Lab #2 Algorithmic State Machine Design

Objectives:

- to demonstrate the incremental design of an algorithmic state machine
- to gain further experience in the computer-aided design of digital logic circuits
- to study the performance characteristics of a binary multiply algorithm
- to gain further experience with the use of a logic analyzer

Preparation:

1. Study the "shift-and-add" binary multiply algorithm and its hardware implementation as described in the lecture notes available on the course web site ("An Example of ASM Design: A Binary Multiplier")

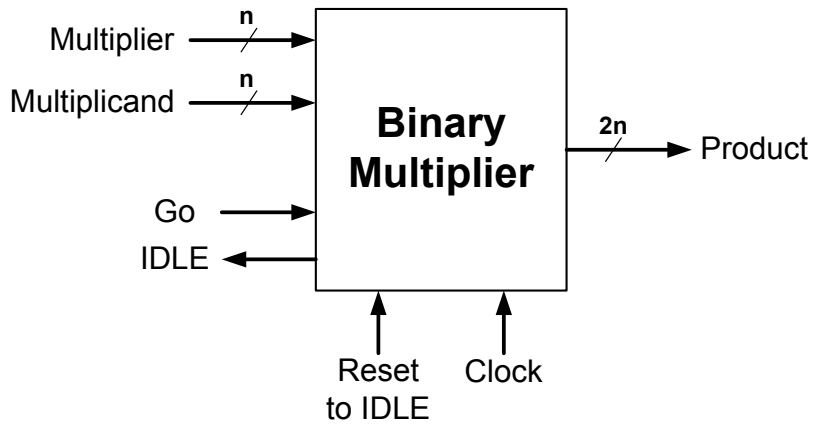
2. Read this entire lab and study the schematics and VHDL definitions.

3. Complete the tables at the end of this lab for each of the cases: **3x2** and **FxF**. These will be verified with the logic analyzer.

Introduction and Background:

An algorithmic state machine (ASM) is a Finite State Machine that uses a sequential circuit (the Controller) to coordinates a series of operations among other units such as counters, registers, adders etc. (the Datapath). In this lab, we use an ASM to demonstrate the design of a binary multiplier. Multiplies feature so frequently in modern embedded digital signal processing, that most microprocessors (DSP chips) have dedicated hardware to achieve fast execution.

In this lab we build an example of a simple algorithmic state machine that implements an unsigned 4-bit by 4-bit binary multiply algorithm in hardware using one-flipflop-per-state hardwired control. The design will illustrate the key concepts of incremental system design. Low-level logic building blocks comprising a combination of VHDL and schematic definitions are connected to build a datapath and a controller for the multiplier. The final circuit (top level) is a connection of the datapath and controller to form the multiplier which itself may then may be viewed as a building block and used in even larger circuits. The multiplier will be programmed in the 3032 CPLD and tested. The top-level view of the system you are to build is as follows:



The circuit produces the 2n-bit product of two n-bit numbers placed on the Multiplier and Multiplicand inputs (in our case **n** = 4). Multiplication is initiated by the first clock pulse that occurs after the "Go" input is set HI. If the output IDLE= 1, the circuit is not busy, i.e. ready to multiply. While a multiply operation is taking place, IDLE = 0. These two signals (IDLE and Go) may be used to coordinate the timing of the binary multiplier with the external user. This is known as "handshaking". The Reset input is used to force the circuit (asynchronously) to the IDLE state at any time (such as at power-up). The clock frequency will of course, determine the multiplication time.

Experiment:

1. The basic logic blocks to be used are:

a full adder (74283 parallel adder) from the Altera mega functions library
a shift register defined in VHDL (shiftreg.vhd), listed below
a binary down counter with zero detect defined in VHDL (counterp.vhd), listed below

These components are connected to form a datapath that will be used in this lab. Open the datapath circuit (bin-multiplier-datapath.gdf) and double-click on each component. You examine/edit the schematic/VHDL code that defines each component. Download the datapath circuit to the 3032 CPLD and verify its operation with switches and LEDs.

2. The one-flipflop-per-state control unit for the multiplier is defined as a schematic (bin-multiplier-controller.gdf). Compile this circuit, download to the 3032 and test it with switches and LEDs. Verify the ASM chart and state transition diagram (shown later in this lab) that it implements. Note that Shift_dec and MUL1 are the same signal.

3. The top-level schematic combining the controller and the datapath is shown in the diagram at the end of this lab (Bin-mult.gdf). Notice that some outputs have been assigned to pins. Be sure that you understand which operation in the datapath is associated with each connection between these two units. Compile the design and program the 3032 CPLD.

4. Connect the toggle switches to the multiplier, multiplicand, Go, Reset_to_idle and Clock inputs of the circuit. Connect LEDs to the eight Product output lines and one LED to the IDLE output. Carefully verify the operation of the muliplier by working through the examples for **3x2** and **FxF** from your preparation using the toggle switch for each clock. Carefully verify the handshaking operation of the Go input, the IDLE output and the clock.

5. Try a number of other examples such as **0xN**, **Nx0**, **1xN**, **Nx1**, **Fx2**, **Fx4**, **Fx8** etc. where **N** is any number you choose. Verify that multiplying any number **N** by $2^{\mathbf{k}}$ is equal to shifting **N** by **k** places to the left.

6. Connect the logic analyzer channels D0-D7 to the multiplier test signals and channels D8-D15 to the eight Product bits. Attach a 1 MHz free-running TTL square wave to the clock input of the multiplier circuit and capture one cycle of the multiply on the 'scope (trigger on IDLE). Carefully verify the **3x2** and **FxF** cases again for all states and outputs.

7. Make a symbol for the entire multiplier and show it on a schematic. Notice that many of the pins are internal state, control and status signals brought to the output pins for testing purposes (so we can look at them with the logic analyser) and need not be present in the final design. Double-clicking on the multiplier symbol allows navigation through the entire design hierarchy. Try it !

Issues that you should think about:

1. What is the multiply time as a function of **n** (measured in clocks) ? _____

2. Explain how the Go input and IDLE output can be viewed as "handshaking" signals.

3. What is the maximum time that the Go input can stay HI to ensure a single iteration of the multiply algorithm ?

   Maximum pulse width on Go input (in clocks): _____

4. Does your circuit still work with a 15 MHz clock (try it !) ? Estimate the highest clock freq: _____

5. What would be required to extend this design to accommodate the multiplication of sign-2's complement numbers ?

6. The entire multiply algorithm can be described using VHDL. An example of a VHDL-only version of a multiplier is available from your textbook (Fig 10.19).

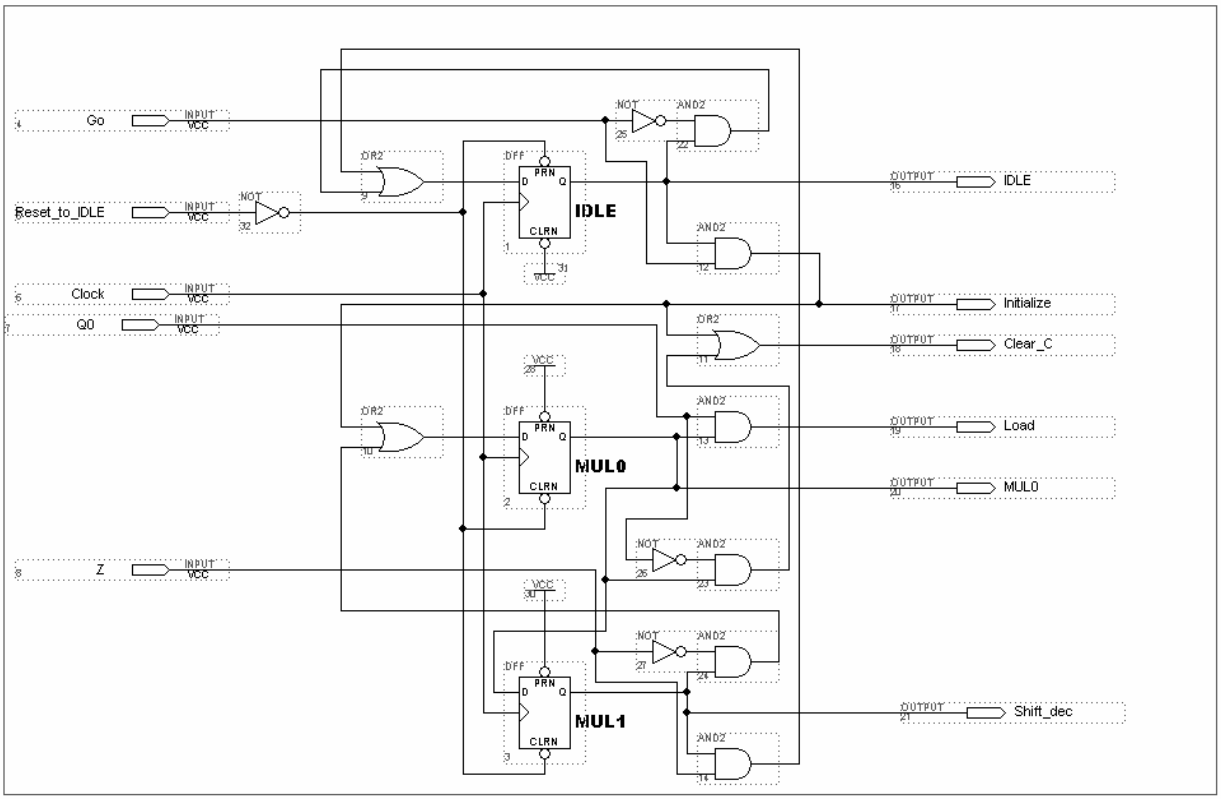
VHDL definition for counter P with Zero detect:

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL ;

ENTITY CounterP IS
        PORT(  CLK, Init: IN STD_LOGIC ;
                        Zero_det: BUFFER STD_LOGIC ;
                        CntP: BUFFER STD_LOGIC_VECTOR (1 DOWNTO 0) );
END CounterP ;

ARCHITECTURE Behaviour OF CounterP IS

BEGIN
        PROCESS ( Init, CLK )
        BEGIN
                IF Init = '1' THEN
                        CntP <= "11" ;
                ELSIF (CLK'EVENT AND CLK = '0') THEN
                        CntP <= CntP - 1 ;
                END IF ;

        END PROCESS ;

        Zero_det <= '1' WHEN CntP = "00" ELSE '0' ;

END Behaviour ;
```

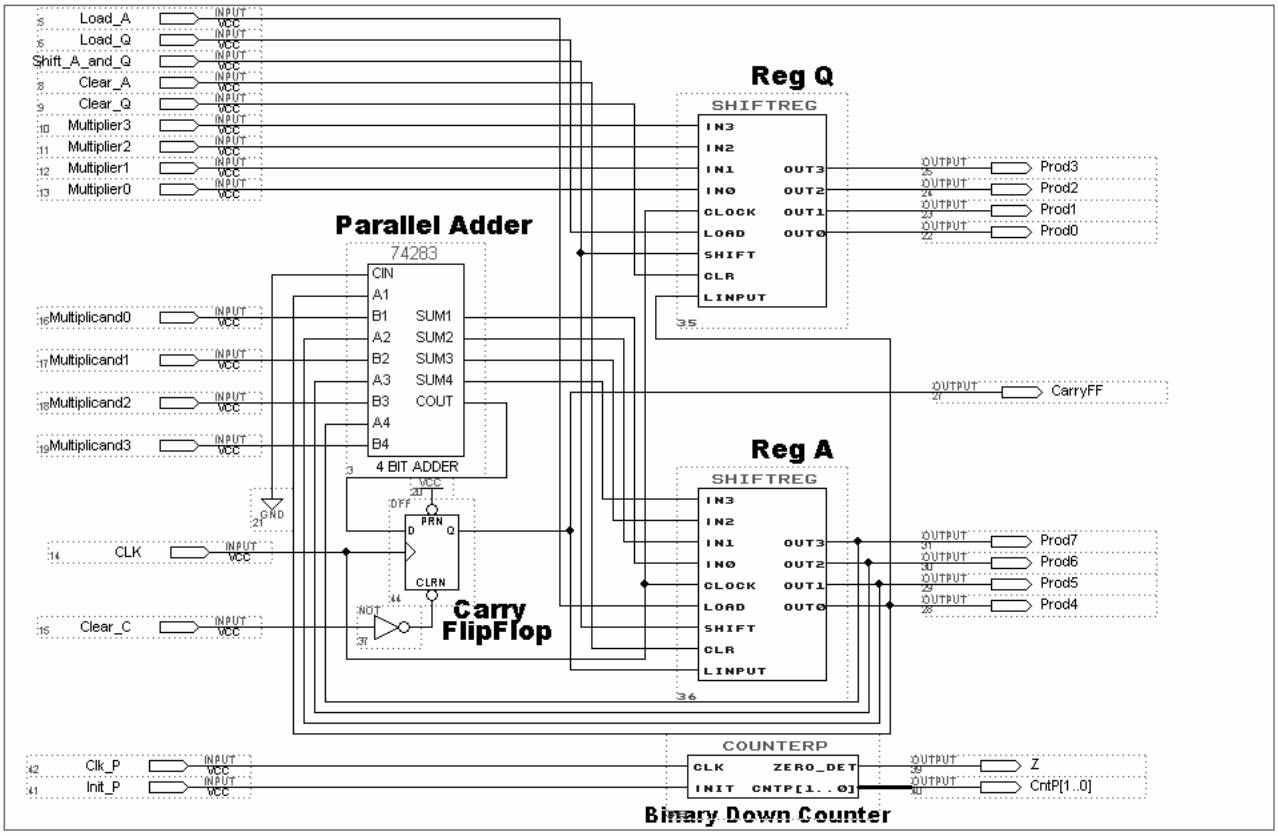
VHDL definitions for shift registers A and Q :

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY shiftreg IS
        PORT (In3,In2,In1,In0: IN          STD_LOGIC;
                Clock : IN   STD_LOGIC ;
                Load, Shift, CLR : IN STD_LOGIC ;
                Linput : IN  STD_LOGIC ;
                Out3,Out2,Out1,Out0  :  BUFFER STD_LOGIC ) ;
END shiftreg ;

ARCHITECTURE Behavior OF shiftreg IS
BEGIN
        PROCESS
        BEGIN
                WAIT UNTIL Clock'EVENT AND Clock = '1' ;

                Out3 <= Out3 ;
                Out2 <= Out2 ;
                Out1 <= Out1 ;
                Out0 <= Out0 ;

                If Load = '1' THEN
                Out3 <= In3 ;
                Out2 <= In2 ;
                Out1 <= In1 ;
                Out0 <= In0 ;

                END IF;

                If Shift = '1' THEN
                        Out0 <= Out1 ;
                        Out1 <= Out2 ;
                        Out2 <= Out3 ;
                        Out3 <= Linput ;
                END IF;

                If CLR = '1' THEN
                        Out3 <= '0';
                        Out2 <= '0';
                        Out1 <= '0';
                        Out0 <= '0';
                END IF;

        END PROCESS ;

END Behavior ;
```

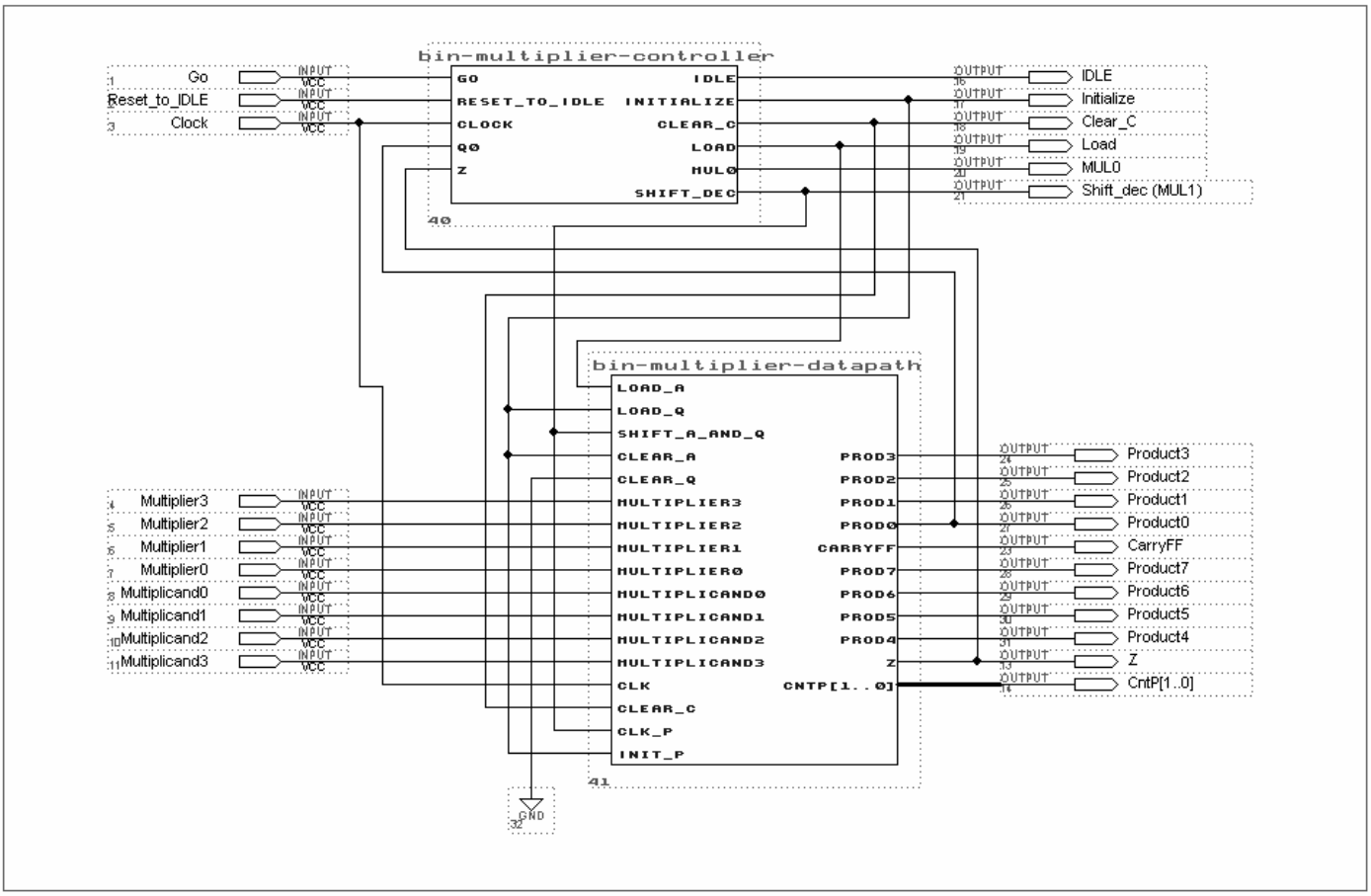
Control Logic with One Flipflop per state (bin-multiplier-controller.gdf):



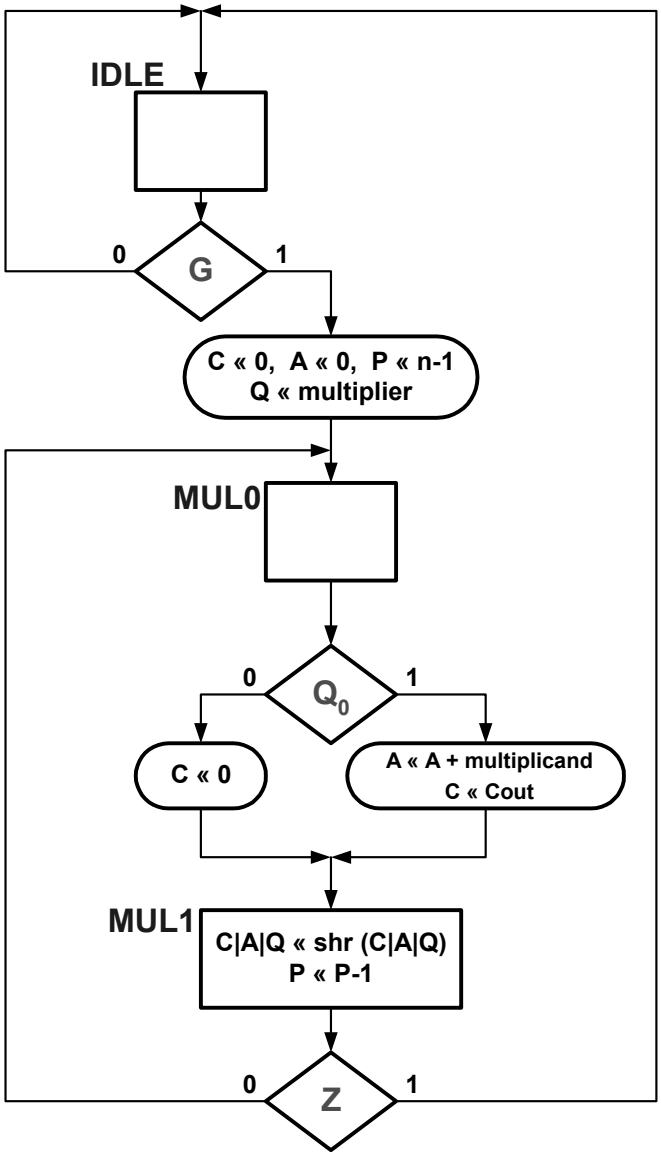
Datapath for Binary Multiplier (bin-multiplier-datapath.gdf):



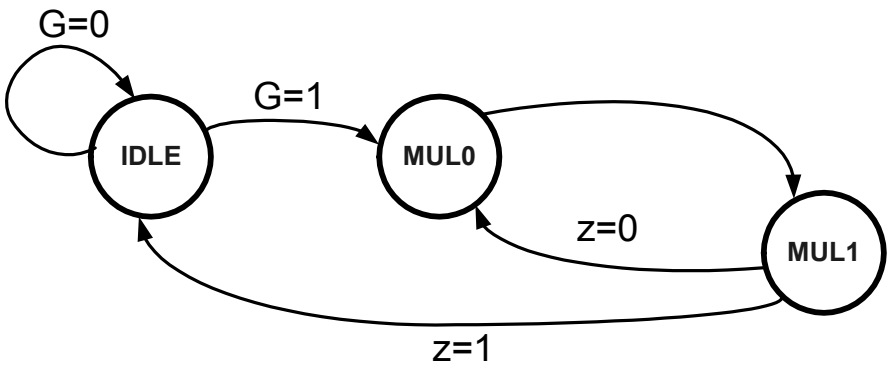
Top level design combining the Controller and Datapath (Bin-mult.gdf):



ASM Chart for Binary Multiplier



State Transition Diagram for Binary Multiplier



Examples:

3 x 2 (assume Go = HI)

Clock pulse	P Count	Z	C	Reg A	Reg Q	STATES			Control Signals			
						IDLE	MUL0	MUL1	Initialize	Clear_C	Load	Shift_dec
	1 1	0	x	x x x x	x x x x	1	0	0	1	1	0	0
1	1 1	0	0	0 0 0 0	0 0 1 0	0	1	0	0	1	0	0
2												
3												
4												
5												
6												
7												
8												
9												

F x F (assume Go = HI)

Clock pulse	P Count	Z	C	Reg A	Reg Q	STATES			Control Signals			
						IDLE	MUL0	MUL1	Initialize	Clear_C	Load	Shift_dec
	1 1	0	x	x x x x	x x x x	1	0	0	1	1	0	0
1	1 1	0	0	0 0 0 0	1 1 1 1	0	1	0	0	0	0	0
2												
3												
4												
5												
6												
7												
8												
9												