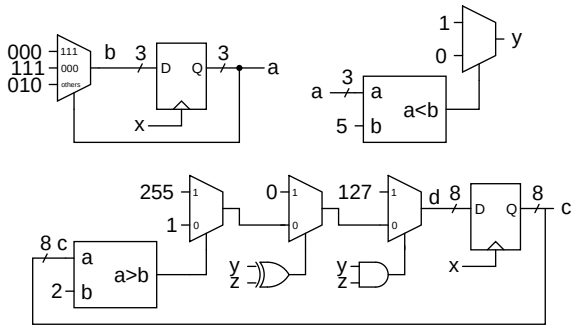
Solutions to Assignment 2

Question 1

The schematic is shown below. Following the conventions used in the course notes, the selected assignment is drawn as a single multi-input multiplexer and the conditional assignment is drawn using a two-input multiplexer. Other symbols could be used for the comparison and logic functions as long as the meaning was unambiguous.



Question 2

The VHDL consists of one selected assignment and one process statement to instantiate the `m` register:

```
library ieee;
use ieee.std_logic_1164.all;

entity sol2q2 is
  port (
    clock : in std_logic ;
    h : out std_logic ;
    m : out std_logic_vector (2 downto 0) ) ;
end sol2q2;

architecture rtl of sol2q2 is
  signal x : std_logic_vector (3 downto 0) ;
  signal m_q : std_logic_vector (2 downto 0);
begin

  h <= x(3) ;
  m <= m_q ;

  with m_q select
    x <=
    x"2" when "000",
    x"4" when "001",
    x"8" when "100",
    x"F" when others;

  process(clock)
  begin
    if clock'event and clock = '1' then
      m_q <= x(2 downto 0) ;
    end if ;
  end process ;

end rtl ;
```

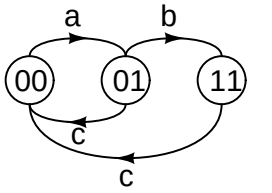
Question 3

The corresponding VHDL is:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sol2q3 is
  port (
    hirate, hold, safe, run, c10k : in std_logic ;
    freq : out unsigned (11 downto 0) );
end sol2q3;

architecture rtl of sol2q3 is
  signal freq_q, next_freq : unsigned (11 downto 0);
begin

  next_freq <=
   to_unsigned(0,freq_q'length) when (not ( safe = '1' and run = '1' ) ) else
    freq_q when hold = '1' else
    freq_q * 2 when hirate = '1' else
    freq_q + 1 ;

  process(c10k)
  begin
    if c10k'event and c10k = '1' then
      freq_q <= next_freq ;
    end if ;
  end process ;

  freq <= freq_q ;

end rtl;
```

Question 4

The state transition diagram is shown below where the least-significant bit of the state represents `on` and the most-significant bit is `fast`:



a: start, b: speed,
c: stop and not start

The corresponding VHDL code is:

```
library ieee;
use ieee.std_logic_1164.all;

entity sol2q4 is
  port (
    start, stop, speed : in  std_logic;
    clock              : in  std_logic ;
    on_out, fast       : out std_logic);
end sol2q4;

architecture rtl of sol2q4 is
```

```
  signal state, next_state : std_logic_vector (1 downto 0) ;
begin

  on_out <= state(0) ;
  fast   <= state(1) ;

  next_state <=
    "01" when state = "00" and start = '1' else
    "11" when state = "01" and speed = '1' else
    "00" when stop = '1' and start = '0' else
    state ;

  process (clock)
  begin
    if clock'event and clock = '1' then
      state <= next_state ;
    end if;
  end process;

end rtl;
```

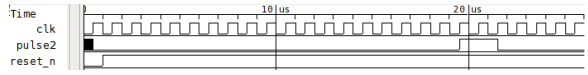
A missing `clock` input was added and the name of the `on` output was changed to `on_out` because `on` is a reserved word in VHDL. To make the code more compact (at the cost of some readability) I used a 2-bit state variable instead of an enumerated type.

Question 5

Here is a simple solution for a resettable clock divider that outputs a 2-period-long output once every 20 clock periods:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sol2q5 is
  port (
    clk, reset_n : in  std_logic;
    pulse2       : out std_logic);
end sol2q5;

architecture rtl of sol2q5 is
  signal count, next_count : unsigned (4 downto 0);
  signal next_pulse2 : std_logic;
begin

  next_count <=
    to_unsigned(0,count'length) when reset_n = '0' or count = 19 else
    count + 1 ;

  next_pulse2 <=
    '1' when count = 18 or count = 19 else
    '0' ;

  process (clk)
  begin
    if clk'event and clk = '1' then
      count <= next_count ;
      pulse2 <= next_pulse2 ;
    end if;
  end process;

end rtl;
```

Note that we register the output so that it is glitch-free. It is now delayed by one clock period relative to the value of `count`. The simulation results are:



Many other solutions are possible:

- since it's often simpler for hardware to check for a value of 0 (and 1) the implementation might be simpler if we counted down from 19 instead of counting up from 0,
- we might (or might not) get a simpler implementation by using a divide-by-2 stage followed by a divide-by-10 (or even divide by-2, -2 and -5) – all synchronous, of course,
- a 5-bit linear-feedback shift-register would replace the adder with an xor gate,
- a one-hot encoding with twenty flip-flops might be the fastest implementation on an FPGA (although probably requiring more hardware).
- at very high speeds (but not in this course) we might use a ripple counter.

Question 6

In the following solution I used an enumerated type instead of `std_logic_vector` values for the state. The outputs are:

State	G	Y
A	1	0
B	1	1
C	0	0

The VHDL code is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity controller is
  port ( en, f, clk : in std_logic ;
         G, Y : out std_logic ) ;
end controller ;

architecture rt of controller is

  type states is (A, B, C) ;
  signal state, next_state : states ;
  signal next_G, next_Y : std_logic;

begin

  next_state <=
    A when state = B and f = '1' and en = '1' else
    A when state = C and en = '1' else
    B when state = A and en = '1' else
    C when state = B and en = '1' else
    state ;

  next_G <= '1' when next_state = A or next_state = B else '0' ;
  next_Y <= '1' when next_state = B else '0' ;

  process (clk)
  begin
    if clk'event and clk = '1' then
      state <= next_state ;
      G <= next_G ;
      Y <= next_Y ;
    end if;
  end process;

end rt;
```