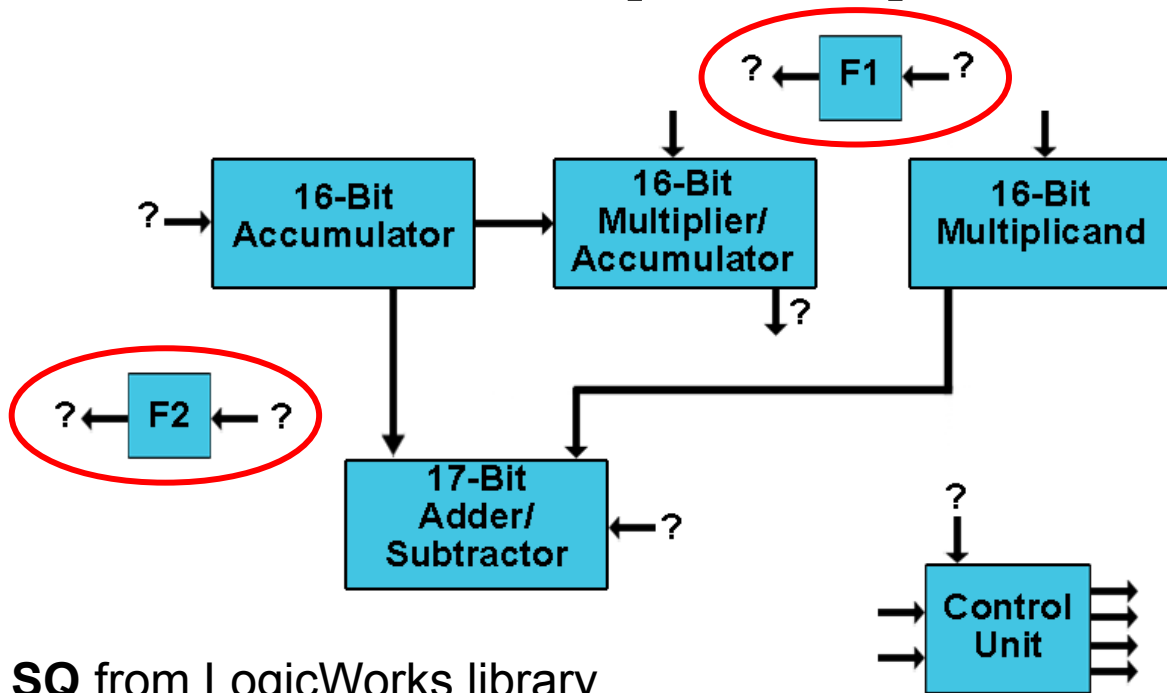


# Overall Guidelines: Flip Flops

| $M_0/F1$ | Operation |
|----------|-----------|
| 00       | No-op     |
| 01       | Add       |
| 10       | Subtract  |
| 11       | No-op     |



- Use **D Flip Flop En wo/ SQ** from LogicWorks library
- F1 Used to hold last-seen LSB of multiplier
  - Initialized to 0 per Booth's Algorithm
- F2 is for you to figure out
  - Use is different in each part of the lab
  - Look at the schematic above carefully...

# Lab 2: Booth's Multiplication

## LogicWorks Implementation

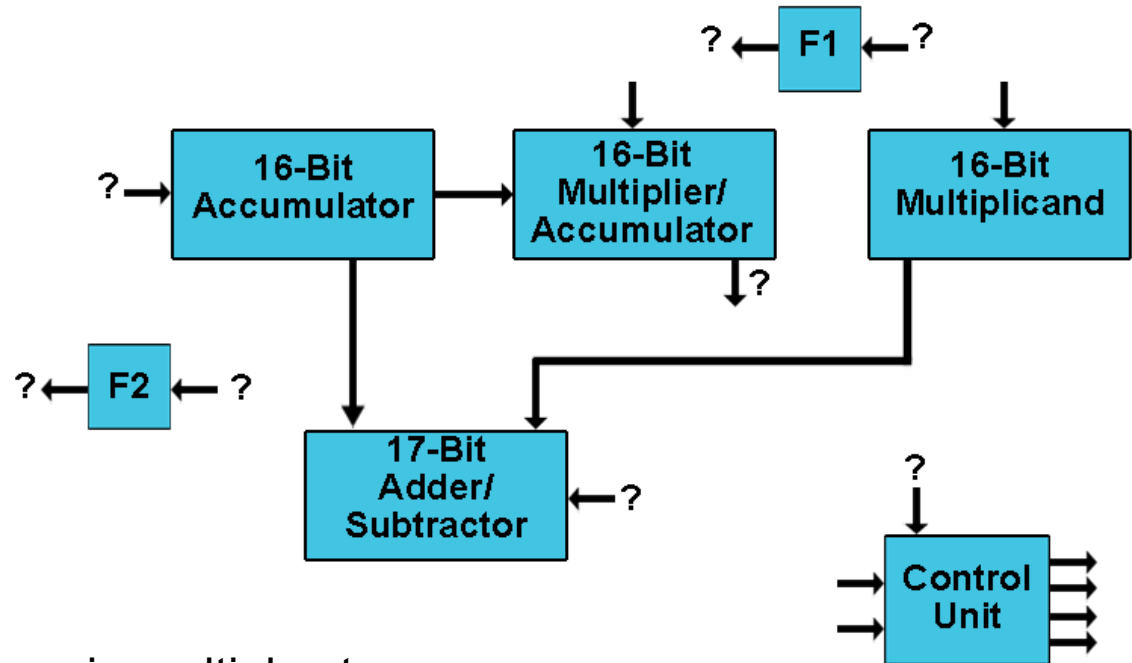
- Part 1: Basic implementation
- Part 2: Optimizations, given input range restrictions
- Part 3: Reintroduce old input range, but use less hardware

## Verilog Implementation

- Re-implement your LogicWorks implementation (Part 3 for full credit, Part 1 or 2 for partial credit) in the Verilog HDL.

# 2's Complement Booth's Multiplication

| $M_0/F1$ | Operation |
|----------|-----------|
| 00       | No-op     |
| 01       | Add       |
| 10       | Subtract  |
| 11       | No-op     |



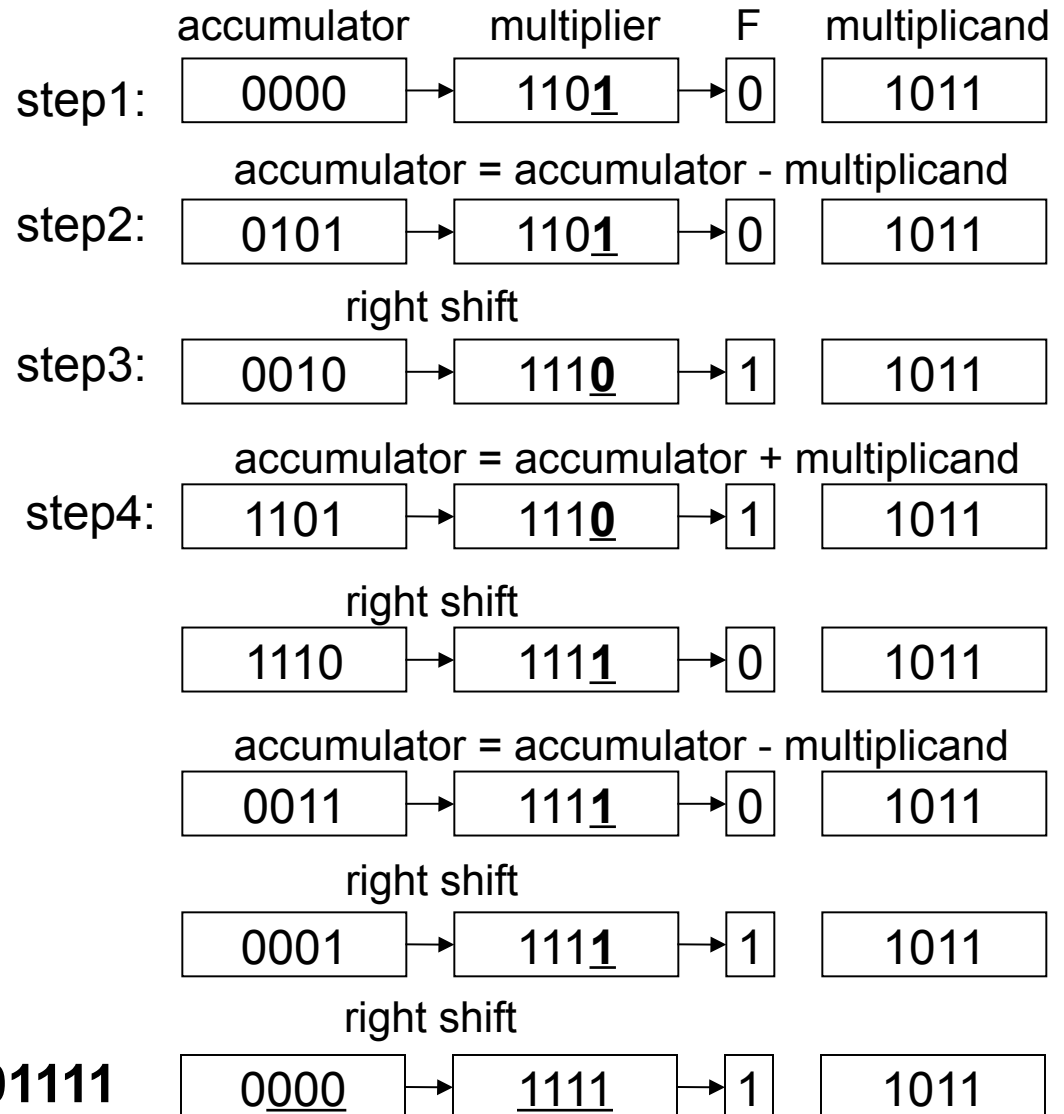
## Basic Idea

- Accumulate a partial sum in multiple steps.
- The rightmost bit of the multiplier and the value of the flip-flop are checked on every step to determine the operation.
- 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 rightmost bit of multiplier, no need to check higher bits;
  - ✓ Lower bits of the partial sum shifted into the multiplier register for storage.

# Booth's Multiplication Example

$$1011_2 * 1101_2$$

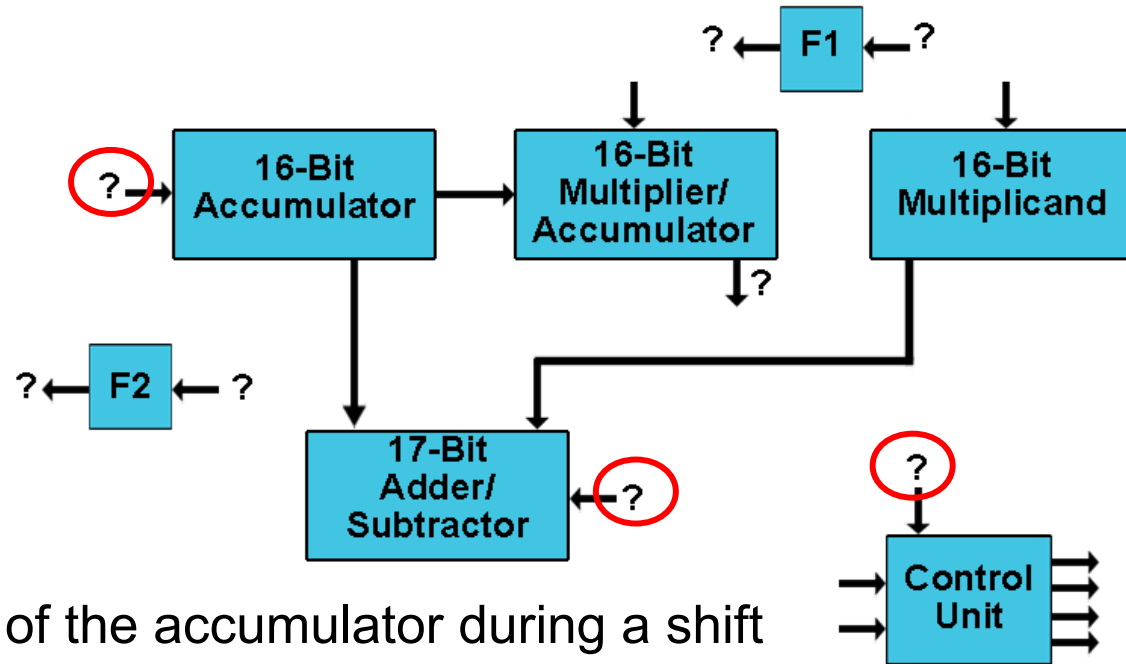
|         |              |
|---------|--------------|
| 1011    | multiplicand |
| 11010   | multiplier   |
| <hr/>   |              |
| 0000101 | (sub 1011)   |
| 001011  | (add 1011)   |
| 00101   | (sub 1011)   |
| <hr/>   |              |
| 0001111 | result       |



**Result: 0001111**

# Overall Guidelines: Glue Logic

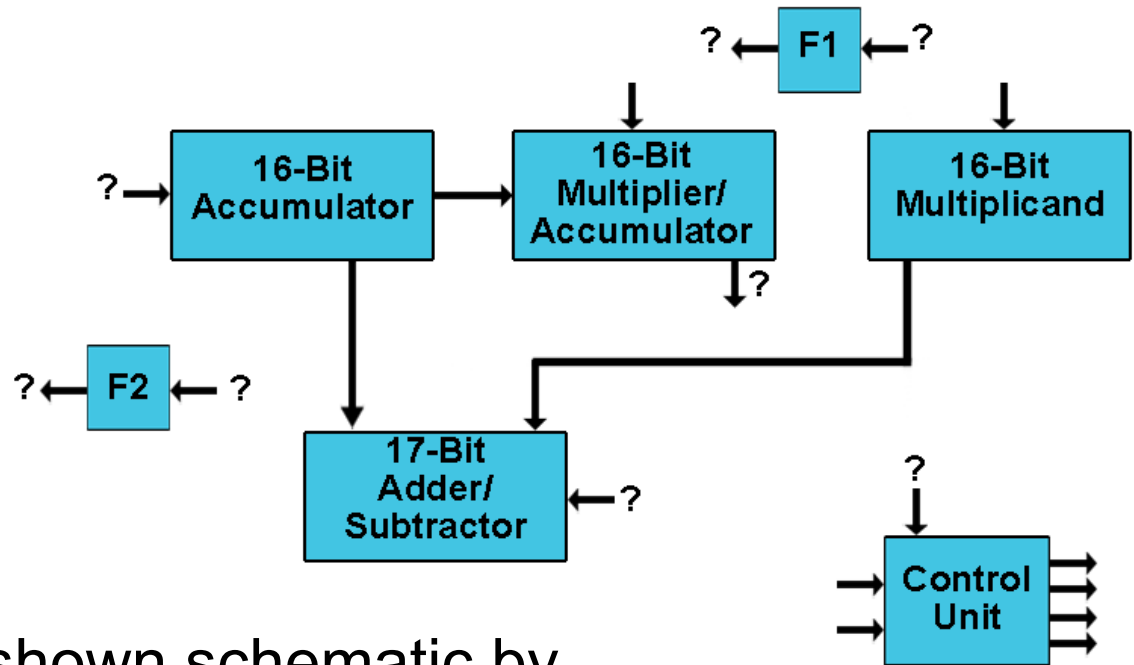
| $M_0/F1$ | Operation |
|----------|-----------|
| 00       | No-op     |
| 01       | Add       |
| 10       | Subtract  |
| 11       | No-op     |



- 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  $A/S = 1$ .
  - You need to drive this signal according to the value of  $F_1$ .
- The *Mult* input of the controller
  - You need to drive it **high** if and only if Booth's algorithm requires an add/subtract operation.

# Overall Guidelines: Flip Flops

| $M_0/F1$ | Operation |
|----------|-----------|
| 00       | No-op     |
| 01       | Add       |
| 10       | Subtract  |
| 11       | No-op     |



- Part 1: Implement shown schematic by completing glue logic and supplying required wiring.
- Part 2: Perform optimizations on Part 1 circuit given that inputs 0 and  $-2^{15}$  can't be inputs.
- Part 3: Fix circuit of Part 2 to ensure it works for input  $-2^{15}$

# LogicWorks Components

- **16-bit Shift Register:**
  - When *LD* is set, the parallel data input is loaded into the shift register;
  - When *SRight* is set, the content of the register is shifted 1-bit right, with the most significant bit fed by *Vlin*.
- **17-bit Adder-Subtractor:**
  - Output  $Q(16:0) = A(15:0) \pm B(15:0)$ ;
    - 16-bit inputs must be sign-extended to produce 17-bit inputs
- **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 logic & connections.**

# Verilog Implementation

- Given Modules:
  - Controller.v – fully specified, no need to modify
  - MultiplierTest.v – tests your implementation. You are not required to modify, but you are encouraged to add your own test cases.
  - Multiplier.v – Your assignment is to fill this in.
- Other Components
  - It is not necessary to implement a separate module for the Shift Register or the Adder/Subtractor, but it may simplify your implementation.