Chapter 8(2)

곱셈/나눗셈 회로

곱셈기 - 카운터를 사용한 상태도 사용

```
module controller (clk, reset, start, m0, load, shift, addshift, sub, ready);
    input clk, reset, start, m0;
    output load, shift, addshift, sub, ready;

    reg state;
    reg [1:0] count;
    localparam S0 = 0, S1 = 1;

    always @ (posedge clk, posedge reset)   // State transitions
        if (reset) begin state <= S0; count <= 0; end
        else
            case (state)
             S0: if (start) begin state <= S1; count <= 3; end
             S1: if (count==0) state <= S0; else count <= count-1;
            endcase

 // control output logic
    assign load = (state==S0) && start;
    assign shift = (state==S1) && ~m0;
    assign addshift = (state==S1) && m0;
    assign sub = (state==S1) && (count==0);
    assign ready = (state==S0) && ~reset;
endmodule
```

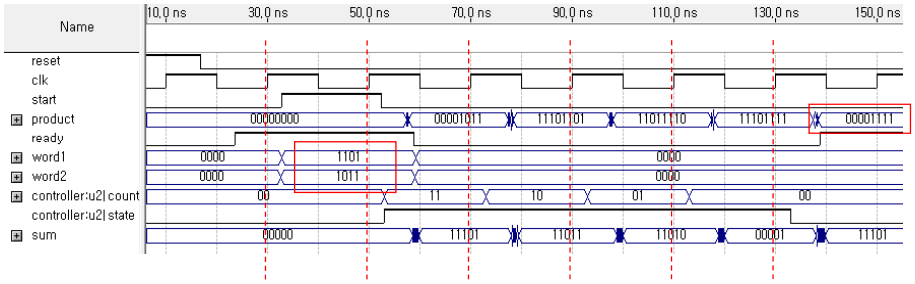
count를 사용하여
4 cycle의 S1 상태를 유지


```
module smul(clk, reset, start, word1, word2, product, ready, sum);
    input clk, reset, start;
    input [3:0] word1, word2;
    output [7:0] product;
    output ready;
    output [4:0] sum;

    wire  m0, load, shift, addshift, sub;

    datapath u1 (clk, reset, load, shift, addshift, sub, word1, word2, product, m0, sum);
    controller u2 (clk, reset, start, m0, load, shift, addshift, sub, ready);
endmodule
```



```
module datapath (clk, reset, load, shift, addshift, sub, word1, word2, product, m0, sum);
    input clk, reset, load, shift, addshift, sub;
    input [3:0] word1, word2;
    output [7:0] product;
    output m0;
    output [4:0] sum;

    reg [7:0] product;      // 9-bit
    reg [3:0] multiplicand; // 5-bit
    wire[4:0] sum;          // 5-bit
    wire [4:0] eproduct, emcand;

    assign m0 = product[0];

    assign eproduct = {product[7],product[7:4]};
    assign emcand = {multiplicand[3],multiplicand};
    assign sum = sub ? (eproduct - emcand) : (eproduct + emcand);
```

```
    always @ (posedge clk or posedge reset) begin
        if (reset) begin multiplicand <= 0; product <= 0; end
        else if (load) begin
            multiplicand <= word1;
            product <= {4'b0, word2};
        end
        else if (shift)     // signed shift-right
            product <= {product[7], product[7:1]};
        else if (addshift)  // signed add & shift right
            product <= {sum, product[3:1]};
    end
endmodule
```

(-3) * (-5) = 15

1101 * 1011 = 00001111

부호없는 정수의 나눗셈기

```
module datapath (clk, reset, load, shift, subshift, word1, word2, quotient, remainder, lt);
    input clk, reset, load, shift, subshift;
    input [7:0] word1;
    input [3:0] word2;
    output [3:0] quotient, remainder;
    output lt;

    reg [7:0] dividend;     // 8-bit
    reg [3:0] divisor;      // 4-bit
    wire[4:0] diff;         // 4-bit
    wire lt;                // less than (borrow)

    assign diff = dividend[7:3] - divisor;  // subtract after shift left
    assign lt = diff[4];    // borrow

    assign quotient = dividend[3:0];
    assign remainder = dividend[7:4];
```



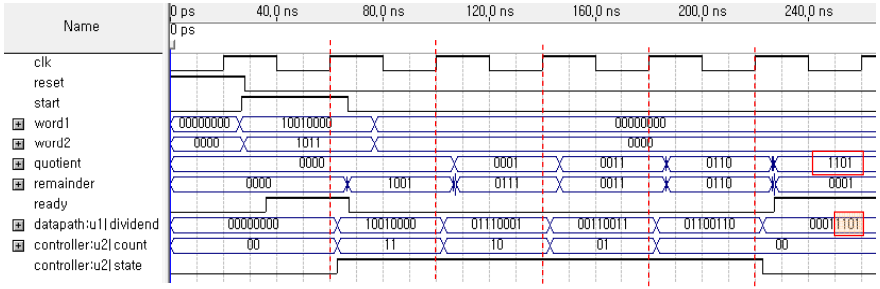
```
    always @ (posedge clk or posedge reset) begin
        if (reset) begin dividend <= 0; divisor <= 0; end
        else if (load) begin
            dividend <= word1;
            divisor <= word2;
        end
        else if (shift)     // shift-left
            dividend <= {dividend[6:0], 1'b0};      // quotient=0
        else if (subshift)  // sub & shift-left
            dividend <= {diff[3:0], dividend[2:0], 1'b1};   // quotient=1
    end

endmodule
```

```
module controller (clk, reset, start, lt, load, shift, subshift, ready);
    input clk, reset, start, lt;
    output load, shift, subshift, ready;

    reg overflow;
    reg state;
    reg [1:0] count;
    localparam S0 = 0, S1 = 1;

    always @ (posedge clk or posedge reset)     // State transitions
        if (reset) begin state <= S0; count <= 0; end
        else
            case (state)
             S0: if (start) begin state <= S1; count <= 3; end
             S1: if (count==0) state <= S0;
                 else count <= count - 1;
            endcase

  // control output logic
    assign load = (state==S0) && start;
    assign shift = (state==S1) && lt;
    assign subshift = (state==S1) && ~lt;
    assign ready = (state==S0) && ~reset;
endmodule
```



```
module div(clk, reset, start, word1, word2, quotient, remainder, ready);
    input clk, reset, start;
    input [7:0] word1;
    input [3:0] word2;
    output [3:0] quotient, remainder;
    output ready;

    wire  load, shift, subshift, lt;

    datapath u1 (clk, reset, load, shift, subshift, word1, word2, quotient, remainder, lt);
    controller u2 (clk, reset, start, lt, load, shift, subshift, ready);
endmodule
```

- overflow 처리 ?



144 / 11 = 13 ... 1

10010000 / 1011 = 1101 ... 0001

부호 있는 정수의 나눗셈기

```
module datapath (clk, reset, load, shift, subshift, ready, word1, word2, quotient,
                 remainder, lt);
    input clk, reset, load, shift, subshift, ready;
    input [7:0] word1;
    input [3:0] word2;
    output [3:0] quotient, remainder;
    output lt;

    reg [7:0] dividend;     // 8-bit
    reg [3:0] divisor;      // 4-bit
    reg sign;               // different sign flag of word1 and word2
    wire [4:0] diff;        // 5-bit
    wire lt;                // less than
    wire [4:0] edivisor;
    wire [4:0] edividend;
    wire [4:0] diff2;       // 5-bit

    assign edivisor = {divisor[3], divisor};    // sign extension of divisor
    assign diff = (dividend[7]^divisor[3]) ?
                  (dividend[7:3] + edivisor) : (dividend[7:3] - edivisor);
    assign lt = (dividend[7]^diff[4]) && (diff != 4'b0); // dividend[7:4] < divisor

    assign quotient = sign ? -dividend[3:0] : dividend[3:0];
    assign remainder =  dividend[7:4];
```

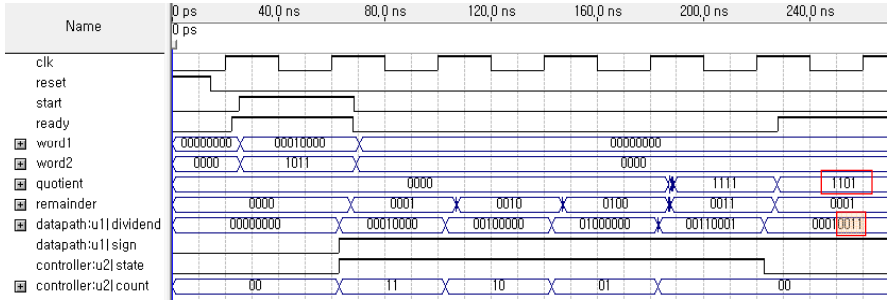
```
    always @ (posedge clk or posedge reset) begin
        if (reset) begin dividend <= 0; divisor <= 0; end
        else if (load) begin
            dividend <= word1;
            divisor <= word2;
            sign <= word1[7] ^ word2[3];
        end
        else if (shift)     // shift left, quotient=0
            dividend <= {dividend[6:0], 1'b0};
        else if (subshift)  // save result & shift left, quotient=1
            dividend <= {diff[3:0], dividend[2:0], 1'b1};
    end
endmodule
```



```
module controller (clk, reset, start, lt, load, shift, subshift, ready);
    input clk, reset, start, lt;
    output load, shift, subshift, ready;

    reg overflow;
    reg state;
    reg [1:0] count;
    localparam S0 = 0, S1 = 1;

    always @ (posedge clk or posedge reset) begin   // State transitions
        if (reset) begin state <= S0; count <= 0; end
        else
            case (state)
             S0: if (start) begin state <= S1; count <= 3; end
             S1: if (count==0) state <= S0;
                 else count <= count-1;
            endcase
    end
 // control output logic
    assign load = (state==S0) && start;
    assign shift = (state==S1) && lt;
    assign subshift = (state==S1) && ~lt;
    assign ready = (state==S0) && ~reset;
endmodule
```



```
module sdiv(clk, reset, start, word1, word2, quotient, remainder, ready);
    input clk, reset, start;
    input [7:0] word1;
    input [3:0] word2;
    output [3:0] quotient, remainder;
    output ready;

    wire  load, shift, subshift, sub, lt;

    datapath u1 (clk, reset, load, shift, subshift, ready, word1, word2, quotient,
                 remainder, lt);
    controller u2 (clk, reset, start, lt, load, shift, subshift, ready);
endmodule
```

16 / (-5) = -3 ... 1

00010000 / 1011 = 1101 ... 0001