PART 3

MSI Based Design

- **Use MSI parts of Part 2**
- **Sequential multiplication**
- **Designing the multiplier**
- **Control and data parts**
- **Testing the multiplier**

In addition:

Will talk about synthesis and how it can help such a design process

MSI_design.parts

AND	gate
74LS377	register
74LS283	adder
74LS299	shift register
74LS157	multiplexer
74LS541	three-state

- Use 74LS parts in a complete design
- Features of this design include
    - Control and data separation
    - Possible to synthesize control part
    - Multiplexed input/output
    - Includes bussing & multiplexing

MSI_design.multiplication

```
A:      1  0  0  1   *
B:      1  0  1  1
---------------------------
        1  0  0  1
     1  0  0  1
  0  0  0  0
1  0  0  1
---------------------------
1  1  0  0  0  1  1
```

- Based on bits of B:
- Add and shift A, or just shift A
- Shift B to see its right hand side bits
- Shift partial add results to add next A

MSI_design.multiplication

0:

TO PERFORM 1001*1011, START WITH :

0	0	0	0	0	1	0	1	**1**

A:	1	0	0	1

1:

ADD 1001+0000 -> 1001, THEN SHIFT

0	0	1	0	0	1	1	0	**1**

A:	1	0	0	1

2:

ADD 1001+0100 -> 1101, THEN SHIFT

0	0	1	1	0	1	1	1	**0**

A:	1	0	0	1

3:

SHIFT ONLY

0	0	0	1	1	0	1	1	**1**

A:	1	0	0	1

4:

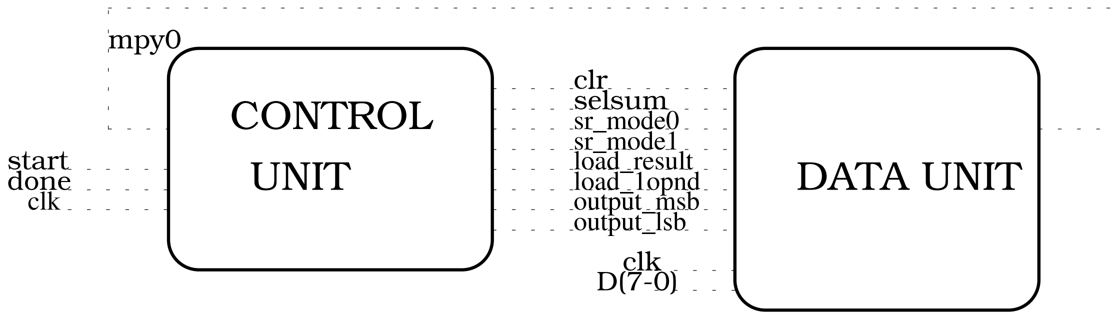
ADD 1001+0011 -> 1100, THEN SHIFT

0	0	1	1	0	0	0	1	1

A:	1	0	0	1

- Shift B and partial add together
- Keep adding A to partial add result
- Repeat for all bits of B

MSI_design.control_and_data



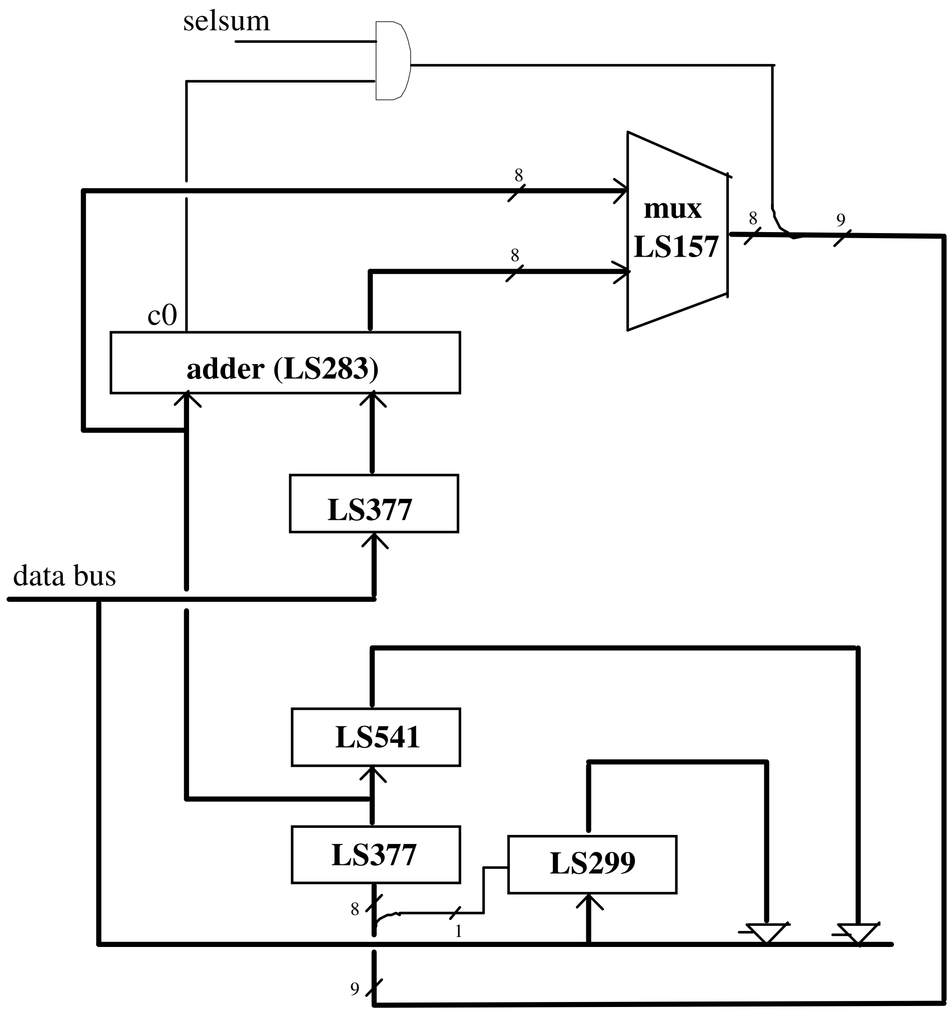
- Partition design into control and data parts
- Control part activates signals to data
- Data part contains registers, busses and logical units

MSI_design.control_and_data

```
USE WORK.q_utilities.ALL;
--
ENTITY shift_and_add IS
    PORT ( clk : IN qit;  databus : INOUT
               wired_qit_vector( 7 DOWNTO 0 ) BUS := "ZZZZZZZZ";
               start : IN qit;  done : OUT qit );
END shift_and_add;
--
ARCHITECTURE structural OF shift_and_add IS
    COMPONENT mult_data_path
        PORT ( clr, selsum : IN qit;
                   databus : INOUT wired_qit_vector( 7 DOWNTO 0 ) := "ZZZZZZZZ";
                   clk, load_1opnd_bar, load_result_bar : IN qit;
                   sr_mode0, sr_mode1 : IN qit;
                   output_lsb_bar, output_msb_bar : IN qit_vector(1 DOWNTO 0);
                   mpy0 : OUT qit );
    END COMPONENT;
    FOR data : mult_data_path USE  ENTITY  WORK.mult_data_path (structural);
    COMPONENT multiplier_controller
        PORT ( clock, start, mpy0 : IN qit;
                   output_msb, output_lsb : OUT qit_vector (1 DOWNTO 0) := "11";
                   sr_mode0, sr_mode1 : OUT qit := '0';
                   load_result, load_1opnd : OUT qit := '1';
                   selsum, clear, done : OUT qit := '0' );
    END COMPONENT;
    FOR ctrl: multiplier_controller USE  ENTITY
        WORK.multiplier_controller(state_machine);
    SIGNAL clr, selsum, mpy0, load_result_bar,load_1opnd_bar: qit;
    SIGNAL sr_mode1, sr_mode0 : qit;
    SIGNAL output_msb_bar, output_lsb_bar  : qit_vector(1 DOWNTO 0);
BEGIN
    data: mult_data_path
       PORT MAP  ( clr, selsum, databus, clk, load_1opnd_bar, load_result_bar,
                       sr_mode1, sr_mode0, output_lsb_bar, output_msb_bar, mpy0 );
    ctrl: multiplier_controller
       PORT MAP  ( clk, start, mpy0, output_msb_bar, output_lsb_bar,
                       sr_mode1, sr_mode0, load_result_bar, load_1opnd_bar,
                       selsum, clr, done );
END structural;
```

- Control and data wiring
- Complete design includes description of both parts

MSI_design.control_and_data



- 74LS parts are used for data part implementation
- *databus* is used as input and output
- Multiplex adder output to add-and-shift or just shift

MSI_design.control_and_data

```
USE WORK.q_utilities.ALL;
ENTITY mult_data_path IS
   PORT ( clr, selsum : IN qit;
            databus : INOUT wired_qit_vector( 7 DOWNTO 0 ) := "ZZZZZZZZ";
            clk, load_1opnd_bar, load_result_bar : IN qit;
            sr_mode0, sr_mode1 : IN qit;
            output_lsb_bar, output_msb_bar : IN qit_vector( 1 DOWNTO 0 );
            mpy0 : OUT qit );
END mult_data_path;
```

- Data interface includes primary input and outputs as well as control signals
- Control signals control bus connections, register transfers, mode selections

MSI_design.control_and_data

```
ARCHITECTURE structural OF mult_data_path IS
  COMPONENT and_gate
     PORT (  i1, i2 : IN qit; o1 : OUT qit );
  END COMPONENT;
  FOR u9: and_gate USE ENTITY WORK.and2( fixed_delay );
  COMPONENT reg_8
     PORT ( clk, g_bar : IN qit; d8 : IN qit_vector( 7 DOWNTO 0 );
            q8 : OUT qit_vector( 7 DOWNTO 0 ) );
  END COMPONENT;
  FOR u1,u3: reg_8 USE ENTITY WORK.ls377( dataflow );
  COMPONENT adder
     PORT ( c0 : IN qit; c4 : OUT qit; a4, b4: IN qit_vector( 3 DOWNTO 0 ) ;
            s4 : OUT qit_vector( 3 DOWNTO 0 ) );
  END COMPONENT;
  FOR u4,u5: adder USE ENTITY WORK.ls283( behavioral );
  COMPONENT transceiver
     PORT ( g_bar : IN qit_vector( 1 DOWNTO 0 );
            a8 : IN qit_vector( 7 DOWNTO 0 );
            y8 : OUT qit_vector( 7 DOWNTO 0 ) );
  END COMPONENT;
  FOR u6: transceiver USE ENTITY WORK.ls541( dataflow );
  COMPONENT multiplexer
     PORT ( g_bar, s : IN qit; a4, b4 : IN qit_vector( 3 DOWNTO 0 );
            y4 : OUT qit_vector( 3 DOWNTO 0 ) );
  END COMPONENT;
  FOR u7,u8 : multiplexer USE ENTITY WORK.ls157( dataflow );
  COMPONENT shifter
     PORT ( clk, clr_bar, lin, rin : IN qit;
            mode1, mode0 : IN qit;
            g_bar : IN qit_vector( 1 DOWNTO 0 );
            qh, qa : OUT qit;
            qq : INOUT wired_qit_vector( 7 DOWNTO 0 ) );
  END COMPONENT;
  FOR u2 : shifter USE ENTITY WORK.ls299( behavioral );
  SIGNAL mcbus, hpbus : qit_vector ( 7 DOWNTO 0 ) := "ZZZZZZZZ";
  SIGNAL sbus, reg_inp:qit_vector ( 7 DOWNTO 0 ) := "ZZZZZZZZ";
  SIGNAL f : qit_vector( 8 DOWNTO 0 );
  SIGNAL s8, c4  : qit;  SIGNAL gnd : qit := '0';  SIGNAL vcc : qit := '1';
  SIGNAL tr_out : qit_vector( 7 DOWNTO 0 ) := "ZZZZZZZZ";
```

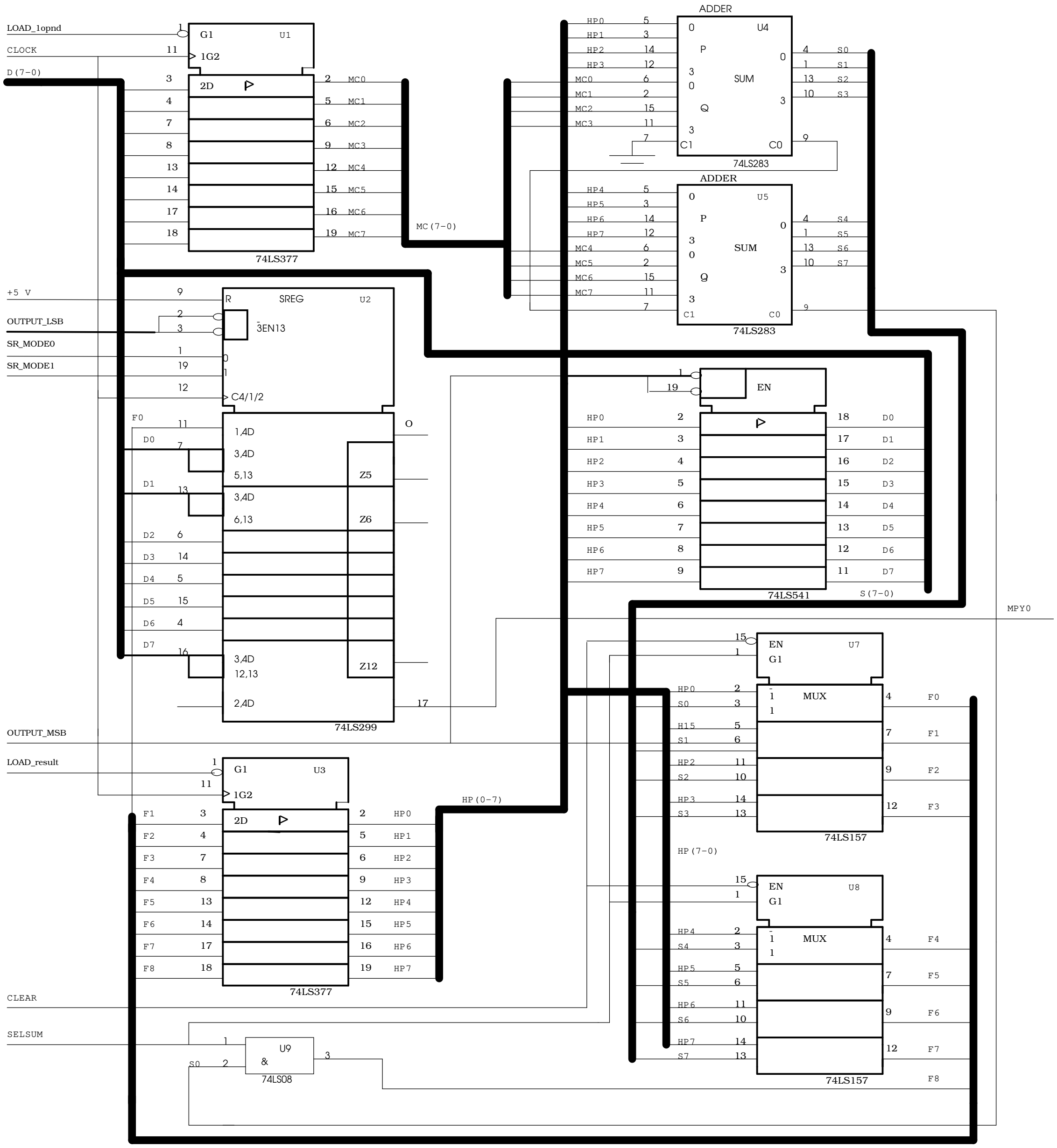
- Declaration of data part includes component declarations
- Components are bound to 74LS parts

MSI_design.control_and_data

```
BEGIN
    -- bus connections  --
    reg_inp <=  qit_vector( databus );     databus <=  wired_qit_vector ( tr_out );
    -- gate connections --
    u9: and_gate PORT MAP ( s8,selsum, f( 8 ) );
    -- register connections --
    u1: reg_8 PORT MAP ( clk, load_1opnd_bar, reg_inp, mcbus );
    u3: reg_8 PORT MAP ( clk, load_result_bar, f( 8 DOWNTO 1 ), hpbus );
    u6: transceiver PORT MAP ( output_msb_bar, hpbus, tr_out );
    -- connection of logical and register structures --
    u4: adder          PORT MAP ( gnd, c4, hpbus( 3 DOWNTO 0 ),
                       mcbus( 3 DOWNTO 0 ),sbus( 3 DOWNTO 0 ) );
    u5: adder          PORT MAP ( c4, s8, hpbus( 7 DOWNTO 4 ),
                       mcbus( 7 DOWNTO 4 ), sbus( 7 DOWNTO 4 ) );
    u2: shifter        PORT MAP ( clk ,vcc, gnd, f(0), sr_mode1, sr_mode0,
                       output_lsb_bar,  mpy0, OPEN, databus );
    u7: multiplexer    PORT MAP ( clr, selsum, hpbus( 3 DOWNTO 0 ),
                       sbus( 3 DOWNTO 0 ), f( 3 DOWNTO 0 ) );
    u8: multiplexer    PORT MAP ( clr, selsum, hpbus( 7 DOWNTO 4 ),
                       sbus( 7 DOWNTO 4 ), f( 7 DOWNTO 4 ) );
END structural;
```

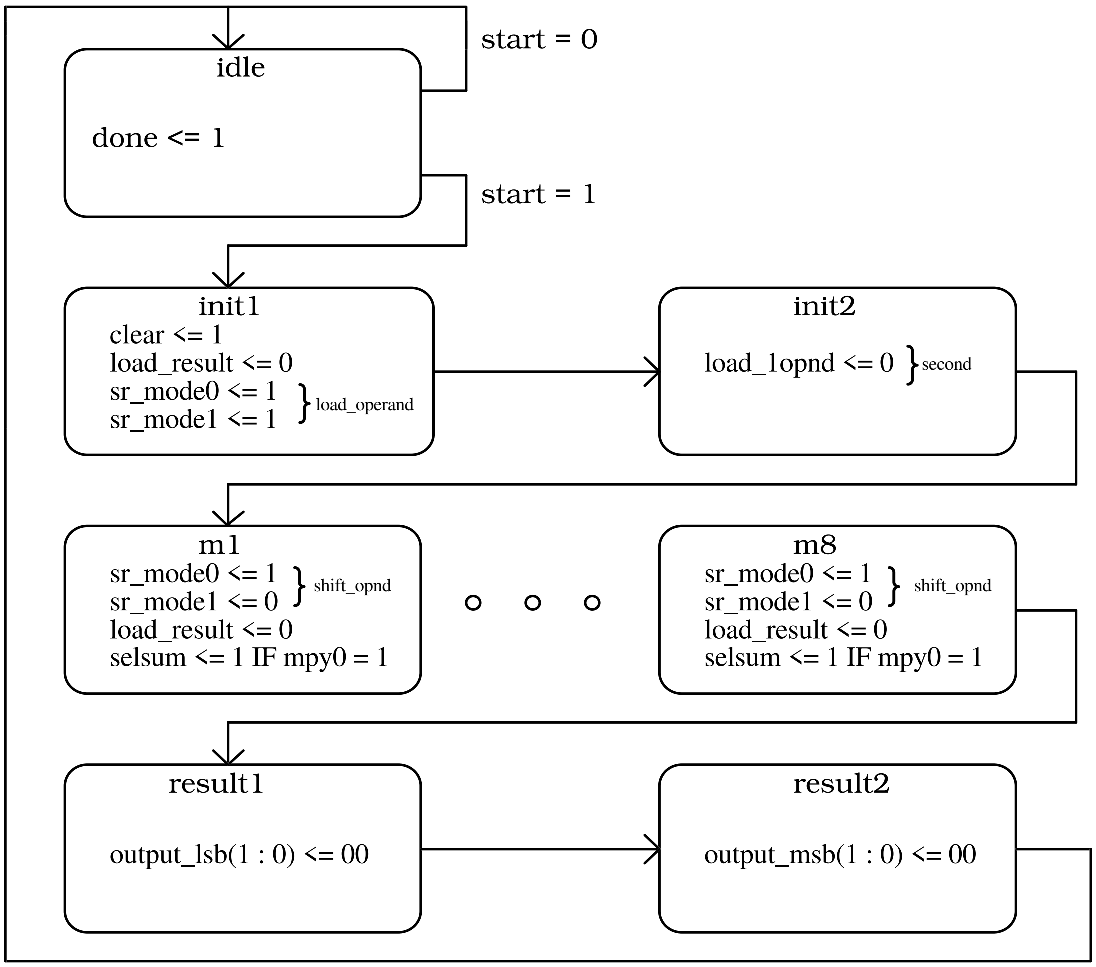
- Wiring components
- Bus connections convert between resolved and unresolved
- Control signals are connected to mode inputs of 74LS parts

MSI_design.control_and_data



- Wiring details using 74LS series
- Figure corresponds to VHDL description of data

MSI_design.control_and_data



- Controller is a simple state machine
- In each state appropriate signals are issued

MSI_design.control_and_data

```
ENTITY multiplier_controller IS
    PORT ( clock, start, mpy0     : IN qit;
                output_msb, output_lsb  : OUT qit_vector (1 DOWNTO 0) := "11";
                sr_mode0, sr_mode1    : OUT qit := '0';
                load_result, load_1opnd       : OUT qit := '1';
                selsum, clear, done       : OUT qit := '0' );
END multiplier_controller;
```

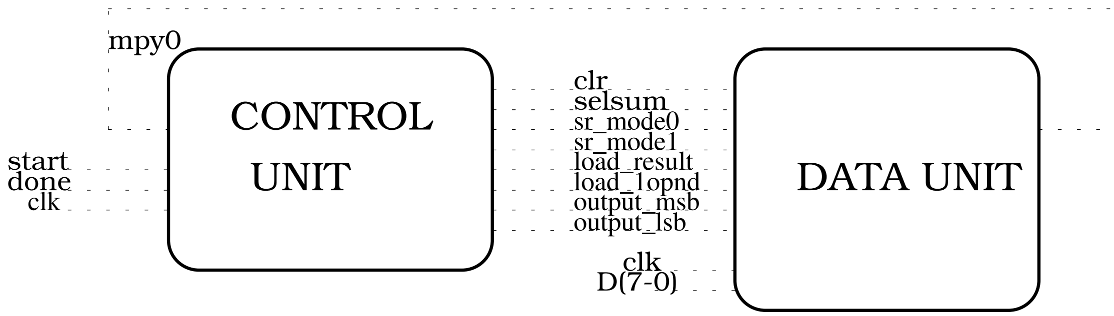
- Interface of controller includes all signals to data part
- The only inputs are clock, start and least significant bit of operand B

MSI_design.control_and_data

```
ARCHITECTURE state_machine OF multiplier_controller IS
    TYPE states IS ( idle, init1, init2,
                     m1, m2, m3, m4, m5, m6, m7, m8,
                     result1, result2 );
    SIGNAL present : states := idle;
BEGIN
    PROCESS  BEGIN
       CASE present IS
          WHEN idle  =>  done <= '1';    WAIT UNTIL clock = '1';
               IF start = '1' THEN      present <= init1;  ELSE present <= idle;
               END IF;
          WHEN init1 =>  clear <= '1';  load_result <= '0';
                sr_mode0 <= '1'; sr_mode1 <= '1';
                WAIT UNTIL clock = '1'; present <= init2;
          WHEN init2 =>  load_1opnd <= '0';
                WAIT UNTIL clock = '1'; present <= m1;
          WHEN m1  | m2  | m3  | m4  | m5  | m6  | m7 =>
                sr_mode0 <= '1'; sr_mode1 <= '0';  load_result <= '0';
                IF mpy0 = '1' THEN selsum <= '1'; END IF;
               WAIT UNTIL clock = '1';  present <= states'RIGHTOF(present);
          WHEN m8 =>   sr_mode0 <= '1';
                sr_mode1 <= '0';   load_result <= '0';
                IF mpy0 = '1' THEN selsum <= '1'; END IF;
               WAIT UNTIL clock = '1';  present <= result1;
          WHEN result1 =>  output_lsb <= "00";
               WAIT UNTIL clock = '1';  present <= result2;
          WHEN result2 =>  output_msb <= "00";
               WAIT UNTIL clock = '1';  present <= idle;
       END CASE;
       WAIT ON present'TRANSACTION;
       done         <= '0';   clear        <= '0';       load_result      <= '1';
       load_1opnd    <= '1';   selsum     <= '0';      sr_mode0    <= '0';
       sr_mode1   <= '0';   output_lsb <= "11";  output_msb  <= "11";
    END PROCESS;
END state_machine;
```

- Wait statements implement clocking
- Style offers flexible clocking

MSI_design.control_and_data



- Data and control are completed
- Can use 74LS parts for data
- Can synthesize the control part
- Individual data parts can be synthesized for an ASIC or full custom design
- The completed design can be simulated

MSI_design.testing

```
USE WORK.q_utilities.ALL;
USE STD.TEXTIO.ALL;
ENTITY multiplier_tester IS END multiplier_tester;
ARCHITECTURE input_output OF multiplier_tester IS
   COMPONENT multiplier_8_bit
       PORT ( clk : IN qit; databus : INOUT wired_qit_vector(7 DOWNTO 0);
               start : IN qit; done : OUT qit );
   END COMPONENT;
   FOR unit: multiplier_8_bit USE ENTITY WORK.shift_and_add (structural);
   SIGNAL clock, start, done : qit;
   SIGNAL data : wired_qit_vector (7 DOWNTO 0)  := "ZZZZZZZZ";
BEGIN
   unit: multiplier_8_bit PORT MAP (clock, data, start, done);
   clk : clock <= NOT clock AFTER 1 US WHEN NOW < 140 US ELSE clock;
   st  : start <= '0','1' after 500 NS, '0' AFTER 5000 NS ;
   read: PROCESS
     FILE data_file : TEXT IS IN "data_in.dat";
     VARIABLE l : LINE;     VARIABLE data_buff1 : bit_vector(7 DOWNTO 0);
     VARIABLE data_buff2 : qit_vector(7 DOWNTO 0);
   BEGIN
      WAIT UNTIL clock = '1';
      IF start = '1' THEN     READLINE ( data_file, l );
        std.textio.READ (l, data_buff1);
        data_buff2 := bit2qit(data_buff1);
        data <= wired_qit_vector(data_buff2);
        WAIT UNTIL clock = '0';   ELSE
        data <= "ZZZZZZZZ";
      END IF;
   END PROCESS;
END input_output;
```

- A testbench instantiates part under test, produces clock, applies data
- READLINE and READ read an external data file

MSI_design.testing

ns+delta	clock	start	done	q8	iq	data bus
0+0	0	0	0	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ
0+1	0	0	1	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ
500+0	0	1	1	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ
1000+0	1	1	1	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ
1000+1	1	1	1	ZZZZZZZZ	ZZZZZZZZ	00101111
1000+2	1	1	0	ZZZZZZZZ	ZZZZZZZZ	00101111
2000+0	0	1	0	ZZZZZZZZ	ZZZZZZZZ	00101111
3000+0	1	1	0	ZZZZZZZZ	ZZZZZZZZ	00101111
3000+1	1	1	0	ZZZZZZZZ	00101111	00111011
3007+0	1	1	0	00000000	00101111	00111011
4000+0	0	1	0	00000000	00101111	00111011
5000+0	1	0	0	00000000	00101111	00111011
5000+1	1	0	0	00000000	00101111	ZZZZZZZZ
6000+0	0	0	0	00000000	00101111	ZZZZZZZZ
7000+0	1	0	0	00000000	00101111	ZZZZZZZZ
7000+1	1	0	0	00000000	10010111	ZZZZZZZZ
7007+0	1	0	0	00011101	10010111	ZZZZZZZZ
8000+0	0	0	0	00011101	10010111	ZZZZZZZZ
9000+0	1	0	0	00011101	10010111	ZZZZZZZZ
9000+1	1	0	0	00011101	01001011	ZZZZZZZZ
9007+0	1	0	0	00101100	01001011	ZZZZZZZZ
10000+0	0	0	0	00101100	01001011	ZZZZZZZZ
11000+0	1	0	0	00101100	01001011	ZZZZZZZZ
11000+1	1	0	0	00101100	10100101	ZZZZZZZZ
11007+0	1	0	0	00110011	10100101	ZZZZZZZZ
12000+0	0	0	0	00110011	10100101	ZZZZZZZZ
13000+0	1	0	0	00110011	10100101	ZZZZZZZZ
13000+1	1	0	0	00110011	01010010	ZZZZZZZZ
13007+0	1	0	0	00110111	01010010	ZZZZZZZZ
14000+0	0	0	0	00110111	01010010	ZZZZZZZZ
15000+0	1	0	0	00110111	01010010	ZZZZZZZZ
15000+1	1	0	0	00110111	00101001	ZZZZZZZZ
15007+0	1	0	0	00111001	00101001	ZZZZZZZZ
16000+0	0	0	0	00111001	00101001	ZZZZZZZZ
17000+0	1	0	0	00111001	00101001	ZZZZZZZZ
17000+1	1	0	0	00111001	10010100	ZZZZZZZZ
17007+0	1	0	0	00011100	10010100	ZZZZZZZZ
18000+0	0	0	0	00011100	10010100	ZZZZZZZZ
19000+0	1	0	0	00011100	10010100	ZZZZZZZZ
19000+1	1	0	0	00011100	11001010	ZZZZZZZZ
19007+0	1	0	0	00101011	11001010	ZZZZZZZZ
20000+0	0	0	0	00101011	11001010	ZZZZZZZZ
21000+0	1	0	0	00101011	11001010	ZZZZZZZZ
21000+1	1	0	0	00101011	11100101	ZZZZZZZZ
21007+0	1	0	0	00010101	11100101	ZZZZZZZZ
21009+0	1	0	0	00010101	11100101	11100101
22000+0	0	0	0	00010101	11100101	11100101
23000+0	1	0	0	00010101	11100101	11100101
23000+3	1	0	0	00010101	11100101	ZZZZZZZZ
23007+1	1	0	0	00010101	11100101	00010101
24000+0	0	0	0	00010101	11100101	00010101
25000+0	1	0	0	00010101	11100101	00010101
25000+2	1	0	1	00010101	11100101	00010101
25000+4	1	0	1	00010101	11100101	ZZZZZZZZ
26000+0	0	0	1	00010101	11100101	ZZZZZZZZ
27000+0	1	0	1	00010101	11100101	ZZZZZZZZ

- Simulation results verify multiplier operation

MSI_design.conclusions

1. Outline: Introduction, Organization, Outline

2. Review: Levels of abstraction, Entity and Architecture, Signal assignments, Guarded signal assignments, Three state bussing, Process statements, Combinational processes, Sequential processes, Multiplexing, Package

3. MSI Based Design: Use MSI par

ERROR: undefined
OFFENDING COMMAND: −0.1cod.000

STACK: