD 5 4 33
[DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
ABCD 1 4 37
[DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
ABCD 13 4 41
[DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
ABCD 9 4 45
[DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]
Round 4 :: I
ABCD 0 6 49
[DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
ABCD 12 6 53
[DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
ABCD 8 6 57
[DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
ABCD 4 6 61
[DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]
Next, increment each of the variables by its starting value.

The Digest will now be (A, B, C, D) in LE format.

It is worth noting that MD5 is not a very good hashing algorithm for cryptography, as collision attacks exist, but still show use for data integrity purposes and such.

3 Implementation

3.1 Approaches

3.1.1 naive

3.1.2 pipelined

3.2 MD5

3.2.1 naive

3.2.2 pipeline 1

4 Benchmarks

5 Discussion

6 Conclusion

References

[1] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666677, August 1978.