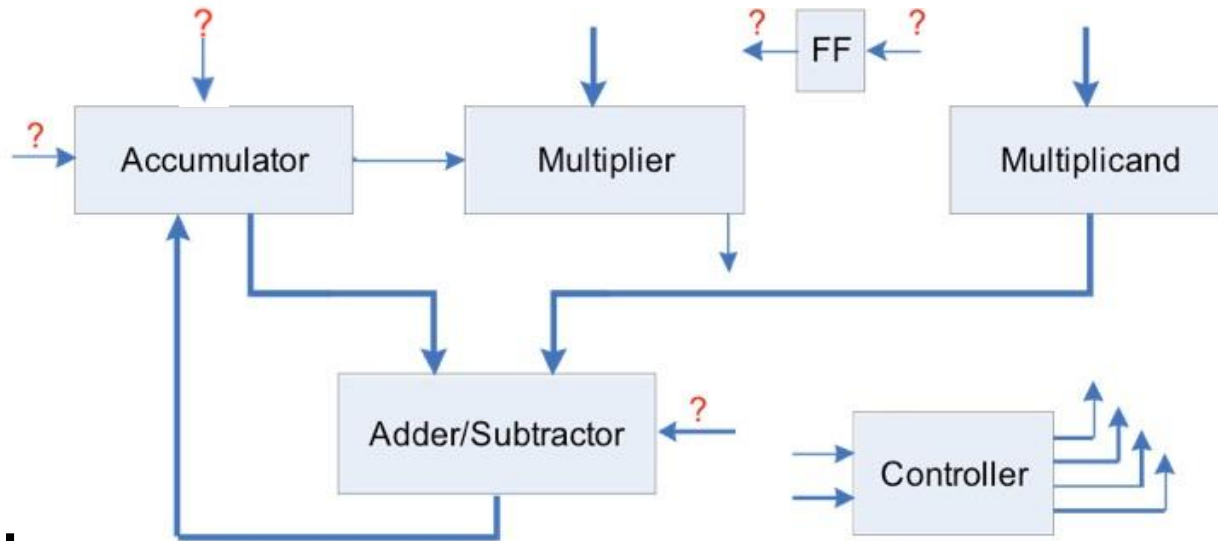


Lab 2:

2's Complement Multiplication and Error Correction

- Section 1: Implement a 19 bit Modified Booth's Multiplier circuit in LogicWorks
- Section 2: Implement an error correction circuit based on Extended Hamming Code

Modified Booth's Multiplication



Basic Idea

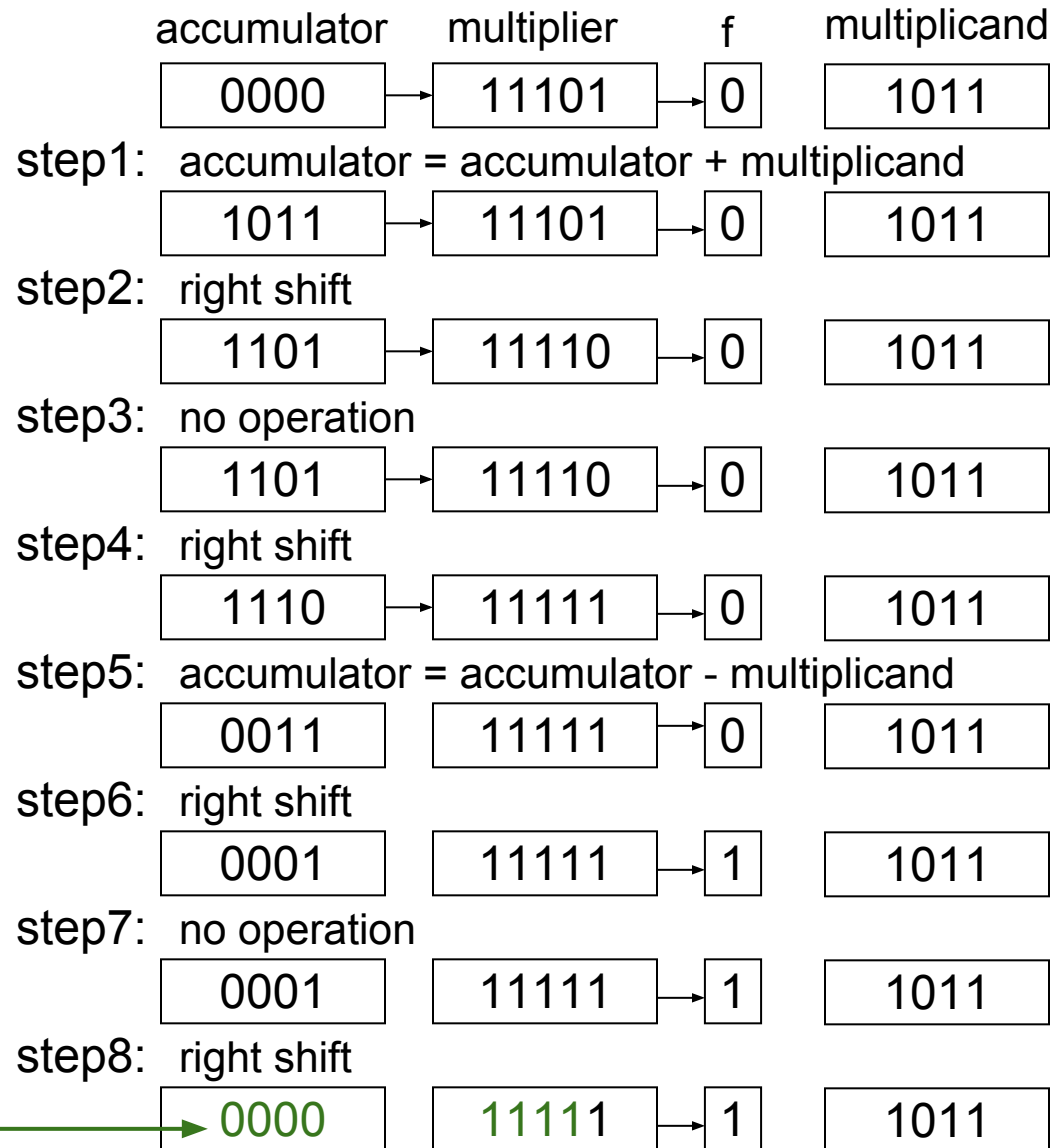
- Accumulate a partial sum in multiple steps.
- The 2 rightmost bits of the multiplier together with FF are checked on every step to determine if the multiplicand should be added or subtracted to the partial sum.
- Shift the partial sum and the multiplier one bit to the right on every step.
 - ✓ Make the partial sum line up with the multiplicand;
 - ✓ Always check the 2 rightmost bits of multiplier along with FF
 - ✓ Lower bits of the partial sum shifted into the multiplier register for storage.

Multiplication Example

$$1011_2 * 1101_2$$

1011	(multiplicand)
11010	(multiplier, F)
0000	(partial sum)
1011	(add)
11011	(partial sum)
----	(no-op)
111011	(partial sum)
0101	(subtract)
0001111	(result)

carryout
ignored



Result

LogicWorks Implementation

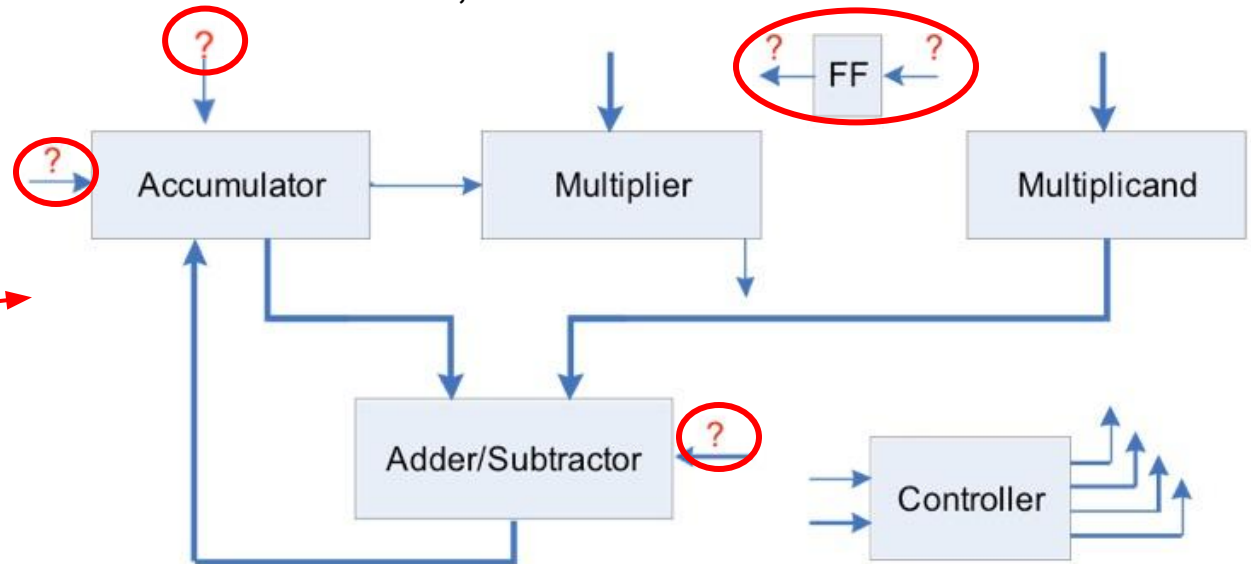
- Part 1: Implementation with $(n+1)$ -bit Adder/Subtractor
- Part 2: Switch to a n -bit Adder/Subtractor, use sign extension for shift in value
- Part 3: Use the carry-out for the shift in value
- Part 4: Use the multiplicand's MSB for the shift in value
- Part 5: Fix issue with using multiplicand's MSB
- Part 6: Fix shift-in issue when accumulator is still zero

Guidelines

M_1M_0/FF	Operation	FF_{new}
000	No-op	0
001	Add	0
010	Add	0
011	No-op	1
100	No-op	0
101	Subtract	1
110	Subtract	1
111	No-op	1

- Use **D Flip Flop En wo/ SQ** from LogicWorks library
- **FF** is used to store the bit M_{-1} of the Multiplier
 - Initialized to '0' per Modified Booth's Algorithm
 - Becomes the value of FF_{new} during the Shift Phase
- The most significant bit of the accumulator during a shift
 - Different in each part of the lab
- The A/S signal for the adder/subtractor.
 - Add if $A/S = 0$; subtract if $A/S = 1$.

Signals left up to you →



LogicWorks Components

- **19-bit Shift Register & 20-bit Shift Register:**
 - When *LD* is set, the parallel data input is loaded into the shift register;
 - When *SH* is set, the content of the register is shifted 1-bit right, with the most significant bit fed by *SI*.
- **19-bit Adder-Subtractor & 20-bit Adder-Subtractor:**
 - If $A/S=0$ then $Q = A+B$
 - If $A/S=1$ then $Q = A+(-B)$
- **Control Unit:**
 - Handles the multi-cycle process by telling each device what to do on every cycle until the entire process is finished.
 - Every other device simply does what it's told on each cycle.

Your job: add necessary glue logic & connections.

Error Detection and Correction

1. Using knowledge of Hamming Code error correction; extrapolate out the underlying Hamming (7,4) mechanics to that of the Hamming (15,11) algorithm
2. Modify the Hamming (15,11) algorithm and corresponding codewords developed above to be Single Error Correction, Double Error Detection (Extended Hamming Code)
3. Implement this error correction algorithm in Verilog

Verilog Implementation

- Provided Modules:
 - hamming_code.v – Your assignment is to fill this in
 - take as input the Extended Hamming codewords based on Hamming (15,11)
 - output codewords with no errors correctly
 - correct all 1 bit errors and provide this corrected codeword as output
 - output a 1 on the badData output when 2 bit errors are detected in the input codeword
 - hamming_code_tb.v – Fully specified for Hamming (15,11)
 - modify this to support Extended Hamming
 - add more test cases