ECE385 Spring 2023 – Lab 4 Report

Ziyuan Chen, Weijie Liang | ziyuanc3, weijiel4

Introduction

The multiplier in this lab takes two 8-bit 2's-complement signed integers and outputs the 16-bit product. It performs in-place operations with an FSM to control the "shift," "add," and "subtract" actions. The product has its higher 8 bits masked out if reused as operand.

Reworked Example

State	X	A	B	M
MASK	0	0000 0000	*00000111*	1
ADD1	1	1100 0101	*00000111*	1
SHIFT1	1	1110 0010	1 *0000011*	1
ADD2	1	1010 0111	1 *0000011*	1
SHIFT2	1	1101 0011	11 *000001*	1
ADD3	1	1001 1000	11 *000001*	1
SHIFT3	0	1100 1100	011 *00000*	0
SHIFT4	0	1110 0110	0011 *0000*	0
SHIFT5	0	1111 0011	00011 *000*	0
SHIFT6	0	1111 1001	100011 *00*	0
SHIFT7	0	1111 1100	1100011 *0*	0
SHIFT8	0	1111 1110	01100011	1

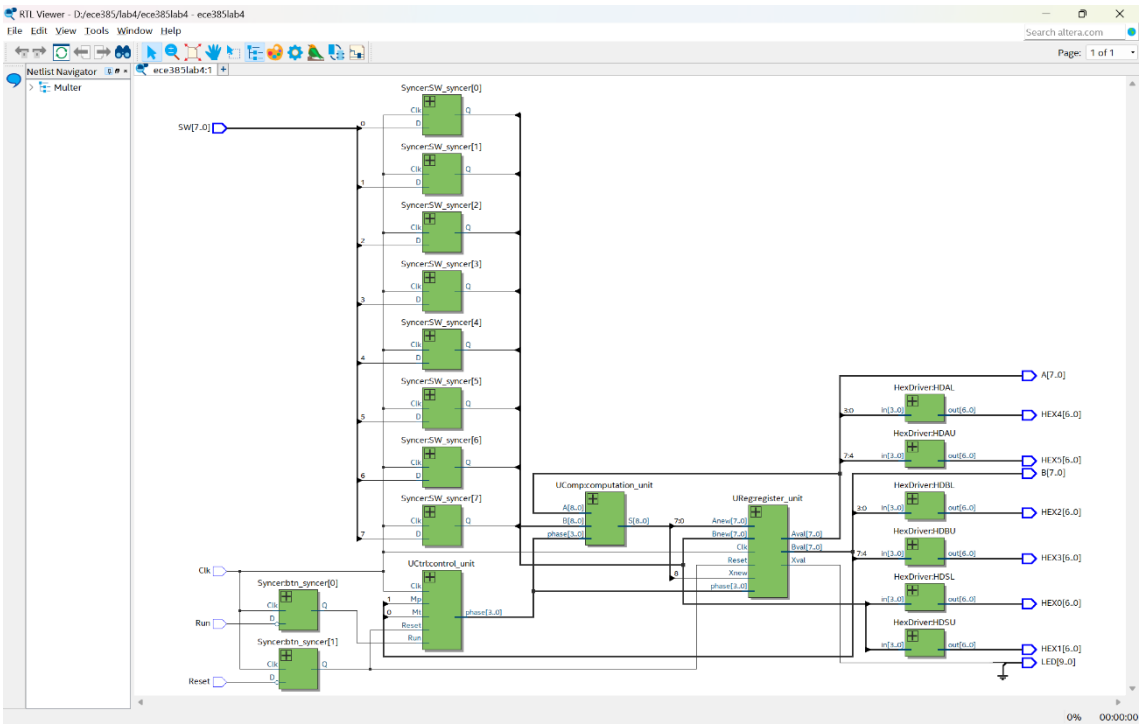
Operating the Multiplier

There are two buttons on the FPGA board, namely `Reset` and `Run`. The user first flips the `SW`itches to the desired `B` value and presses `Reset`, at which signal the multiplier (1) clears register `A`, (2) loads `SW` into `B`, and (3) resets the FSM to `STOP` state. The user then sets `SW` to the desired `A` and presses `Run`, where the multiplier (1) initiates the operation, (2) performs 10~17 state transitions depending on the `M` value after each register shift, and (3) pauses at `DONE` state until the user releases `Run`. `Reset` overrides `Run` when both buttons are pressed.

In order to save memory, the multiplier adopts a modified version of the Grade School Multiplication algorithm that adds and shifts the registers in place.

- **Shifts:** All shifts are arithmetic where `XAB` is viewed as a 17-bit number – that is, `XAB <- SEXT(XAB[16:1])`. The least significant bit of `B` is simply dropped.
- **Arithmetic operations:** `A` and (synchronized) `SW` are sign-extended before entering the 9-bit adder, whose output routes to `{X, A}` and the carry bit is discarded.
- **State transitions:** If `M` (a.k.a. `B[0]`) is `1` by the end of `SHIFTi`, the FSM moves into `ADDi` ($0 < i < 7$) or `SUB` ($i = 7$). This indicates that a shifted addend is "written" on the imaginary "paper." Otherwise, the next state will be `SHIFT(i+1)`.

Two FSM states deserve additional explanations. `SUB` is in the position of a non-existent "`ADD8`" since the operands are signed integers and the result should be subtracted in case the highest bit of `B` is `1`, indicating a negative number. `MASK` takes over the action of "clearing register `A`" from `DONE` since the machine stays in `DONE` most of the time and displays the 16-bit product, whose higher bits should not be masked. Instead, the machine forcefully clears `A` each time it initiates a new operation cycle of multiplication.



Top-level block diagram, generated by the RTL Viewer of Quartus Prime 18.1 Lite

SystemVerilog Modules *(Array sizes are abbreviated. E.g.,* `[7:0]` *is written as* `[8]`*)*

`FullAdder`

Inputs: `A B Ci`

Outputs: `S Co`

Description: Standard 1-bit full adder.

`uReg`

Inputs: `Xnew Anew[8] Bnew[8] phase[4] Reset Clk`

Outputs: `Xval Aval[8] Bval[8]`

Description: At posedge of `Clk`, `{Xval, Aval, Bval}` are updated into

- (if `Reset`) `{0, 8'h0, Bnew}` *("Clear-and-Load")*
- (if `Mask`) `{0, 8'h0, Bval}` *(only B retains)*
- (if `Shift`) `{X, {X, A[7:1]}, {A[0], B[7:1]}}`
- (if `Add` or `Sub`) `{Xnew, Anew, Bval}`

`Mask`, `Shift`, `Add`, `Sub` are encoded in the 4 bits of `phase`. These control signals are programmed to be one-hot.

Purpose: Register unit that manages the operands and results.

`uComp`

Inputs: `A[9] B[9] phase[4]`

Outputs: `S[9]`

Description: Performs parallel addition and outputs the result. Has built-in `XOR` gates that reverse `B` if `phase` = `Sub`. The `Sub` bit is also inputted into `Ci` to perform subtraction when requested.

Purpose: Computation unit that provides updated values for `X` and `A`.

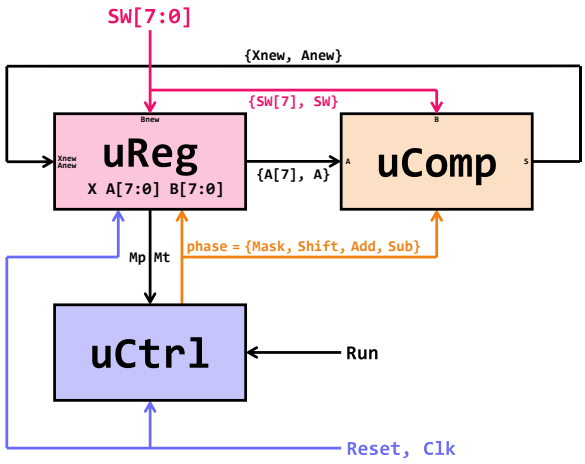
`uCtrl`

Inputs: `Mp Mt Run Reset Clk`

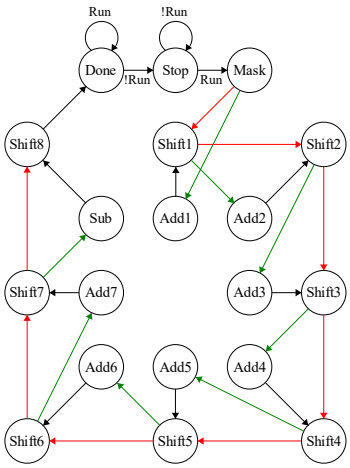
Outputs: `phase[4]`

Description: Implements the FSM that gives the proper `phase` signals at the right time. Accepts two `M`s (`Mp` = `B[1]`, `Mt` = `B[0]`) to account for the actual shift that occurs *at the end of* a SHIFT state, where M is not updated until the next state.

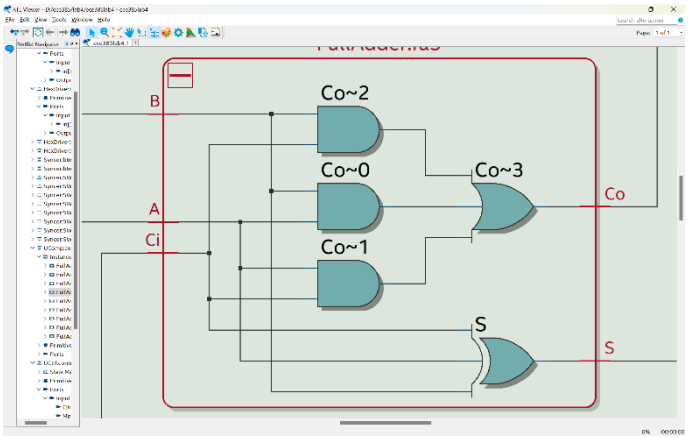
Purpose: Control unit that orchestrates the computation cycle.



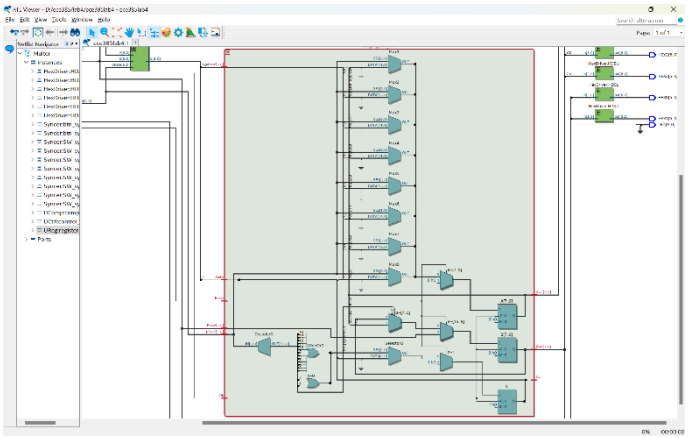
Top-level block diagram of interconnected SystemVerilog modules, simplified and labeled with signals



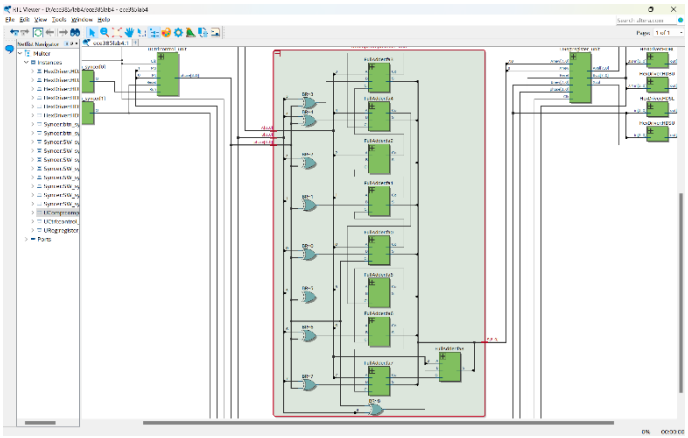
*State diagram for the control unit (**Green** = if `M`, **Red** = if `!M`, unlabeled **Black** = always)*



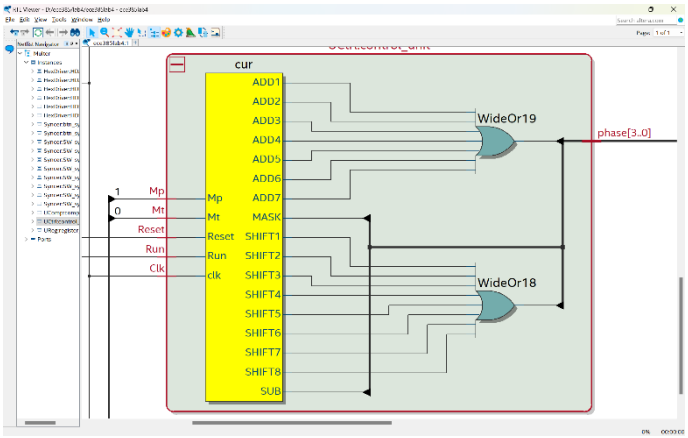
Full Adder



Register Unit



Computation Unit



Control Unit

Expanded RTL diagrams of individual modules (Clear images are in our [GitHub repository](#))

Annotated ModelSim waveforms showing step-by-step operations of 0x3B × 0x07 = 0x0019D, 0x3B × 0xF9 = 0x1FE63

Post-Lab Problems

- Design Resources and Statistics

	Multiplier
LUT	617
DSP	0
Flip-Flop	733
Memory (BRAM)	3,456
Frequency/MHz	89.02
Static Power/mW	89.95
Dynamic Power/mW	1.48
Total Power/mW	100.41

To increase the maximum operating frequency, we can substitute the current CRA for a more time-efficient CSA or CLA at the cost of extra space. To decrease the total register or gate count, we can encode `phase` from the FSM with only two bits instead of four one-hot bits. Everything else seems to be optimal, especially the seemingly redundant `MASK` state – merging the operation of masking A into `STOP` causes the higher two HEX digits to always be 0.

- The purpose of the `X` register

  The 8-bit operands are sign-extended to 9-bit, and `X` holds the most significant bit of the result. When performing a register shift, its value will be copied to the most significant bit of the 8-bit register `A`. Since adding produces a carry-out bit, the adder should also be sign-extended to 9 bits, and we need an additional register `X` to hold the out-of-range bit. `X` gets **set** when the 9-bit adder generates a new value and is **cleared** when `Reset` is pressed.

- What if we update `X` with the carry-out of an 8-bit adder?

  The results will be the same for the `ADD` and `SHIFT` operations but will not be correct in the final `SUB` phase when the sign bit of `B` comes into effect.

- Limitations of continuous multiplications

  If we continuously multiply a number whose absolute value exceeds 1, B (holding the result) eventually overflows and yields incorrect values since it only has 16 bits. Valid results should be within the range of [-32768, 32767].

- (Dis)advantages over the Grade School Multiplication algorithm

  The pencil-and-paper method needs more memory to hold the intermediate values from each step of multiplication, which is redundant since they are merely shifted versions of the operand. On the contrary, this *in-place* algorithm makes full use of the registers by shifting.

Conclusion

In this lab, we designed a multiplier that consists of 3 parts: a register unit of grouped `X`, `A`, and `B`, a 9-bit adder that adds the sign-extended `A` and `S` and loads the result to `A` and `X` (that temporarily holds the most significant bit), and a finite state machine that provides control logic to the circuit like the `Mask` (a.k.a. CLR, Clear-Load-Reset), `Shift`, `Add`, and `Sub` signals. We attempted to use the carry bit of the 8-bit adder to update the register X, and, as we later analyzed, it fails at the final subtraction step.