

Homework 6B – SystemC AMS Modeling - TDF

Taghizadeh,
810198373

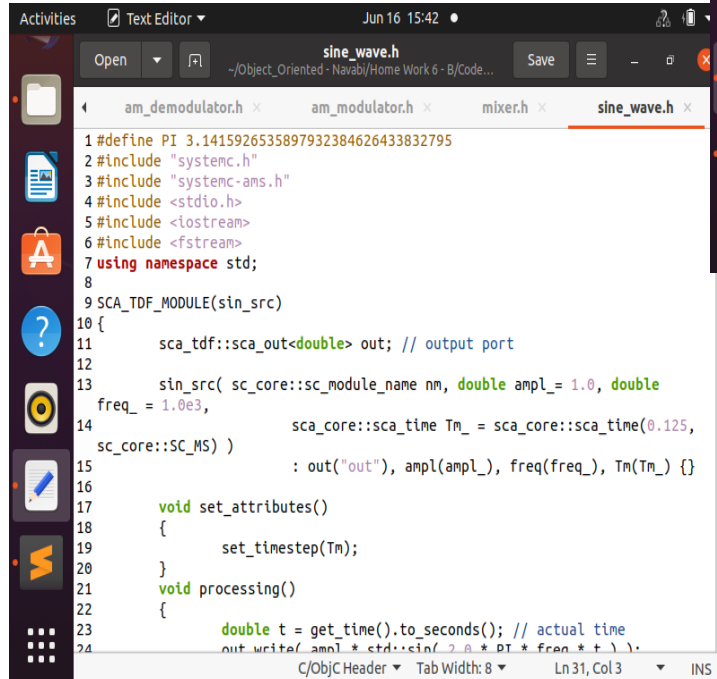
I. INTRODUCTION

In this homework, we are about to make an AM Modulation using *SystemC-AMS* TDF and ELN elements.

II. BAND PASS FILTER IMPLEMENTATION

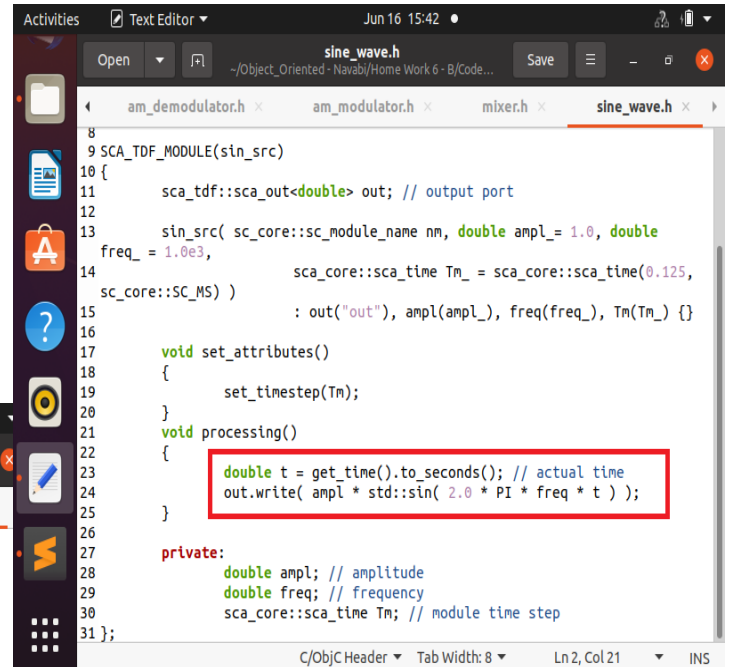
A. Message and Carrier Sine Wave in SystemC-AMS

To produce the *Message* and *Carrier* sine wave, we simply measure $\sin(2\pi ft)$ which f is the frequency of the signal and t is the current time.



```
1 #define PI 3.1415926535897932384626433832795
2 #include "systemc.h"
3 #include "systemc-ams.h"
4 #include <stdio.h>
5 #include <iostream>
6 #include <fstream>
7 using namespace std;
8
9 SCA_TDF_MODULE(sin_src)
10 {
11     sca_tdf::sca_out<double> out; // output port
12
13     sin_src( sc_core::sc_module_name nm, double ampl_ = 1.0, double
14             freq_ = 1.0e3,
15             sca_core::sca_time Tm_ = sca_core::sca_time(0.125,
16             sc_core::SC_MS )
17             : out("out"), ampl(ampl_), freq(freq_), Tm(Tm_) {}
18
19     void set_attributes()
20     {
21         set_timestep(Tm);
22     }
23     void processing()
24     {
25         double t = get_time().to_seconds(); // actual time
26         out.write( ampl * std::sin( 2.0 * PI * freq * t ) );
27     }
28
29 private:
30     double ampl; // amplitude
31     double freq; // frequency
32     sca_core::sca_time Tm; // module time step
33 }
```

Fig. 1 Sine Wave Generator SystemC-AMS



```
8
9 SCA_TDF_MODULE(sin_src)
10 {
11     sca_tdf::sca_out<double> out; // output port
12
13     sin_src( sc_core::sc_module_name nm, double ampl_ = 1.0, double
14             freq_ = 1.0e3,
15             sca_core::sca_time Tm_ = sca_core::sca_time(0.125,
16             sc_core::SC_MS )
17             : out("out"), ampl(ampl_), freq(freq_), Tm(Tm_) {}
18
19     void set_attributes()
20     {
21         set_timestep(Tm);
22     }
23     void processing()
24     {
25         double t = get_time().to_seconds(); // actual time
26         out.write( ampl * std::sin( 2.0 * PI * freq * t ) );
27     }
28
29 private:
30     double ampl; // amplitude
31     double freq; // frequency
32     sca_core::sca_time Tm; // module time step
33 }
```

Fig. 2 Sine Wave Generator SystemC-AMS

B. Mixer in SystemC-AMS

First in the method “*set_attributes*” the sampling rate of the *Message* and the *Carrier* signals are defined. Due to the Nyquist theorem the sampling frequency must be at least 2 times greater than the frequency of the original signal, so the rate of the *Message* signal is set to 20KHz (2*10KHz) and the rate of the *Carrier* signal is set to 2MHz (2*1MHz).



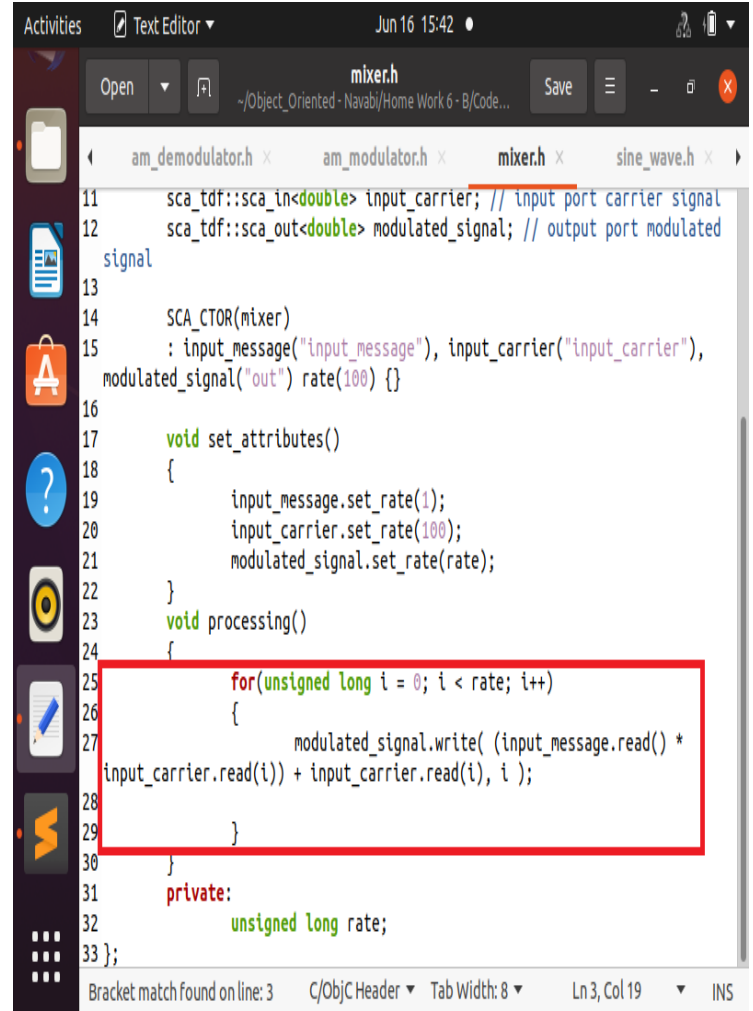
```

1 #include "systemc.h"
2 #include "systemc-ams.h"
3 #include <stdio.h>
4 #include <iostream>
5 #include <fstream>
6 using namespace std;
7
8 SCA_TDF_MODULE(mixer)
9 {
10     sca_tdf::sca_in<double> input_message; // input port message signal
11     sca_tdf::sca_in<double> input_carrier; // input port carrier signal
12     sca_tdf::sca_out<double> modulated_signal; // output port modulated
13     signal
14
15     SCA_CTOR(mixer)
16     : input_message("input_message"), input_carrier("input_carrier"),
17       modulated_signal("out") rate(100) {}
18
19     void set_attributes()
20     {
21         input_message.set_rate(1);
22         input_carrier.set_rate(100);
23         modulated_signal.set_rate(rate);
24     }
25
26     void processing()
27     {
28
29     }
30 }
31
32 private:
33     unsigned long rate;

```

Fig. 3 Mixer SystemC-AMS

Then in a loop, $C(t) + m(t)*C(t)$ is calculated for each sample and written in the output (*modulated_signal*). As the *Carrier* frequency is 100 times greater than the *Message* frequency, in this loop we use the same *Message* sample for all *Carrier* samples, because whenever 100 samples of the *Carrier* is received, just 1 sample of the *Message* is collected so this sample is used for all *Carrier* samples.



```

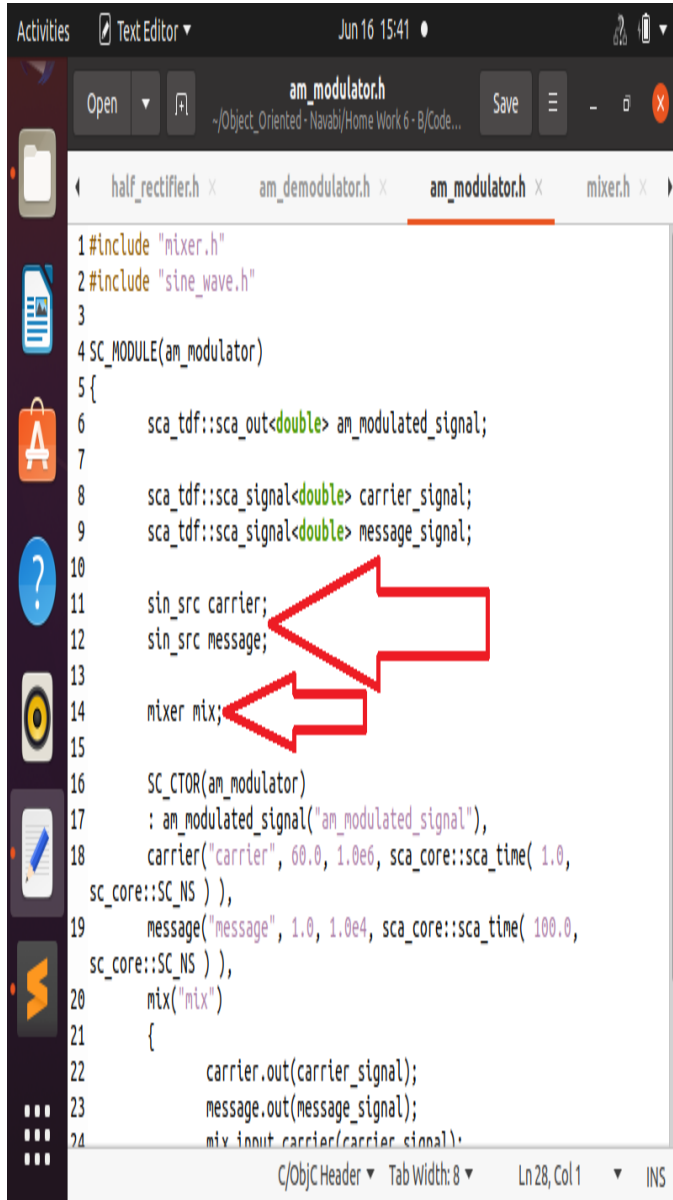
11     sca_tdf::sca_in<double> input_carrier; // input port carrier signal
12     sca_tdf::sca_out<double> modulated_signal; // output port modulated
13     signal
14
15     SCA_CTOR(mixer)
16     : input_message("input_message"), input_carrier("input_carrier"),
17       modulated_signal("out") rate(100) {}
18
19     void set_attributes()
20     {
21         input_message.set_rate(1);
22         input_carrier.set_rate(100);
23         modulated_signal.set_rate(rate);
24     }
25
26     void processing()
27     {
28         for(unsigned long i = 0; i < rate; i++)
29         {
30             modulated_signal.write( (input_message.read() *
31                                     input_carrier.read(i) + input_carrier.read(i), i );
32         }
33     }
34
35     private:
36         unsigned long rate;

```

Fig. 4 Mixer SystemC-AMS

C. AM Modulator in SystemC-AMS

First two sine wave sources are made to produce the 10KHz *Message* and the 1MHz *Carrier* signals. Then a *Mixer* is made to mix the *Message* and *Carrier* signals, so the outputs of the sine sources are connected to the inputs of this module. The output of the *Mixer* is the AM modulated signal of the *Message*.



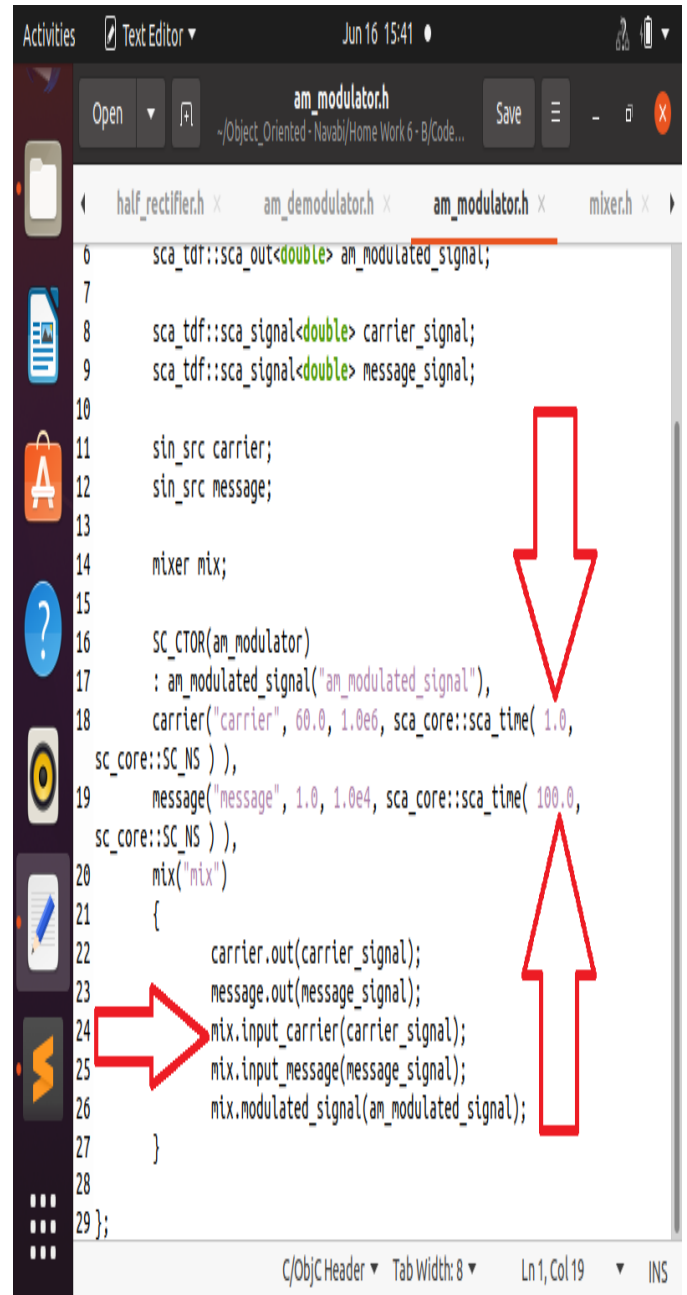
```

1#include "mixer.h"
2#include "sine_wave.h"
3
4SC_MODULE(am_modulator)
5{
6    sca_tdf::sca_out<double> am_modulated_signal;
7
8    sca_tdf::sca_signal<double> carrier_signal;
9    sca_tdf::sca_signal<double> message_signal;
10
11    sin_src carrier;
12    sin_src message;
13
14    mixer mix;
15
16    SC_CTOR(am_modulator)
17    : am_modulated_signal("am_modulated_signal"),
18      carrier("carrier", 60.0, 1.0e6, sca_core::sca_time( 1.0,
19        sc_core::SC_NS ) ),
20      message("message", 1.0, 1.0e4, sca_core::sca_time( 100.0,
21        sc_core::SC_NS ) ),
22      mix("mix")
23    {
24        carrier.out(carrier_signal);
25        message.out(message_signal);
26        mix.inout_carrier(carrier_signal);
27        mix.inout_message(message_signal);
28        mix.modulated_signal(am_modulated_signal);
29    }

```

Fig. 5 AM Modulator SystemC-AMS

Carrier is much faster than the *Message* sine wave so the time step of the *Carrier* must be much less than the *Message* signal, because of that the time step of the *Carrier* and *Message* signals are set to 1 and 100 nano seconds respectively.



```

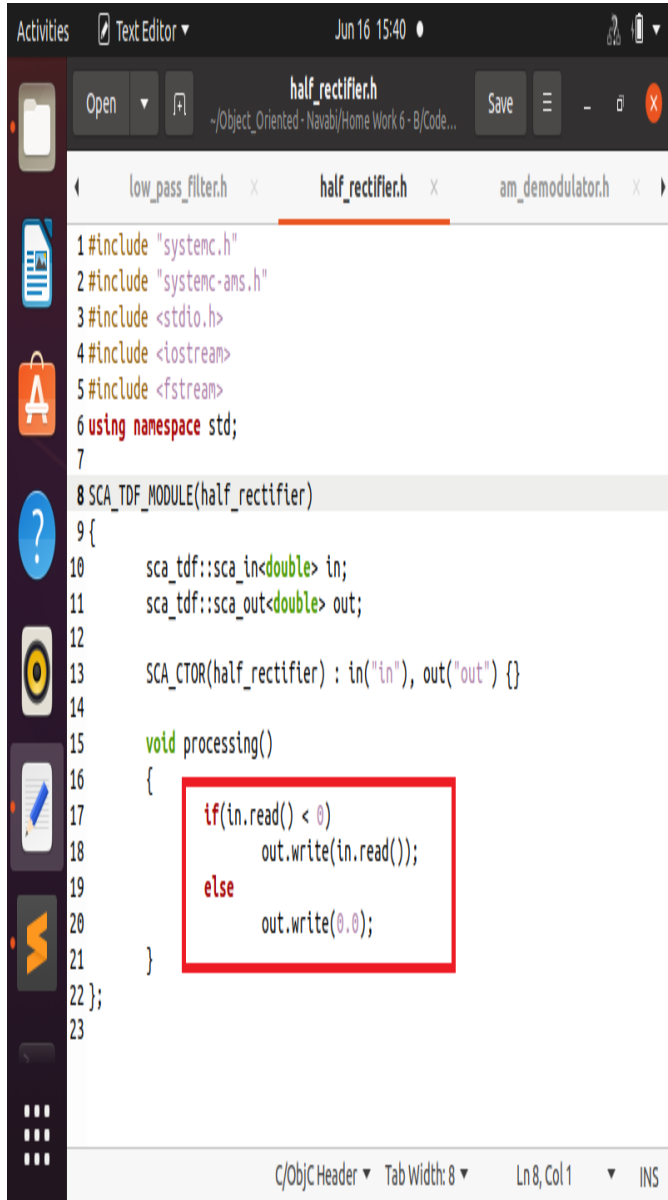
6    sca_tdf::sca_out<double> am_modulated_signal;
7
8    sca_tdf::sca_signal<double> carrier_signal;
9    sca_tdf::sca_signal<double> message_signal;
10
11    sin_src carrier;
12    sin_src message;
13
14    mixer mix;
15
16    SC_CTOR(am_modulator)
17    : am_modulated_signal("am_modulated_signal"),
18      carrier("carrier", 60.0, 1.0e6, sca_core::sca_time( 1.0,
19        sc_core::SC_NS ) ),
20      message("message", 1.0, 1.0e4, sca_core::sca_time( 100.0,
21        sc_core::SC_NS ) ),
22      mix("mix")
23    {
24        carrier.out(carrier_signal);
25        message.out(message_signal);
26        mix.input_carrier(carrier_signal);
27        mix.input_message(message_signal);
28        mix.modulated_signal(am_modulated_signal);
29    }

```

Fig. 6 AM Modulator SystemC-AMS

D. Half-Wave Rectifier in SystemC-AMS

To half rectify the input signal, we simply check the input. If the input is negative (less than 0) the output will be equal to the input, else the output will be zero



```

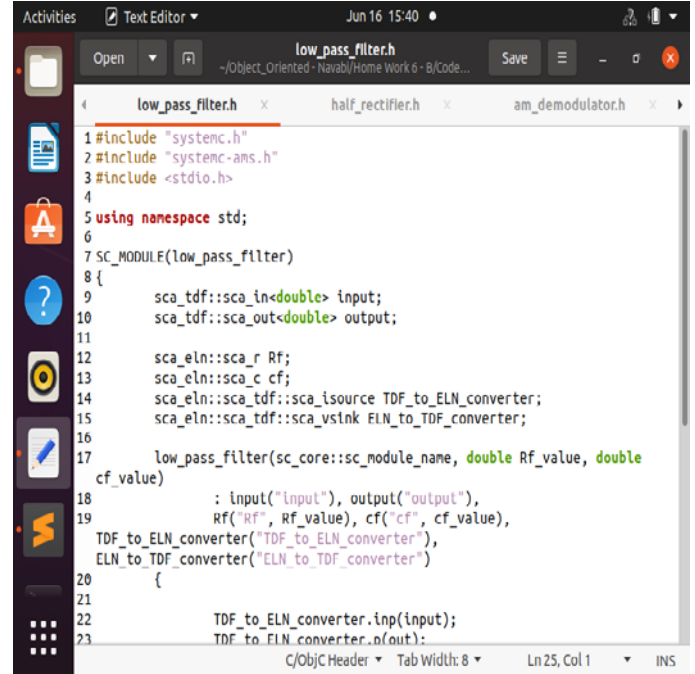
1#include "systemc.h"
2#include "systemc-ams.h"
3#include <stdio.h>
4#include <iostream>
5#include <fstream>
6using namespace std;
7
8SCA_TDF_MODULE(half_rectifier)
9{
10    sca_tdf::sca_in<double> in;
11    sca_tdf::sca_out<double> out;
12
13    SCA_CTOR(half_rectifier) : in("in"), out("out") {}
14
15    void processing()
16    {
17        if(in.read() < 0)
18            out.write(in.read());
19        else
20            out.write(0.0);
21    }
22};
23

```

Fig. 7 Half-Wave Rectifier SystemC-AMS

E. RC Low-Pass Filter in SystemC-AMS

To filter the modulated signal and recover the message, a *RC Low-Pass Filter* is needed. To design this *RC Low-Pass Filter*, a resistor (Rf) and a capacitor (Cf) are defined. To convert the tdf input signal to an eln signal and convert the output eln signal to a tdf signal, a tdf to eln converter and an eln to tdf converter are used.

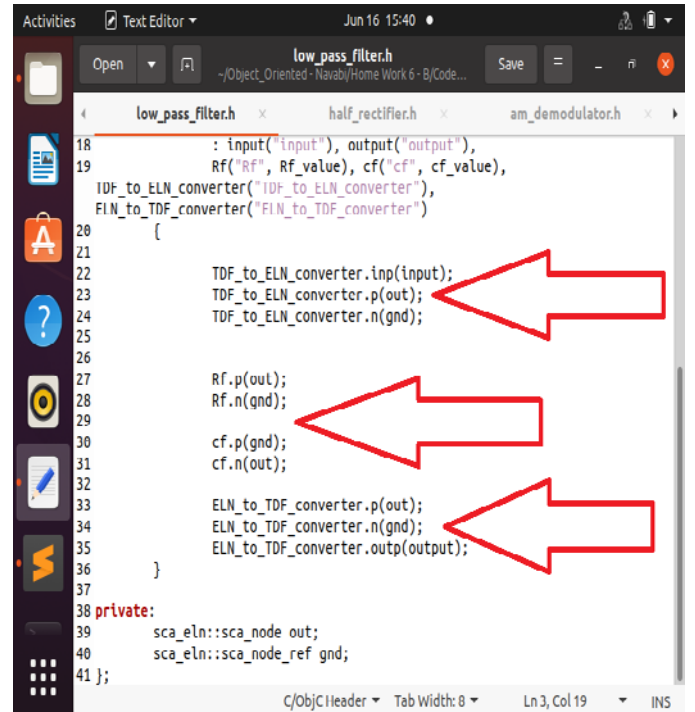


```

1#include "systemc.h"
2#include "systemc-ams.h"
3#include <stdio.h>
4
5using namespace std;
6
7SCA_MODULE(low_pass_filter)
8{
9    sca_tdf::sca_in<double> input;
10    sca_tdf::sca_out<double> output;
11
12    sca_eln::sca_r Rf;
13    sca_eln::sca_c Cf;
14    sca_eln::sca_tdf::sca_isource TDF_to_ELN_converter;
15    sca_eln::sca_tdf::sca_vsink ELN_to_TDF_converter;
16
17    low_pass_filter(sc_core::sc_module_name, double Rf_value, double
18        cf_value)
19        : input("input"), output("output"),
20          Rf("Rf", Rf_value, Cf("Cf", Cf_value),
21            TDF_to_ELN_converter("TDF_to_ELN_converter"),
22            ELN_to_TDF_converter("ELN_to_TDF_converter"))
23    {
24        TDF_to_ELN_converter.inp(input);
25        TDF_to_ELN_converter.out(out);
26    }
27
28    void processing()
29    {
30        Rf.p(out);
31        Rf.n(gnd);
32        Cf.p(gnd);
33        Cf.n(out);
34        ELN_to_TDF_converter.p(out);
35        ELN_to_TDF_converter.n(gnd);
36        ELN_to_TDF_converter.outp(output);
37    }
38private:
39    sca_eln::sca_node out;
40    sca_eln::sca_node_ref gnd;
41};

```

Fig. 8 RC Low-Pass Filter SystemC-AMS



```

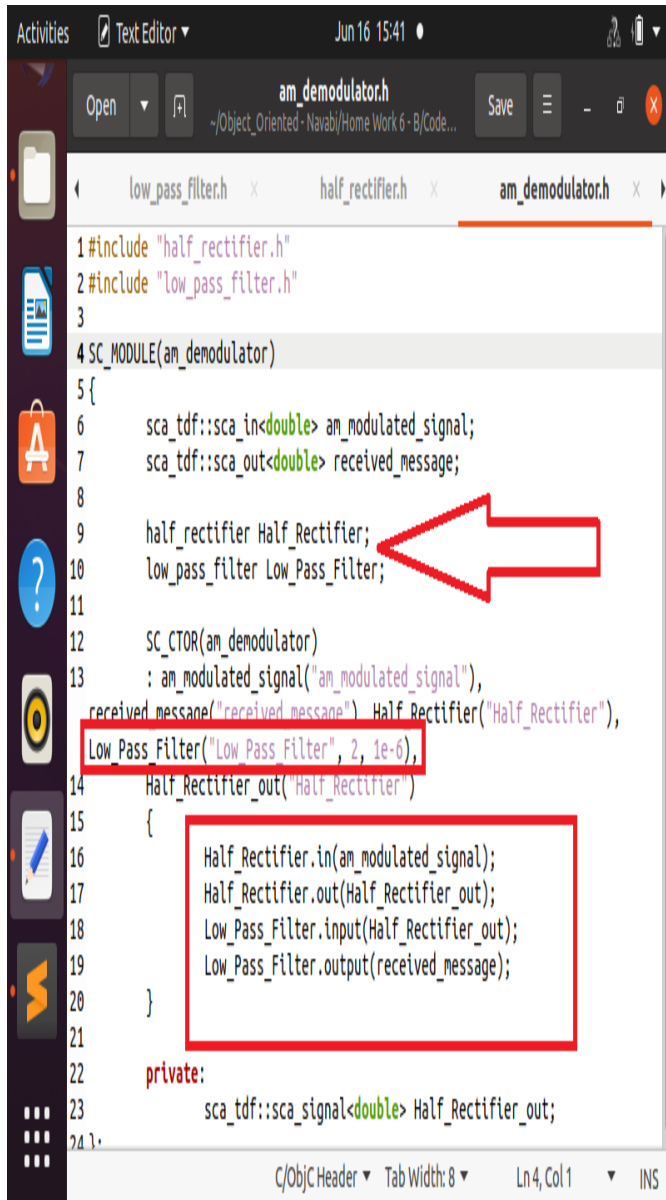
18    : input("input"), output("output"),
19      Rf("Rf", Rf_value, Cf("Cf", Cf_value),
20        TDF_to_ELN_converter("TDF_to_ELN_converter"),
21        ELN_to_TDF_converter("ELN_to_TDF_converter"))
22    {
23        TDF_to_ELN_converter.inp(input);
24        TDF_to_ELN_converter.out(out);
25
26        Rf.p(out);
27        Rf.n(gnd);
28
29        Cf.p(gnd);
30        Cf.n(out);
31
32        ELN_to_TDF_converter.p(out);
33        ELN_to_TDF_converter.n(gnd);
34        ELN_to_TDF_converter.outp(output);
35    }
36private:
37    sca_eln::sca_node out;
38    sca_eln::sca_node_ref gnd;
39};

```

Fig. 9 RC Low-Pass Filter SystemC-AMS

F. AM Demodulator in SystemC-AMS

Simply an instance of *Half-Wave Rectifier* and *RC Low-Pass Filter* are taken. The output of the *Half-Wave Rectifier* (*Half_Rectifier_out*) is connected to the input of the *RC Low-Pass Filter*. The output of the *RC Low-Pass Filter* is the recovered *Message* signal. As the cutoff frequency of an RC Low-Pass Filter is equal to $1/2\pi RC$, and the filter must pass frequencies between *Message* and *Carrier* frequencies, so we choose $R = 2$ and $C = 1\mu$ to have a 2MHz cutoff frequency, to pass the frequencies between *Message* and *Carrier* frequencies through the filter.



```

1#include "half_rectifier.h"
2#include "low_pass_filter.h"
3
4SC_MODULE(am_demodulator)
5{
6    sca_tdf::sca_in<double> am_modulated_signal;
7    sca_tdf::sca_out<double> received_message;
8
9    half_rectifier Half_Rectifier;
10   low_pass_filter Low_Pass_Filter;
11
12   SC_CTOR(am_demodulator)
13   : am_modulated_signal("am_modulated_signal"),
14     received_message("received_message") Half_Rectifier("Half_Rectifier"),
15     Low_Pass_Filter("Low_Pass_Filter", 2, 1e-6),
16     Half_Rectifier_out("Half_Rectifier_out")
17   {
18       Half_Rectifier.in(am_modulated_signal);
19       Half_Rectifier.out(Half_Rectifier_out);
20       Low_Pass_Filter.input(Half_Rectifier_out);
21       Low_Pass_Filter.output(received_message);
22   }
23
24private:
25    sca_tdf::sca_signal<double> Half_Rectifier_out;
26}

```

Fig. 10 AM Demodulator SystemC-AMS

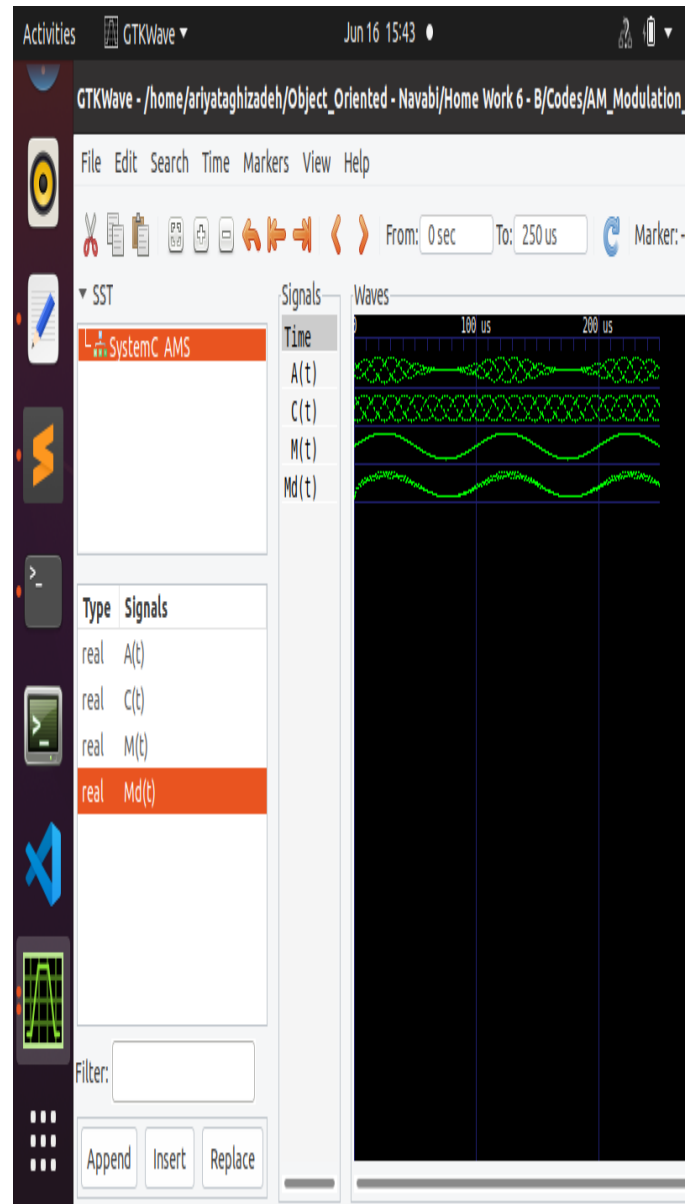


Fig. 11 AM Modulation Test Bench Results