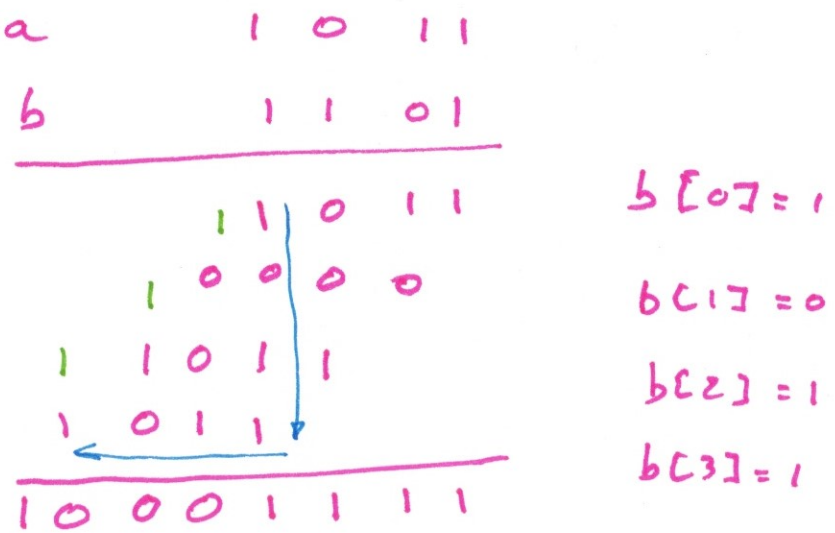
4-Bit MULTIPLY



b[0] = 1

b[1] = 0

b[2] = 1

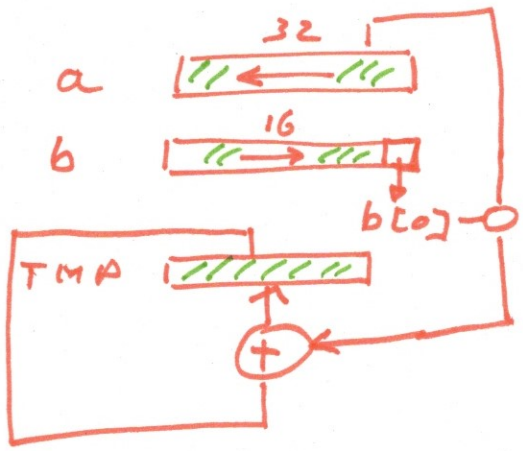
b[3] = 1

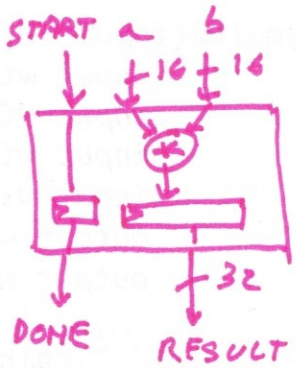
```
module mymult2(input  wire        clk,
               input wire     reset,
               input wire [15:0]  a,
               input wire [15:0]  b,
               input wire     start,
               output wire [31:0] result,
               output wire        done);

   (* multstyle = "logic" *) reg [31:0] r, next_r;
   reg                  d, next_d;

   always @(posedge clk)
     begin
        r <= (reset) ? 32'h0 : next_r;
        d <= (reset) ? 1'b0  : next_d;
     end

   // IMPLEMENTATION 2
   reg [31:0] tmp[15:0];
   reg [31:0] tmpa;
   integer    j;
   always @(*)
     begin
        tmp[0] = b[0] ? {16'b0, a} : 32'b0;
        for (j = 1; j<16; j=j+1)
          begin
          tmpa = {15'b0, a} << j;
          tmp[j] = tmp[j-1] + (b[j] ? tmpa : 32'b0);
        end
        next_r = tmp[15];
        next_d = start;
     end

   assign result = r;
   assign done   = d;

endmodule
```

- PARTIAL PRODUCT ACCUMULATOR
- SHIFTED VERSION OF A

MULT 3

SHFT REGISTER






```
module mymult1(input  wire        clk,
               input wire     reset,
               input wire [15:0]  a,
               input wire [15:0]  b,
               input wire     start,
               output wire [31:0] result,
               output wire    done);

   (* multstyle = "logic" *) reg [31:0] r, next_r;
   reg           d, next_d;

   always @(posedge clk)
     begin
       r <= (reset) ? 32'h0 : next_r;
       d <= (reset) ? 1'd0  : next_d;
     end

   // IMPLEMENTATION 1
   always @(*)
     begin
       next_r = {16'b0,a} * {16'b0,b};
       next_d = start;
     end

   assign done   = d;
   assign result = r;

endmodule
```