University of Victoria

ECE 466

Project Report

2019 Summer Semester

Diffie-Hellman Key Exchange

Multiplier Hardware-Software Co-simulation

Matthew Wegener

V00826848

mwegener@uvic.ca

Submitted to:

Dr. Daler Rakhmatov

Table of Contents

[Introduction 3](#_Toc15548685)

[Project Tasks [1] 3](#_Toc15548686)

[Hardware-Software Handshaking Protocol 4](#_Toc15548687)

[Hardware Handshaking 4](#_Toc15548688)

[Software Handshaking 4](#_Toc15548689)

[Hardware Multiplier 4](#_Toc15548690)

[Structural Description 4](#_Toc15548691)

[Controller Description 4](#_Toc15548692)

[Results 4](#_Toc15548693)

[Simple FSM implementation 4](#_Toc15548694)

[Structural implementation 4](#_Toc15548695)

[Conclusion 4](#_Toc15548696)

[Recommendations 4](#_Toc15548697)

[References 4](#_Toc15548698)

List of Figures

# List of Tables

**No table of figures entries found.**

# Introduction

The project uses a Diffie-Hellman key exchange protocol, originally written in C and distributed as a part of the NetBench benchmark suite [1]. The purpose of the project is to use SystemC and model one of its computationally intensive functions, NN\_DigitMult, as a hardware module [1]. On the course website the following 7 files have been made available to assist in this task: Makefile, dhdemo.cpp, dh\_sw.h, dh\_sw.cpp, dh\_hw\_mult.h, dh\_hw\_mult.cpp, digit.h [1]. With the provided files, they can be compiled using the Makefile and the SystemC executable main.x can be run. The output looks like this:

\*\*\*Agreed Key: 09 2a f1 41 e2 93 61 d5

\*\*\*Agreed Key: 64 30 94 c5 da d2 f6 da 49 6d 67 f1 16 55 b3 ea ee a2 c0 30 2b b5 4f 05 9e a4 58 ac 97 3b b9 a0 25 b7 56 fe 82 73 bb 22 d4 31 36 60 7f 41 e9 47 97 b9 5e 27 99 3e 73 f0 28 da b5 25 da e4 61 84

[1]

In the project files there are two control signals exchanged between software and hardware: enable and done. Currently, software and hardware are synchronized using only enable and explicit timing delays. The delays used are 100 ns for computation and 10 ns for communication. The enable signal is generated by software to enable hardware multiplication, and the done signal is generated by hardware to indicate that multiplication has been completed.

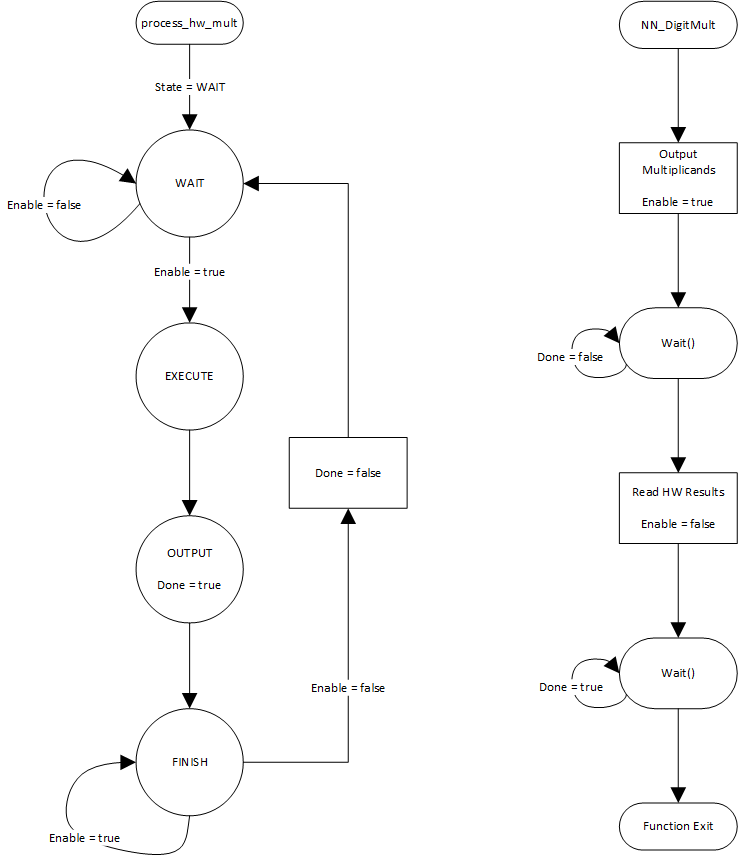
# Project Tasks [1]

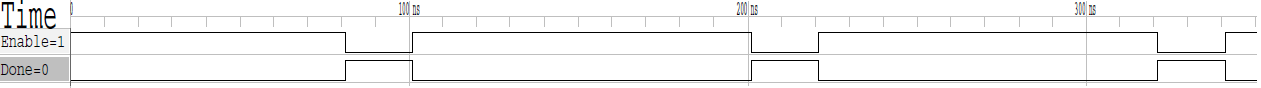
The first task is to replace timed waits with the enable-done handshaking protocol in both hardware software. For handshaking to occur, first the hardware should wait for enable signal to be asserted. Once enable has been asserted, the hardware should execute the multiplication followed with outputting the result and asserting the done signal. The hardware should deassert done only if enable has been deasserted. To implement this a clocked input needs to be added in the hardware and make it a CTHREAD; and in the hardware module an FSM with 4 states:

1. *WAIT:* Wait for the enable signal to be asserted.
2. *EXECUTE:* Multiply the two inputs (using the multiplication code as is).
3. *OUTPUT:* Write to the module's output ports, assert the done signal.
4. *FINISH:* Check if enable is deasserted; if so, deassert done.

The second task is to design the datapath and controller of the hardware multiplier. First, extract the multiplication code inside the EXECUTE state and convert it to the structural description. Then, split the EXECUTE state into as many states as necessary to control the datapath. This makes the datapath controller becomes "embedded" into the handshaking FSM. In the final design implantation, it should produce the same output as the original code without any timed waits.

# Hardware-Software Handshaking Protocol



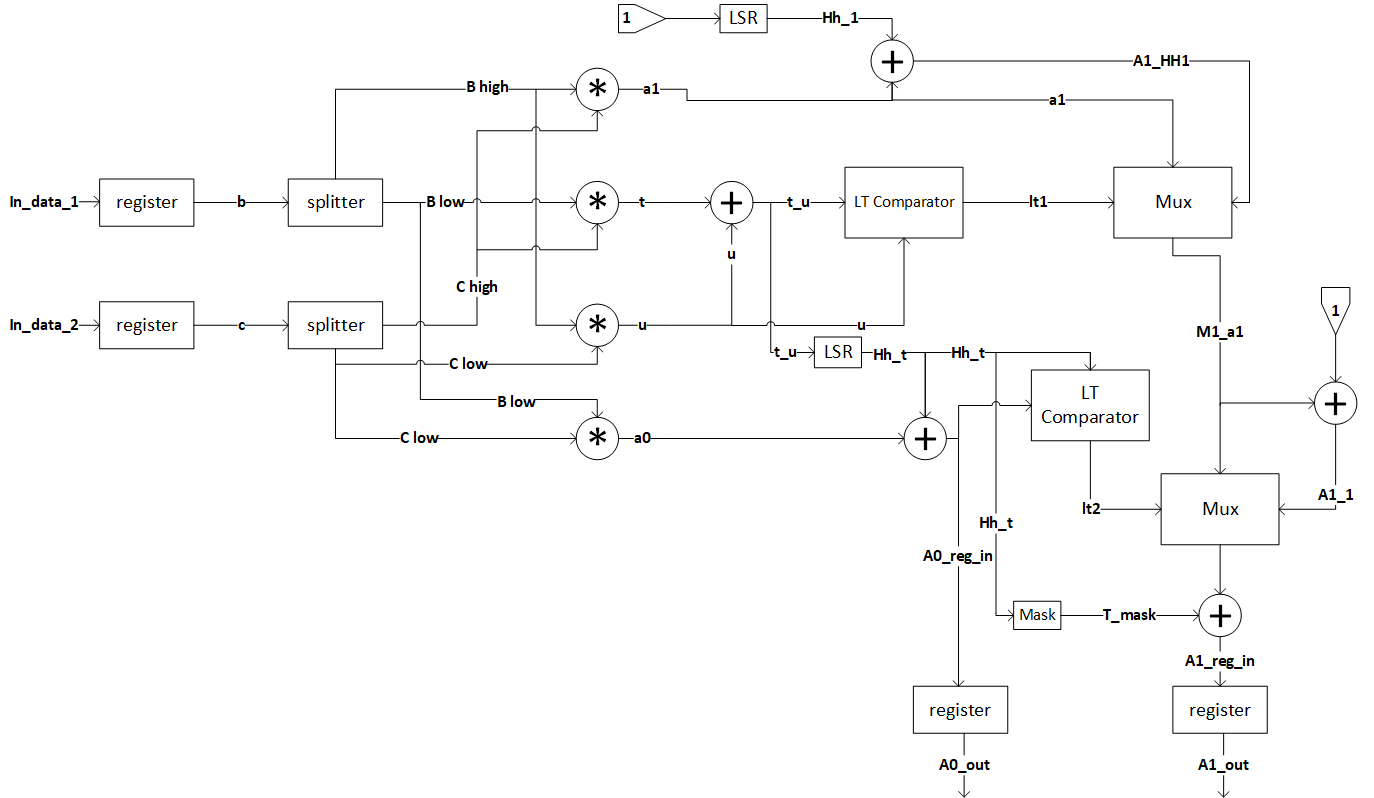


## Hardware Handshaking

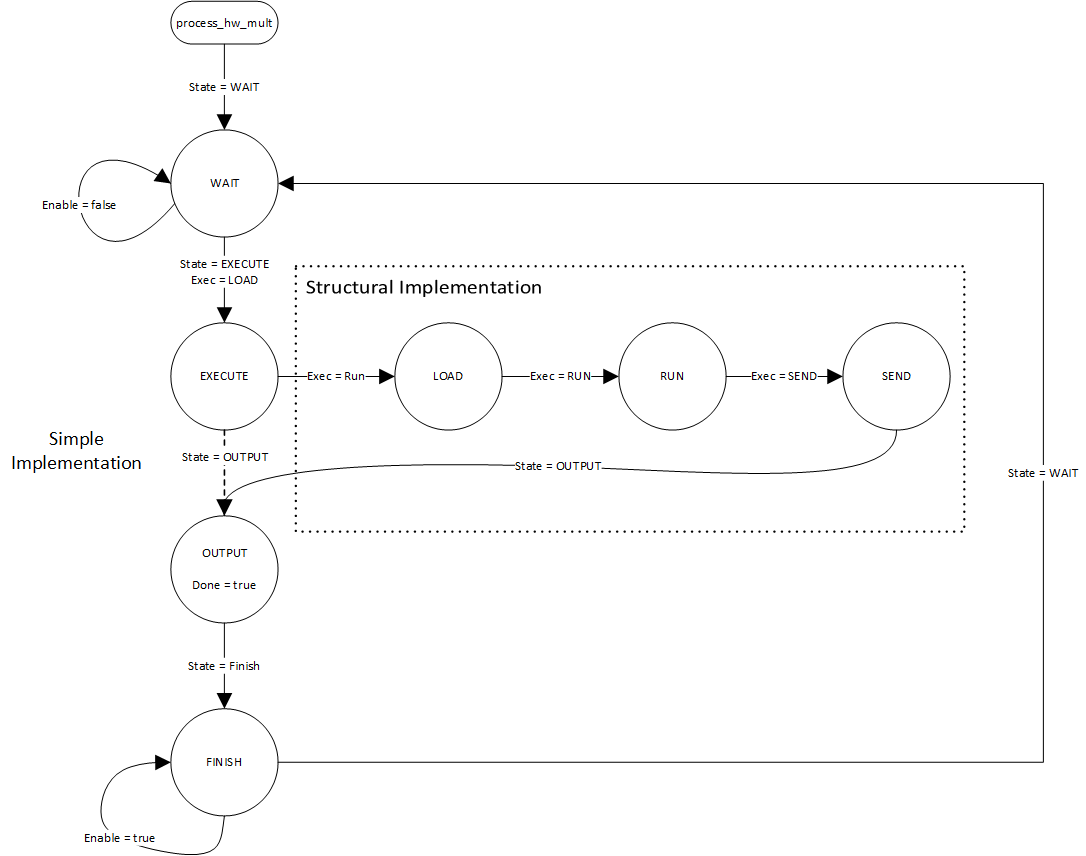
## Software Handshaking

# Hardware Multiplier

## Structural Description



## Controller Description



# Results

## Simple FSM implementation

## Structural implementation

# Conclusion

# Recommendations

# References

|  |  |
| --- | --- |
| [1] | D. Rakhmatov, "Project," [Online]. Available: https://www.ece.uvic.ca/~daler/courses/ece466/. [Accessed 1 August 2019]. |