Question 1 – Water Tank Controller

Design a VHDL code to control the enable of a pump which fills a water tank. As shown in the figure 1 below the inputs to the controller are X and Y which are fed from two sensors connected to the water tank. The controller should issue a pump enable signal E. Note that X, Y and E are 1 bit signals.

![Figure 1](image)

You should assume that the sensor gives a logic high signal in presence of water. The controller should be designed in a way to enable the pump ONLY when the water level falls below the lower limit Y and switches off ONLY when the water level has risen beyond the upper limit X. Note that the controller should be synchronized to a clock and should have an asynchronous reset.

Question 2 – Sequence Detector

Design a sequential logic circuit using Verilog which can detect an input sequence of ‘11010’. Data (din) is fed serially in on every clock cycle. As soon as it detects a correct sequence, an output alert (sync_out) is issued and it will restart scanning the input for another correct sequence. Note that din and sync_out are 1 bit signals.

![Figure 2](image)
Question 3 – 4-bit BCD/Binary Counter with 7-segment display driver

(a) Describe a HEX to seven segment display decoder as a VHDL code. The value assigned to the output vector is to be determined according to Figure 3, where “a” should be placed as the MSB of your output vector.

(b) Design a counter with the following specifications:
• 4 bit output
• BCD/binary counter
• Carry output signals
• Asynchronous Reset
• Synchronous Pre-Programming capability of the output value.

Functional description:
When \( \text{async\_rst} = 1 \), \( \text{cntr\_out} \) is resetted to zero.
When \( \text{pdata\_en} = 1 \), \( \text{cntr\_out} \) is set to \( \text{pdata\_in} \) in the next clock cycle.
When \( \text{cntr\_en} = 1 \), then \( \text{cntr\_out} \) is incremented on every clock cycle.

In BCD mode (\( \text{bcd\_mode} = 1 \)), the counter cycles from 0 to 9
In Binary mode (\( \text{bcd\_mode} = 0 \)), the counter cycles from 0 to 15

(c) Connect the blocks designed in parts (a) and (b) in a structural VHDL architecture and perform adequate testing.
Question 4 – 4-bit Integer Multiplier

Theory:

Integer Multiplication Algorithm

The block diagram in Figure 6 shows the hardware arrangement for sequential multiplication. This circuit performs multiplication by using a single n-bit adder n times to implement the spatial addition performed in the standard long multiplication of numbers.

Registers A and Q combined hold the $i^{th}$ partial product while multiplier bit $q_i$ generates the Add/NoAdd signal. If $q_i = 1$ then Add else NoAdd. This signal controls the addition of the multiplicand $M$ to the $i^{th}$ partial product to generate the $i+1^{st}$ partial product. The product is computed in $n$ cycles. The partial product grows in length by one bit per cycle from the initial vector PP0, of $n$ zeros in register A. The carry-out from the adder is stored in flip-flop C.

At the beginning, the multiplier is loaded into register Q, the multiplicand into register M, while C and A are cleared to 0.

At the end of each cycle, C, A and Q are shifted right one bit position to allow for growth of the partial product as the multiplier is shifted out of register Q. After $n$ cycles the higher-order half of the product is held in A and the low-order half is in register Q.

![Figure 6 – Sequential Multiplier](image-url)
**Preparation:**

Work out the following using the above algorithm on a paper:

\[1101_2 \times 1011_2\]

showing the contents of the respective registers after each cycle.

**Implementation:**

![Multiplier Diagram](image)

*Figure 7 – Input/Output Requirements of Multiplier*

When `mult_start` signal is enabled for one clock cycle, the multiplier load the `multiplicand_in` and `multiplier_in` in the respective registers and starts the operation. An output signal `mult_ready` will show when the result is ready.

Describe the multiplier algorithm with a **VHDL** code.

Produce the required test benches to test your implementation. Use the example given in the preparation to verify correct operation.

Finally implement the multiplier algorithm on the Spartan-3 FPGA available in the HDL laboratory.
Question 5 – 4-bit Integer Divider

Theory:

An algorithm that implements division by a longhand method is as follows: it positions the divisor appropriately with the dividend and performs a subtraction. If the remainder is zero or positive, a quotient bit of 1 is determined, the remainder is extended by another bit of the dividend, the divisor is repositioned, and another subtraction is performed. On the other hand, if the remainder is negative, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction. This is known as the restoring division algorithm. Another algorithm was developed, known as the non-restoring division algorithm which avoids the need of restoring the dividend after an unsuccessful subtraction.

Restoring Division

Figure 8 shows a logic circuit arrangement that implements restoring division. An n-bit positive divisor is loaded into register M and an n-bit positive dividend is loaded into register Q at the start of operation. Register A is set to 0. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. The extra bit position at the left of both A and M accommodates the sign bit during subtractions. The following algorithm performs restoring division. Do the following for n times:

1. Shift A and Q left one binary position
2. Subtract M from A and place answer back in A
3. If the sign of A is 1, set q_i to 0 and add M back to A; otherwise set it to 1

Figure 8 – Sequential Divider
Non-restoring Division

Subtraction is said to be unsuccessful if the result is negative. Consider the sequence of operations that take place after the subtraction operation in the preceding algorithm. If A is positive, we shift left and subtract M, that is we perform $2^*A - M$. If A is negative, we restore it by performing $A+M$ and then shift it left and subtract M, that is, $2(A+M) - M = 2^*A + M$. The qi bit is appropriately set to 1 or 0 after the correct operation has been performed. This can be summarized as follows:

**Step 1:** Do the following n times:

1. If the sign of A is 0, shift A and Q left one bit position and subtract M and A; otherwise shift A and Q left and add M to A.

2. Now if the sign of A is 0, set $q_i$ to 1; otherwise set it to 0.

**Step 2:** If the sign of A is 1, add M to A.

**Preparation**

Work out the following using one of the above algorithms:

$1000_2 \div 0011_2$

showing the contents of the respective registers after each cycle.

**Implementation**

Implement one of the above algorithms with a Verilog code.

Produce the required test benches to test your implementation. Use the example given in the preparation to verify correct operation.

Finally implement the divider algorithm on the Spartan-3 FPGA available in the HDL laboratory.