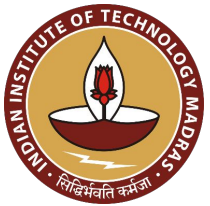
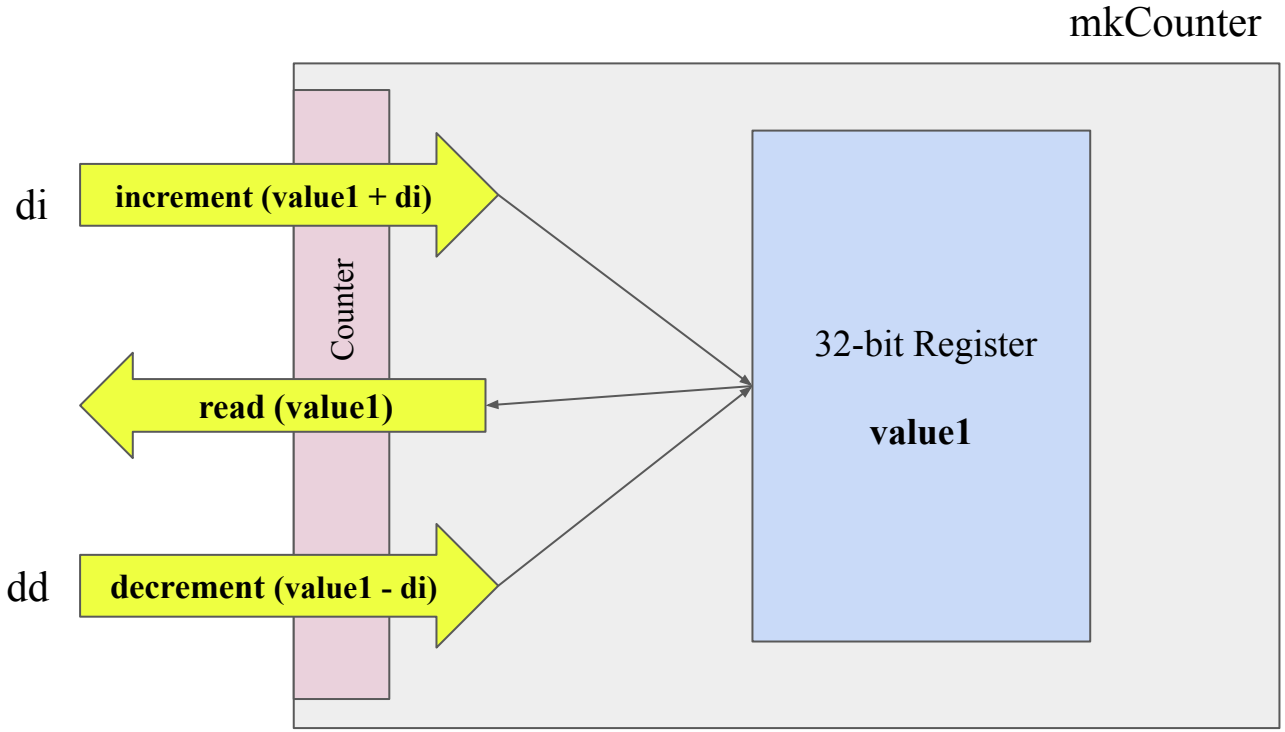


CS6230

BLUESPEC DESIGN DEMO

Up/Down Counter

Counter design



How to handle conflicts while accessing methods?

- Counter operates on int values (32 bits)
- Initial value of Register : value1 = 0
- Pass the increment/decrement values (di/dd) using increment/decrement methods
- Read the register value using read method
- increment, decrement methods : write onto value1 register
- read method : reads from value1 register
- So, cannot call both increment and decrement methods simultaneously in a single rule - will cause conflict
- Then, how to instantiate and use both methods of mkCounter?

Handling method conflicts

```
// Instantiate the counter
Counter myCounter <- mkCounter_v1;

// Register to track the current mode: 0 for counting up, 1 for counting down
Reg#(Bool) countingUp <- mkReg(True);

// Rule to increment the counter by 1 each cycle if counting up
rule up_counter (countingUp && myCounter.read() < 10);
   myCounter.increment(1);
   let value = myCounter.read();
   $display("Counter Value (Up): %0d", value);
   if (value == 9) begin
      countingUp <= False;  // Switch to counting down on next cycle
   end
endrule

// Rule to decrement the counter by 1 each cycle if counting down
rule down_counter (!countingUp && myCounter.read() > 0);
   myCounter.decrement(1);
   let value = myCounter.read();
   $display("Counter Value (Down): %0d", value);
   if (value == 1) begin
      countingUp <= True;  // Optionally reset or end simulation
      // You could end the simulation here with `$finish` or reset to start over
      $finish;
   end
endrule
```