Lab_2: Add and Shift Multiplier

Introduction

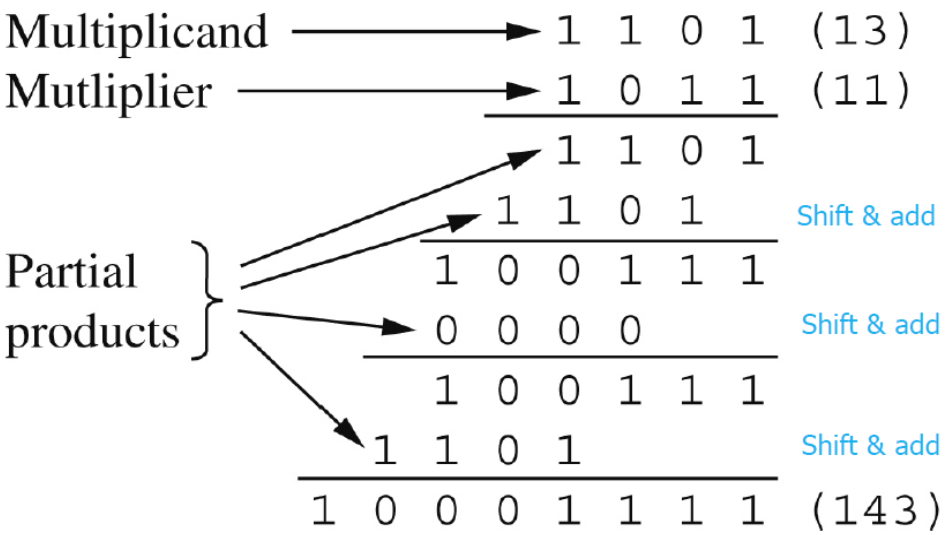
In this experiment, you will need to implement a $m \times n$ multiplier **from scratch**. Note that "starting from scratch" means that you cannot directly use the methods already available in the standard library. As the name implies, you can **only "add" and "shift" data.**

This is not a course on digital circuits, so we will detail the calculation method, which you will just need to implement in a hardware description language.

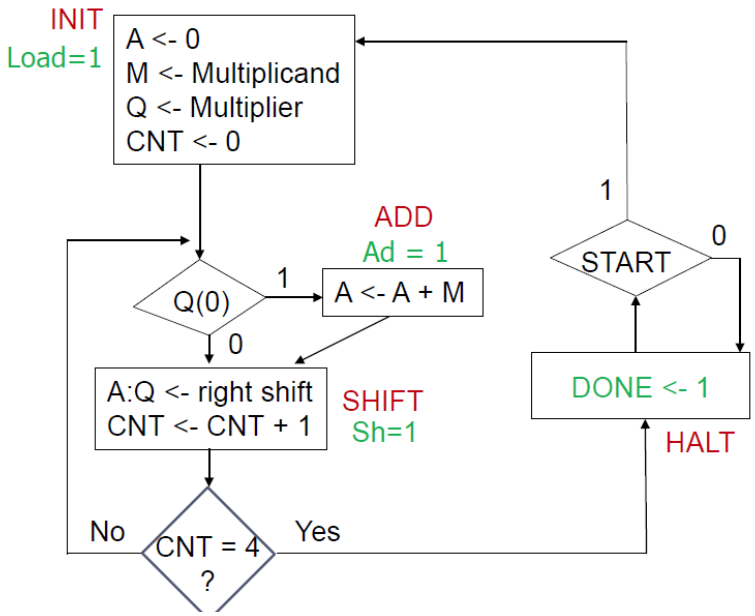
You can also compare the predefined multiplex implementations with your implementation at the end of the experiment, which will help you understand the importance of "Don't reinvent the wheel"

Algorithm

How to hand computation binary multiplication



Algorithm block diagram (Moore model)



Example: $6 \times 5 = 110 \times 101$

M	A	Q	CNT	State	
110	0000	101	0	INIT	Multiplicand->M, 0->A, Multiplier->Q, CNT=0
	+ 110			ADD	(Since Q0=1) A = A+M
	0110	101	0		
	0011	010	1	SHIFT	Shift A:Q, CNT+1=1 (CNT not 3 yet)
					(skip ADD, since Q0 = 0)
	0001	101	2	SHIFT	Shift A:Q, CNT+1=2 (CNT not 3 yet)
	+ 110			ADD	(Since Q0 = 1) A = A+M
	0111	101	2		
	0011	110	3	SHIFT	Shift A:Q, CNT+1=2 (CNT= 3)
	0011	110	3	HALT	Done = 1

P = 30

Declare of top level

```
entity MUL is
    Generic(
            nq : INTEGER;
            nm : INTEGER);
    Port (
            Multiplicand : in STD_LOGIC_VECTOR (nm-1 downto 0);
            Multiplier : in STD_LOGIC_VECTOR (nq-1 downto 0);
            Start : in STD_LOGIC;
            CLK : in STD_LOGIC;
            Done : out STD_LOGIC;
            Product : out STD_LOGIC_VECTOR (nm+nq-1 downto 0));
end MUL;
```

Behavior of test-bench

```
    for i in (2**nm)-1 downto 0 loop
        for j in (2**nq)-1 downto 0 loop
            multiplicand_sig <= STD_LOGIC_VECTOR(TO_UNSIGNED(i,nm));
            multiplier_sig <= STD_LOGIC_VECTOR(TO_UNSIGNED(j,nq));
            product_ref <= STD_LOGIC_VECTOR(TO_UNSIGNED(i*j,nm+nq));
            start_sig <= '1';
            wait until rising_edge(clk_sig);
            start_sig <= '0';
            wait until done_sig='1';
            wait for 1 ns;
            if product_sig = product_ref then
                is_right := '1';
            else
                is_right := '0';
            end if;
            wait until rising_edge(clk_sig);
        end loop;
    end loop;
```

Requirements

1. The algorithm can be divided into **three** parts: **addition**, **shift**, and **control**. This experiment requires you to use **three different modules** to implement each of the three parts. **Each component should implement and can only implement its own**

functionality!

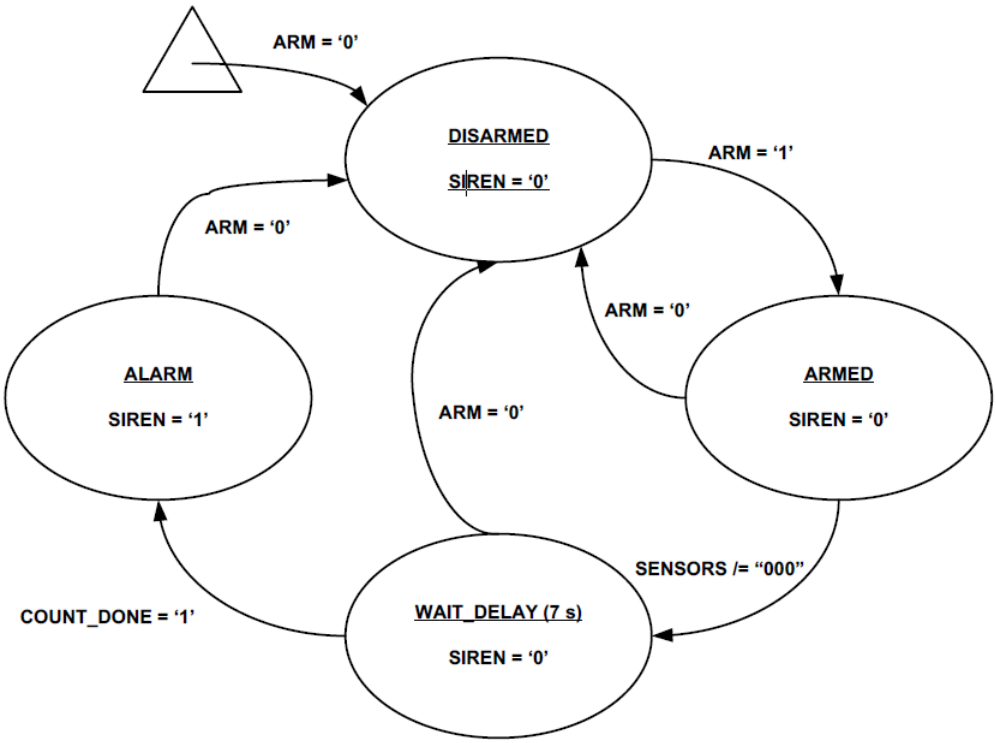
2. You need to use finite state machines (**FSM**) to implement control functions. For finite state machines, you can find examples in Professor Ha's courseware.

3. The experiments in this course forces you to use a structure that **separates the timing logic from the combinatorial logic**. You can refer to the courseware to write the finite state machine as **two or three processes**. The caveat is that you **have to completely separate** the combinatorial logic from the timing logic. It is not acceptable to use clocks in the combinatorial logic or to add too much logic judgment to the timing logic.

4. Note that this lab we should write a multiplier for $m \times n$ bits, so the use of **GENERIC** is necessary. Remember the counter (ADDR_CNTR) that was written in the last experiment? Make a change to its architecture and you'll get a good component that can handle the "shift" option. To conserve resources, **No more than $O(m+n)$ registers** may be used, and **no more than $O(m+n)$ adders may be used.**

5. Draw state transition diagram and system block diagram (such as Figure 1 in Lab_1.pdf) **before** you start writing code.

6. There is a reference of state transition diagram as follows:



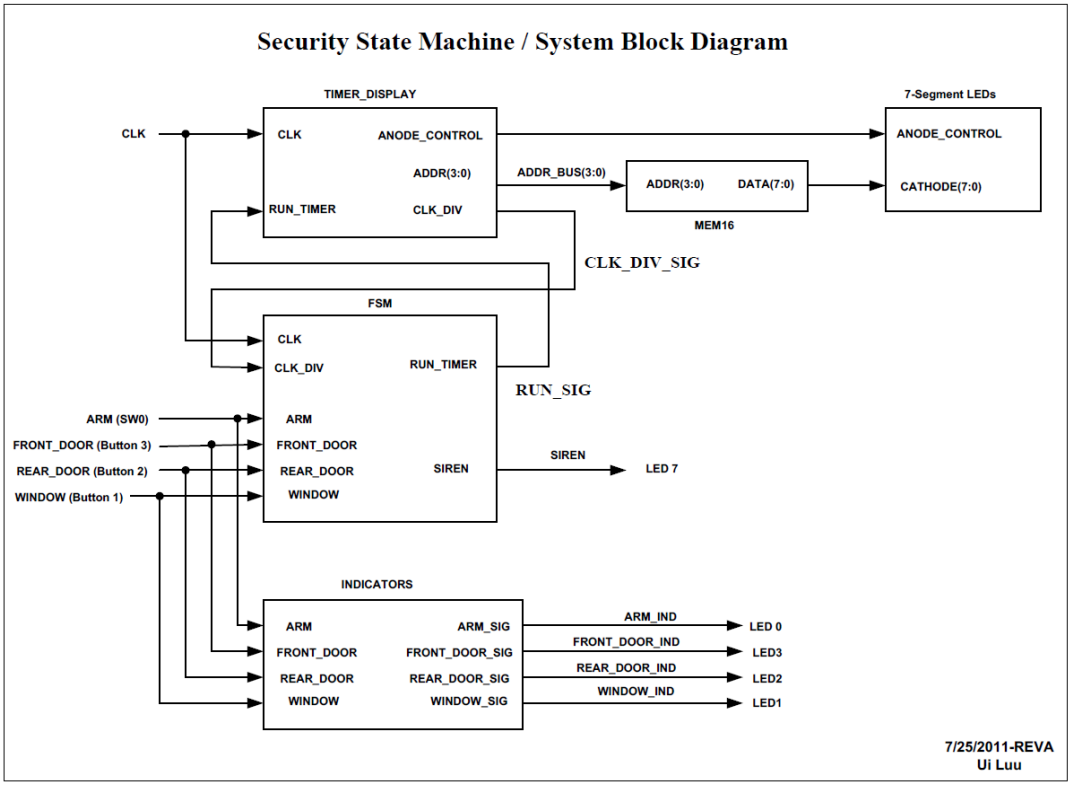
7. The state transition diagram should contain the following key points and distinguish them in different formats:

   a) Name of state

   b) Conditions for transitioning from one state to another

c) Value of control signal in each state (or during the transition from one state to another)

8. There is a reference of the system block diagram as fellow:



9. The system block diagram should contain the following key points:

a) Components and name of each component

b) Signals and name of each signal

c) Ports and name of each port

Code format

1. Keywords (except data type) are **all lowercase**.
2. The library names and data types are **all capitalized**.
3. Naming must be **meaningful** (except for the prescribed name). There are no restrictions on naming. You can reasonably use uppercase letters and underscores to make your name easier to read and understand.
4. Use **four Spaces** or **a Tab** to indent your code. Reasonable indentation makes your code easier to read and understand.

First Submit

1. Compress the diagrams and **name the compressed package as follow the format**:

   Lab2_[YourNameInEnglish]_[YourStudentNumber].zip

2. You should submit:

   - State transition diagram
     - Draw it by computer rather than your hand
   - System block diagram/schematic
     - Draw it by computer rather than your hand

3. **File Organization Schema in Package**:

   Lab2_[YourName]_[YourStudentNumber].zip
   ├──State_Transition_Diagram.pdf
   └──System_Block_Diagram.pdf

4. Upload the .zip file to related lab folder on BB.

5. The reference State Transition **Sketch** and System Block **Sketch** will be given after submission deadline.

First Submit Deadline: 2019-10-7 17:30

Second Submit

1. Compress the HDL codes and **name the compressed package as follow the format**:

   Lab2_[YourNameInEnglish]_[YourStudentNumber].zip

2. You should submit:

   - Source codes
     - All of the source codes
   - Test bench codes
     - All of the test bench codes

3. **File Organization Schema in Package**:

   Lab2_[YourName]_[YourStudentNumber].zip
   ├──[The name of the module where you implemented the addition function].vhd
   ├──[The name of the module where you implemented the control function].vhd
   ├──[The name of the module where you implemented the shift function].vhd
   ├──[The name of top-level].vhd

└──testbench.vhd

4. Enter the sutomatic scoring system and **upload your .zip file at lab_2.**

   https://autolab.sist.shanghaitech.edu.cn/auth/users/sign_in

Second Deadline: 2020-10-14 17:30

- Submit on time, get all scores.
- Submission time does not exceed 24 hours of the deadline, get half of the score.
- Submitted more than 24 hours from the deadline, get no score.

Any Question?

Any questions on course or labs can be proposed in the Discussing Forum on BB.

We recommend you subscribe the forum to receive the newest topics on time.