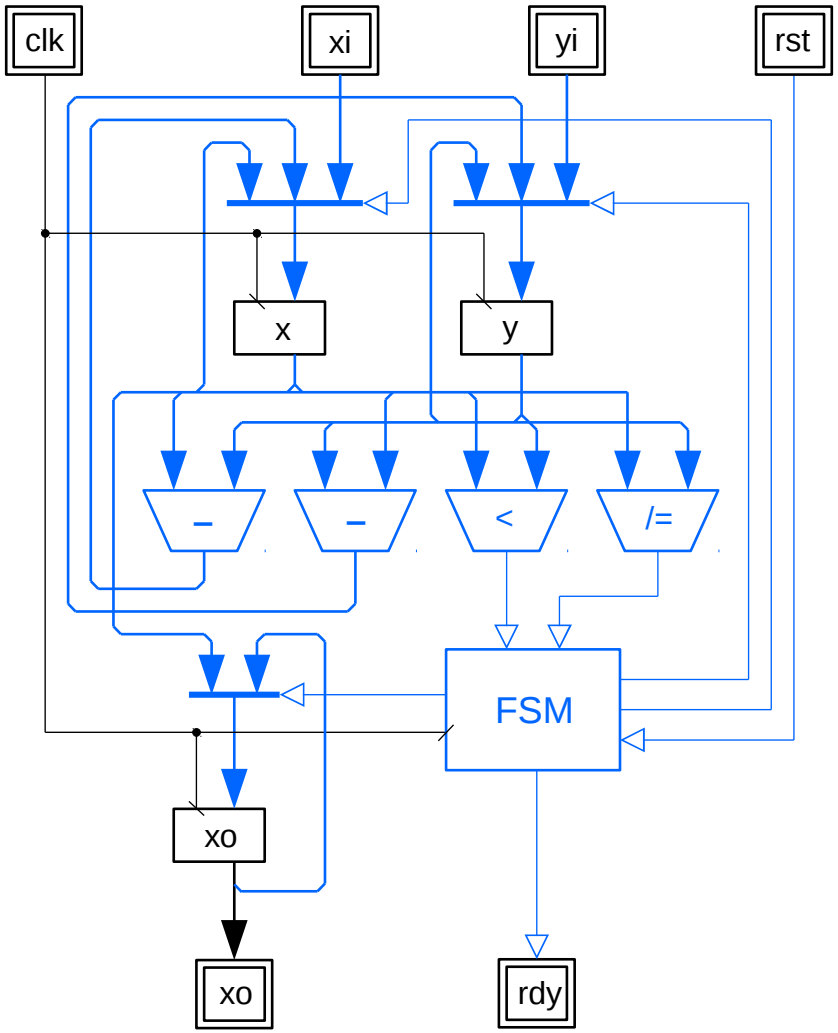
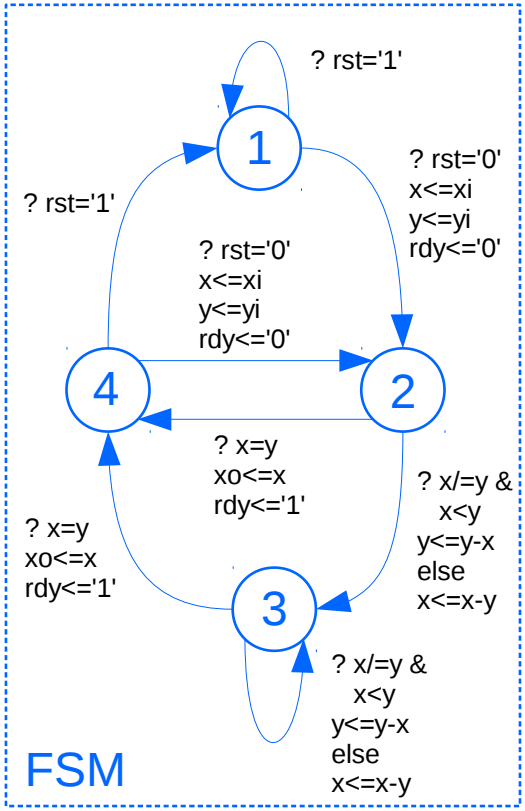
Clocked behavioral (gcd-bhvc.sv)

```
module gcd (
  input logic unsigned [15:0] xi, yi,
  input logic rst,
  output logic unsigned [15:0] xo,
  output logic rdy,
  input logic clk );

  logic unsigned [15:0] x, y;

  always begin
    // Wait for the new input data
    while ( rst == 1 ) @(posedge clk);

    x = xi;    y = yi;    rdy = 0;
    @(posedge clk);

    // Calculate
    while ( x != y ) begin
      if ( x < y )  y = y - x;
      else          x = x - y;
      @(posedge clk);
    end

    // Ready
    xo = x;    rdy = 1;
    @(posedge clk);
  end

endmodule
```



Only registers (signals) are described explicitly.

The rest (datapath & FSM) are described implicitly – subject for interpretations by synthesizers.

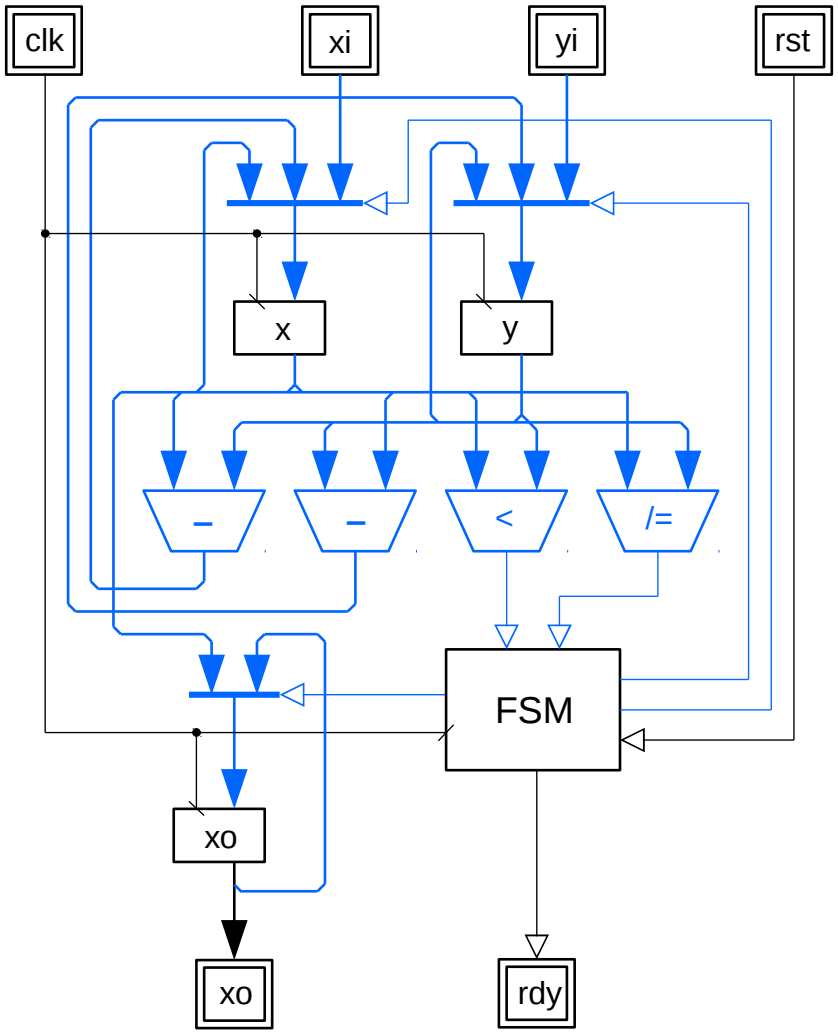


2 subtracters, 2 comparators
[1 clock step per iteration]

ASIC: 961 e.g. / 20.0 ns
FPGA: not synthesizable

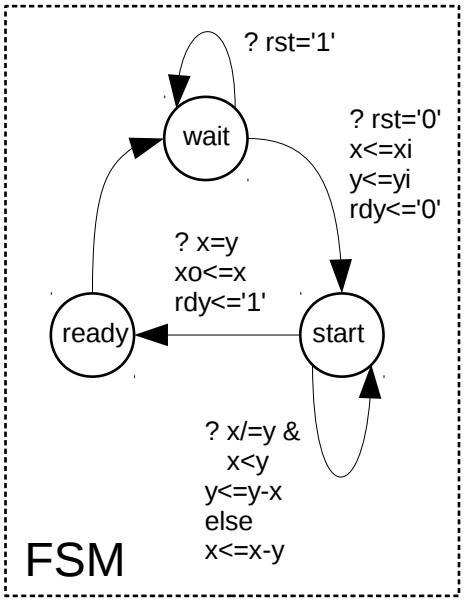
Behavioral state machine (gcd-bfsm.sv)

```
module gcd (
  input logic unsigned [15:0] xi, yi,
  input logic rst,
  output logic unsigned [15:0] xo,
  output logic rdy,
  input logic clk );

  // State - type & signals
  enum {S_wait, S_start, S_comp, S_ready}
    state, next_state;

  logic unsigned [15:0] x, y;

  always @(posedge clk)
    case (state)
      // Wait for the new input data
      S_wait :
        if (rst==0) begin
          x <= xi;  y <= yi;  rdy <= 0;
          state <= S_start;
        end
      // Calculate
      S_start :
        if ( x != y ) begin
          if ( x < y )  y <= y - x;
          else          x <= x - y;
          state <= S_start;
        end
        else begin
          xo <= x;  rdy <= 1;
          state <= S_ready;
        end
      // Ready
      default : state <= S_wait;
    endcase

endmodule
```



Registers (signals) and FSM are described explicitly.

The datapath is described implicitly – subject for interpretations by synthesizers.



2 subtracters, 2 comparators
[1 clock step per iteration]

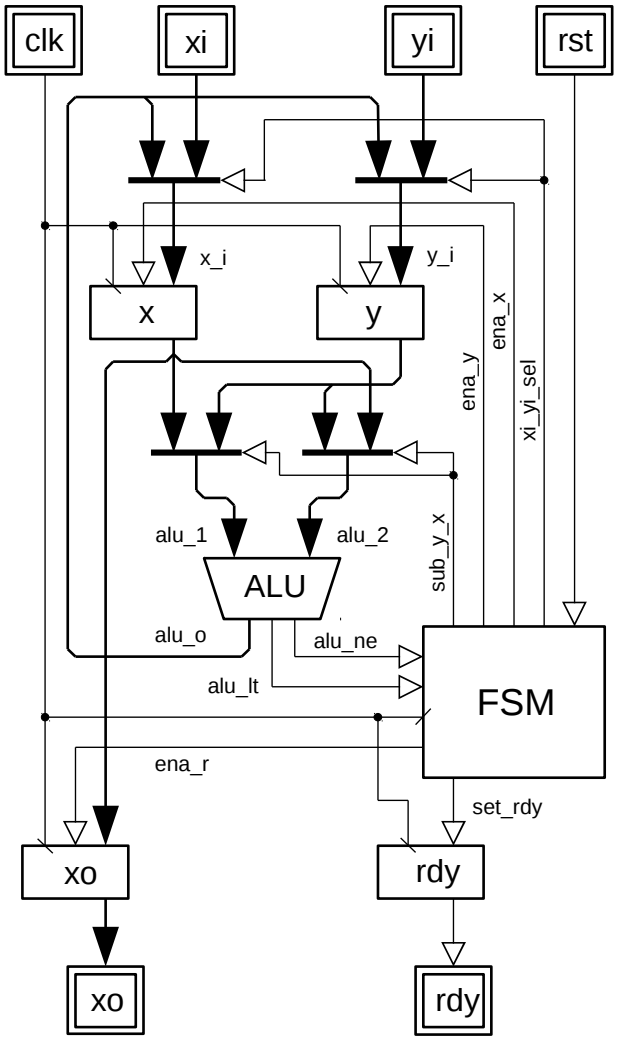
ASIC: 911 e.g. / 19.4 ns
FPGA: 108 SLC / 9.9 ns

RTL #1: (gcd-rtl1.sv)

```
module gcd ( /* ... */ );
  // State - type & signals
  enum {S_wait, S_start, S_comp, S_sub_x_y, S_sub_y_x, S_ready}
    state, next_state;
  logic unsigned [15:0] x, y;
  logic unsigned [15:0] alu_1, alu_2, alu_o, x_i, y_i;
  logic alu_lt, alu_ne, ena_x, ena_y, ena_r, set_rdy;
  logic xi_yi_sel, sub_y_x;

  // Next state function of the FSM
  always @(state, rst, alu_ne, alu_lt) begin
    ena_x <= 0;    ena_y <= 0;    ena_r <= 0;
    set_rdy <= 0;    xi_yi_sel <= 0;    sub_y_x <= 0;
    next_state <= state;
    case (state)
      // Wait for the new input data
      S_wait :
        if (rst==0) begin
          xi_yi_sel <= 1;    ena_x <= 1;    ena_y <= 1;
          next_state <= S_start;
        end
      // Loop: ready?
      S_start :
        if (alu_ne==1) next_state <= S_comp;
        else           next_state <= S_ready;
      // Loop: compare
      S_comp :
        if (alu_lt==1) next_state <= S_sub_y_x;
        else           next_state <= S_sub_x_y;
      // Loop: y-x
      S_sub_y_x : begin
        ena_y <= 1;    sub_y_x <= 1;    next_state <= S_start;
      end
      // Loop: x-y
      S_sub_x_y : begin
        ena_x <= 1;    sub_y_x <= 0;    next_state <= S_start;
      end
      // Ready
      default : begin
        ena_r <= 1;    set_rdy <= 1;    next_state <= S_wait;
      end
    endcase
  end

  // ALU: subtract / less-than / not-equal
  assign alu_o = alu_1 - alu_2;
  assign alu_lt = alu_o[15];
  assign alu_ne = | alu_o;
  // Multiplexers
  assign x_i = (xi_yi_sel==1) ? xi : alu_o;
  assign y_i = (xi_yi_sel==1) ? yi : alu_o;
  assign alu_1 = (sub_y_x==1) ? y : x;
  assign alu_2 = (sub_y_x==1) ? x : y;

  // Registers
  always @(posedge clk) begin
    state <= next_state;
    if (ena_x==1) x <= x_i;
    if (ena_y==1) y <= y_i;
    if (ena_r==1) xo <= x;
    rdy <= set_rdy;
  end
endmodule
```

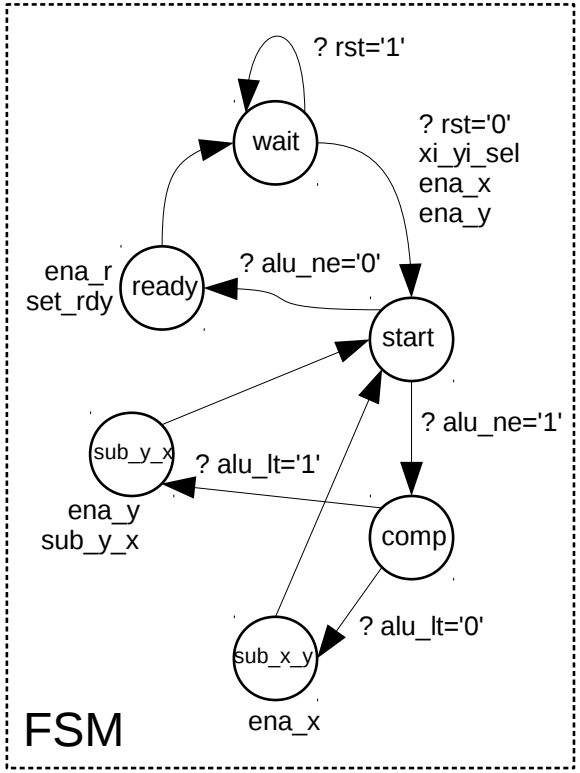
Single ALU (“–”, “<”, “/=”)
[3 clock steps per iteration]

ASIC: 986 e.g. / 19.8 ns
FPGA: 50 SLC / 10.8 ns



Register-transfer level description with universal ALU (operation reuse).

Default value for the control signals is '0'

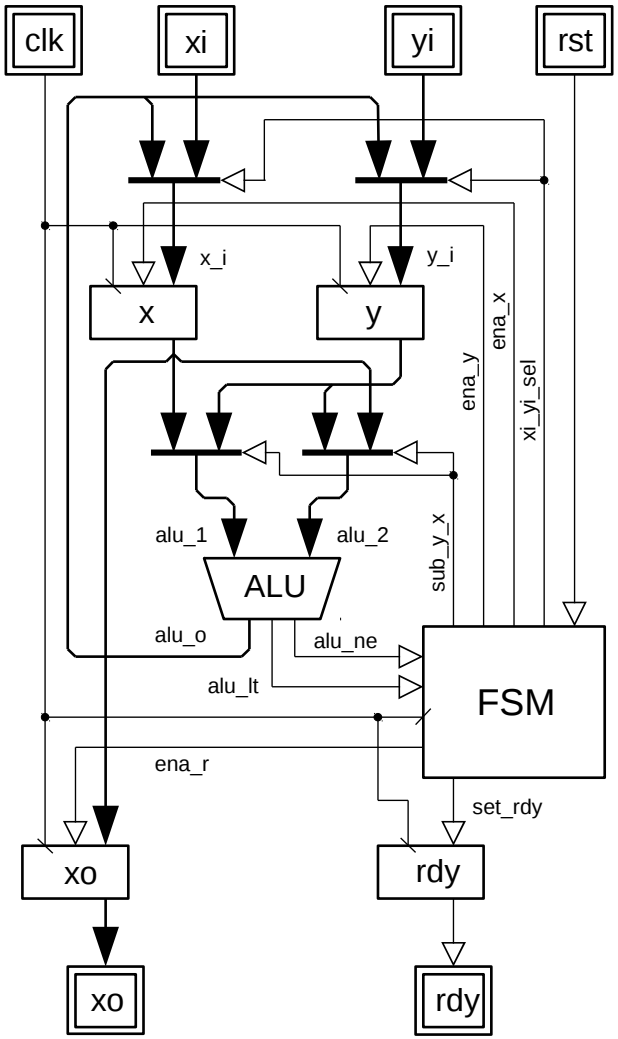


RTL #2: (gcd-rtl2.sv)

```
module gcd ( /* ... */ );
  // State - type & signals
  enum {S_wait, S_start, S_sub_x_y, S_sub_y_x, S_ready} state, next_state;

  logic unsigned [15:0] x, y;
  logic unsigned [15:0] alu_1, alu_2, alu_o, x_i, y_i;
  logic alu_lt, alu_ne, ena_x, ena_y, ena_r, set_rdy;
  logic xi_yi_sel, sub_y_x;

  // Next state function of the FSM
  always @(state, rst, alu_ne, alu_lt) begin
    ena_x <= 0;    ena_y <= 0;    ena_r <= 0;
    set_rdy <= 0;    xi_yi_sel <= 0;    sub_y_x <= 0;
    next_state <= state;
    case (state)
      // Wait for the new input data
      S_wait :
        if (rst==0) begin
          xi_yi_sel <= 1;    ena_x <= 1;    ena_y <= 1;
          next_state <= S_start;
        end
      // Loop: ready?
      S_start :
        if (alu_ne==1)
          if (alu_lt==1) next_state <= S_sub_y_x;
          else           next_state <= S_sub_x_y;
        else             next_state <= S_ready;
      // Loop: y-x
      S_sub_y_x : begin
        ena_y <= 1;    sub_y_x <= 1;    next_state <= S_start;
      end
        // Loop: x-y
      S_sub_x_y : begin
        ena_x <= 1;    sub_y_x <= 0;    next_state <= S_start;
      end
      // Ready
      default : begin
        ena_r <= 1;    set_rdy <= 1;    next_state <= S_wait;
      end
    endcase
  end

  // ALU: subtract / less-than / not-equal
  assign alu_o = alu_1 - alu_2;
  assign alu_lt = alu_o[15];
  assign alu_ne = | alu_o;
  // Multiplexers
  assign x_i = (xi_yi_sel==1) ? xi : alu_o;
  assign y_i = (xi_yi_sel==1) ? yi : alu_o;
  assign alu_1 = (sub_y_x==1) ? y : x;
  assign alu_2 = (sub_y_x==1) ? x : y;

  // Registers
  always @(posedge clk) begin
    state <= next_state;
    if (ena_x==1) x <= x_i;
    if (ena_y==1) y <= y_i;
    if (ena_r==1) xo <= x;
    rdy <= set_rdy;
  end
endmodule
```

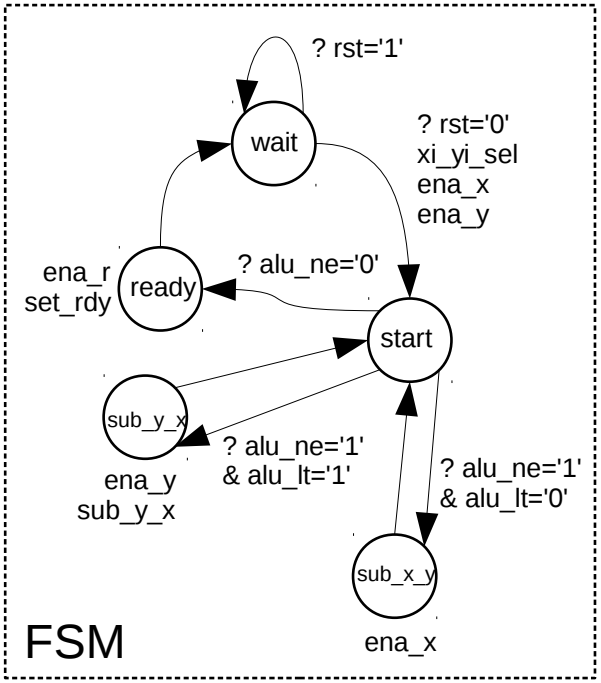
Single ALU (“–”, “<”, “/=”)
[2 clock steps per iteration]

ASIC: 931 e.g. / 19.9 ns
FPGA: 48 SLC / 10.8 ns



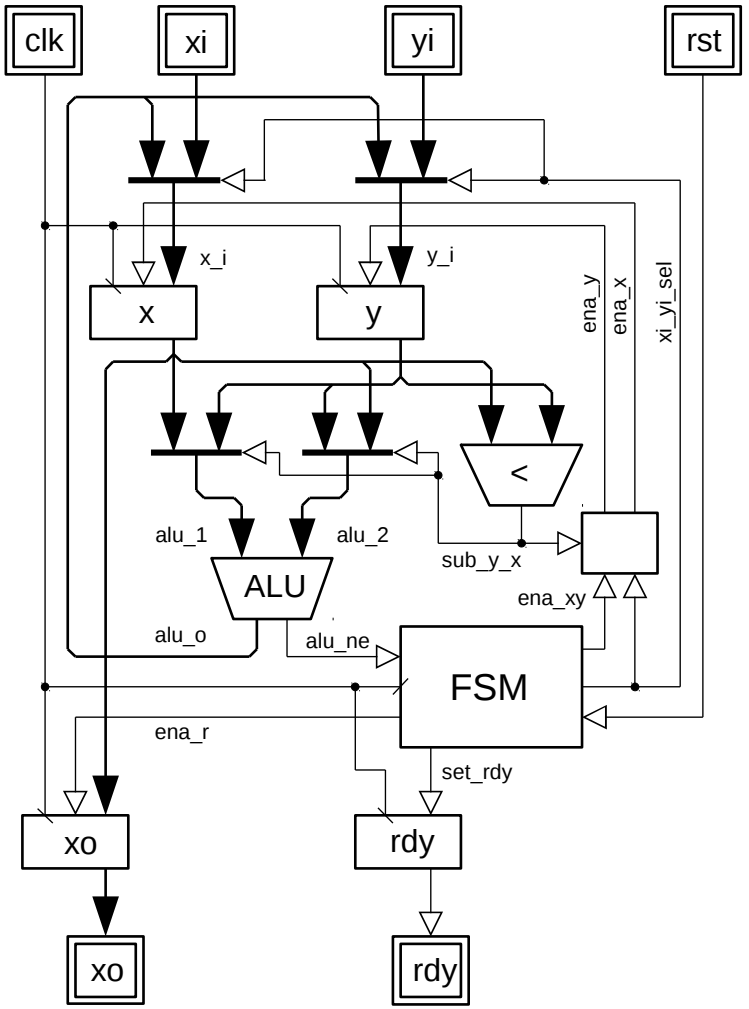
Register-transfer level description with universal ALU (operation reuse).

Default value for the control signals is '0'.



RTL #3: (gcd-rtl3.sv)

```
module gcd ( /* ... */ );
  // State - type & signals
  enum {S_wait, S_start, S_ready} state, next_state;

  logic unsigned [15:0] x, y;
  logic unsigned [15:0] alu_1, alu_2, alu_o, x_i, y_i;
  logic alu_ne, ena_xy, ena_x, ena_y, ena_r, set_rdy;
  logic xi_yi_sel, sub_y_x;

  // Next state function of the FSM
  always @(state, rst, alu_ne) begin
    ena_xy <= 0;    ena_r <= 0;    set_rdy <= 0;    xi_yi_sel <= 0;
    next_state <= state;
    case (state)
      // Wait for the new input data
      S_wait :
        if (rst==0) begin
          xi_yi_sel <= 1;    ena_xy <= 1;     next_state <= S_start;
        end
      // Calculate
      S_start :
        if (alu_ne==1) begin  ena_xy <= 1;  next_state <= S_start;  end
        else begin  ena_r <= 1;  set_rdy <= 1;  next_state <= S_ready; end
      // Ready
      default : begin
        ena_r <= 1;    set_rdy <= 1;    next_state <= S_wait;
      end
    endcase
  end

  // Comparator (less-than)
  assign sub_y_x = ( x < y ) ? 1 : 0;

  // Subtracter (+not-equal)
  assign alu_o = alu_1 - alu_2;
  assign alu_ne = | alu_o;

  // Multiplexers
  assign x_i = (xi_yi_sel==1) ? xi : alu_o;
  assign y_i = (xi_yi_sel==1) ? yi : alu_o;
  assign alu_1 = (sub_y_x==1) ? y : x;
  assign alu_2 = (sub_y_x==1) ? x : y;
  assign ena_x = ((sub_y_x==0 & ena_xy==1) | xi_yi_sel==1) ? 1 : 0;
  assign ena_y = ((sub_y_x==1 & ena_xy==1) | xi_yi_sel==1) ? 1 : 0;

  // Registers
  always @(posedge clk) begin
    state <= next_state;
    if (ena_x==1) x <= x_i;
    if (ena_y==1) y <= y_i;
    if (ena_r==1) xo <= x;
    rdy <= set_rdy;
  end

endmodule
```

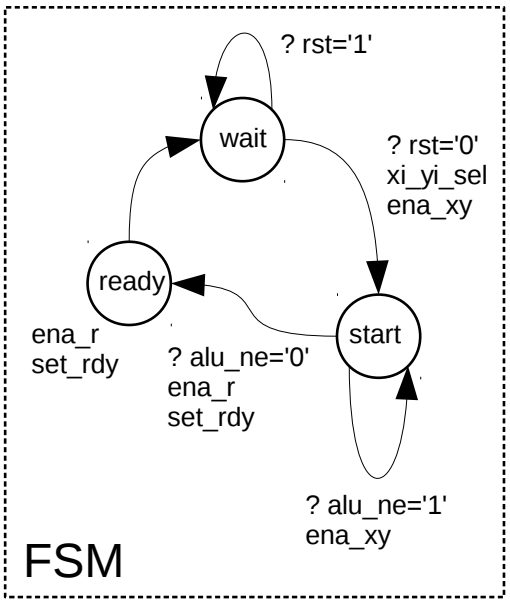


Register-transfer level description with comparator controlling subtractions.

1 ALU (“–”, “/=”), 1 comparator
[1 clock step per iteration]

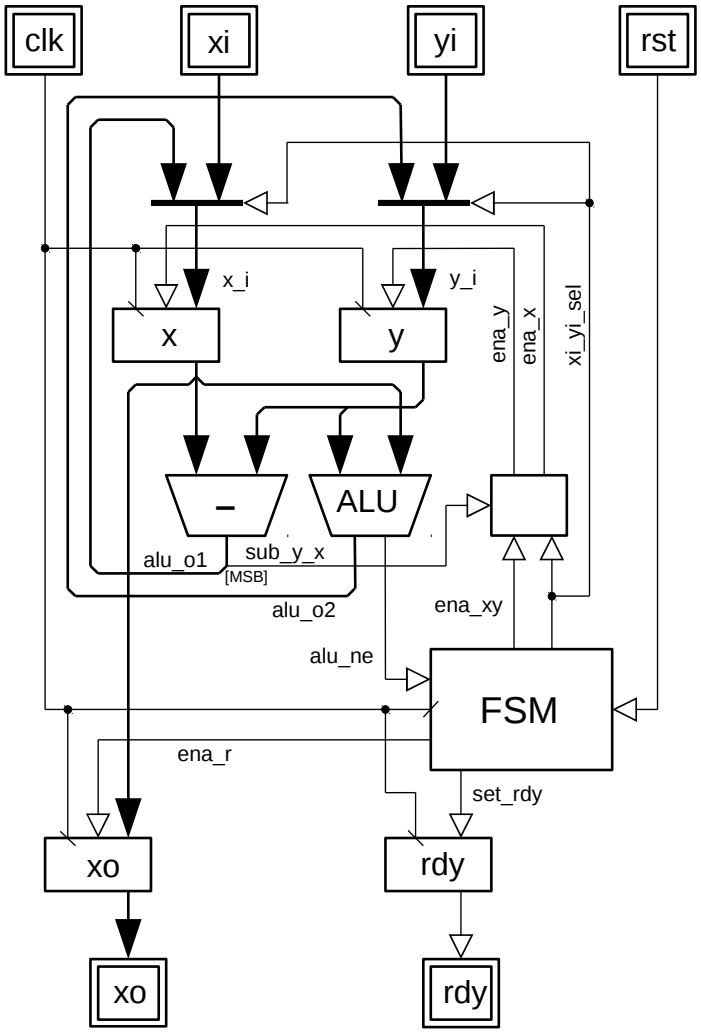
ASIC: 1134 e.g. / 20.0 ns
FPGA: 58 SLC / 17.0 ns

Default value for the control signals is '0'.



RTL #4: (gcd-rtl4.vhdl)

```
module gcd ( /* ... */ );
  // State - type & signals
  enum {S_wait, S_start, S_ready} state, next_state;

  logic unsigned [15:0] x, y;
  logic unsigned [15:0] alu_o1, alu_o2, x_i, y_i;
  logic alu_ne, ena_xy, ena_x, ena_y, ena_r, set_rdy;
  logic xi_yi_sel, sub_y_x;

  // Next state function of the FSM
  always @(state, rst, alu_ne) begin
    ena_xy <= 0;    ena_r <= 0;    set_rdy <= 0;    xi_yi_sel <= 0;
    next_state <= state;
    case (state)
      // Wait for the new input data
      S_wait :
        if (rst==0) begin
          xi_yi_sel <= 1;    ena_xy <= 1;     next_state <= S_start;
        end
      // Calculate
      S_start :
        if (alu_ne==1) begin  ena_xy <= 1;   next_state <= S_start;  end
        else begin  ena_r <= 1;  set_rdy <= 1;  next_state <= S_ready;  end
      // Ready
      default : begin
        ena_r <= 1;    set_rdy <= 1;    next_state <= S_wait;
      end
    endcase
  end

  // Subtracter (x-y) / comparator (x<y)
  assign alu_o1 = x - y;
  assign sub_y_x = alu_o1[15];

  // Subtracter (y-x) / comparator (y!=x)
  assign alu_o2 = y - x;
  assign alu_ne = | alu_o2;

  // Multiplexers
  assign x_i = (xi_yi_sel==1) ? xi : alu_o1;
  assign y_i = (xi_yi_sel==1) ? yi : alu_o2;
  assign ena_x = ((sub_y_x==0 & ena_xy==1) | xi_yi_sel==1) ? 1 : 0;
  assign ena_y = ((sub_y_x==1 & ena_xy==1) | xi_yi_sel==1) ? 1 : 0;

  // Registers
  always @(posedge clk) begin
    state <= next_state;
    if (ena_x==1) x <= x_i;
    if (ena_y==1) y <= y_i;
    if (ena_r==1) xo <= x;
    rdy <= set_rdy;
  end

endmodule
```

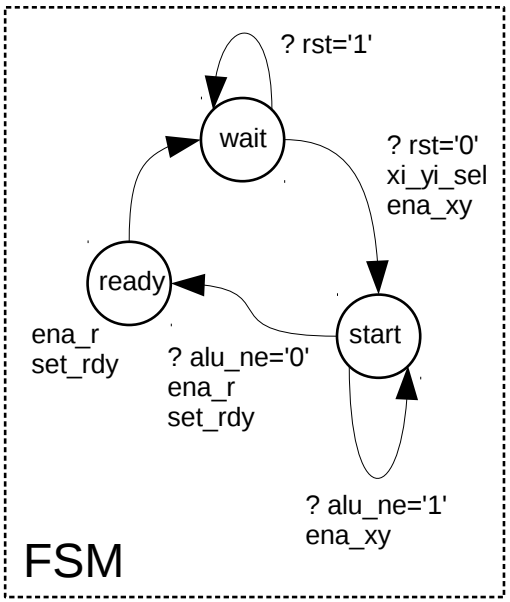


Register-transfer level description with out-of-order subtractions. Only data-path differs from RTL #3.

1 subtracter, 1 ALU (“–”, “/=”)
[1 clock step per iteration]

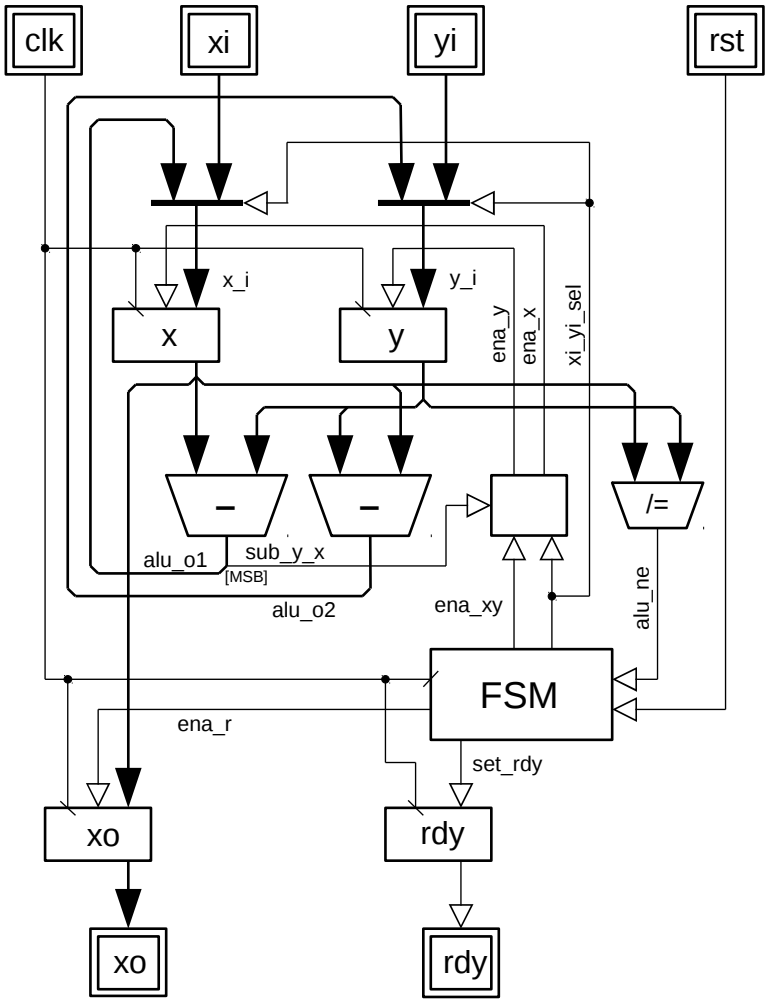
ASIC: 976 e.g. / 19.9 ns
FPGA: 78 SLC / 12.6 ns

Default value for the control signals is '0'.



RTL #5: (gcd-rtl5.vhdl)

```
module gcd ( /* ... */ );
  // State - type & signals
  enum {S_wait, S_start, S_ready} state, next_state;

  logic unsigned [15:0] x, y;
  logic unsigned [15:0] alu_o1, alu_o2, x_i, y_i;
  logic alu_ne, ena_xy, ena_x, ena_y, ena_r, set_rdy;
  logic xi_yi_sel, sub_y_x;

  // Next state function of the FSM
  always @(state, rst, alu_ne) begin
    ena_xy <= 0;    ena_r <= 0;    set_rdy <= 0;    xi_yi_sel <= 0;
    next_state <= state;
    case (state)
      // Wait for the new input data
      S_wait :
        if (rst==0) begin
          xi_yi_sel <= 1;    ena_xy <= 1;     next_state <= S_start;
        end
      // Calculate
      S_start :
        if (alu_ne==1) begin  ena_xy <= 1;   next_state <= S_start;  end
        else begin  ena_r <= 1;  set_rdy <= 1;  next_state <= S_ready;  end
      // Ready
      default : begin
        ena_r <= 1;    set_rdy <= 1;    next_state <= S_wait;
      end
    endcase
  end

  // Subtracter (x-y) / comparator (x<y)
  assign alu_o1 = x - y;
  assign sub_y_x = alu_o1[15];

  // Subtracter (y-x)
  assign alu_o2 = y - x;

  // Comparator (y!=x)
  assign alu_ne = ( x != y ) ? 1 : 0;

  // Multiplexers
  assign x_i = (xi_yi_sel==1) ? xi : alu_o1;
  assign y_i = (xi_yi_sel==1) ? yi : alu_o2;
  assign ena_x = ((sub_y_x==0 & ena_xy==1) | xi_yi_sel==1) ? 1 : 0;
  assign ena_y = ((sub_y_x==1 & ena_xy==1) | xi_yi_sel==1) ? 1 : 0;

  // Registers
  always @(posedge clk) begin
    state <= next_state;
    if (ena_x==1) x <= x_i;
    if (ena_y==1) y <= y_i;
    if (ena_r==1) xo <= x;
    rdy <= set_rdy;
  end

endmodule
```



Register-transfer level description with out-of-order subtractions. Only data-path differs from RTL #3 & #4.

2 subtracters, 1 comparator
[1 clock step per iteration]

ASIC: 915 e.g. / 20.0 ns
FPGA: 58 SLC / 8.0 ns

Default value for the control signals is '0'.