

Department of Electrical and Computer Engineering

University of Victoria

ELEC 466

**Project Report**

Course: Elec 466

Submitted to: Daler Rakhmatov

Report Submitted on: August 4, 2015

Name: **Daniel Apperloo (V00759448)**

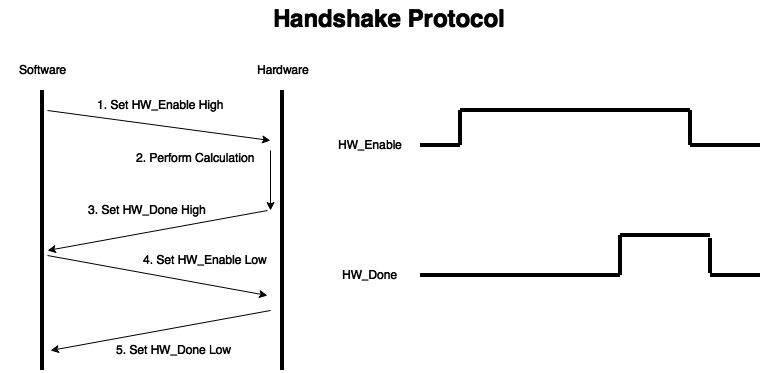
1. Problem Description

The objective of this project is to model a hardware assisted implementation of the Diffie-Hellman key exchange. In particular, the computationally heavy function NN\_DigitMult was simulated as a hardware module. To accomplish this task, three components had to be designed. First, a handshaking protocol between the Software algorithm and the Hardware module had to be implemented. Then, the function was implemented as a Hardware datapath with a controller unit. The resulting solution was then simulated and the results compared to the pure software solution.

The solution only had two restrictions. The SW handshaking component could not have any clock signals. Lastly, there must not be any timed waits in the final code.

2. Handshaking Protocol

To ensure the Hardware component is active when required, and that the software algorithm waits until a solution is computed, a simple handshaking protocol was designed. Figure 1 outlines the protocol. The left hand figures shows the overall flow required by both the software and the hardware. There are two signals that govern the protocol, HW\_Enable and HW\_Done. To begin the Hardware calculation, SW sets HW\_Enable to high. The HW then performs the calculation and when complete, sets HW\_Done to high. Software can then read the computed output and deassert HW\_Enable. Hardware can then deassert HW\_Done and wait for new inputs. The desired signal levels are shown in the right side of Figure 1.



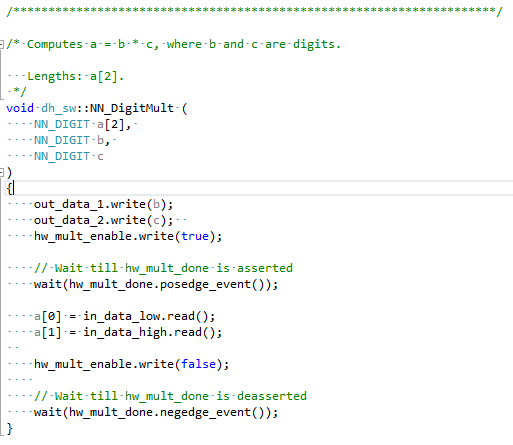
**Figure 1. Handshaking Protocol**

Figure 2 reiterates the handshaking procedure in software. As can be seen, the software handshake contains two waits. The first wait allows Hardware to complete the calculation. The second wait ensures that Hardware resets to the ready state by waiting for HW\_Done to be deasserted.



**Figure 2. SW Handshaking Flowchart**

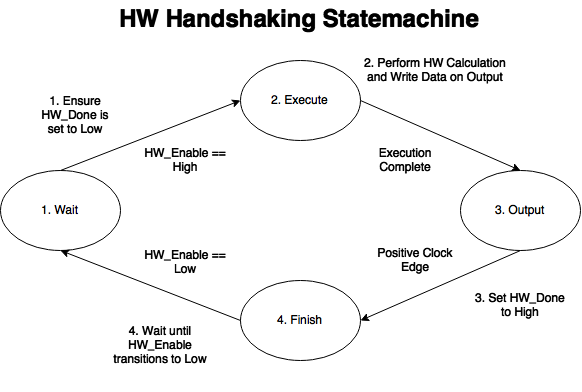
Figure 3 shows the SystemC functions in the Software algorithm. Instead of timed waits, which do not represent a viable and stable real system, wait-until events are used. These wait-until calls represent wakeup on signal edges in a real system.



**Figure 3. SW Handshaking Code**

3. Hardware Multiplier Design

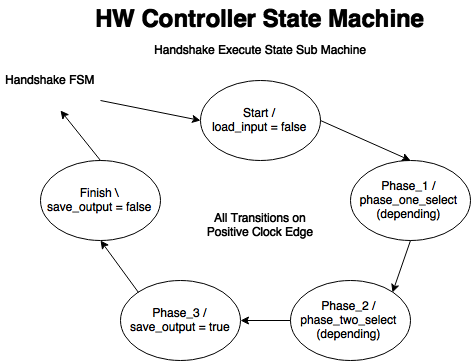
The hardware solution is split into two components, the controller and the datapath. The controller implements both the handshaking protocol and the execution state-machine. Figure 4 shows the states used for handshaking, as well as the events which cause a state changes. As seen, states 3 and 4 are clock-based transitions. State 2, Execute contains a submachine which takes 5 clock cycles to finish the calculation.



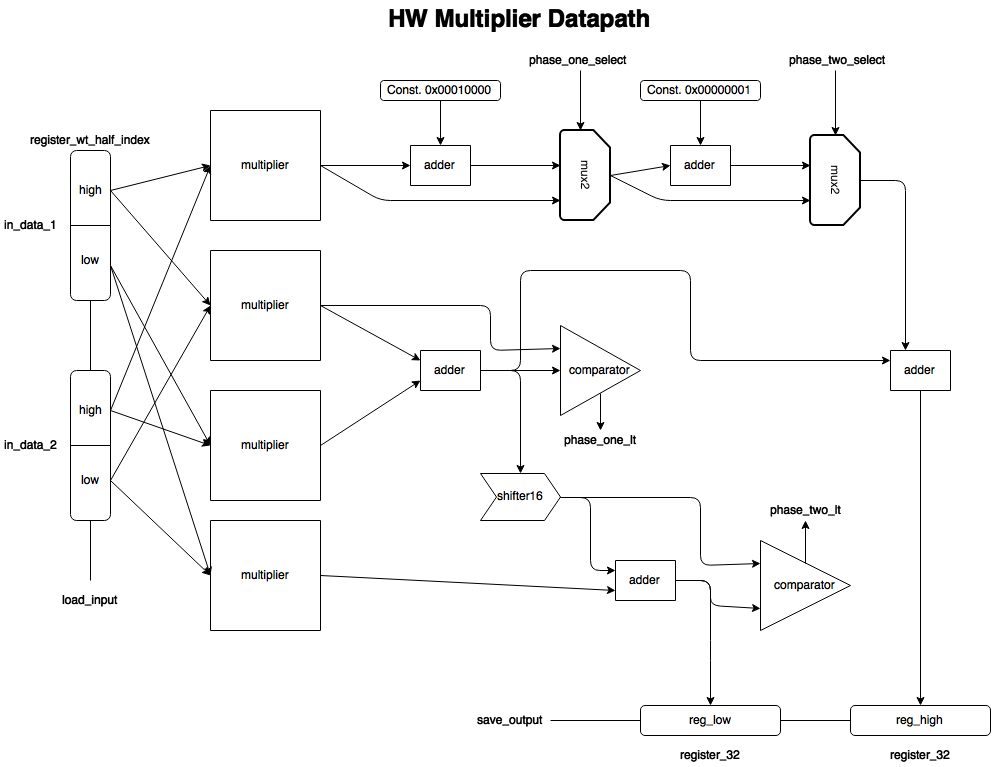
**Figure 4. HW Multiplier Handshake State-machine**

Figure 5 shows the datapath control state-machine. This state-machine exists as substates inside of the Execute state in the Handshaking protocol. The datapath controller has 4 main states. The first, ‘Start’, saves the input into registers. Secondly, the output of the first mux2 is chosen based on comparator input into the controller. The third state, phase\_2, selects the output of the second mux2. Finally, the output is stored into a register which drives the output data lines.

Lastly, the datapath is described in block format by Figure 6. The designed datapath takes 4 cycles to complete before the output can be read. This is due to the use of registers to store the input and output, as well as the requirement of conditionally addition based on the output of the comparators. The SystemC code for the datapath is a structural description. All of the used components can be found in hw\_const.h and hw\_components.h. To facilitate interfacing with software, an adaptor component was used to convert the input NN\_Digit, which is a uint32\_t, to a sc\_uint<32>, which is more representative of real hardware data. Since the control lines are simple, bool values were used. This reduced the complexity by eliminating conversion to sc\_logic.



**Figure 5. HW Multiplier Datapath Controller State-machine**



**Figure 6. HW Multiplier Datapath**

4. Recommendations