Unit 4 Data Path and Controller design (Part 3)

Example 3: Booth's Multiplication

Booth's algorithm is an improvement whereby we can avoid the additions whenever consecutive 0's or 1's are detected in the multiplier → makes process faster.

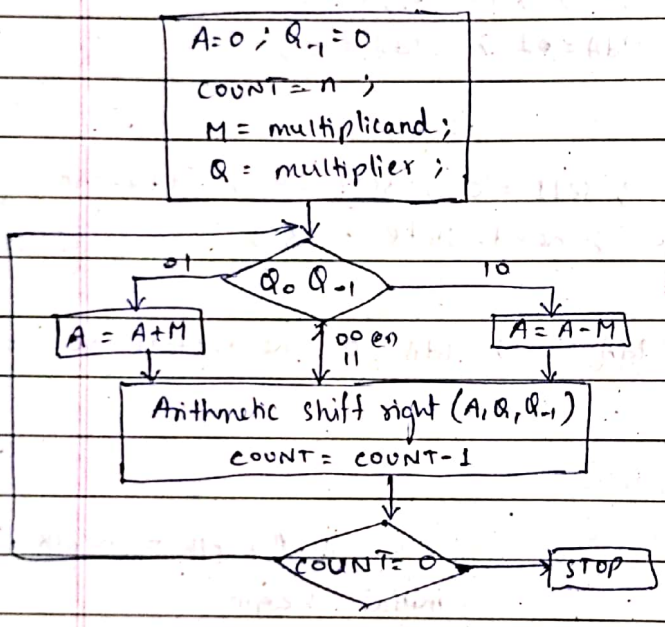
* We inspect bits of the multiplier (Q_i, Q_{i-1}) at a time.

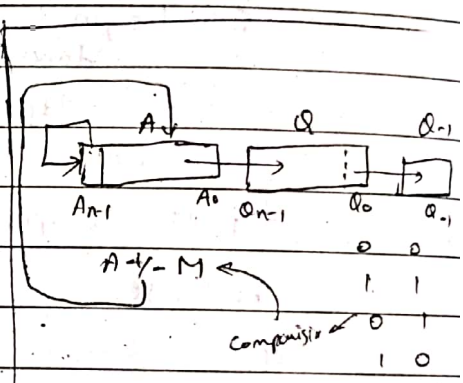
$(Q_i\ Q_{i-1})$	
00 or 11	: only shift the partial product
01	: do addition then shift
10	: do subtraction then shift

* Q_{-1} is assumed to be equal to 0.

Algorithm: Start

M : n-bit multiplicand
Q : n-bit multiplier
A : n-bit temporary register
Q_{-1} : 1 bit flip flop



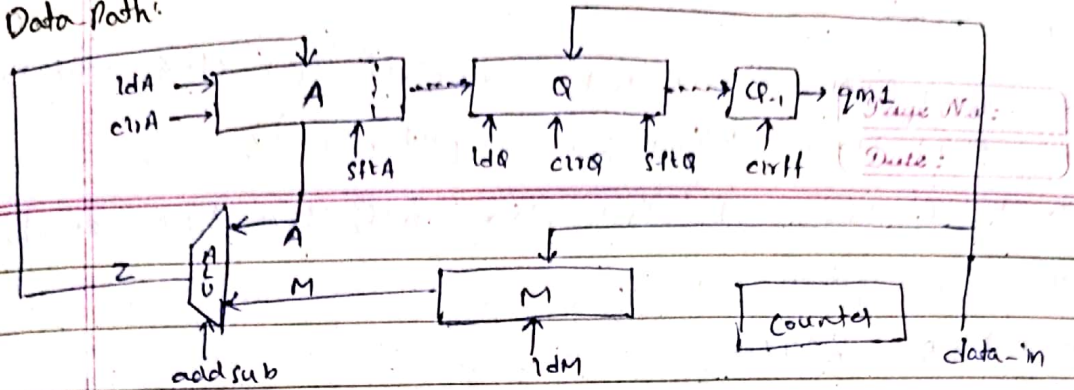


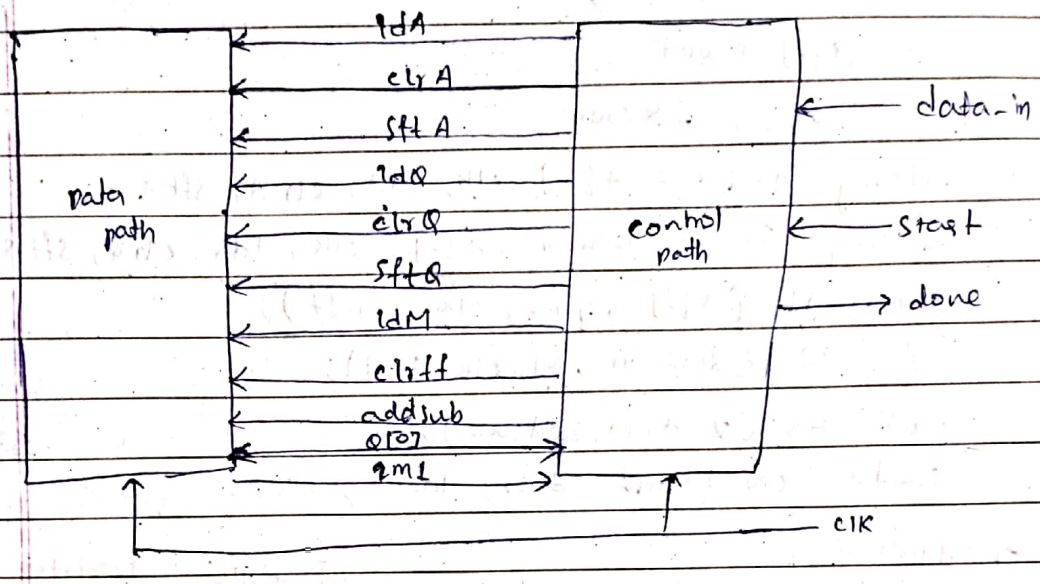
example 1: (-10) x (13)

Assume: 5-bit numbers
M: $(10110)_2$
-M: $(01010)_2$
Q: $(01101)_2$
product = -130
$= (1\,0\,1\,1\,1\,1\,1\,1\,0)_2$

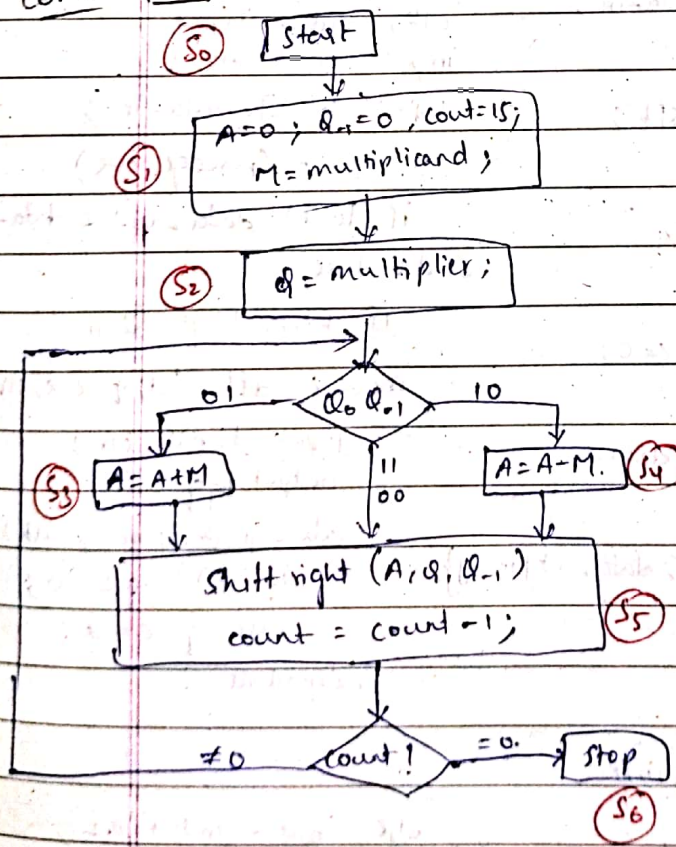
A	Q	Q_{-1}		
0 0 0 0 0	0 1 1 0	1	0	Initialization
0 1 0 1 0	0 1 1 0 1	0		A = A - M
0 0 1 0 1	0 0 1 1 0	1		shift — step 1
1 1 0 1 1	0 0 1 1 0	1		A = A + M
1 1 1 0 1	1 0 0 1 1	0		shift — step 2
0 0 1 1 1	1 0 0 1 1	0		A = A - M
0 0 0 1 1	1 1 0 0 1	1		shift — step 3
0 0 0 0 1	1 1 1 0 0	1		shift — step 4
1 0 1 1 1	1 1 1 0 0	1		A = A + M
1 1 0 1 1	1 1 1 1 0	0		shift — step 5

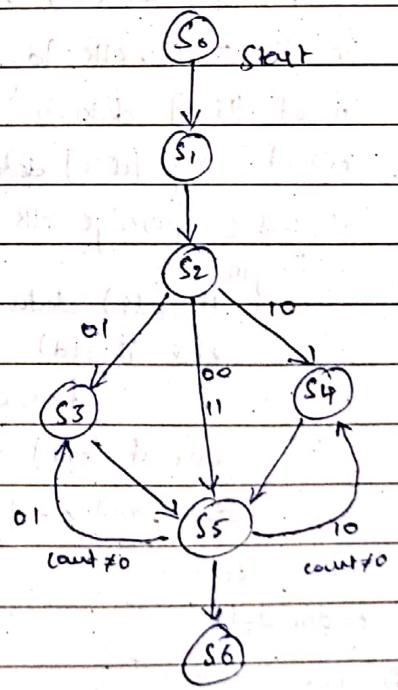
Data Path:





Control path





Data Path :

①
```
module BOOTH (ldA, ldQ, ldM, clrA, clrQ, clrff, sftA, sftQ, addsub,
      decr, ldcnt, data-in , clk , qm1 , eqz);
input ldA, ldQ , ldM, clrA, clrQ, clrff, sftA, sftQ , addsub, clk ;
input [15:0] data-in ;
output qm1 , eqz ;
wire [15:0] A, M, Q, Z ;
wire [4:0] count ;
assign eqz = ~| count ;
Shiftreg AR (A, Z, A[15], clk, ldA, clrA, sftA);
Shiftreg QR (Q, data_in , A[0]; clk, ldQ, clrQ, sftQ);
dff QM1 (Q[0] , qm1 , clk, clrff );
PIPO MR (data-in , M, clk , ldM);
ALU AS (Z, A, M, addsub );
Counter CN (count, decr, ldcnt , clk);
endmodule
```

s_in = serial in

② ~~Shift register~~
```
module shiftreg (data_out, data_in,
  s_in, clk, ld, clr, sft );
input s_in , clk; ld, clr, sft ;
input [15:0] data-in ;
output reg [15:0] data-out ;
always @ (posedge clk)
  begin
    if (clr) data_out <= 0;
    else if (ld)
        data_out <= data-in;
    else if (sft)
    data_out <= { s_in , data_out[15:1]};
  end
endmodule
```

③ PIPO register
```
module PIPO (data_out , data-in ,
  clk, load); ~~input~~
input load , clk ;
input [15:0] data-in ;
always @ (posedge clk)
  if (load) data_out <= data-in;
endmodule
```

④ Flip flop (D-ff)
```
module dff (d, q , clk, clr);
  input d, clk, clr ;
  output reg q ;
  always @ (posedge clk)
    if (clr) q <= 0 ;
    else q <= d ;
endmodule
```

⑤ ALU design
```
module ALU (out, in1, in2, addsub);
  input [15:0] in1, in2 ;
  input addsub;
  output reg [15:0] out ;
  always @ (*)
    begin
      if (addsub == 0) out = in1 - in2;
      else out = in1 + in2 ;
    end
endmodule
```

⑥ counter

```
module counter (data_out, decr, ldcnt, clk);
  input decr, clk;
  output [4:0] data_out;

  always @(posedge clk)
    begin
      if (ldcnt) data_out <= 5'b10000;
      else if (decr) data_out <= data_out - 1;
    end
endmodule
```

The control Path

```
module controller (
input clk, q0, qm1, start;
output reg ldA, clrA, sftA, ldQ, clrQ, sftQ, ldM, clrff,
           addsub, decr, ldcnt, done);

reg [2:0] state;
parameter s0 = 3'b000, s1 = 3'b001, s2 = 3'b010, s3 = 3'b011,
          s4 = 3'b100, s5 = 3'b101, s6 = 3'b110;

always @(posedge clk)
  begin
    case (state)
      s0 : if (start) state <= s1;
      s1 : state <= s2;
      s2 : #2 if ({q0, qm1} == 2'b01) state <= s3;
           else if ({q0, qm1} == 1'b10) state <= s4;
           else state <= s5;
      s3 : state <= s5;
      s4 : state <= s5;
      s5 : #2 if ((({q0, qm1} == 2'b01) && !eq2)
                state <= s3;
              else if ((({q0, qm1} == 2'b10) && !eq2)
                state <= s4;
              else if (eq2) state <= s6;
      s6 : state <= s6;
      default : state <= s0;
    endcase
end
```

```
always @(state)
  begin
    case (state)
    S0: begin clrA=0; ldA=0; sftA=0; clrQ=0; ldQ=0;
              sftQ=0; ldM=0; clrff=0; done=0; end
    S1: begin clrA=1; clrff=1; ldcnt=1, ldM=1; end
    S2: begin clrA=0; clrff=0; ldcnt=0; ldM=0; ldQ=1; end
    S3: begin ldA=1; addsub=1; ldQ=0; sftA=0; sftQ=0;
              decr=0; end
    S4: begin ldA=1; addsub=0; ldQ=0; sftA=0; sftQ=0;
              decr=0; end
    S5: begin sftA=1; sftQ=1; ldA=0; ldQ=0; decr=1; end
    S6: done=1;
    default: begin clrA=0; sftA=0; ldQ=0; sftQ=0; end
    endcase
endmodule
```