EECS 270 W25
Discussion 12

April 10th, 2025

By: Mick Gordinier

1. Course Evaluations
2. RTL Design Recap
3. Project 7 Full Discussion
4. P7 Example XInit → Loading Value
5. P7 Booth Multiplication
6. ModelSim Demo of Booth Mul
7. P7 Division

Course Evaluations

Please Do Course Evaluations!!

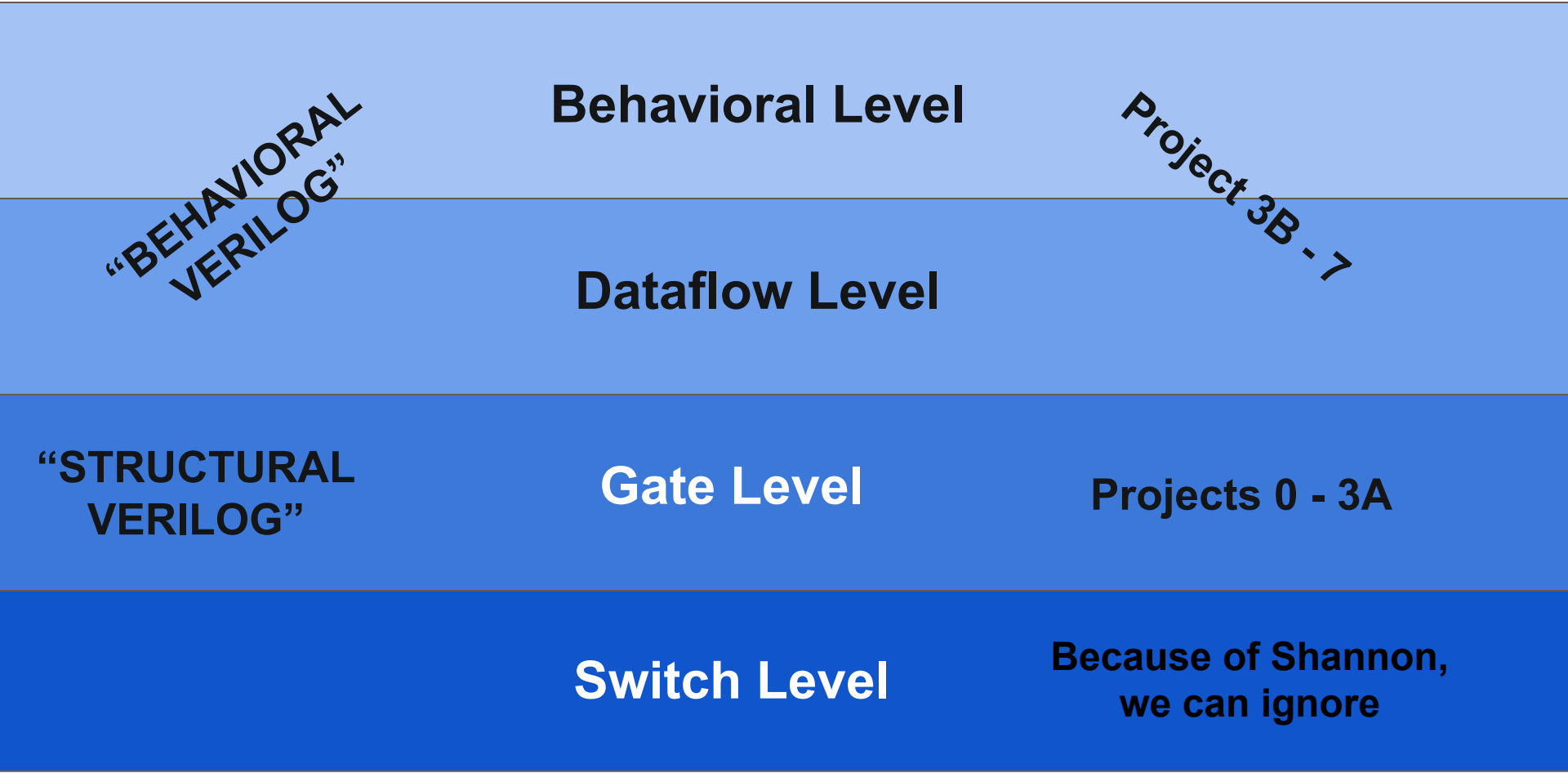
- We are always continuing to improve upon the course and give the best 270 experience possible!

- We want to hear from you, understand your 270 experience, and want to know what worked and didn’t

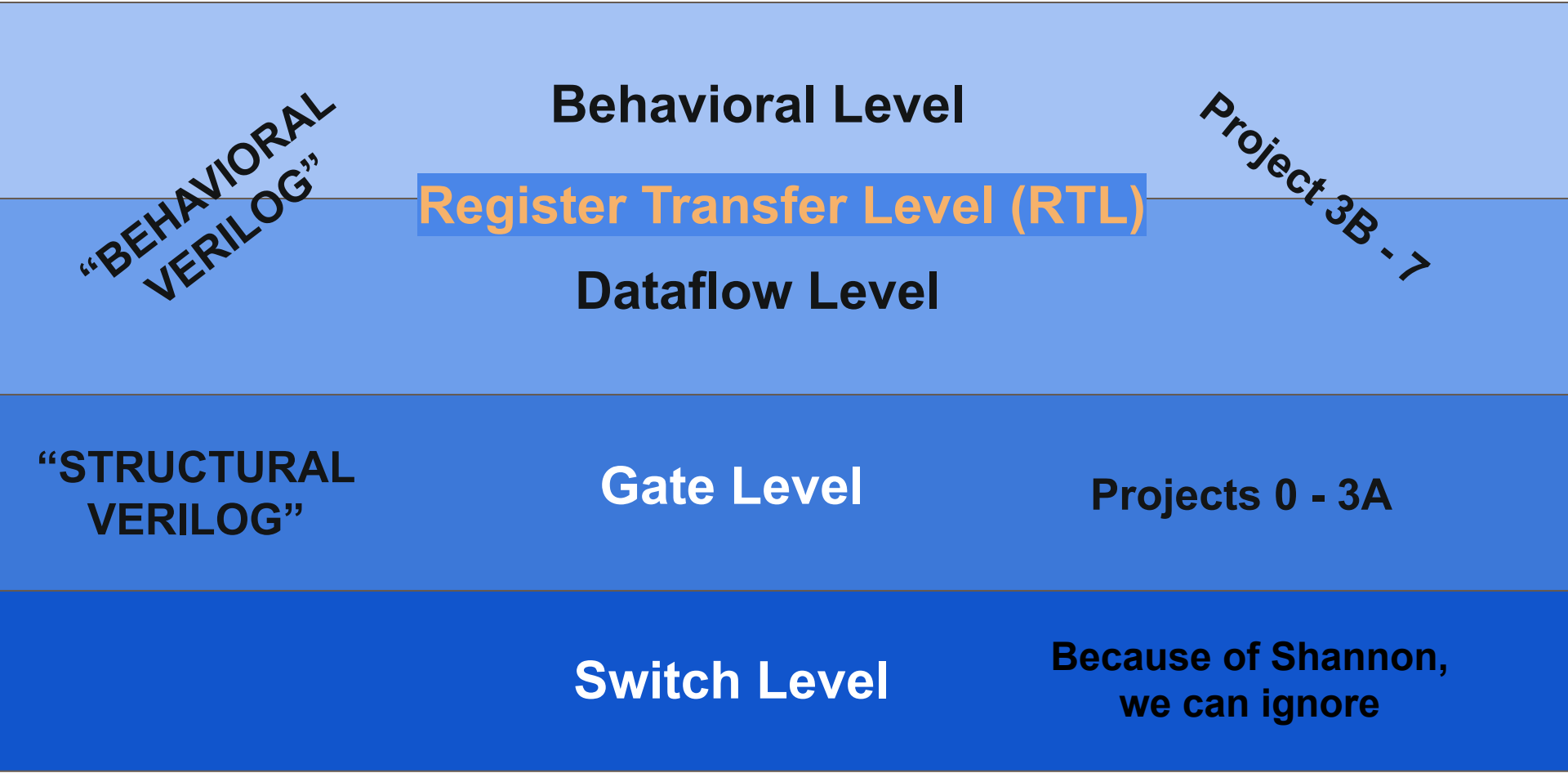
- For discussion comment/improvements, please let me know under ‘Additional comments for Mick Gordinier’

Register Transfer Level (RTL) Design Recap

Verilog's Many Levels of Abstraction



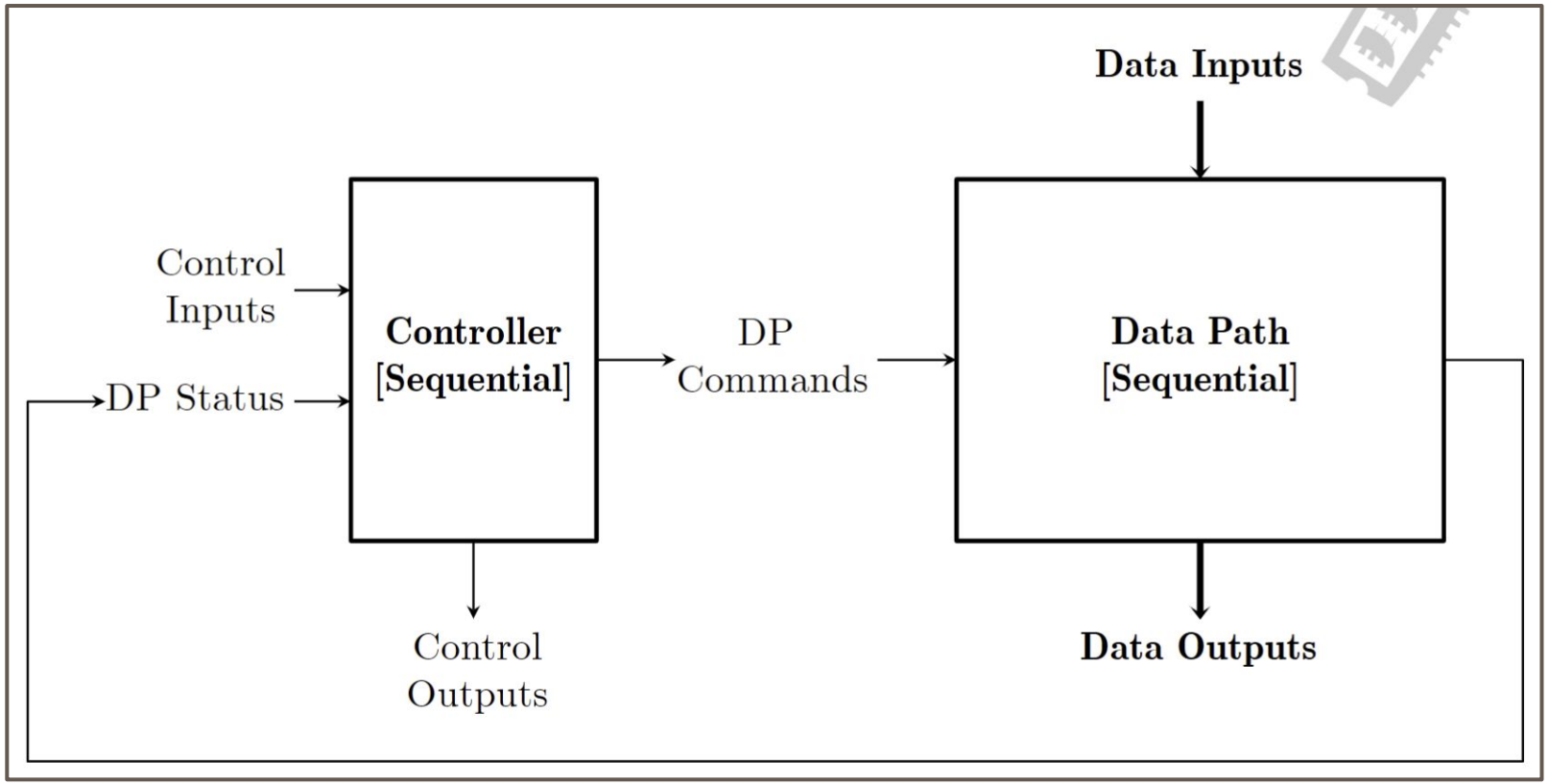
Verilog's Many Levels of Abstraction



RTL Design Overview

- Splitting up design in Controller and Datapath
- **Controller** - Controls and orchestrates the Datapath operations
- **Datapath** - Computation on the signals
  - Arithmetic - Add/Subtract/Multiple/Count/…
  - Logical - Shift Left / Shield Right / Arithmetic Shift / bitwise Ops
  - Other - Clear / Load / Hold

RTL Sequential Circuit



Main Differences with Behavioral / RTL Design

Feature	Behavioral Verilog	RTL Verilog
Abstraction Level	Very High (Nearly Software)	Medium (Closer to Hardware)
Synthesizable	Sometimes (Not everything can be synthesized to hardware)	Always
Main Use Case	Simulation and quick modeling	Actual hardware implementation
Need to Explicitly Describe Underlying Gates/Hardware	No	No

Project 7 Full Discussion (Sequential Calculator)

Project 7 Deadlines

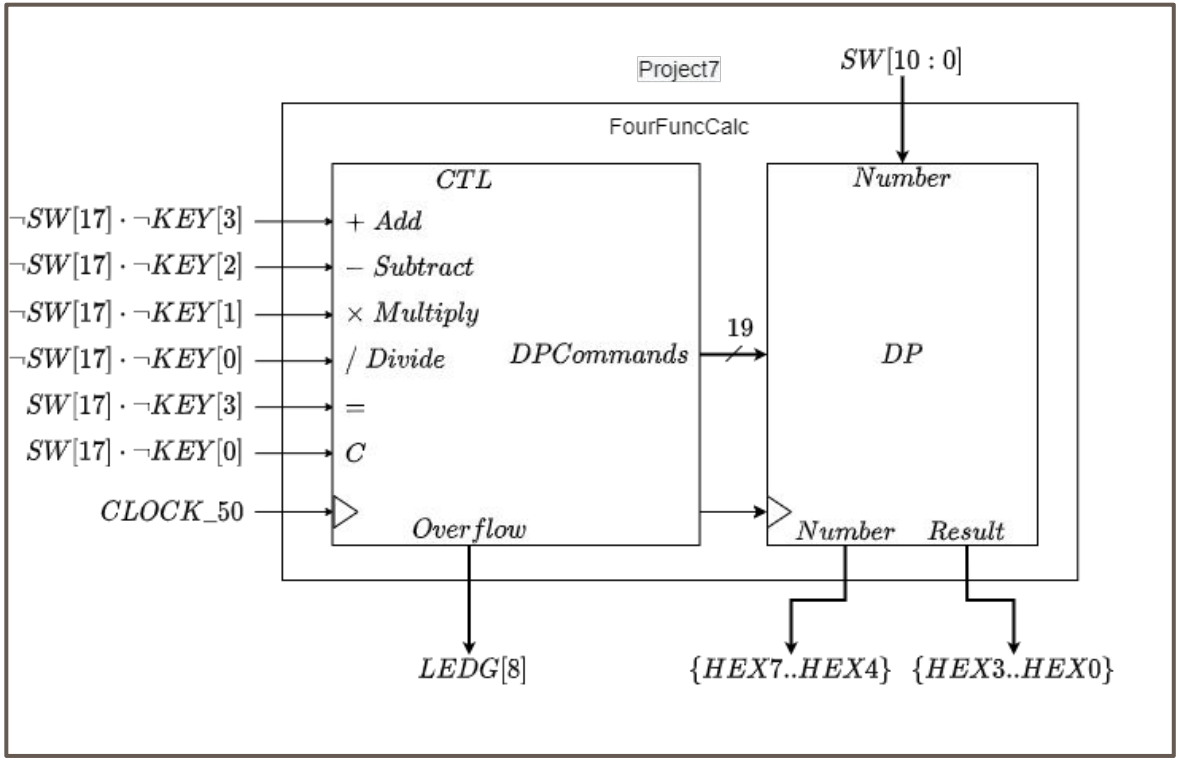
- Project 7 Checkpoint (Due: 4/15)
  - Implemented Adding and Subtracting in your Four Func Calculator
  - **Worth 10% of your grade**
  - Will require a testbench for adding and subtracting
  - **Allows us to run your project on FPGA to ensure no timing issues**

- Project 7 Final (Due: 4/25)
  - Implementing Multiplication and Division in your Four Func Calculator
  - Requires full testbench
  - **Worth 90% of your grade**

- **NOTE: THERE IS NO IN-PERSON SIGN OFFS FOR THIS PROJECT**

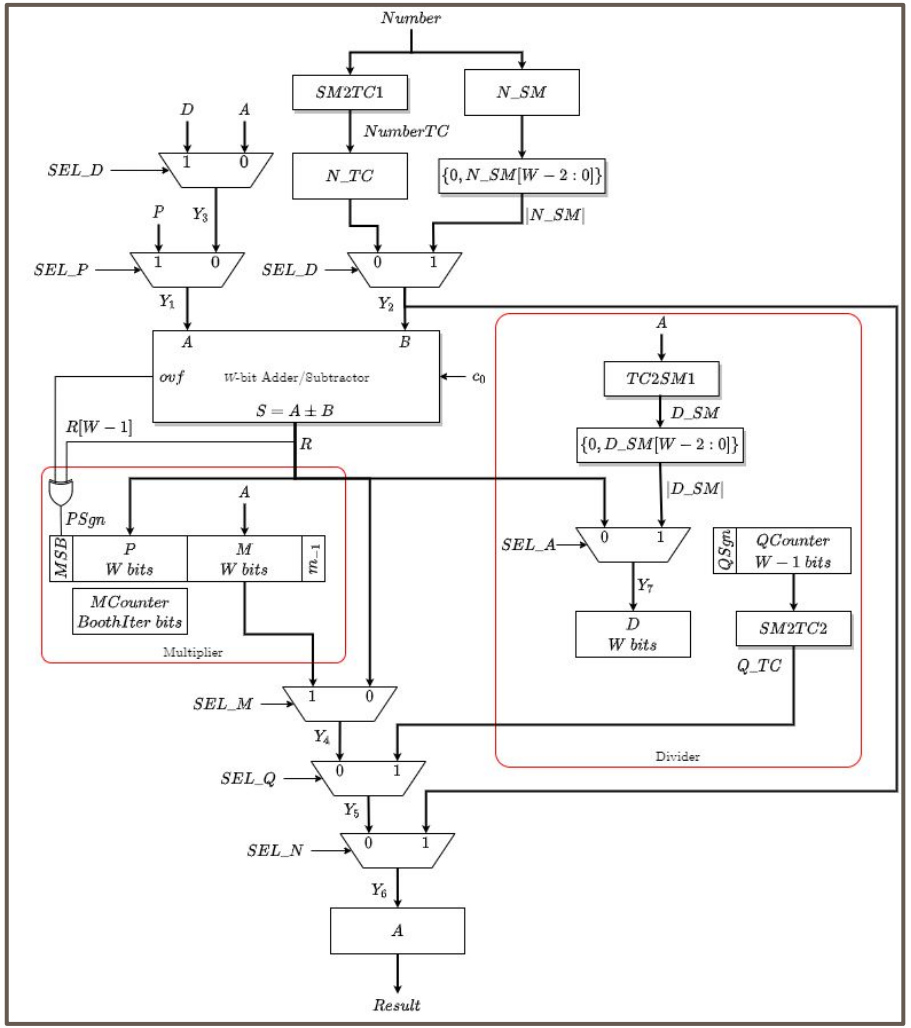
Four Function Calculation (Checkpoint) Overview

- Controller Input
  - **+, -, x, /, =, C**
  - 50 MHz Clock
- Datapath Input
  - DPCommands
  - Number
- Outputs
  - HEX Displays
  - Overflow bit



FFC Datapath (CP)

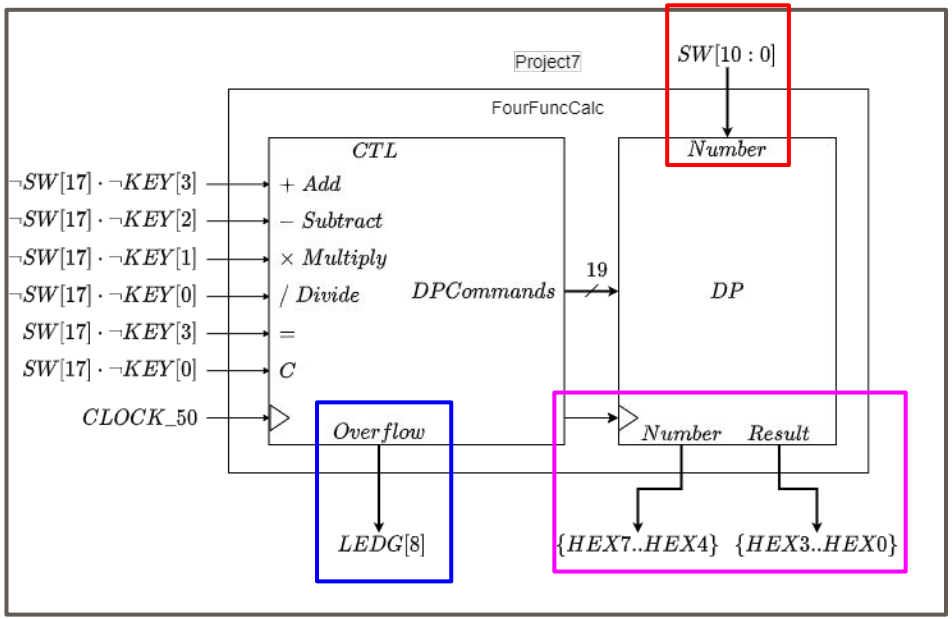
- Only need to focus on implementing add/sub datapaths
- **NOTE:** Must only have 1 instantiation of the AddSub modules



I/O Interface Notes

- Dealing with 11-bit numbers
- **Number SWs inputted as Signed-Magnitude**
- But computation in datapath assumes 2's complement

- Overflow - Outside 11-bit 2C range
  - [-1024, 1023]
- vs. Display "----" if can't be displayed
  - [-999, 999]



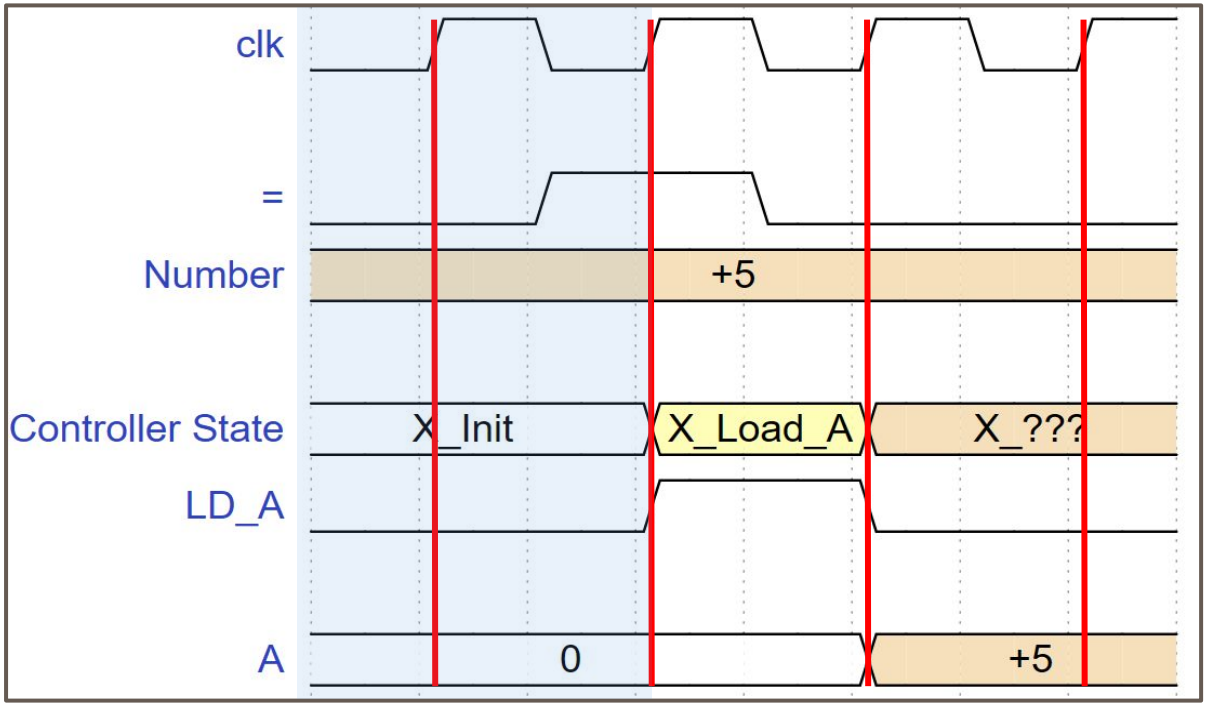
I/O Interface

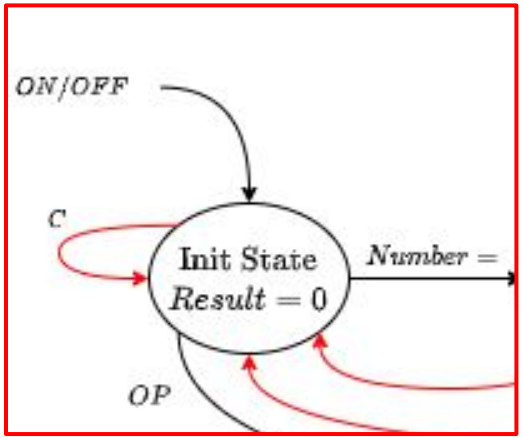
Figure 1 shows the datapth/control decomposition and the DE2-115 interface of the calculator.

- **Entering and displaying numbers**: Signed-magnitude numbers, in the range $[-1023, 1023]$, can be entered using SW[10:0] and should be displayed on {HEX7, HEX6, HEX5, HEX4}. Operation results should be displayed on {HEX3, HEX2, HEX1, HEX0}. Numbers outside the range $[-999, 999]$ should be displayed as − − −− indicating "can't be displayed'.
- **Entering operations**: The four arithmetic operations can be entered by pressing a pushbutton KEY with SW[17] set to 0. Refer to the mapping in Figure1.
- **Entering commands**: Besides entering numbers, two KEYs are used to enter the following commands when SW[17] is set to 1:
  - = (equals): Pressing KEY[3] displays the result of the operation on {HEX3, HEX2, HEX1, HEX0}.
  - C (clear): Pressing KEY[0] clears the result display and returns the calculator to its initial state.
- **Overflow indicator**: Overflow should be indicated by LEDG[8] if any operation result is outside the range of 11-bit two's complement numbers, ie., $[-1024, 1023]$ (would have been better if this was a red LED because of its location between the HEX displays!)

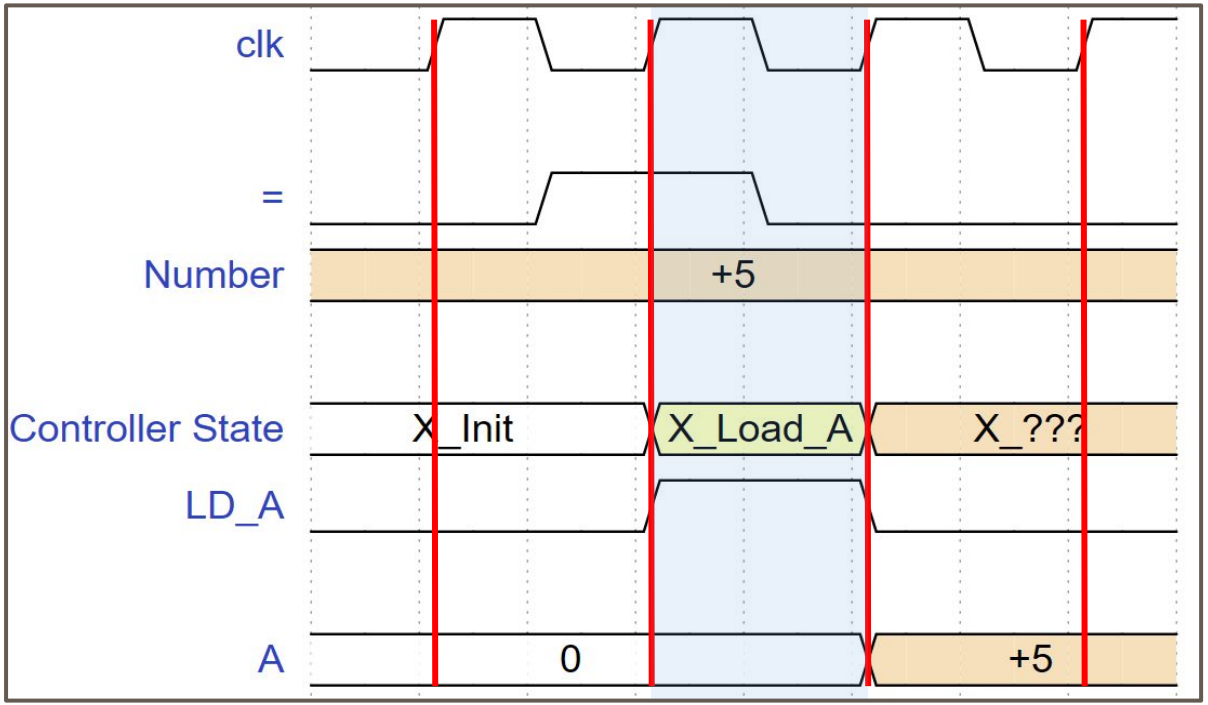
P7 Example XInit → Loading Value (Ignoring N_TC Reg)

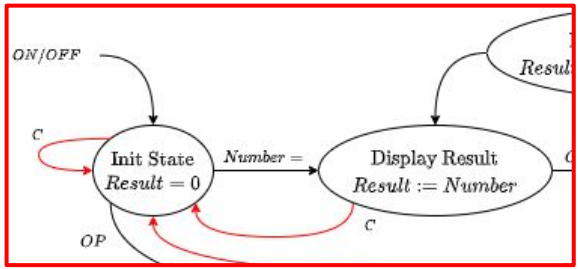
1. On Initialization, Should stay Init State





2. Once '=' Seen on Posedge clk, Begin Loading



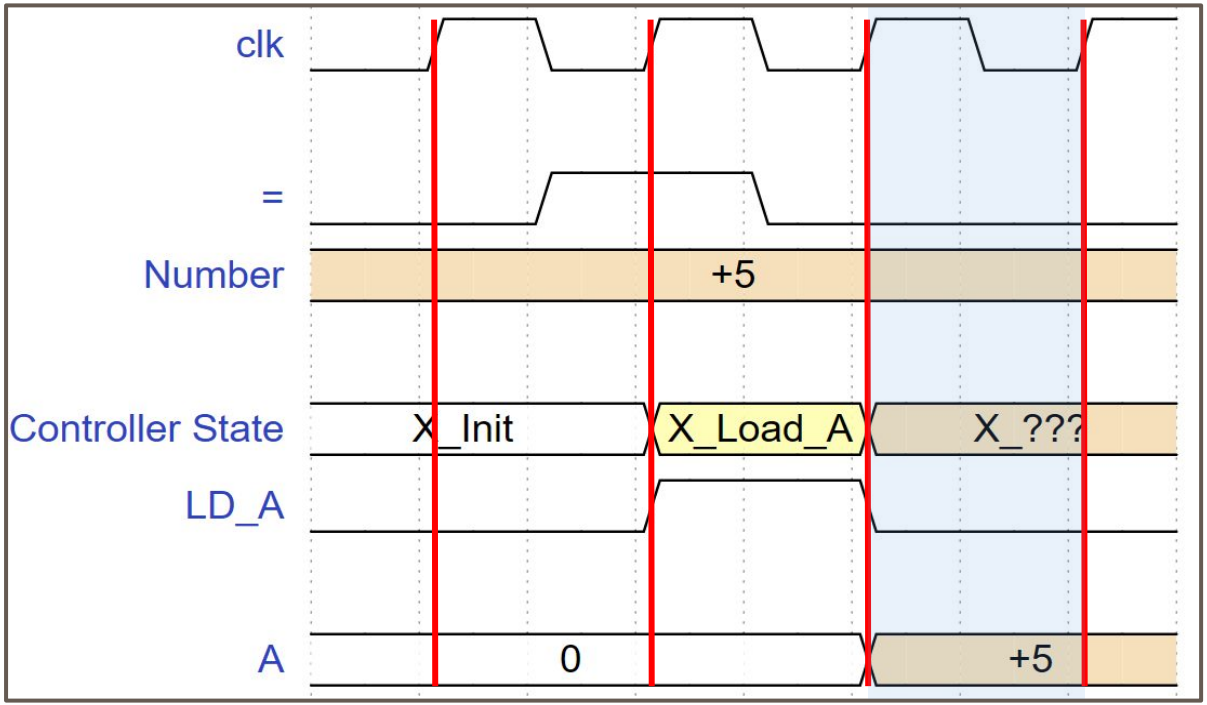


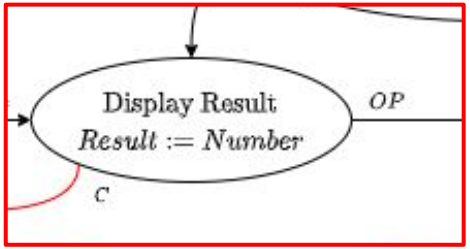
NOTES:

Only CONTROLLER has seen the '='.

Controller needs to inform datapath.
Controller updates its internal state
→ Changes command bits

3. LD_A Command Seen by Datapath NEXT Cycle



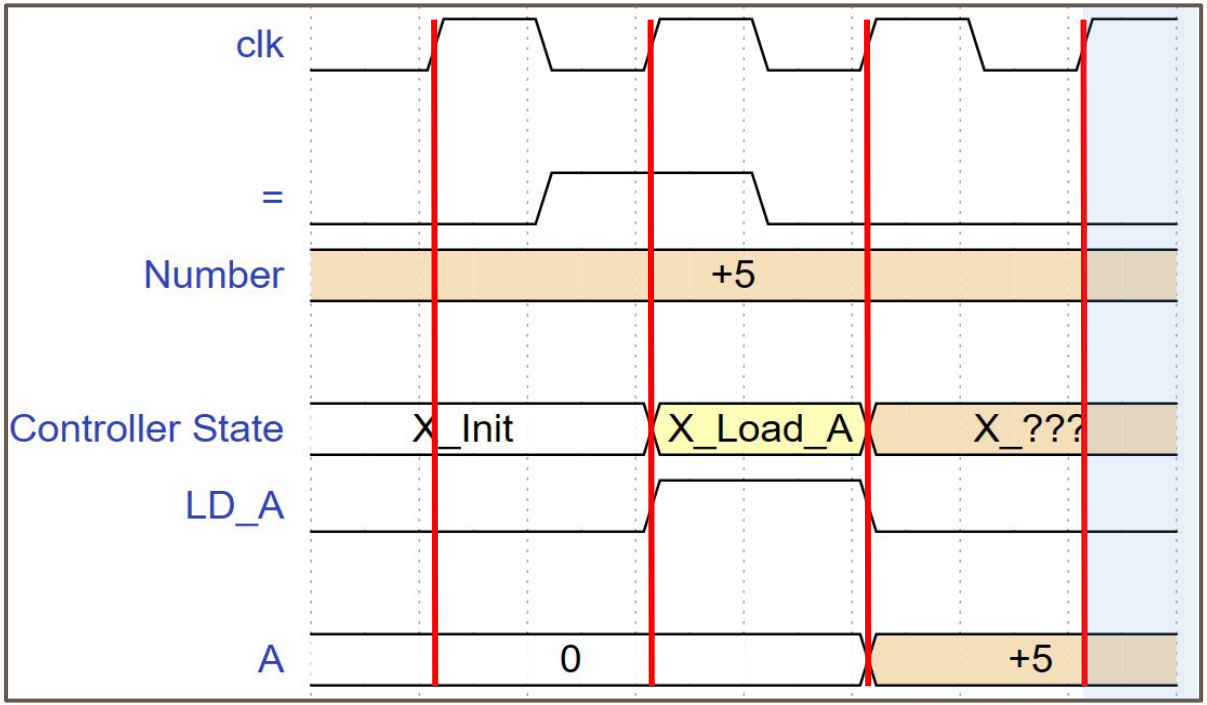


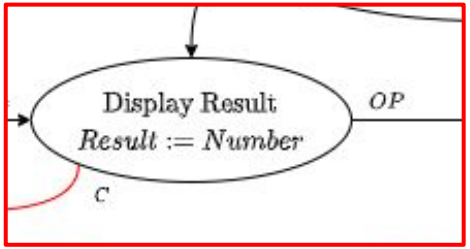
NOTES:

Datapath sees 'LD_A' command CYCLE AFTER it was set by the controller.

Controller want to ensure 'A' only loads for ONE CYCLE.

4. Accumulator Hold Onto Value Until Next Command





P7 Booth Multiplication

Understanding Booth Multiplication In-depth

- Watch Lecture Session 19
- Go through Lecture Slides
- Discussion today is focussed on implementation, not understanding Booth Multiplication

Booth Multiplication in P7

- Optimized W-bit x W-bit multiplication
- Requires multiple cycles to complete (Iterates ‘CNT’ times)

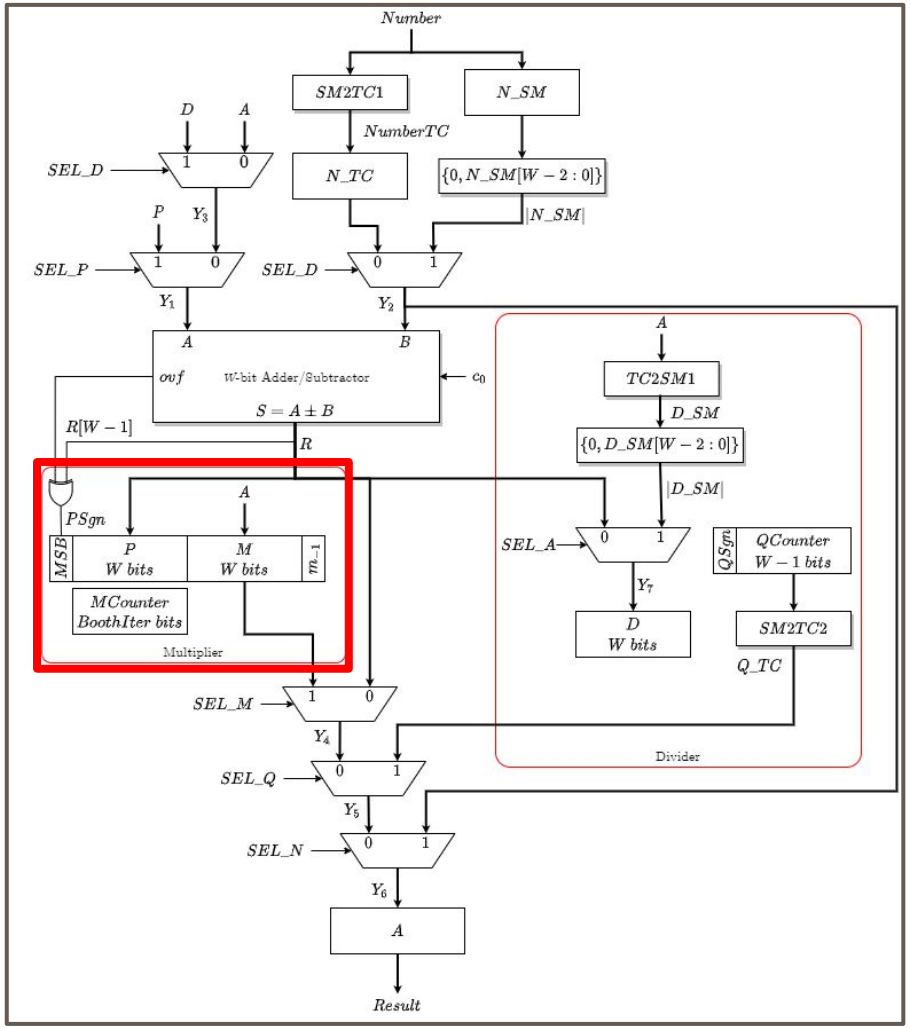
- **WE PROVIDE YOU WITH A COMPLETED PARAMETERIZED BOOTH MUL**

- Your Job: Integrate Booth Mul with your four function calculator

- **REMEMBER: YOU CAN ONLY USE 1 ADD/SUB MODULE IN FFC**
  - **You must integrate Booth Mul with using the same add/sub module**

Booth Mul Registers

- Performing: **A * Number**
  - **Both are W-bits wide**
  - **Both are 2's complement**
- W-bit P, M registers
  - Hold final 2W-bit result
- Additional 1-bit MSB, m_{-1}
- MCounter register
  - Need to keep track of how many iterations we have left



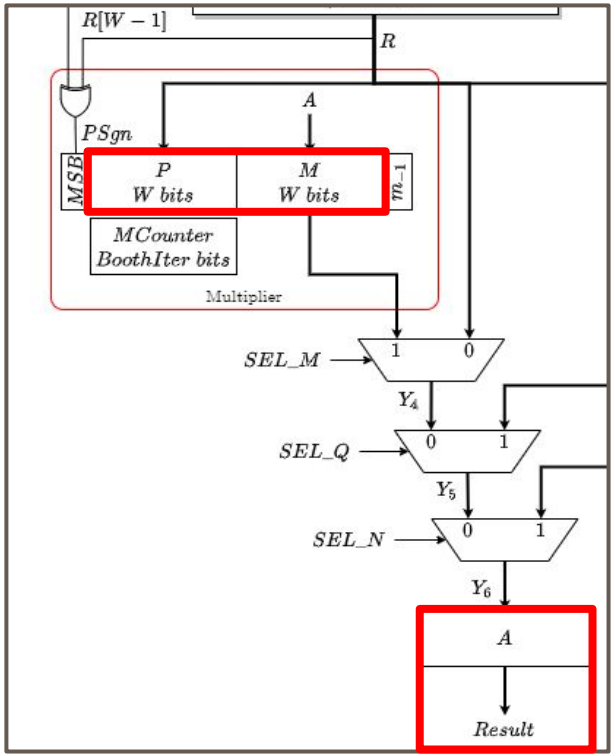
IMPORTANT: Storing Result & Overflow (Multiplication)

- Performing: **A * Number**
  - Both are W-bits wide
  - Both are 2's complement signed

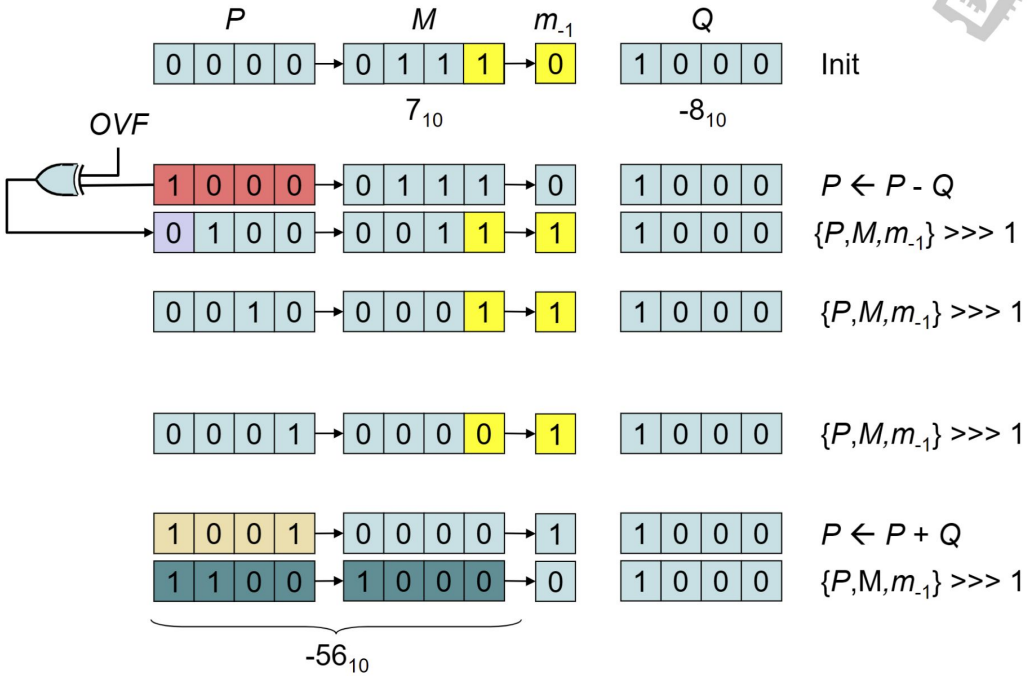
- Your final result is **2W-bits {P, M}**

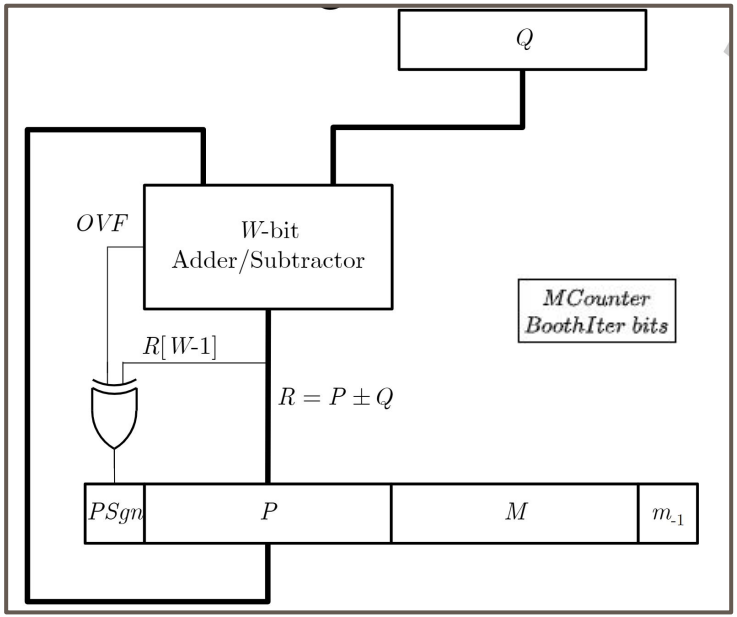
- **Problem: Accumulator is only W-bits wide**

- **How can we handle this?**
- **How do we determine if our result can fit in A?**
- **What do we do if we can't fit our final result in A?**



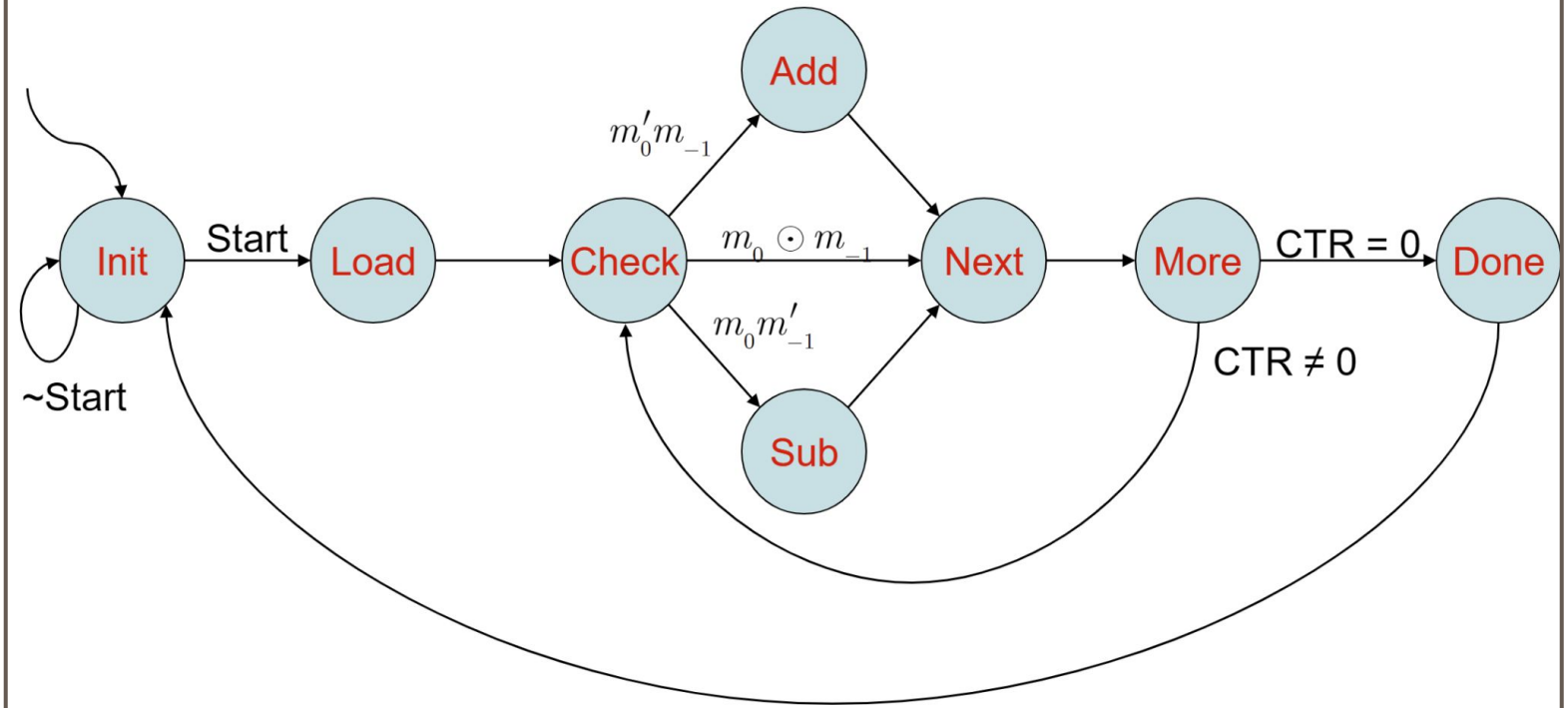
Multiplier V2.0 Execution Trace





What is an arithmetic vs. logical shift?

Controller State Diagram



Use of ‘signed’ in Verilog

```
// Datapath State Update
  wire signed [W:0] ZERO;           // (W+1)-bit 0 since `(W+1)'d0 does not work
  assign ZERO = 'd0;
  always @(posedge Clock)
    if (Reset)
      begin
        PM[WW+1:W+1] <= 'd0;
        PM[0] <= 0;
        CTR <= W;
      end
    else
      begin
        PM <=
          (M_LD? $signed({ZERO, M, 1'b0}) :       // Load M
            (P_LD ? $signed({PSgn, R, PM[W:0]}) : // Add/Sub
              (PM_ASR ? PM >>> 1 :                // ASR
               PM                                 // Unchanged
              )
            )
          );
        CTR <= CTR_DN ? CTR - 1 : CTR;
      end

// Datapath Output Logic
  assign P = AllDone? PM[WW:1] : 'd0;
endmodule // BoothMul
```

Telling Verilog to treat register as 2C value when doing arithmetic operations.

THE BITS OF THE REGISTER REMAIN UNCHANGED

NOTE: Only operation this pertains to is the arithmetic shifting (>>>)

NOTE: AddSub module does not care that the number is signed as it perform BITWISE operations

ModelSim Demo of Booth Mul

P7 Division

Division in P7

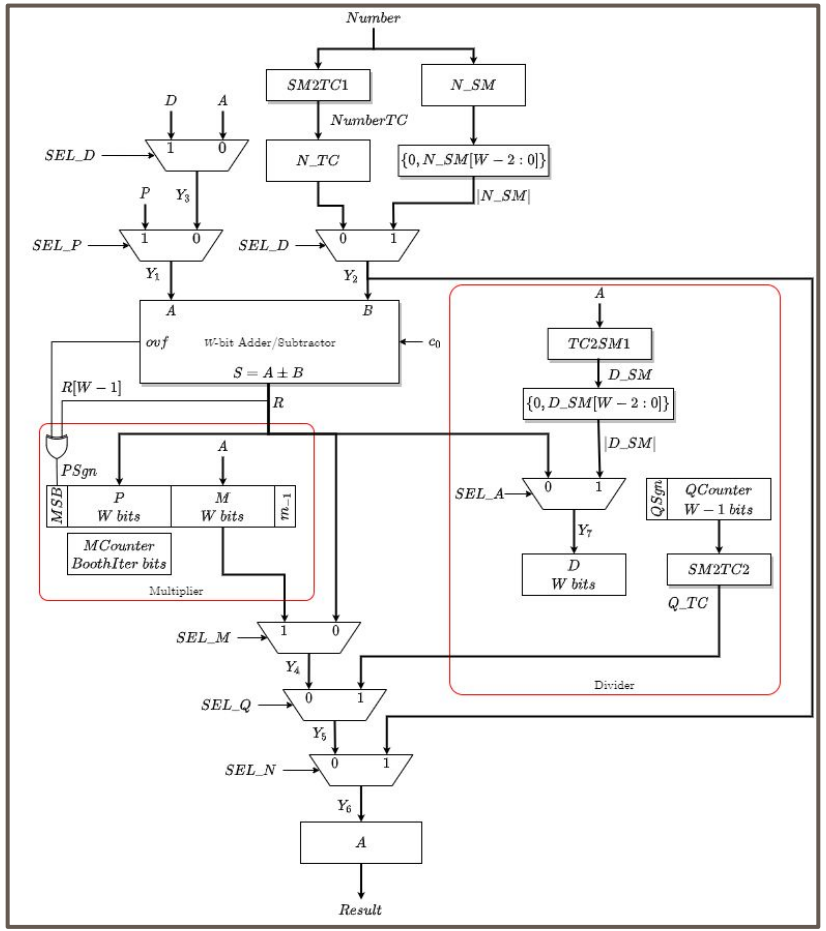
- Will NOT be covering in lecture
- Obtaining **Quotient MAGNITUDE** by repeatedly subtracting the **MAGNITUDE** of the **Divisor** from the **MAGNITUDE** of the **Dividend**
- **Quotient = # Subtraction performed**

	$Quotient = 0$
$7 - 2 = 5$	$Quotient = 0 + 1 = 1$
$5 - 2 = 3$	$Quotient = 1 + 1 = 2$
$3 - 2 = 1$	$Quotient = 2 + 1 = 3$
$1 - 2 = -1$	Stop

$Quotient = \frac{Dividend}{Divisor}$. For example $\frac{7}{2} = 3$, $\frac{-11}{5} = -2$, $\frac{-26}{-6} = 4$, and $\frac{15}{-3} = -5$

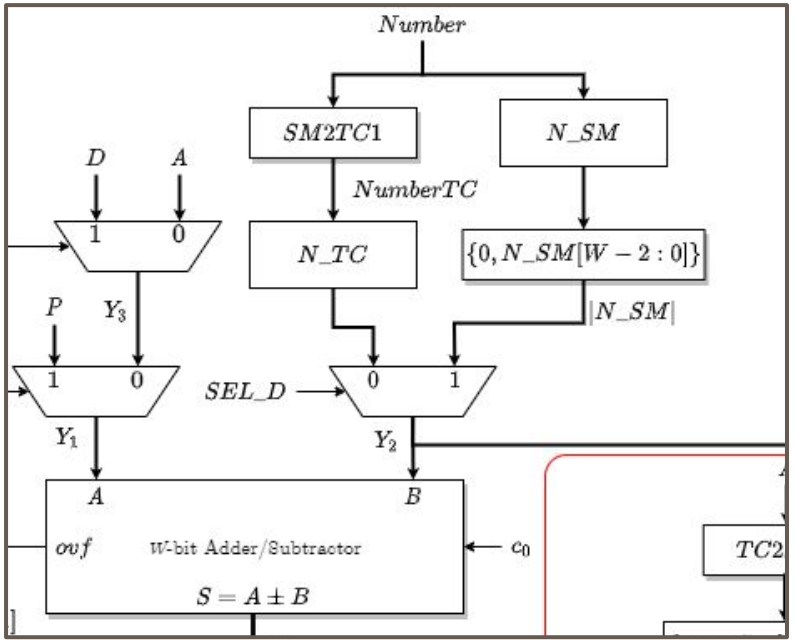
Obtaining Sign of Quotient

- After obtaining magnitude of quotient, **need to determine sign**
- **How can we do that?**



Why Bother with Both N_SM and N_TC?

- **Why not just have one over the other?**
- **Are there benefits to having both?**



THANK YOU