VHDL to Circuit Schematics

Task 1

Question 1: The missing `else` or default statement will create a latch instead of a MUX. This happens because without the `else` or default statement the update of the output is only triggered when `SelxSI = '1'` and when `SelxSI` has any other value the new value of `OutxDO` is not defined so it will have to keep its old value, that is, the value of `OutxDO` should not change from what it was before. This implies a memory element, which in this case is a latch.

Question 2: The code can be fixed in two different ways:

- Add an `else` in the process along with the associated assignment to `OutxDO` from `In0xDI` as shown in Listing 1.
- Add a default signal assignment `OutxDO <= In0xDI` in the process as shown in Listing 2.

Question 3: This just requires a MUX with inputs `In0xDI` and `In1xDI` at input numbers 0 and 1, respectively, and output `OutxDO` along with the select signal `SelxSI`.

Listing 1: VHDL solution for Task 1 with `else`.

```
process(all)
begin
  if SelxSI = '1' then
    OutxDO <= In1xDI;
  else
    OutxDO <= In0xDI;
  end if;
end process;
```

Listing 2: VHDL solution for Task 1 with default signal assignment.

```
process(all)
begin
  OutxDO <= In0xDI;

  if SelxSI = '1' then
    OutxDO <= In1xDI;
  end if;
end process;
```

Task 2

Question 1: The syntax errors are the missing `else` in the 2nd last line and the missing semicolon for the last line. The typo is the `ResxDN <= ResxDP`. The solution is shown in Listing 3.

Listing 3: VHDL solution for Task 2.

```
  signal ResxDN, ResxDP : unsigned(8-1 downto 0);

begin

  process(CLKxCI, RSTxRI)
  begin
    if (RSTxRI = '1') then
      ResxDP <= (others => '0');
    elsif (CLKxCI'event and CLKxCI = '1') then
      ResxDP <= ResxDN;
    end if;
  end process;

  ResxDN <= AxDI + BxDI     when CxDI + DxDI > 1 else
            AxDI - BxDI - 1 when CxDI > DxDI and DxDI /= 0 else
            AxDI + 1;
```

Question 2: The circuit schematic is shown in Figure 1.

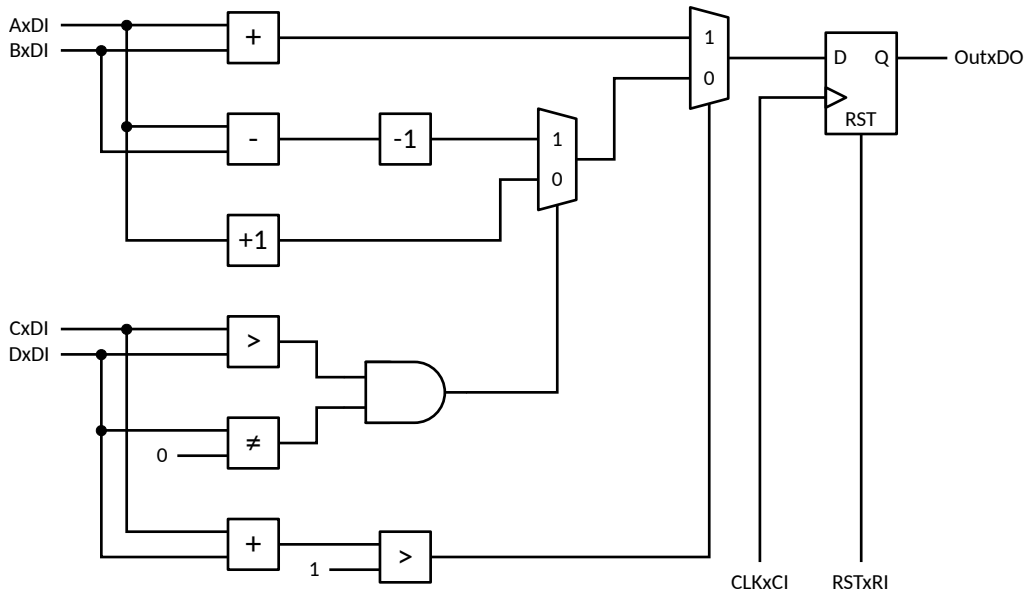

Figure 1: Circuit schematic for Task 2.

Task 3

Question 1: The errors are: Empty process sensitivity list for combinational logic, 2 missing `then`. The process for the registers is missing the `RSTxRI`. The solution is shown in Listing 4.

Listing 4: VHDL solution for Task 3.

```
  signal ResxDN, ResxDP : unsigned(8-1 downto 0);

begin

  process(CLKxCI, RSTxRI)
  begin
    if (RSTxRI = '1') then
      ResxDP <= (others => '0');
    elsif (CLKxCI'event and CLKxCI = '1') then
      ResxDP <= ResxDN;
    end if;
  end process;

  process(all)
  begin
    if Sel0xSI = '1' then
      ResxDN <= AxDI + BxDI;
    elsif Sel1xSI = '1' then
      ResxDN <= AxDI + CxDI;
    else
      ResxDN <= DxDI + 1;
    end if;
  end process;
```

Question 2: The solution is shown in Figure 2.

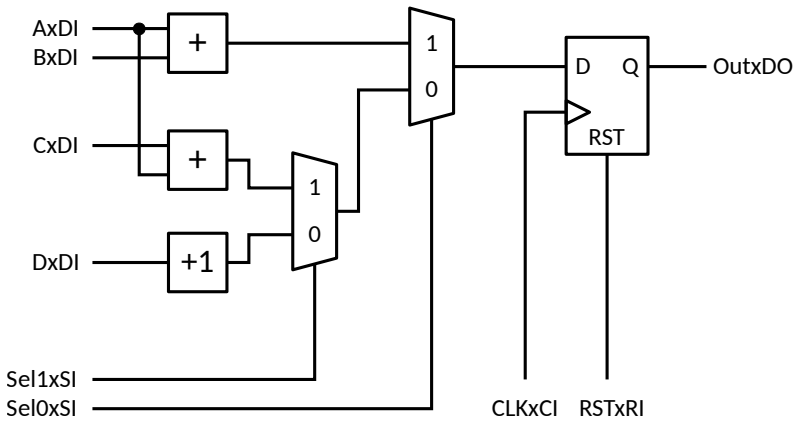

Figure 2: Circuit schematic for Task 3.

Task 4

Question 1: Recall that a signal in a process is not updated until the end of the process. So, while the update `CNTxDP <= CNTxDP + 1` is scheduled at the rising edge of the clock, it does not take effect until the end of the process. This means that if `CNTxDP = "1110"` when we enter the process, the if statement for `MaxPulsexSO` does not use `CNTxDP + 1 = "1111"`. Instead, it uses the current value `CNTxDP = "1110"` from when we entered the process. As the process is only triggered by the clock (and the reset), a change to `CNTxDP` (which happens at the end of the process) will not trigger the process again and so we wait until the next rising edge of the clock for the if statement to be evaluated again and the `MaxPulsexSO` to be assigned `'1'` now that it sees `CNTxDP = "1111"`.

Note that there is a difference between how to think of scheduled events in a process during simulation and how the inferred hardware looks like. In terms of the hardware, this style of coding has the effect of inferring a 1-bit register for `MaxPulsexSO` which delays the signal by one clock cycle after `CNTxDP` has attained the value `"1111"`. You can verify this by looking at the LUT which sits in front of the `MaxPulsexSO` register in Vivado if you synthesize `Task4.vhdl`.

This style also creates other issues with the hardware when synthesizing it with Vivado. Vivado returns us an FDSE block for the register `MaxPulsexSO`, which is a *D Flip-Flop with Clock Enable and Synchronous Set*. This FDSE block does not have an asynchronous reset and instead uses a clock enable which is based on the reset being low. Normally, we expect an FDCE block, which is a *D Flip-Flop with Clock Enable and Asynchronous Clear* which uses the asynchronous reset. If you look at the code it should also be clear as to why this happens, as `MaxPulsexSO` is not reset to `'0'` when the reset is asserted. **Therefore, it is critical that you never write code like this!**

Question 2: This style of coding is considered bad practice because it is much harder to take the VHDL code and compare it to a circuit schematic and the result is not what you may expect, as described in the previous question. Imagine a much larger process with many signals and several nested if statements. It would be much more difficult to keep track of the state of the signals and the order in which they are updated compared to writing combinational logic inside `process(all)`, separate from the register description in `process(CLKxCI, RSTxRI)`.

Question 3: The solution is to change the process to define only the register for the counter and then move the code for the counting and pulse outside as concurrent assignments (or in a `process(all)`). This can be seen in Listing 5. Now that the if statement is a concurrent assignment, whenever `CNTxDP` changes value the change is observed immediately instead of being delayed and no register is inferred for `MaxPulsexSO`.

Question 4: The corrected circuit is shown in Figure 3.

Listing 5: VHDL solution for Task 4.

```
  signal CNTxDN, CNTxDP : unsigned(4-1 downto 0);

begin

  process(CLKxCI, RSTxRI)
  begin
    if (RSTxRI = '1') then
      CNTxDP <= (others => '0');
    elsif (CLKxCI'event and CLKxCI = '1') then
      CNTxDP <= CNTxDN;
    end if;
  end process;

  CNTxDN      <= CNTxDP + 1;
  MaxPulsexSO <= '1' when CNTxDP = "1111" else
                 '0';
```

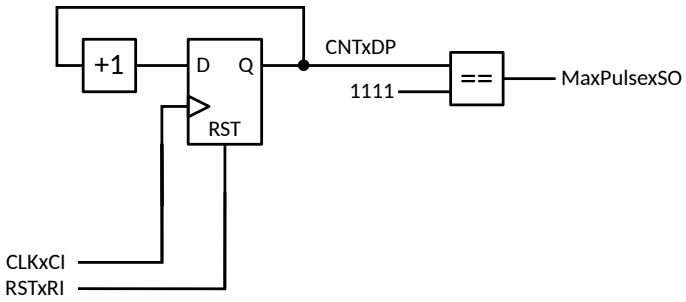

Figure 3: Circuit schematic for Task 4.

Circuits to VHDL

Task 5

Question 1: The solution is shown in Listing 6. This solution use `(others => 'X')` as the default instead of don't care (`'-'`) or `'0'`. The reason why you may want to use `(others => 'X')` is because it allows your component to output an `'X'` when `SelxSI` has an `'X'` in it, that is, your component does not swallow an `'X'` but allows it to propagate through. This can make it easier to perform debugging when you have a larger hiearchical design where you want to ensure that any error happening in a sub-component can be seen at the output. If you picked `(others => '0')` as the default, then, even if `SelxSI` has an `'X'` in it, you would not observe it on the output but would only see an output that is all zero.

Listing 6: VHDL solution for Task 5.

```
  signal ResxDN, ResxDP : unsigned(8-1 downto 0);

begin

  process(CLKxCI, RSTxRI)
  begin
    if (RSTxRI = '1') then
      ResxDP <= (others => '0');
    elsif (CLKxCI'event and CLKxCI = '1') then
      ResxDP <= ResxDN;
    end if;
  end process;

  with SelxSI select
    ResxDN <= AxDI + 1         when "00",
              AxDI - BxDI - 1  when "01",
              AxDI + BxDI      when "10" | "11",
              (others => 'X') when others;

  OutxDO <= ResxDP;
```

Task 6

Question 1: This block is used to find the smallest input value. It does not return the index, only the smallest value.

Question 2: The solution is shown in Listing 7.

Question 3-4: The testbench is given in file `tb_task6_sol.vhdl`.

Listing 7: VHDL solution for Task 6.

```
  signal Res0xD, Res1xD, Res2xD : unsigned(8-1 downto 0);
  signal Comp0xS, Comp1xS, Comp2xS : std_logic;

begin
  Comp0xS <= '1' when In0xDI < In1xDI else '0'; -- Comparators
  Comp1xS <= '1' when In2xDI < In3xDI else '0';
  Comp2xS <= '1' when Res0xD < Res1xD else '0';

  Res0xD <= In0xDI when Comp0xS = '1' else In1xDI; -- Mux
  Res1xD <= In2xDI when Comp1xS = '1' else In3xDI;
  Res2xD <= Res0xD when Comp2xS = '1' else Res1xD;

  OutxDO <= Res2xD;
```

Task 7

Question 1: The circuit schematic for Task 7 shows how an adder can be reused for many different input signal combinations instead of having multiple adders for each combination like in Task 3. This is called **resource sharing**, and is quite commonly done in RTL design. In this case, we only use 1 adder instead of 3!

Question 2: The solution is shown in Listing 8.

Listing 8: VHDL solution for Task 7.

```
  signal Src0xD, Src1xD : unsigned(8-1 downto 0);
  signal ResxDN, ResxDP : unsigned(8-1 downto 0);

begin

  process(CLKxCI, RSTxRI)
  begin
    if (RSTxRI = '1') then
      ResxDP <= (others => '0');
    elsif (CLKxCI'event and CLKxCI = '1') then
      ResxDP <= ResxDN;
    end if;
  end process;

  with SelxSI select
    Src0xD <= AxDI             when "00"|"01",
              DxDI             when "10"|"11",
              (others => 'X') when others;

  with SelxSI select
    Src1xD <= BxDI             when "00",
              CxDI             when "01",
              "00000001"       when "10"|"11",
              (others => 'X') when others;

  ResxDN <= Src0xD + Src1xD;
  OutxDO <= ResxDP;
end rtl;
```