SPH 336
TUTORIAL 2

13th November 2015

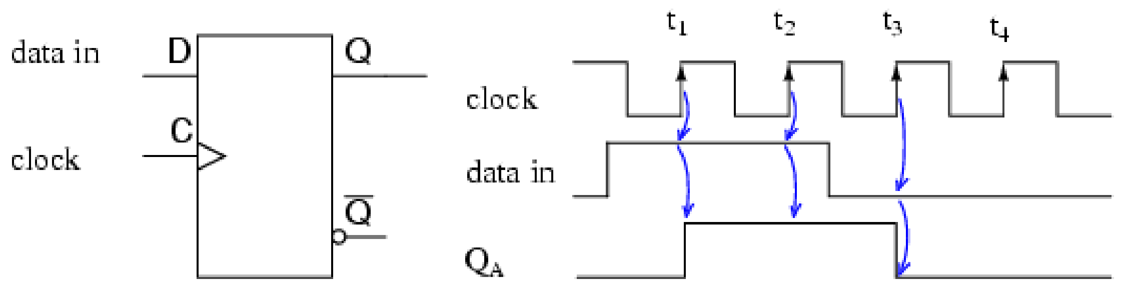
8 Bit Serial in Serial Out Register

Group Members
1.Thiong'o Chelmis Muthoni I39/2210/2013
2.Andati Lexy Amwayi A.L. I39/2218/2013

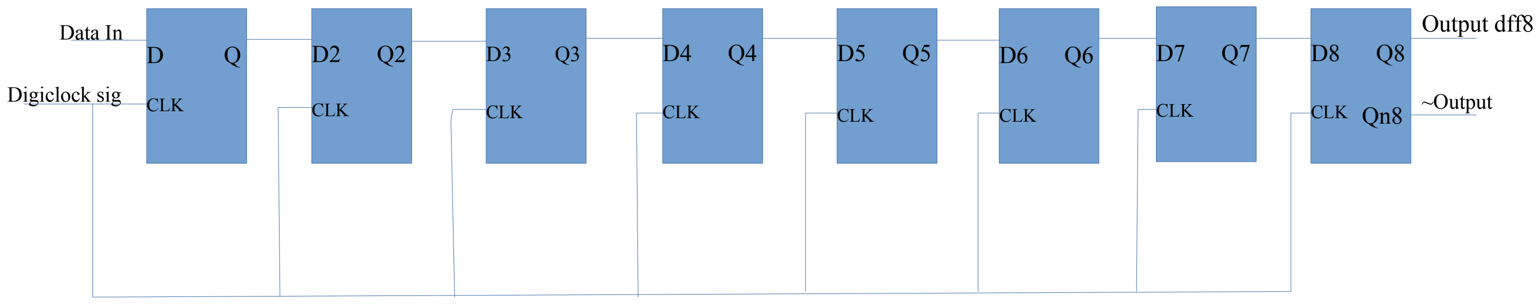
Objective
To model and simulate the operation of an 8-bit serial in serial Out (SISO) register using systemC.

Introduction
A register is a memory cell that holds data in a computer or any other computing device. An 8 bit register means that this register can hold 8 bits of data at a given time. Therefore, a serial in serial out register is a register whereby an input is fed into the register at one end in some order, and an output comes out of the end of that register in the exact same order in which they were fed into the register.

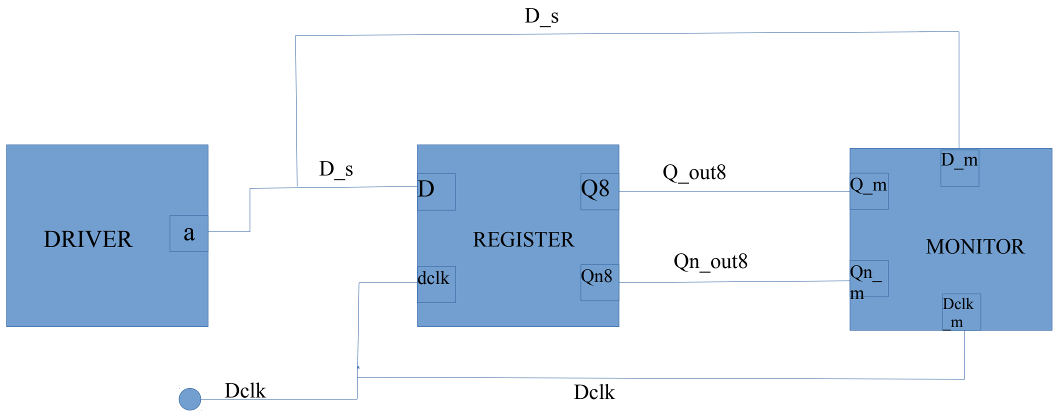
Methodology
It was decided that a D-flipflop would be used to build this type of register. A D-flipflop is a device that is capable of holding one bit of data at any single time. Therefore if cascaded with other D-flip flops, may be able to hold more bits of information at any given time. The following is a schematic of a D- flip flop:



It has an data input pin (D) a clock input pin (C), a data output (Q) and the inverted output(Qo). The initial output is low. Therefore, if this is a positive edge triggered D flip-flop, then when the clock goes from low to high and the data input D is high at this point, then the output will change to high, the same value as the input, else if the clock is low at this point then the output (Q) will not change. The same applies if the data signal is low when the clock is going from low to high, the output will change from high to low if it was high, or will remain low if it was already low. Thus by virtue of the output (Q) remaining the same despite the change in clock from high to low until the next clock cycle, we say that this flip flop “ stores data.” Therefore to create our register, we cascaded 8 of these flipflop forming the following circuit.



From this circuit, we were able to see how the data from one D flip-flop will flow into the next. And from that, we were able to design a model of computation as below:



<u>Driver Module</u>
The input to register is produced from here. It is communicated through the D_s signal

<u>Register Module</u>
This module consists of the implementation of the register using the flip flops. The D flip-flops were cascaded at this point in order to allow for data transfer from one flip-flop to the next. The clock sensitivity of the D flip-flop (ie. Which clock edge the output of the D flip-flop, Q, responds to) was also configured here.

<u>Monitor</u>
This was used to monitor the timing diagrams of the input to the register, the clock signal and the output of the register to determine if the simulation was working properly.

From the above model we were then able to easily develop our SystemC code shown below:

```
//driver.h
#include "systemc.h"

SC_MODULE(driver){
sc_out<bool> a;

SC_CTOR(driver){
      SC_THREAD(drive);
}

void drive(void){
      while(1){
            a=0;
            wait(1.5,SC_MS);
            a=1;
            wait(1.5,SC_MS);
      }
}
};
```

//dff.h

```
#include"systemc.h"

//declaring all the Dffs here. They are 8 in number. Each output of one Dff is
an input to the next hence the similarity in the naming of the ports. The last
outputs are those of dff8 hence the need to have the final Qnot at that point.
SC_MODULE (Dff)
{
      sc_in<bool> D, dclk;
      sc_out<bool> Q;

      SC_CTOR (dff)
      {
            SC_METHOD(decode);
            sensitive<<dclk.pos();
            dont_initialize();
      }

      ~dff(){}

      void decode(void)
      {
            Q = D.read();
            //Qn =!(D.read()) ;
      }
};
SC_MODULE (dff2)  //2nd dff
{
      sc_in<bool> Q,dclk2;  //input of this dff is the output Q of dff 1.
      sc_out<bool> Q2;

      SC_CTOR (dff2)
      {
            SC_METHOD(decode);
            sensitive<<dclk2.pos();
            dont_initialize();
      }

      ~dff2(){}

      void decode(void)
      {
            Q2 = Q.read();
            //Qn2 =!(Q.read()) ;
      }
};
```

```
SC_MODULE (dff3)  //3rd dff
{
      sc_in<bool> Q2,dclk3;
      sc_out<bool> Q3;

      SC_CTOR (dff3)
      {
            SC_METHOD(decode);
            sensitive<<dclk3.pos();
            dont_initialize();
      }

      ~dff3(){}

      void decode(void)
      {
            Q3 = Q2.read();
            //Qn3 =!(Q.read()) ;
      }
};
SC_MODULE (dff4) // dff 4
{
      sc_in<bool> Q3,dclk4;
      sc_out<bool> Q4;

      SC_CTOR (dff4)
      {
            SC_METHOD(decode);
            sensitive<<dclk4.pos();
            dont_initialize();
      }

      ~dff4(){}

      void decode(void)
      {
            Q4 = Q3.read();
            //Qn4 =!(Q.read()) ;
      }
};
SC_MODULE (dff5) // dff 5
{
      sc_in<bool> Q4,dclk5;
      sc_out<bool> Q5;

      SC_CTOR (dff5)
      {
            SC_METHOD(decode);
            sensitive<<dclk5.pos();
            dont_initialize();
      }
```

```
        ~dff5(){}

        void decode(void)
        {
              Q5 = Q4.read();
              //Qn5 =!(Q.read()) ;
        }
  };
  SC_MODULE (dff6)  // dff6
  {
        sc_in<bool> Q5,dclk6;
        sc_out<bool> Q6;

        SC_CTOR (dff6)
        {
              SC_METHOD(decode);
              sensitive<<dclk6.pos();
              dont_initialize();
        }

        ~dff6(){}

        void decode(void)
        {
              Q6 = Q5.read();
              //Qn6 =!(Q.read()) ;
        }
  };
  SC_MODULE (dff7)  // dff 7
  {
        sc_in<bool> Q6,dclk7;
        sc_out<bool> Q7;

        SC_CTOR (dff7)
        {
              SC_METHOD(decode);
              sensitive<<dclk7.pos();
              dont_initialize();
        }

        ~dff7(){}

        void decode(void)
        {
              Q7 = Q6.read();
              //Qn7 =!(Q.read()) ;
        }
  };
  SC_MODULE (dff8)  // dff8
  {
        sc_in<bool> Q7,dclk8;
        sc_out<bool> Q8,Qn8;
```

```
      SC_CTOR (dff8)
      {
            SC_METHOD(decode);
            sensitive<<dclk8.pos();
            dont_initialize();
      }

      ~dff8(){}

      void decode(void)
      {
            Q8 = Q7.read();
            Qn8 =!(Q7.read()) ;
      }
};
```

//monitor.h

```
#include<iostream>
#include"systemc.h"

using namespace std;

SC_MODULE(monitor)
{
      sc_in<bool> D_m, dclk_m, Q_m, Qn_m;

      SC_CTOR(monitor)
      {
            SC_METHOD(monita);
            sensitive<<Q_m<<Qn_m;
            dont_initialize();
      }

      void monita(void){
      cout<<"at "<<sc_time_stamp()<<" D is: "<<D_m<<" Q8 is: "<<Q_m8<<" notQ
is: "<<Qn_m<<endl;
      }
};
```

//main.cc

```
#include"dff.h"
#include"driver.h"
#include"monitor.h"
#include"systemc.h"

int sc_main(int argc, char** argv)
{
      //signals
      sc_signal<bool> D_s,
Q_out,Q_out2,Q_out3,Q_out4,Q_out5,Q_out6,Q_out7,Q_out8, Qn_out8; //D_s signal
```

```
from driver, Q_out signal from dff1 output, Q_out2 signal from dff2 output etc.
      sc_clock dclk ("the_clock", 1.3,SC_MS ,0.7,0,SC_MS);
      //module instances
      dff df("dff_instance");
      dff2 df2("dff2_instance");
      dff3 df3("dff3_instance");
      dff4 df4("dff4_instance");
      dff5 df5("dff5_instance");
      dff6 df6("dff6_instance");
      dff7 df7("dff7_instance");
      dff8 df8("dff7_instance");
      driver dr("driver");
      monitor mn("monitor");
      //interconnections btn modules
      dr.a(D_s);
      df.D(D_s); //input of dff1
      mn.D_m(D_s);

        df.Q(Q_out); //Q-out is the output of dff1 and input of dff 2
        df2.Q(Q_out); //and input of dff 2
      df2.Q2(Q_out2); //Q2 here is output of dff2

      df3.Q2(Q_out2);
      df3.Q3(Q_out3); //Q2 here is output of dff3

      df4.Q3(Q_out3);
      df4.Q4(Q_out4); //Q2 here is output of dff4

      df5.Q4(Q_out4);
      df5.Q5(Q_out5); //Q2 here is output of dff5

      df6.Q5(Q_out5);
      df6.Q6(Q_out6); //Q2 here is output of dff6

      df7.Q6(Q_out6);
      df7.Q7(Q_out7); //Q2 here is output of dff7

      df8.Q7(Q_out7);
      df8.Q8(Q_out8); //Q2 here is output of dff8
      mn.Q_m(Q_out8);

      df8.Qn8(Qn_out8);
      mn.Qn_m(Qn_out8);

      df.dclk(dclk);
      df2.dclk2(dclk);
      df3.dclk3(dclk);
      df4.dclk4(dclk);
      df5.dclk5(dclk);
      df6.dclk6(dclk);
      df7.dclk7(dclk);
      df8.dclk8(dclk);
      mn.dclk_m(dclk);
```

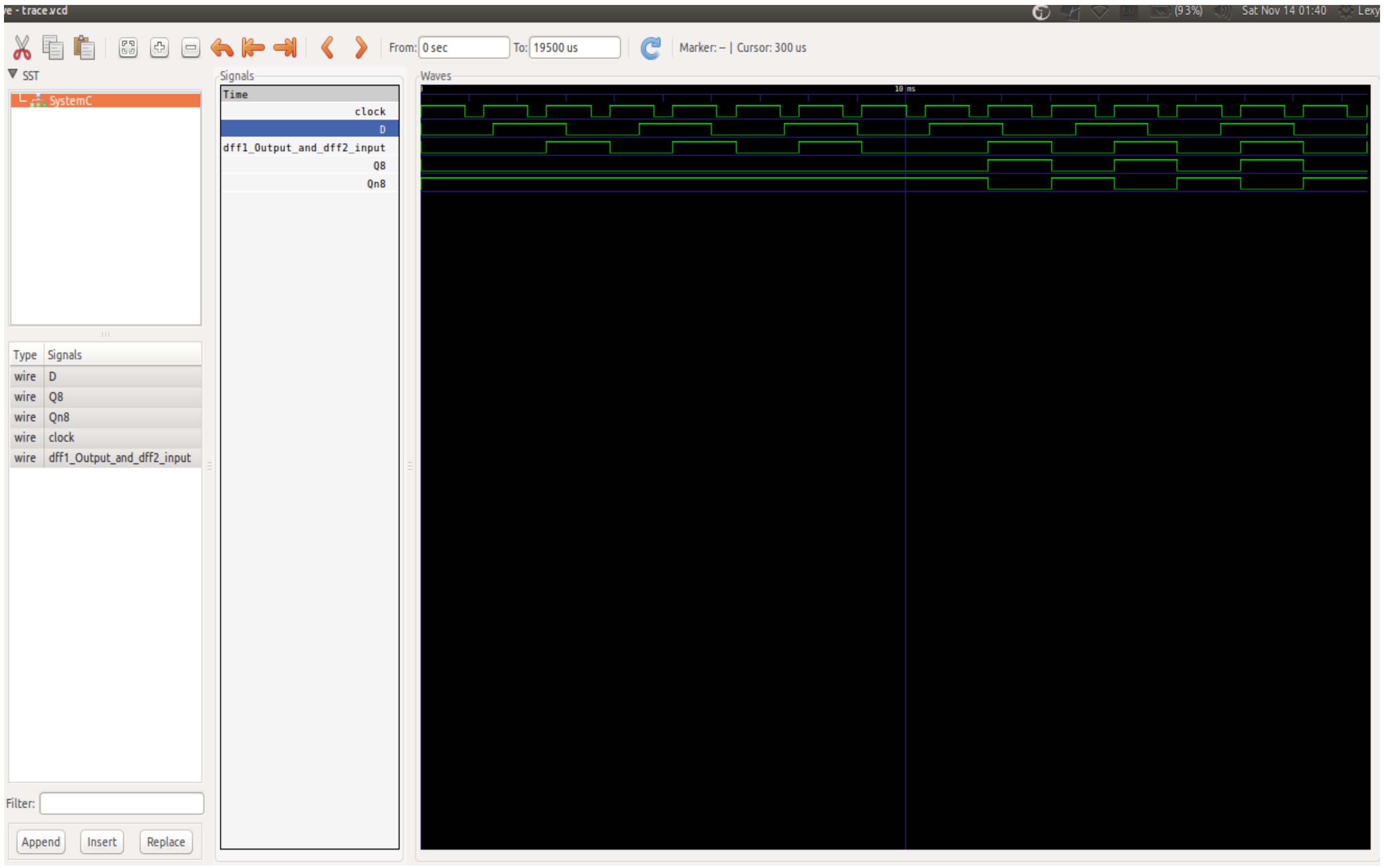
```
        //create a trace file with nanosecond resolution
        sc_trace_file *tf;
        tf = sc_create_vcd_trace_file("trace");
        tf->set_time_unit(1, SC_NS);
        //trace the signals interconnecting modules
        sc_trace(tf, dclk, "clock"); // signals to be traced
        sc_trace(tf, D_s, "D");
        sc_trace(tf, Q_out, "dff1_Output_and_dff2_input");
        sc_trace(tf, Q_out8, "Q8");
        sc_trace(tf, Qn_out8, "Qn8");
        //sc_time_unit(SC_NS);
        //run a simulation for 10 systemc milli-seconds
        sc_start(20,SC_MS);
        //close the trace file
        sc_close_vcd_trace_file(tf);
        return 0;

}
```

Results

The command line output:

```
Info: (I703) tracing timescale unit set: 1 ns (trace.vcd)
at 0 s D is: 0 Q8 is: 0 notQ8 is: 1
at 11700 us D is: 1 Q8 is: 1 notQ8 is: 0
at 13 ms D is: 0 Q8 is: 0 notQ8 is: 1
at 14300 us D is: 1 Q8 is: 1 notQ8 is: 0
at 15600 us D is: 0 Q8 is: 0 notQ8 is: 1
at 16900 us D is: 1 Q8 is: 1 notQ8 is: 0
at 18200 us D is: 0 Q8 is: 0 notQ8 is: 1
gtkwave trace.vcd
```

The trace file:



<u>Discussion</u>

From the final output of the register, it is clear that the first input appears here after 8 complete clock cycles and that the output only changes to the value of the input *only* when the clock moves from low to high.

<u>Conclusion</u>

We thus concluded that our objective was met as we were able to model and simulate the functioning of an 8 bit SISO register.

<u>References</u>

1.SystemC: From the Ground Up , 2010 ed
D.C. Black, J. Donovan, B. Bunton, A. Keist