Mid-Term Exam Solutions

Question 1

The simplest solution uses the approach described in the hints: an unsigned 9-bit value is incremented on each `clk` rising edge and is reset when the `tdc` signal is asserted. The `spark` output is a combinational function of the `hispeed` input and the angle. The schematic generated for this architecture by Synopsis' *Design Compiler* is shown in Figure 1.

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

entity ignition is
   port (
   clk : in std_logic ;
   tdc, hispeed : in std_logic ;
   spark : out std_logic ) ;
end ignition ;

architecture rtl of ignition is
   signal angle, nextangle : unsigned (8 downto 0) ;
begin
   -- crankshaft angle counter
   nextangle <=
      conv_unsigned(0,9) when tdc = '1' else
      angle + conv_unsigned(1,9) ;

   process(clk)
   begin
      if clk'event and clk = '1' then
         angle <= nextangle ;
      end if ;
   end process ;

   -- spark output
   spark <=
      '1' when (
         hispeed = '1' and
         angle >= conv_unsigned(355,9) and
         angle <= conv_unsigned(359,9) ) else
      '1' when (
         hispeed = '0' and
         angle >= conv_unsigned(0,9) and
         angle <= conv_unsigned(4,9) ) else
      '0' ;
end rtl ;
```

Although this solution is acceptable, it has three flaws: (1) if the `hispeed` input changes while the angle is between 355 and 4 degrees the output would not correspond to either of the desired outputs, (2) the output is not registered and thus will contain glitches, and (3) the use of arithmetic comparisons will probably result in an implementation requiring four 9-bit comparators. A better solution requires registering the `hispeed` input before it is used, registering the `spark` output and using a state machine instead of arithmetic comparisons:

```
architecture rtl of ignition is
   signal advance, nextadvance : std_logic ;
   signal ospark, nextospark : std_logic ;
   signal angle, nextangle : unsigned (8 downto 0) ;
begin
   -- crankshaft angle counter
   nextangle <=
      conv_unsigned(0,9) when tdc = '1' else
      angle + conv_unsigned(1,9) ;

   process(clk)
   begin
      if clk'event and clk = '1' then
         angle <= nextangle ;
      end if ;
   end process ;

   -- latch 'hispeed' as 'advance' at 354 degrees
   nextadvance <=
      hispeed when angle = conv_unsigned(353,9) else
      advance ;

   process(clk)
   begin
      if clk'event and clk = '1' then
         advance <= nextadvance ;
      end if ;
   end process ;

   -- spark output state machine
   nextospark <=
      '1' when advance = '1' and
         angle = conv_unsigned(354,9) else
      '0' when advance = '1' and
         angle = conv_unsigned(359,9) else
      '1' when advance = '0' and
         angle = conv_unsigned(359,9) else
      '0' when advance = '0' and
         angle = conv_unsigned(  4,9) else
      ospark ;

   process(clk)
   begin
      if clk'event and clk = '1' then
         ospark <= nextospark ;
      end if ;
   end process ;

   -- registered spark control output
   spark <= ospark ;
```

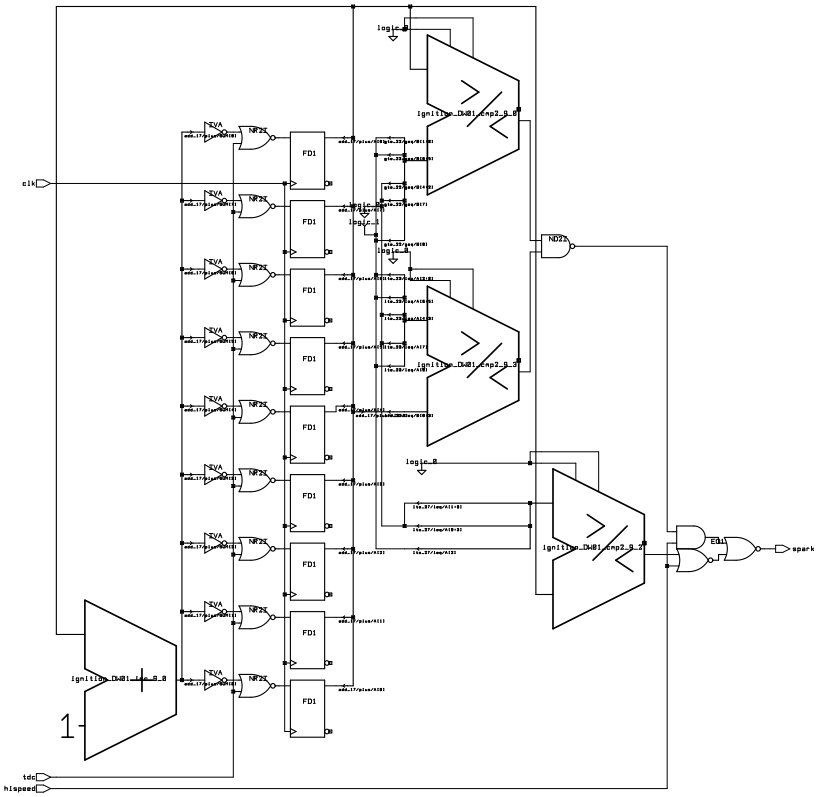

Figure 1: Simple Solution for Question 1.

```
end rtl ;
```

Which results in the simpler and more reliable design shown in Figure 2.

Question 2

```
;
; copy 64-byte buffer at C4000H to port 220H
; ELEC 379 Mid-Term Exam Question 2
; Ed Casas, 1998/11/26
;

copyout:
    push  ax       ; save working registers
    push  bx
    push  cx
    push  dx
    push  ds

    mov   ax,c400  ; DS:AX points to buffer
    mov   ds,ax
    mov   bx,0

loop:
    mov   al,[bx]  ; get byte from buffer

    mov   cl,220   ; write to O/P port
    out   cl,al

    add   bx,1     ; point to next byte

    cmp   bx,64    ; repeat 64 times
    jnz   loop

    pop   ds       ; restore registers
    pop   dx
    pop   cx
    pop   bx
    pop   ax

ret
```

Question 3

(a) The 12 LS bits ($4\text{kB} = 2^{10} \times 2^2$) are used to select a byte within the 4 kB region, so the decoder uses the $(16 - 12 = 4)$ MS bits. The MS 4 bits of the start address, 4000H, are 0100. The answer is:

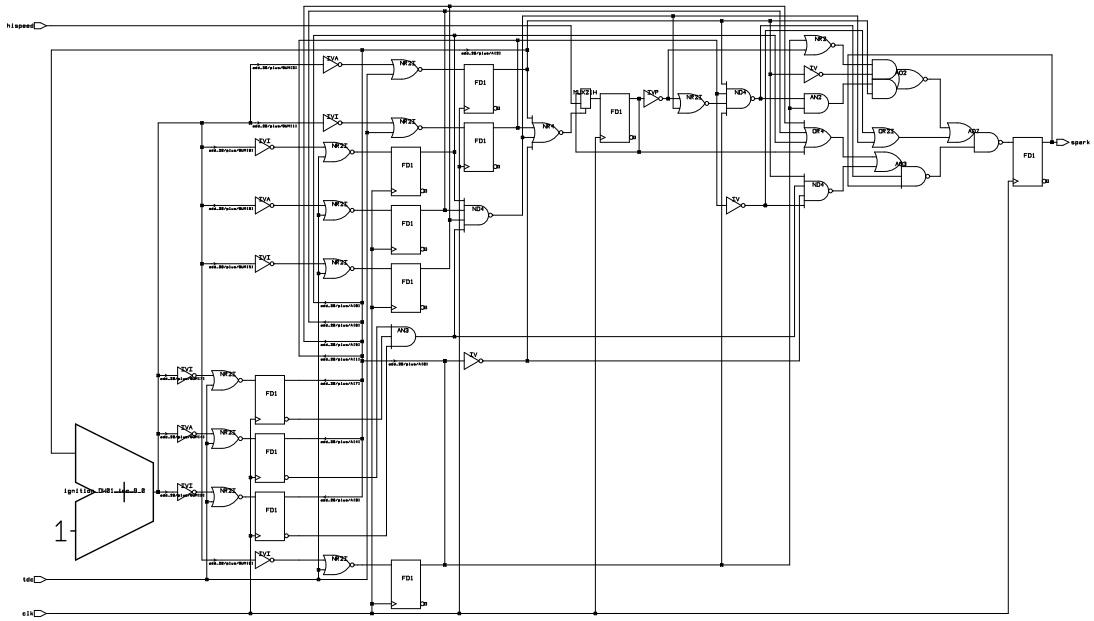

Figure 2: Improved Solution for Question 1.

```
en <=
   '0' when a(15 downto 12) = "0100" else
   '1' ;
```

(b) The `mov [45],ax` instruction stores two bytes starting at location 45, so locations 45 and 46 will be altered.

The 80386SX CPU has a 16-bit data bus which allows simultaneous access to two bytes if they are at an even/odd address pair. However, address 45 is the MS byte of the word at address 44/45, while byte 46 is the LS byte of the word at address 46/47, so the CPU will execute two write cycles to execute these instructions.

The Intel 80x86 processors store multi-byte values in little-endian order so location 45 will be set to the LS byte, 34H and location 46 will be set to the MS byte, 12H.

The last instruction loads AL, the LS 8 bits of AX, with the value at location 46, 12. The final value in AX is thus 1212H.

(c) The two choices for RAM are SRAM and DRAM. SRAM, although more expensive on a per-bit basis, has faster access times and and does not require complex refresh circuitry. For a small and fast memory system SRAM would be the better choice.