Problem-Set #1 – COE838

Introduction and SystemC

Q1. Suppose we have four different processors; each does 25% of the application. If we improve two of the processors by 10 times, what would be the overall application speedup?

Let 100 percent application is completed in Y time units
P1 and P2 runs 10 time faster the 50% of the application will execute in (Y/2)/10 time units
50% application execute in Y/2 time units
Total time of application = Y/2 + Y/20 = 11Y/20 Speedup = Y/(11Y/20) = 20/11 = 1.818 time

(Also done in the class)

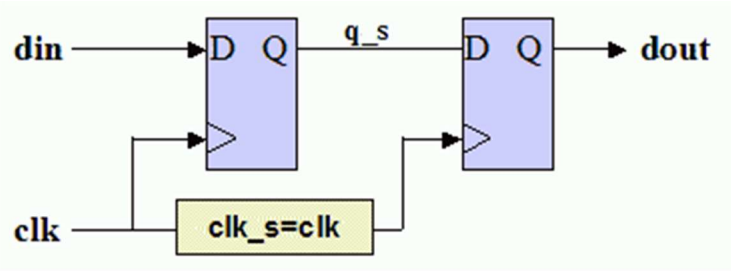
Q2. Write a System module to model a D-type flip-flop.
(Do it Your self as done in the Lab)

Q3. Identify the main function of the following SystemC module. Explain the module briefly.

```
SC_MODULE(what) {
   sc_in<bool >  clk;
   sc_out<bool > dout1;
   sc_out<bool > dout2;
   sc_signal<bool> sig1;
   bool var2 ;
     void p1() {    // using signals
                    sig1.write(!sig1.read());
                    dout1.write(sig1.read());
     }
    void p2() {   // using variables
                    var2 = !var2;
                    dout2.write(var2);
    }
   SC_CTOR(what) {
          sig1.write(true);   // init
          var2=false;        // init
          SC_METHOD(p1);
                    sensitive << clk.pos();
                    dont_initialize();
          SC_METHOD(p2);
                    sensitive << clk.pos();
                    dont_initialize();
   }
};
    }
};
```

(Done in the class) It is Toggle Flip Flop

Q4. Model the following two flip-flop hardware that by writing the code of a module. The module can have multiple METHODS or processes.




```
SC_MODULE(delta) {
   sc_in<bool >  clk;
   sc_in<bool >  din;
   sc_out<bool > dout;

   sc_signal<bool > q_s;
   sc_signal<bool > clk_s;

   void p1() {
      q_s.write(din.read());
   }
   void p2() {
      dout.write(q_s.read());
   }
   void p3() {
     clk_s.write(clk.read());
   }

   SC_CTOR(delta) {
      SC_METHOD(p1);
        sensitive << clk.pos();
      SC_METHOD(p2);
        sensitive << clk_s.posedge_event();
      SC_METHOD(p3);
        sensitive << clk;
   }
};
```

Q5. SystemC code of a counter module is given below. Identify the type of counter and which signal it counts. Justify your answer.

```
#include "systemc.h"
SC_MODULE (which_counter) {
        //-----------Input Ports---------------
        sc_in    <bool> enable, clk, reset ;
        //-----------Output Ports---------------
        sc_out   <sc_uint<8> > out ;
        //------------Internal Variables--------
        sc_uint<8>  count ;

        //-------------Code Starts Here---------
        void counter () {
          if (reset.read()) {
                count = 0 ;
          } else if (enable.read()) {
                count = count + 1 ;
            }
         out.write(count) ;
        }

 SC_CTOR(which_counter) {
        SC_METHOD (counter) ;
        sensitive << clk.pos() ;
 }
};
```

It is an up counter

Q6. Design and write the SystemC code of a down-counter module that will initialize its count at 1024 and then count the number of input pulses.

Do it yourself by following the above SystemC implementation of an up-counter in Q5.

Q7. Design and write the SystemC code for asynchronous read/write random access memory (RAM) module. The memory size is 512 bytes where the word size of the memory is of 16-bits. The memory has separate data-in and data-out ports of 16-bits each. In addition to memory address signals, the memory module has chip select (cs), output enable (oe) and write enable (we) input signals.

A solution is given below.

// Asynchronous read write RAM

```
#include "systemc.h"

#define DATA_WIDTH      16
#define ADDR_WIDTH      8
#define RAM_WIDTH       8

SC_MODULE (ram_sp_ar_aw) {
 sc_in   <sc_uint<ADDR_WIDTH> > address ;
 sc_in   <bool>             cs  ;
 sc_in   <bool>             we ;
 sc_in   <bool>             oe  ;
 sc_in   <sc_uint<DATA_WIDTH> > data_in ;
 sc_out   <sc_uint<DATA_WIDTH> > data_out ;

 //-----------Internal variables----------------
 sc_uint <DATA_WIDTH> mem [RAM_DEPTH] ;

 // Memory Write Block
 // Write: When we = 1, cs = 1
 void write_mem () {
  if (cs.read() && we.read()) {
    mem[address.read()] = data_in.read() ;
  }
 }

 // Memory Read Block
 // Read: When we = 0, oe = 1, cs = 1
 void read_mem () {
  if (cs.read() && !we.read() && oe.read())  {
    data_out.write(mem[address.read()]);
  }
 }

 SC_CTOR(ram_sp_ar_aw) {
  SC_METHOD (read_mem);
    sensitive << address << cs << we << oe ;
  SC_METHOD (write_mem);
    sensitive << address << cs << we << data_in ;
 }
};
```