Solution to Assignment 3
RTL Design

Question 1

Preliminary Design

This solution follows the approach suggested in the notes titled *Hints for Assignment 3*. The register names and widths are:

register	width
rc	33
rb	31
ra	15

the required operations and register transfers are:

- set `rc` to zero, `ra` to `a`, and `rb` to `b`
- set `rc` to the sum of `rc` and `rb`
- shift `ra` right by 1 bit and shift `rb` left by 1 bit

one set of datapath control signals is:

- `load`
- `add`
- `shift`

and status signals:

- `azero` - register `a` is zero
- `lsbset` - L.S. bit of `a` is a '1'

one set of controller states is:

state	operations
`init`	none
`add_shift`	add `rb` to `rc` if LS bit of `ra` is 1 and shift `ra` and `rb`
`done`	none

and the state transition table is:

state	reset	azero	lsbset	next state
X	1	X	X	`init`
`init`	0	0	X	`add_shift`
`init`	0	1	X	`done`
`add_shift`	0	0	X	`add_shift`
`add_shift`	0	1	X	`done`

Note that the controller must be a Mealy state machine. This is because the current state is unknown during the first cycle (which is the only time when the `a` and `b` inputs are valid and the `reset` input is asserted). Therefore the `load` output cannot be made a function of the current state. In this case it is a function only of the `reset` input (in fact, it is the same as `reset`). The `init` state could be removed since it serves no purpose in this design.

VHDL Code

This design can be coded in VHDL as follows:

```
-- ELEC 379 Assignment 3 Solution
-- 16x16 bit unsigned multiplier, RTL-style
-- Ed Casas, Feb 4 1998

-- multiplier type declarations

package mult_types is
   -- controller states
   type states is ( init, add_shift, done ) ;
end mult_types ;

-- datapath

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

entity datapath is
   port (
   a, b : in unsigned (15 downto 0) ;
   c : out unsigned (31 downto 0) ;
   load, add, shift : in std_logic ;
   azero, lsbset : out std_logic ;
   clk : in std_logic ) ;
end datapath ;

architecture rtl of datapath is
   -- registers and register inputs ('next' values)
   signal rc, nrc : unsigned (31 downto 0) ;
   signal rb, nrb : unsigned (30 downto 0) ;
   signal ra, nra : unsigned (15 downto 0) ;
begin

   process(load,add,shift,a,b,ra,rb,rc)
   begin
      -- load operation
      if load = '1' then
```

```
         nra <= conv_unsigned(a,nra'length) ;
         nrb <= conv_unsigned(b,nrb'length) ;
         nrc <= conv_unsigned(0,nrc'length) ;
      else
         -- shift ra and rb
         if shift = '1' then
            nra <= '0' & ra (ra'left downto 1) ;
            nrb <= rb (rb'left-1 downto 0) & '0' ;
         else
            nra <= ra ;
            nrb <= rb ;
         end if ;
         -- add rb to rc
         if add = '1' then
            nrc <= rc + rb ;
         else
            nrc <= rc ;
         end if ;
      end if ;
   end process ;

   -- registers
   process(clk,nra,nrb,nrc)
   begin
      if clk'event and clk = '1' then
         ra <= nra ;
         rb <= nrb ;
         rc <= nrc ;
      end if ;
   end process ;

   -- "a is zero" output (converting to std_logic
   -- avoids warnings when comparing)
   azero <= '1' when std_logic_vector(ra) =
                     conv_std_logic_vector(0,ra'length)
                else '0' ;

   -- "LS bit of ra set" output
   lsbset <= '1' when ra(0) = '1' else '0' ;

   -- product output
   c <= rc ;

end rtl ;
```

The code for the design of the state machine is:

```
-- controller

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use work.mult_types.all ;

entity controller is
   port (
   azero, lsbset, reset : in std_logic ;
   load, add, shift : out std_logic ;
   clk : in std_logic ) ;
end controller ;

architecture rtl of controller is
   signal s, ns : states ;
begin

   -- next-state logic
   ns <=
      init when reset = '1' else
      done when azero = '1' else
      add_shift ;

   -- state register
   process(clk,ns)
   begin
      if clk'event and clk = '1' then
         s <= ns ;
      end if ;
   end process ;

   -- datapath control
   load  <= '1' when reset = '1' else '0' ;
   add   <= '1' when s = add_shift and lsbset = '1' else '0' ;
   shift <= '1' when s = add_shift else '0' ;

end rtl ;
```

The two entities are placed in a package:

```
-- component package

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use work.mult_types.all ;

package mult_components is

   component datapath
      port (
      a, b : in unsigned (15 downto 0) ;
      c : out unsigned (31 downto 0) ;
      load, add, shift : in std_logic ;
      azero, lsbset : out std_logic ;
      clk : in std_logic ) ;
   end component ;

   component controller
      port (
      azero, lsbset, reset : in std_logic ;
      load, add, shift : out std_logic ;
      clk : in std_logic ) ;
   end component ;

end mult_components ;
```

The last part is the multiplier top-level entity which just instantiates the datapath and controller:

```
-- multiplier

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use work.mult_components.all ;

entity mult is
   port (
   a, b : in unsigned (15 downto 0) ;
   c : out unsigned (31 downto 0) ;
   clk, reset : in std_logic ) ;
end  mult ;
```

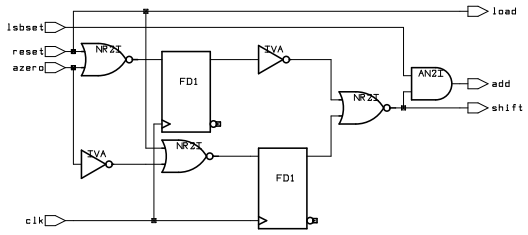
```
architecture rtl of mult is

   signal azero, lsbset : std_logic ;
   signal load, add, shift : std_logic ;

   for all : datapath use entity work.datapath (rtl) ;
   for all : controller use entity work.controller (rtl) ;

begin

   d1: datapath port map (
      a, b,
      c,
      load, add, shift,
      azero, lsbset,
      clk ) ;

   c1: controller port map (
      azero, lsbset, reset,
      load, add, shift,
      clk ) ;

end rtl ;
```

Schematics

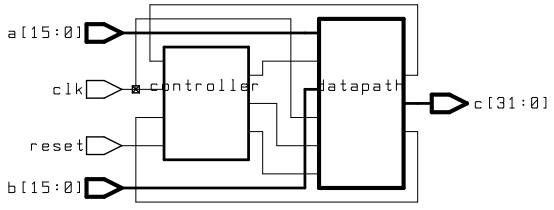
One schematic will be generated for each of the two design units (datapath and controller) in the hierarchy and one for the top level (you may also have gotten a separate schematic for the adder).

The schematic for the datapath is the largest one since the three registers require 79 flip-flops plus the combinational logic to multiplex their inputs. It is not shown here to save space.

The state machine schematic is much simpler:



and the top-level schematic simply shows how the other entities are connected:



Simulation Log

Finally, the simulation log shows how the product 11×26 is computed:

```
# run
       Test: 0
      Input: a=11=0000000000001011 b=26=0000000000011010 clk=0 reset=1
     Output: c=?=UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
       Test: 1
      Input: a=11=0000000000001011 b=26=0000000000011010 clk=1 reset=1
     Output: c=0=00000000000000000000000000000000
       Test: 2
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=0 reset=0
     Output: c=0=00000000000000000000000000000000
       Test: 3
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=1 reset=0
     Output: c=0=00000000000000000000000000000000
       Test: 4
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=0 reset=0
     Output: c=0=00000000000000000000000000000000
       Test: 5
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=1 reset=0
     Output: c=26=00000000000000000000000000011010
       Test: 6
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=0 reset=0
     Output: c=26=00000000000000000000000000011010
       Test: 7
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=1 reset=0
     Output: c=78=00000000000000000000000001001110
       Test: 8
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=0 reset=0
     Output: c=78=00000000000000000000000001001110
       Test: 9
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=1 reset=0
     Output: c=78=00000000000000000000000001001110
       Test: 10
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=0 reset=0
     Output: c=78=00000000000000000000000001001110
       Test: 11
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=1 reset=0
     Output: c=286=00000000000000000000000100011110
       Test: 12
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=0 reset=0
     Output: c=286=00000000000000000000000100011110
       Test: 13
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=1 reset=0
     Output: c=286=00000000000000000000000100011110
       Test: 14
      Input: a=0=0000000000000000 b=0=0000000000000000 clk=0 reset=0
     Output: c=286=00000000000000000000000100011110
(vhdlsim): Simulation complete, time is 1500000000 FS.
# quit
```