Greatest Common Divisor (GCD)

Task:

Implement a GCD algorithm that is able to handle any combination of 8-bit (sign bit included) numbers. Use two's complement format to represent negative values. Provide the circuit with an interface for repetitive data input (using buttons and switches) and result output (using LEDs). The design should be developed in accordance with the same methodology as described in the example below:

- prepare the flowchart of the algorithm that is based on the selected GCD computation method
- prepare GSA and state transition table for the control part of the design
- develop a suitable datapath

Simulate and implement the design on FPGA development board.

Greatest Common Divisor Example

The greatest common divisor of two non-zero integers is the largest positive integer that divides both numbers without remainder. One way to compute GCD is to use Euclidean algorithm. The flowchart of Euclidean algorithm is presented on Figure 1. For this example it is assumed that input operands are unsigned 8-bit numbers and none of them is zero.

GCD circuit includes a control unit (FSM), which is complemented with a datapath. On this stage the datapath consists of two registers (RG1 and RG2), multiplexers for selecting the source of data to be stored in these registers and comparator logic. The remainder computation is treated as “black box”.

As most of the basic FPGAs do not feature any dedicated logic for performing division, it must be synthesized separately. A simple division algorithm is presented on Figure 2. At each step it checks whether divisor, which is multiplied by corresponding power of two (starting from the maximum possible), fits into dividend. If it doesn’t, then the dividend is

restored.

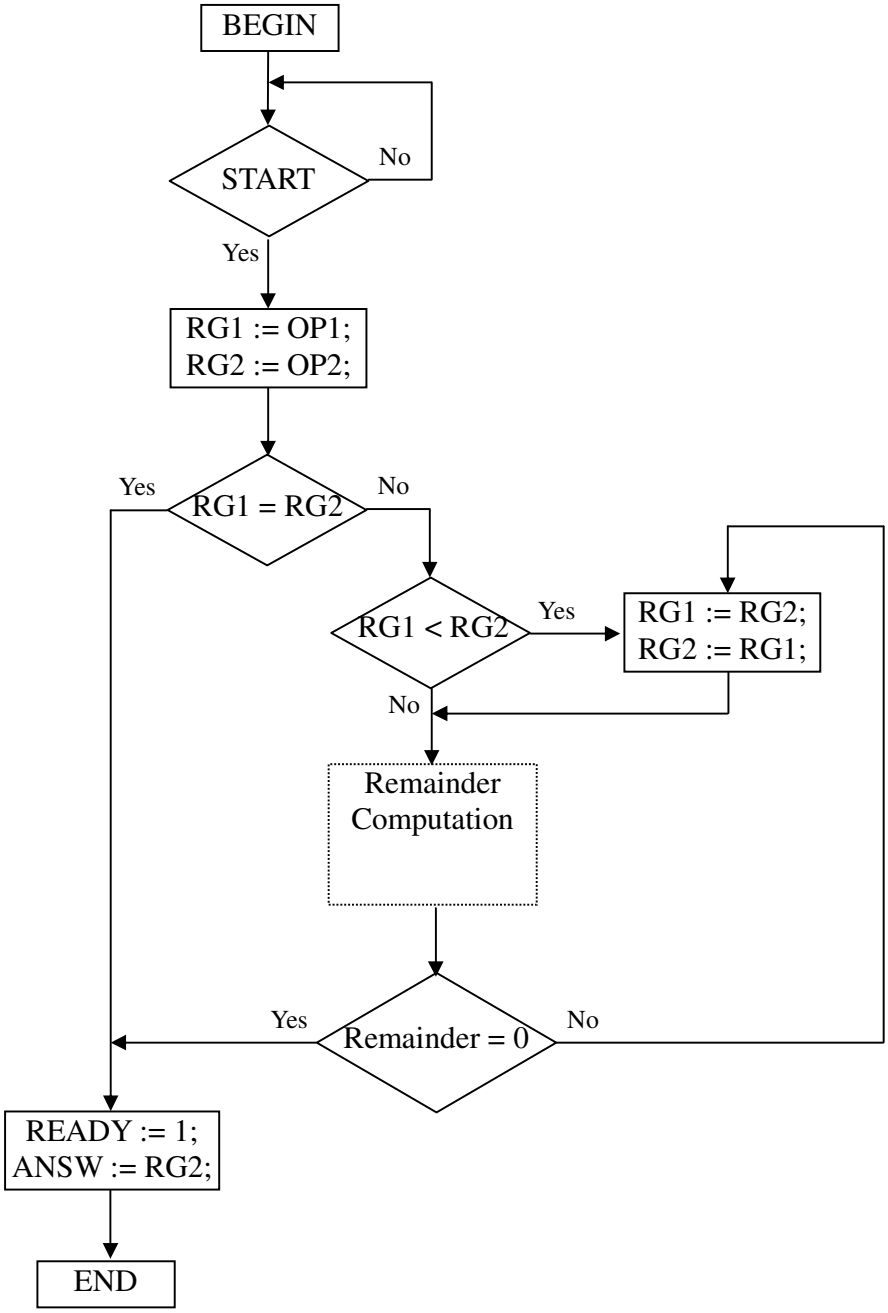

Figure 1: GCD Algorithm

Register RG1 is loaded with dividend, while register RG2 – with divisor. At first the divisor is normalized. As the quotient is of no interest, its computation is omitted from the algorithm. After the final run, RG1 should contain the remainder, while divisor in RG2 is

back to normal. Note, that RG1 is a 9-bit registers, as it should include a sign bit as well.

Remainder computation algorithm invokes additional elements in the datapath: Arithmetic Logic Unit (ALU) and up/down counter. ALU is, in a sense, a custom block, its functionality is design specific. For this particular algorithm ALU should handle addition, subtraction and right/left shift operations, as well as comparison. The block scheme of the complete datapath is presented on Figure 3.

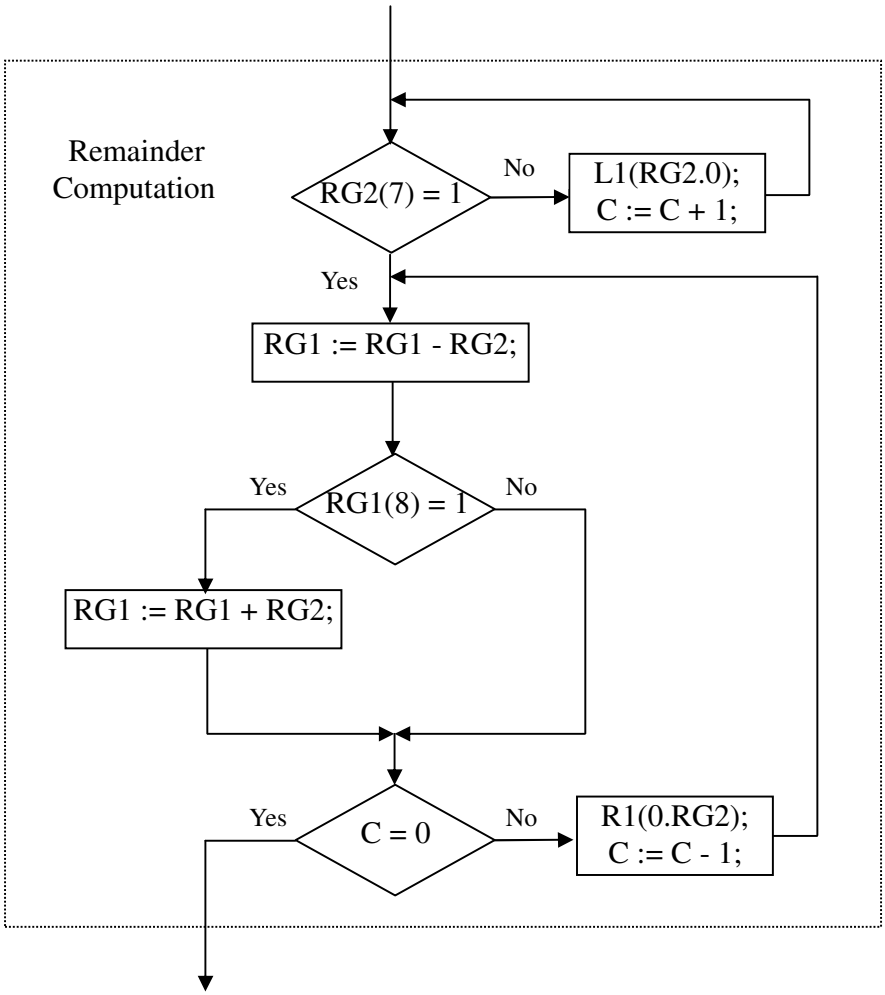

Figure 2: Remainder Computation Algorithm

Elements of the datapath are controlled via eleven (Y10-Y0) signals – a control word. Y0, Y1 and Y2 are enable signals for the registers (RG1 and RG2) and the counter. Y3 signal

selects the counting direction. Y7-Y4 control the multiplexers, which feed data to registers RG1 and RG2. Y8 and Y9 select the ALU operation (addition, subtraction or right/left shift). Y10 enables the ANSW output, when the result is ready.

The control unit gets feedback from datapath via six (X6-X1) signals. X1 indicates whether the content of RG1 equals the content of RG2. X2 signals whether the value stored in RG1 is greater than the value of RG2. X3 is the most significant bit of RG2. X4 is the sign bit of RG1. Signals X5 and X6 indicate when content of counter and RG1 equal zero.

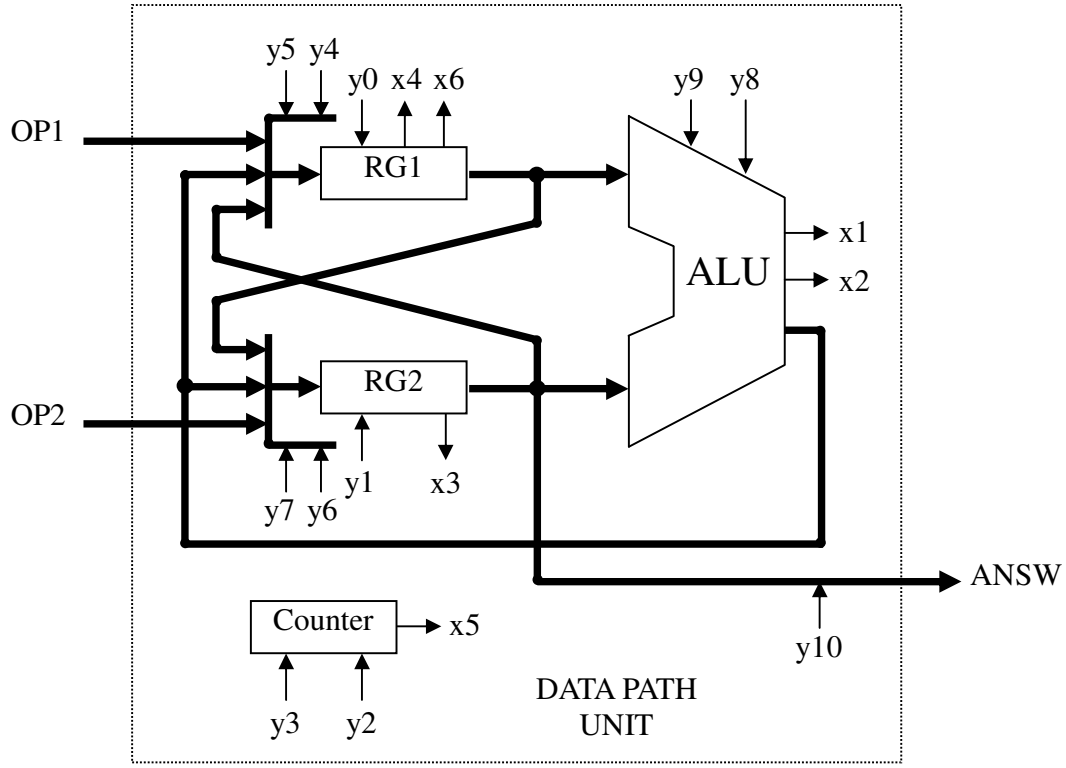

Figure 3: Datapath Of GCD Circuit

At this stage it is possible to represent the control unit by means of graph-scheme of algorithm (GSA). In this GSA the computational statements (actions of ALU and counter) are replaced with the corresponding control signals (Y-s) and the conditions - with binary conditions signals (X-s). Simultaneously executed statements are grouped into common blocks. The resultant GSA is presented on Figure 4. It can be synthesized as either Moore or Mealy FSM to form a control unit for the GCD circuit.

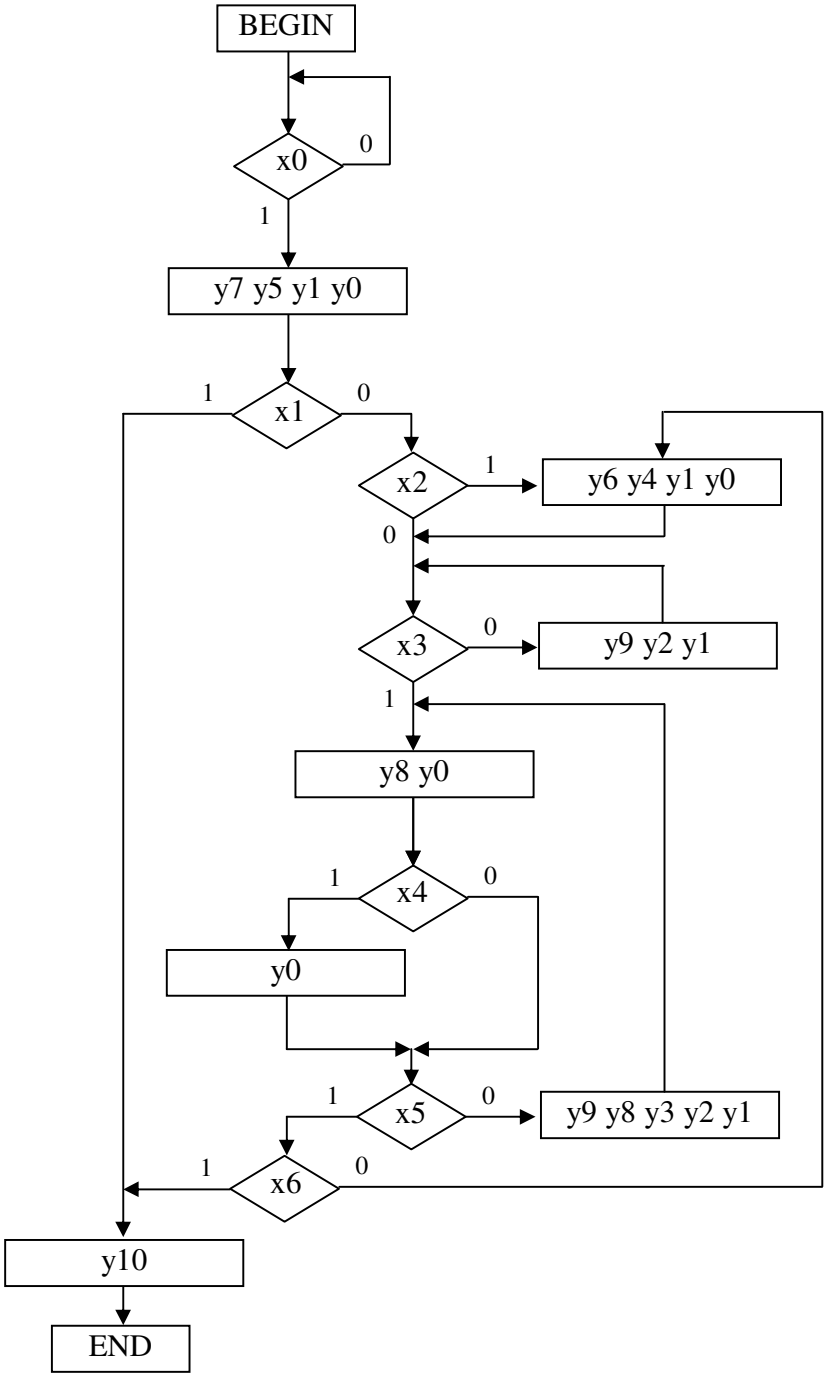

Figure 4: GSA of GCD Algorithm

VHDL descriptions of the ALU and up/down counter are provided in Listing 1 and Listing 2, respectively. ALU is a purely combinational circuit, thus it reacts to the change on any of its inputs. It performs all the necessary arithmetical/logical operation (addition, subtraction

and right/left shift) and does the comparison of the operands. The counter can both decrement and increment. The counter should definitely be implemented as a signal, because its value must be available to other processes (components in the datapath).

Listing 1: VHDL Description of ALU

```
process (OP1, OP2, op_sel)
begin

case op_sel is
  when "00"   => result <= std_logic_vector(unsigned(OP1) + unsigned(OP2));
  when "01"   => result <= std_logic_vector(unsigned(OP1) - unsigned(OP2));
  when "10"   => result <= OP2(7 downto 0)&'0';
  when "11"   => result <= '0'&OP2(8 downto 1);
  when others => result <= (others => '0');
end case;

if OP1 = OP2 then
  equal <= '1';
else
  equal <=  '0';
end if;

if OP1 < OP2 then
  greater <= '1';
else
  greater <=  '0';
end if;

end process;
```

Listing 2: VHDL Description of Up/Down Counter

```
process (clk)
begin
if clk'event and clk = '1' then
  if counter_en = '1' then
    if count_direction = '1' then
      counter_value <= std_logic_vector(unsigned(counter_value) - 1);
    else
      counter_value <= std_logic_vector(unsigned(counter_value) + 1);
    end if;
  end if;
end if;
end process;
```