SPH 336
COUMPUTING LAB II REPORT

T FLIP-FLOP REGISTER

GROUP MEMBERS

I39/2207/2013 OWOKO PHILIP ONYANGO
I39/2221/2013 KIPLAGAT HILLARY
I39/2215/2013 AKELLO LAZARUS OTIENO

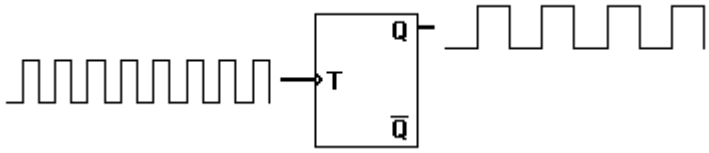
OBJECTIVE

The objective of this report was to design and to implement a Toggle (T) flip-flop register.

INTRODUCTION

A flip flop is a device having two states and a feedback path that allows it to store a bit of information. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a register. The n-bit register will consist of n number of flip-flop and it is capable of storing an n-bit word.

A T flip-flop is a device which swaps or "toggles" state every time it is triggered if the *T* input is asserted, otherwise it holds the current output.
T flip flop is useful for constructing binary counters, frequency dividers, and general binary addition devices. It is a single input device: T (trigger). Two outputs: Q and Q' (where Q' is the inverse of Q) as shown below;



Its characteristic table has 3 columns. The first column is the value of T, a control input. The second column is the current state, that is the current value being output by Q. The third column is the next state, that is, the value of Q at the next positive edge. It's labeled with Q and the superscript plus sign.
The T flip flop has two possible values. When T = 0, the flip flop does a hold. A hold means that the output, Q is kept the same as it was before the clock edge. When T = 1, the flip flop does a toggle, which means the output Q is negated after the clock edge, compared to the value before the clock edge.
This is illustrated in the following truth table

T Flip-flop

Symbol

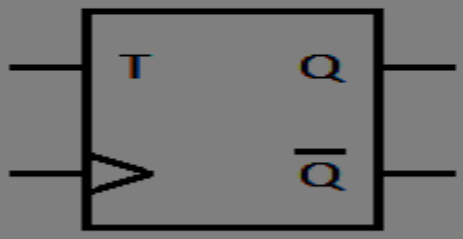


Table of truth:

T	Q	$\overline{Q}$
0	Q	$\overline{Q}$
1	$\overline{Q}$	Q
0	$\overline{Q}$	Q
1	Q	$\overline{Q}$

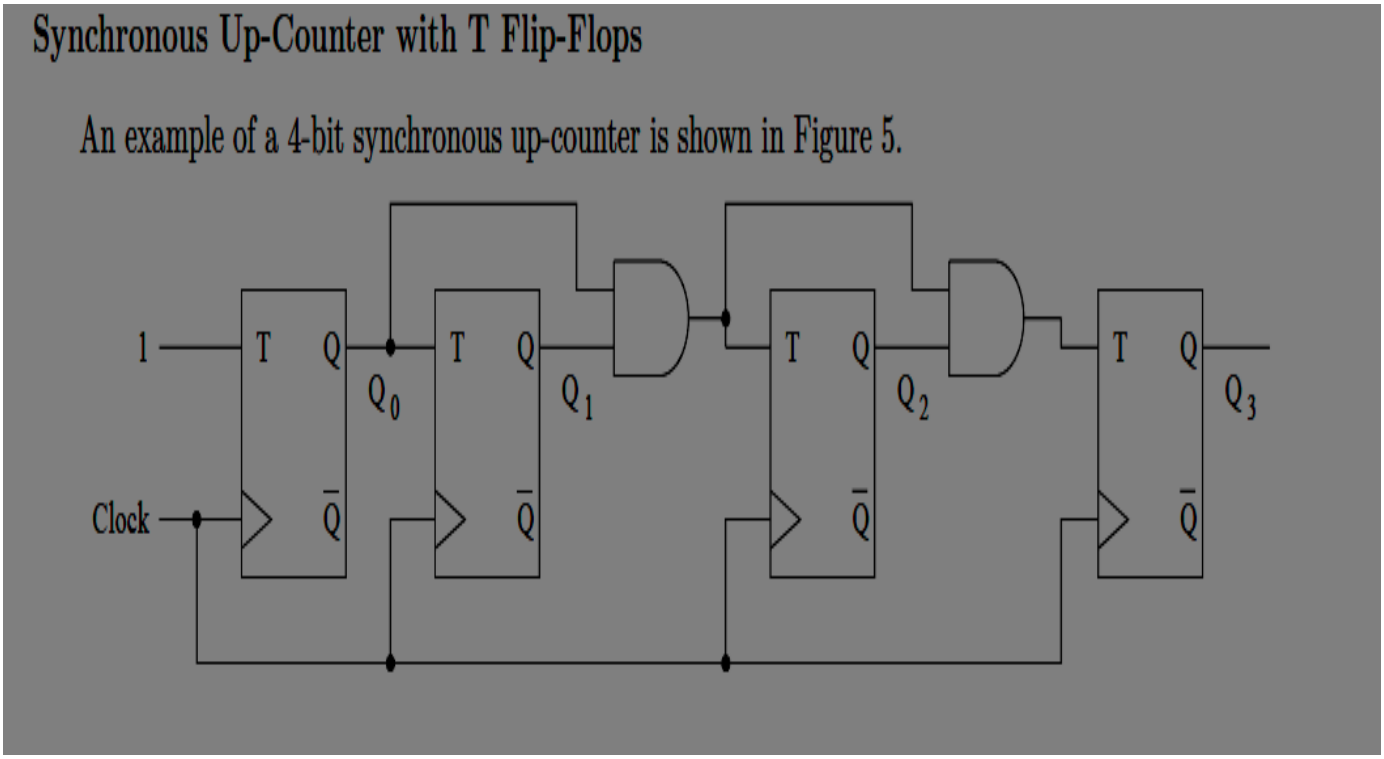
Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

The chosen design for the 4-bit counter is a simple 4-bit synchronous counter with synchronous set and reset option and input and output carry option. The following are some of the advantages of Synchronous Counters which can be made from Toggle or T-type flip-flops;

- Synchronous counters are easier to design than asynchronous counters.
- They are called synchronous counters because the clock input of the flip-flops are all clocked together at the same time with the same clock signal.
- Due to this common clock pulse all output states switch or change simultaneously.

- With all clock inputs wired together there is no inherent propagation delay.
- Synchronous counters are sometimes called parallel counters as the clock is fed in parallel to all flip-flops.
- The inherent memory circuit keeps track of the counters present state.
- The count sequence is controlled using logic gates.
- Overall faster operation may be achieved compared to Asynchronous counters.

They are called synchronous counters because the clock input of the flip-flops are all clocked together at the same time with the same clock signal.



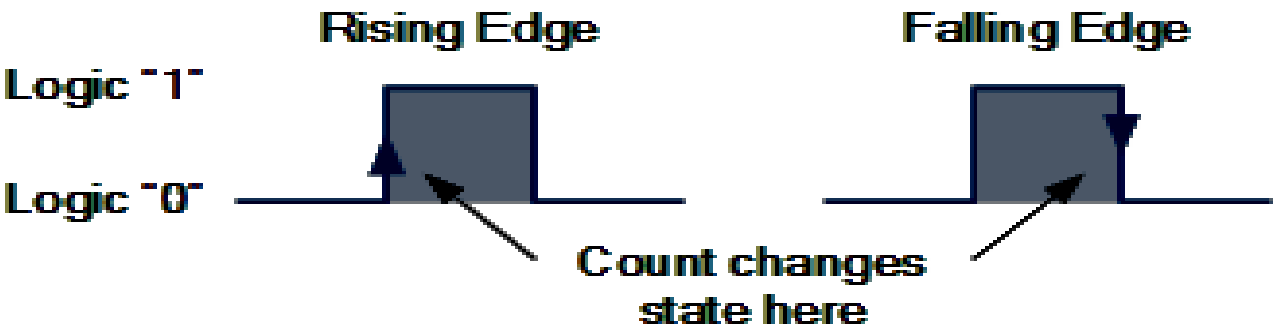
Synchronous T-FF Schematic diagram

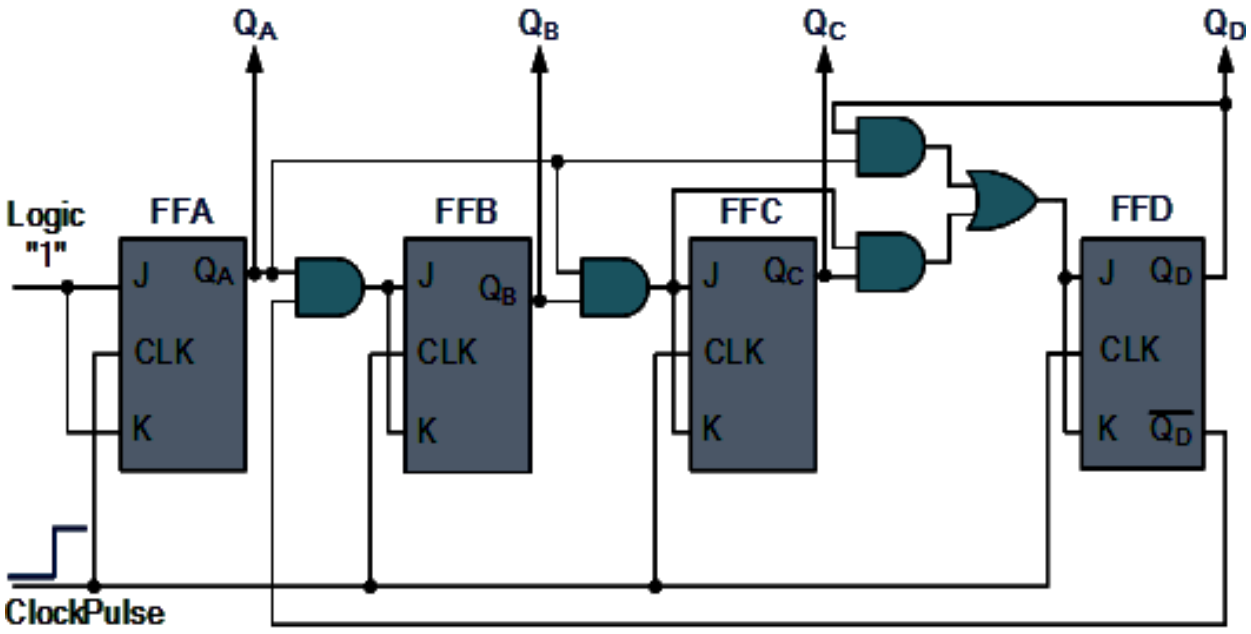
METHODOLOGY

Triggering A Synchronous Counter

Synchronous Counters use edge-triggered flip-flops that change states on either the “positive-edge” (rising edge) or the “negative-edge” (falling edge) of the clock pulse on the control input resulting in one single count when the clock input changes state.

Generally, synchronous counters count on the rising-edge which is the low to high transition of the clock signal and asynchronous ripple counters count on the falling-edge which is the high to low transition of the clock signal.





The additional AND gates detect when the counting sequence reaches “1001”, (Binary 10) and causes flip-flop FF3 to toggle on the next clock pulse. Flip-flop FF0 toggles on every clock pulse. Thus, the count is reset and starts over again at “0000” producing a synchronous decade counter.

The implementation of theT-FF was done using four critical modules namely, *tff_module tff_main, tff_driver,* and *tff_monitor.*

The four modules were implemented in SystemC and inter-included with the corresponding modules which were then fetched into the main function for the ultimate implementation.

The modules that were used to implement the systemC code were as follows;-

Driver Module

The input to register is produced from here ;-it drives input to the data ,clock, reset port of the T-flipflop.

tff_driver.h

```
#ifndef TFF_DRIVER_H_
#define TFF_DRIVER_H_

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

SC_MODULE(tff_driver){

        sc_out <bool> data, clk, reset;

        SC_CTOR(tff_driver){
                SC_THREAD(driveData);
                SC_THREAD(driveClock);
                SC_THREAD(driveReset);
        }

        /**
         * Drives data to input port
         */
        void driveData(){
      while(1){
        data.write(0);
                        wait(5, SC_NS);
                        data.write(1);
                        wait(5,SC_NS);
      }
        }
        /**
         * Drives data to the clock port
         */
        void driveClock(){
```

```
		while(1){
			clk.write(0);
			wait(7,SC_NS);
			clk.write(1);
			wait(7,SC_NS);
		}
	}
	/**
	 * Drives data to the Reset port
	 */
	void driveReset(){
		while(1){
			reset.write(0);
			wait(4,SC_NS);
			reset.write(1);
			wait(4, SC_NS);
		}

	}
};


#endif /* TFF_DRIVER_H_ */
```

The main function module

The main function instantiates the tff_driver, tff_monitor, tff_module modules, creates the trace file, and initializes simulation.

tff_main.cc

```
#include "systemc.h"
#include <systemc>
#include "tff_module.h"
#include "tff_driver.h"
#include "tff_monitor.h"

/**
 * Instantiate the T flipflop
 */
int sc_main (int argv, char * argc[]){
	sc_signal<bool> data, clk, reset, output;

	//Start module instances
	tff_driver tff_d("driver instance");
	tff_module tff_mod("tff_module instance");
	tff_monitor tff_mon("tff_monitor instance");
```

```
	tff_d.data(data);
	tff_d.clk(clk);
	tff_d.reset(reset);
	tff_mon.data(data);
	tff_mod.data(data);

	tff_mon.clock(clk);
	tff_mod.clk(clk);

	tff_mon.reset(reset);
	tff_mod.reset(reset);

	tff_mon.output(output);
	tff_mod.q(output);

	/**
	 * Creating a trace file
	 * Trace the signals, and name the various signals
	 */

	sc_trace_file *tf;
	tf = sc_create_vcd_trace_file("timingDiagram"); //sets the filename .vcd file
	tf ->set_time_unit(1, SC_NS);

	sc_trace(tf, data, "data_input");
	sc_trace(tf, clk, "clock_signal");
	sc_trace(tf, reset, "reset_switch");
	sc_trace(tf, output, "Output");

	if (!sc_pending_activity())
		sc_start(75, SC_NS);

	sc_close_vcd_trace_file(tf);

	return 0;
}
```

The module module

This module consists of the implementation of the register using the flip flop. This module does the actual flipflop workings, by evaluating the input values hence determining the outputs.

tff_module.h

```
#ifndef TFF_MODULE_H_
#define TFF_MODULE_H_


#include "systemc.h"
#include <systemc>
SC_MODULE(tff_module)
	{
	sc_in<bool> data, clk, reset;
	sc_out<bool> q;

	bool q_l;

	void tff () {
		if (reset.read()) {
			q_l = 0;
		} else if (~data.read()) {
			q_l =  ! q;
		}else {}
		q.write(q_l);
	}

	SC_CTOR(tff_module) {
		q_l = 0;
		SC_METHOD (tff);
		sensitive << clk.pos();
	}
};

#endif /* TFF_MODULE_H_ */
```

The monitor module

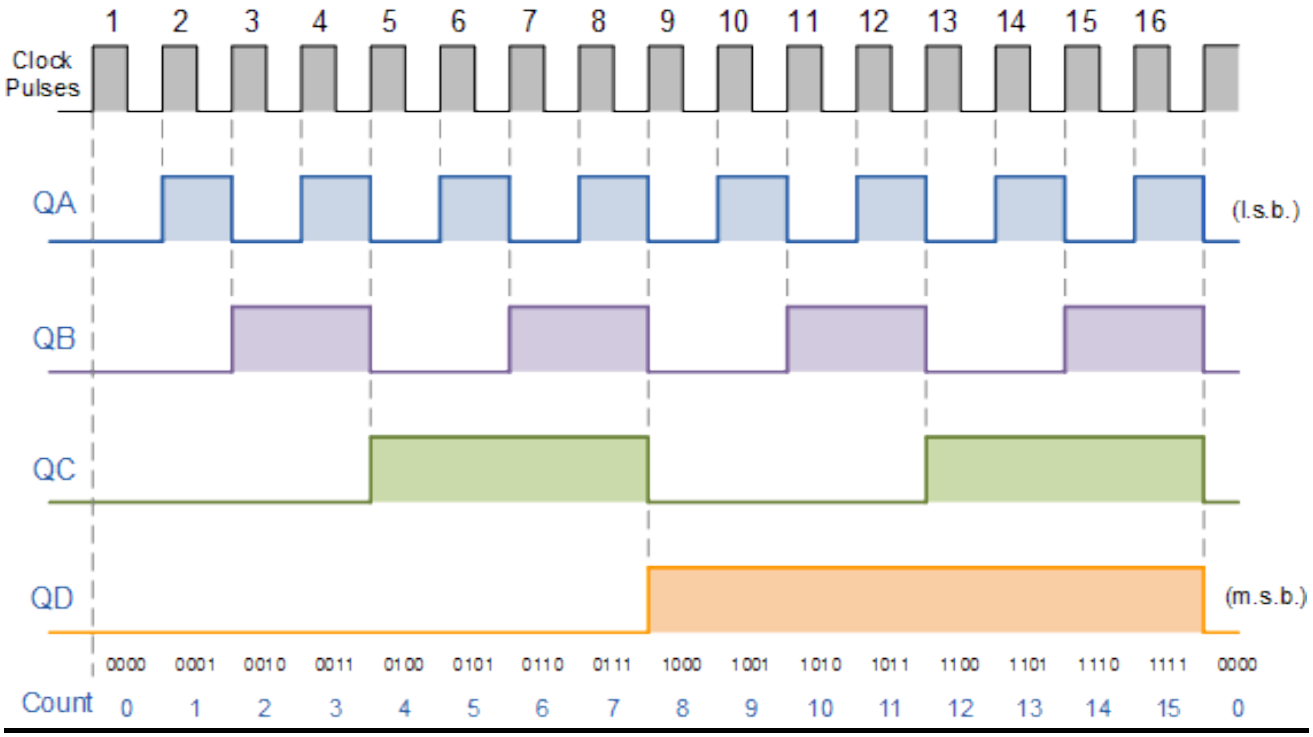
This was used to monitor the timing diagrams of the input to the register, the clock signal and the output of the register to determine if the simulation was working properly.

tff_monitor.h

```
#ifndef  TFF_MONITOR_H_
#define TFF_MONITOR_H_
```

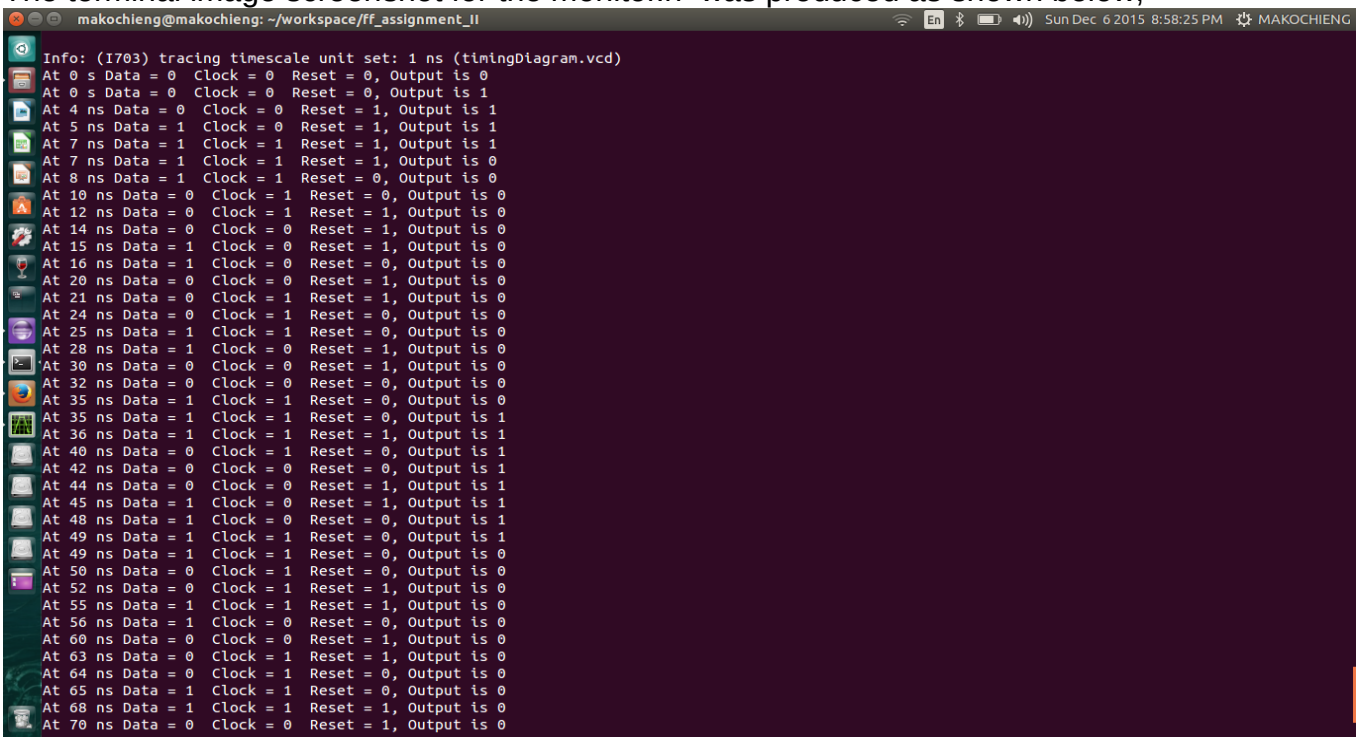
```
#include "systemc.h"
#include <systemc>

SC_MODULE(tff_monitor){

	sc_in<bool> data, clock, output, reset;

	SC_CTOR(tff_monitor){
		SC_METHOD(trackTff);
		sensitive<<data<<clock<<output<<reset;
	}
	/**
	 * Displays values of the input , clock, reset and the output to the screen
	 */
	void trackTff(){
		cout<<"At "<<sc_time_stamp()<<" Data = "<<data<<"  Clock = "<<clock<<"
Reset = "<<reset<<", Output is "<<output<<endl;

	}

};

#endif /* TFF_MONITOR_H_ */
```

RESULTS

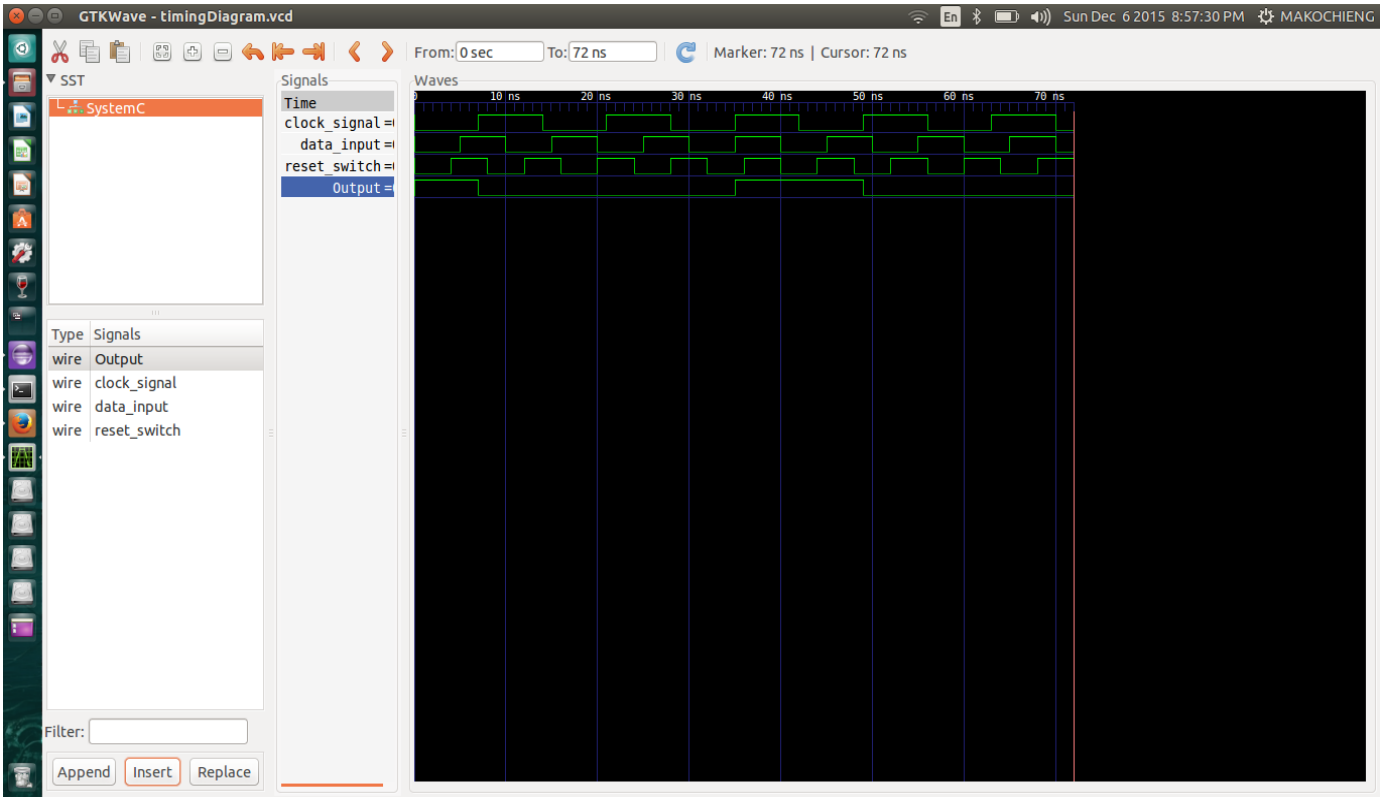


Because this 4-bit synchronous counter counts sequentially on every clock pulse the resulting outputs count upwards from 0 (0000) to 15 (1111).

The terminal image screenshot for the monitor.h was produced as shown below;



The output was also shown from the gtkWave and a waveform of the form as shown below was then produced. Below is the vcd timing diagram of the wave;



DISCUSSIONS

The implementation of the t-ff was done in systemC using three modules, namely driver, monitor and main. This was done with the help of eclipse-cdt plugin in eclipse that was used for the task.
From our results and the research we did on the schematic diagram and the truth table, we realized that T flip flops toggles its outputs on a rising edge and otherwise keeps its present state. The result of synchronization is that all the individual output bits changing state at exactly the same time in response to the common clock signal with no ripple effect and therefore, no propagation delay.

CONCLUSION

The t flip-flop designed above consisted of the following files: *tff_module.h, tff_main.cc, tff_driver.h,* and *monitor.h.* The codes were compiled and tested in systemC and were found to run successfully. Therefore the objective of the experiment were accomplished as the 4-bit register T-FF was implemented in such a way that the T flip flops toggled its outputs on a rising edge and otherwise kept its present state.

REFERENCES

- Structured Computer Organization (Andrew Tunenbaum)
- https://github.com/Muriukidavid/cplusplus_examples
- Fundamentals of Logic Design*, 6th Edition*, by Roth and Kinney
- https://en.wikipedia.org/wiki/Special:Search?search=4+bit+synchronous+counter+d+latch&sourceid=Mozilla-search