POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

INTEGRATED SYSTEMS ARCHITECTURE

Laboratory 3

Bernunzo Angela 198721
Busignani Fabio 197883
Gianoglio Emanuele 200090

February 9, 2014

CONTENTS

INTRODUCTION

The aim of this third laboratory is to design with *SystemC* a *4-state add-compare-select* (acs) unit and simulate it in different ways to achieve a significant test-bench.
This project is divided into four different steps, three of them are simulations:

1. Description of our architecture in SystemC (Sec.1).
2. First simulation using *G++* and *GTKwave* (Sec.2).
3. Second simulation using *Modelsim* (Sec.3).
4. Third simulation using *Modelsim mixed SystemC-verilog languages* (Sec.4).

In order to achieve this project we used the following softwares:

- **G++ Compiler** to compile the source code of Viterbi decoder;
- **GTKwave** to view the waveforms generated by source code for behavioral simulation;
- **Modelsim** to compile and simulate SystemC and verilog source codes of the architecture;
- **Matlab** and **Code::Blocks** to verify results obtained by first simulation.

In the following sections the steps that we have followed are shown.

1 HARDWARE DESCRIPTION

1.1 A BRIEF INTRODUCTION TO VITERBI ALGORITHM

The Viterbi algorithm is widely used to check sequential error-control codes, and to detect symbol in channels with memory.
It is useful to find the most likely noiseless sequence given a noise corrupted finite-state sequence.
To realize this behavior, this algorithm is described by a Mealy finite state machine, in which for every input an output signal is associated (Fig.1).

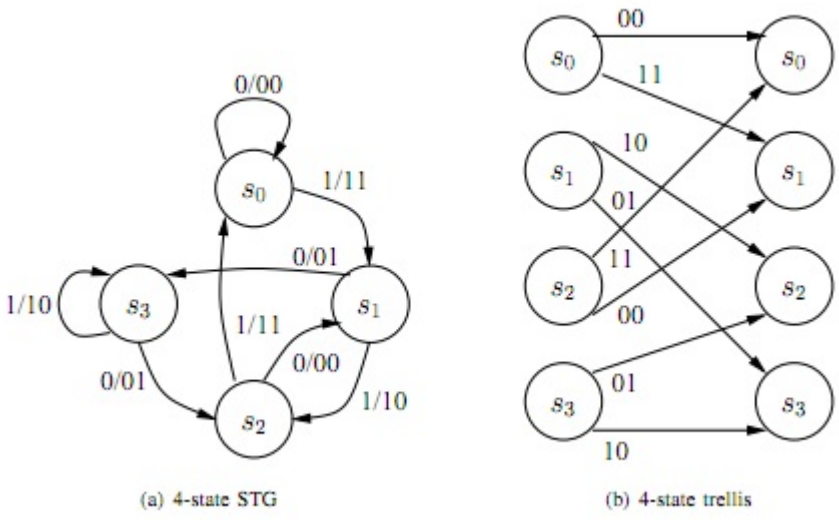

Figure 1.: 4-state STG and trellis

The state transition diagram (*STG*) in (Fig.1) can also be described using a time indexed equivalent *trellis diagram*, which describes all the state transition.
If a sequence of symbols, gnerated from a trellis, is corrupted by noise, the Viterbi algorithm can find the closest sequence of symbols in the given trellis.
Anyway, in this laboratory experience, we had to design an operation in trellis representation: the **Add-Compare-Select** function (*ACS*). This function is used to compute the new state metrics of the trellis.

1.2 DESCRIPTION OF SYSTEMC SOURCES CODE

For the realization of *SystemC* sources code we followed the given document to understand the design steps.

1.2.1 Study of the architecture

First of all we studied the architecture of *ACS4*, shown in (Fig.2).

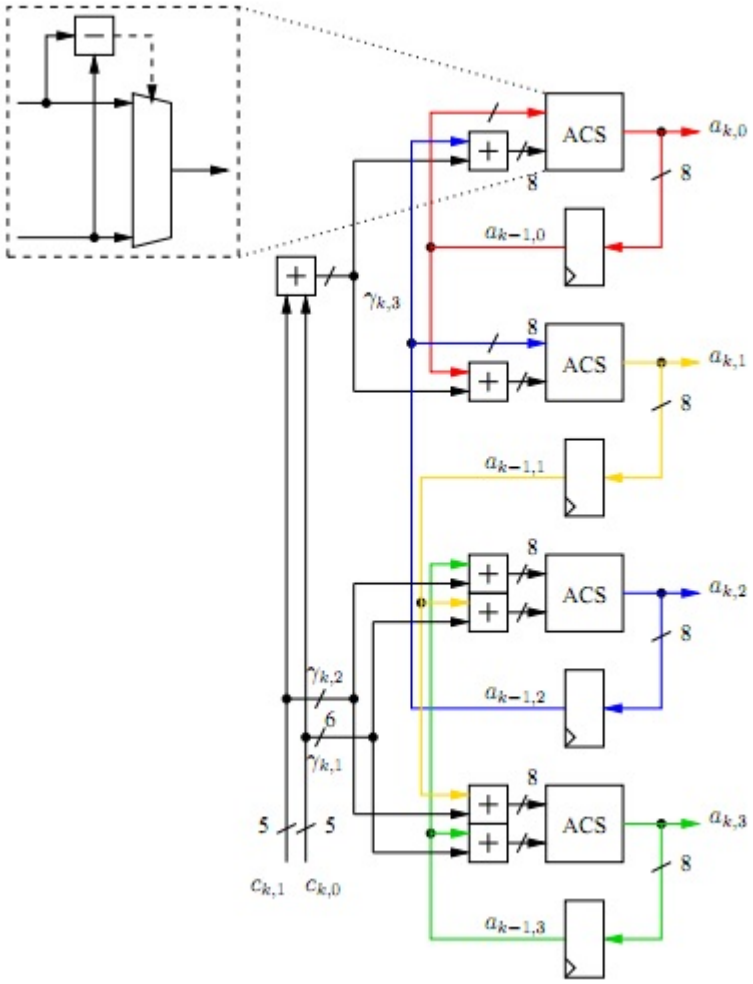

Figure 2.: a_k computation architecture

By looking at (Fig.2) we observed that the *ACS4* is made of:

- 7 adders;
- 4 *ACS* blocks;
- 4 registers.

We could divide all this structure in two macro block: combinational one and sequential one. However our choice is to divide each combinational component (each adder and

ACS) in order to achieve a simulation which is as close as possible to verilog one. Because every component has different sensitivity list.
In fact, looking at (Sec.A.1.2), twelve function can be seen: eleven for combinational components, and one for sequential components.

1.2.2 Realization of source codes

After that the decisions about how to divide hardware in blocks are taken, we wrote the SystemC code.
To make a parametric design we created a header file containing the value of parameters (Sec.A.1.1).
ACS4 architecture is described through two file shown in (Sec.A.1.2):

- Header, in which we declared the function declarations of internal blocks, the internal, input and output signals. For each internal block we had to wrote the sensitivity list.
- Cpp, in which we wrote the code that implements the methods.

In our codes two different model can be seen: *synthesizable* and *not-synthesizable* divided by preprocessor directives. The first model can be used for physical realization of the architecture (that is not the goal of this laboratory experience). While, the second model is used for hardware simulation of the system.
To realize the delays there isn't an assignment in SystemC. But, we exploited the **event-driven** simulation. In that way, we can signal an event with method `notify()`, delayed of a given amount of time. Each function works on dummy signals and every time that an event occurs, the values of these signal are put on the real ones.
There are two different delay values, one for propagation delay (t_{pd}) of combinational components and *clock-to-output delay* (t_{co}) of sequential components.

2 FIRST SIMULATION WITH G++

Before starting with the simulation, we needed to create a test-bench.
We had to provide a number of signal for testing the *ACS4*:

- a clock;
- an asynchronous reset;
- a synchronous clear;
- a couple of input signal.

Following the given example we realized a structural test-bench. This means that we built different classes:

1. **Starter**, shown in (Sec.A.1.6). It generates the asynchronous reset signal.
2. **Data Generation**, shown in (Sec.A.1.4). It generates the `CLEAR` and input (`C0,C1`) signals.
3. **Data Writer**, shown in (Sec.A.1.5). It implements a sequential process that allows to write the output of *ACS4* on a file.

The objects of all these class are connected together in the **test-bench** file, shown in (SecA.1.3).

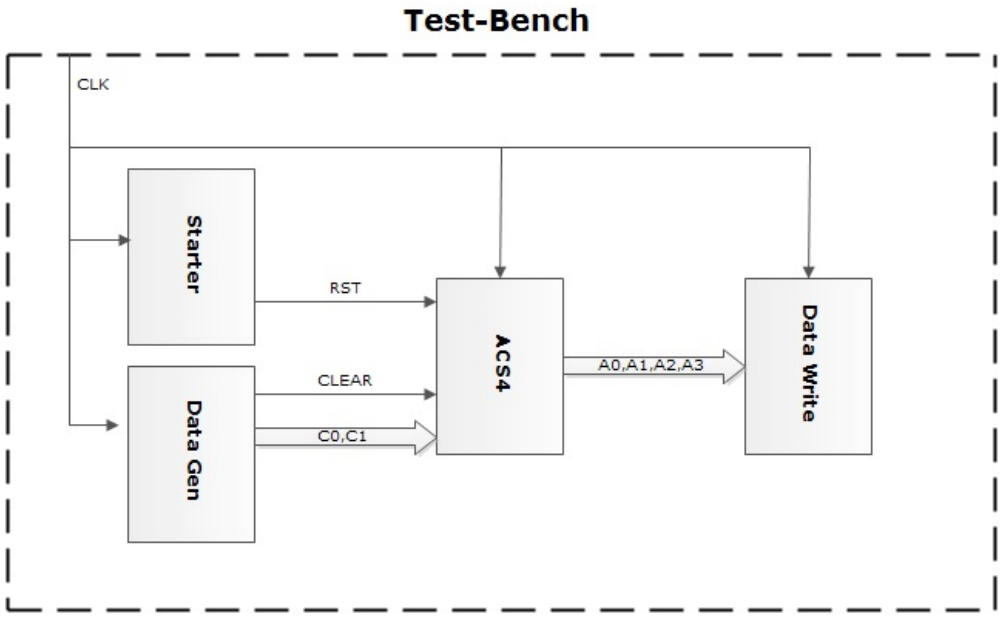

Figure 3.: Test-bench block diagram

(Fig.3) shows how the different blocks are connected in *test-bench*.

To simulate with *G++* compiler we wrote the `Makefile`, which is a file containing rules to manage the creation of an executable file. The content of this file is:

```
CC=g++
#CFLAGS=-O3 -Wall
CFLAGS=-O0 -g -Wall
#LFLAGS=-lsystemc -lm
LFLAGS=-lsystemc

IDIR=/software/SystemC/systemc-2.2.0/include
LDIR=/software/SystemC/systemc-2.2.0/lib-linux64
#IDIR=/usr/local/systemc-2.2.0/include
#LDIR=/usr/local/systemc-2.2.0/lib-linux
#LDIR=/usr/local/systemc-2.2.0/lib-cygwin

SRCDIR=./src
OBJDIR=./obj
BINDIR=./bin
TBDIR=./tb

$(BINDIR)/tb_acs4 :: $(OBJDIR)/acs4.o $(OBJDIR)/starter.o \
                     $(OBJDIR)/data_gen.o $(OBJDIR)/data_writer.o \
                     $(OBJDIR)/tb_acs4.o
                     $(CC) $? -o $@ -L $(LDIR) $(LFLAGS)

$(OBJDIR)/tb_acs4.o :: $(TBDIR)/tb_acs4.cpp
        $(CC) $(CFLAGS) -c $? -o $@ -I $(IDIR) -I $(SRCDIR)

$(OBJDIR)/starter.o :: $(TBDIR)/starter.cpp
        $(CC) $(CFLAGS) -c $? -o $@ -I $(IDIR) -I $(SRCDIR)

$(OBJDIR)/data_gen.o :: $(TBDIR)/data_gen.cpp
        $(CC) $(CFLAGS) -c $? -o $@ -I $(IDIR) -I $(SRCDIR)

$(OBJDIR)/data_writer.o :: $(TBDIR)/data_writer.cpp
        $(CC) $(CFLAGS) -c $? -o $@ -I $(IDIR) -I $(SRCDIR)

$(OBJDIR)/acs4.o :: $(SRCDIR)/acs4.cpp
        $(CC) $(CFLAGS) -c $? -o $@ -I $(IDIR) -I $(SRCDIR)

clean ::
        rm -rf $(BINDIR)/* $(OBJDIR)/*
```

The waveforms resulted by this simulation are viewed using *GTKwave*, and show in (Fig.4)

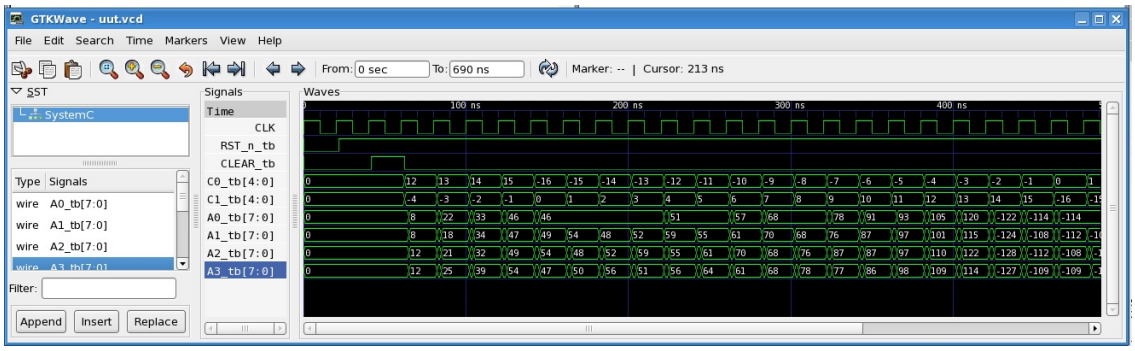

Figure 4.: G++ compiler and GTKwave viewer

The results are stored in `results.txt`. This file is used by us to check the correct behavior of our *ACS4*. In fact, using *Matlab* we created another high level description of the architecture (shows in (Sec.A.2)) which store the results in `output.txt`. Finally using the C program shows in (Sec.A.3) we obtained that the results are correct.
So, we could proceed with the others two simulation.

3 SECOND SIMULATION WITH MODELSIM

The second simulation use *Modelsim* both to compile and to view the results.
After a few modifications of test-bench file we started the simulation, obtaining as result the waveforms shown in (Fig.5)

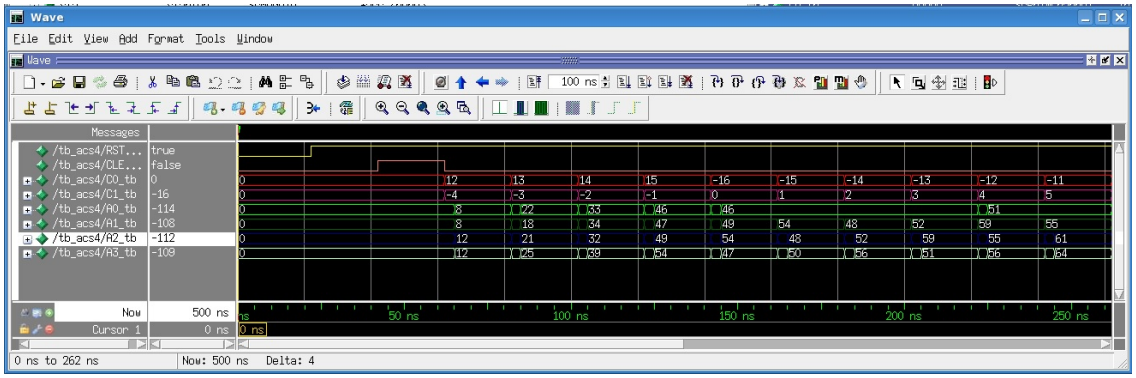

Figure 5.: Modelsim simulation

Of course, the waveforms are the same that previous simulation, because the source codes are also the same.

4 THIRD SIMULATION WITH VERILOG SOURCE

For this last simulation we had substituted the *SystemC ACS4* module with the corresponding verilog one.
Following the steps indicated in the given example, that include the creation of a *SystemC* wrapper for connecting the verilog component (shows in (Sec.A.1.7)), we performed this last simulation using *Modelsim*.
The result is shown in (Fig.6).

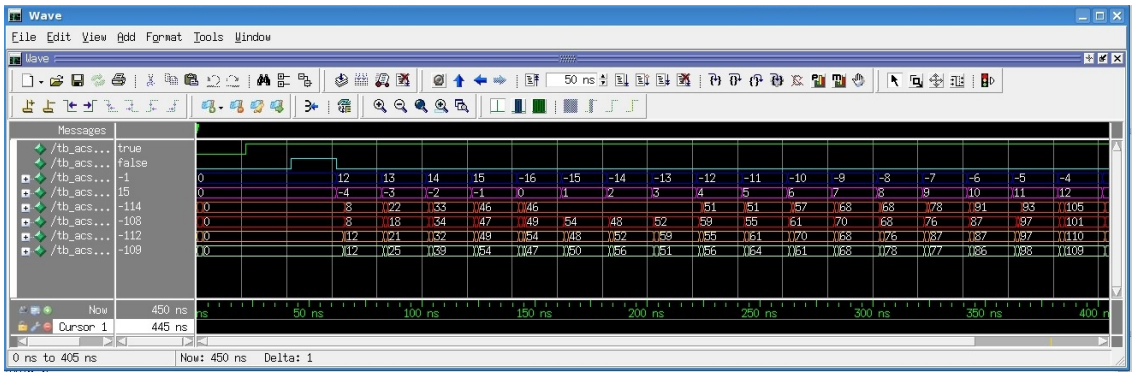

Figure 6.: Modelsim simulation with verilog source

As we can see by (Fig.6) the simulation give us the same waveform than the previous ones. Also the width of output glitch, due to internal delay, is the same.
This means that our *SystemC* source codes describes the same architecture of verilog one.

A APPENDIX - CODE

A.1 SYSTEMC

A.1.1 Constants

```
#ifndef __CONSTANTS_H
#define __CONSTANTS_H
#define Cbit 5
#define Abit 8
#define Gbit 6

#define NPROC_TCO 1
#define NPROC_TPD 2

#define TB_CLK_PERIOD 20
#define TB_TPD 2
#define TB_TCO 1

#define SIM_CYCLES_OFFSET 5

#endif
```

A.1.2 ACS4

Header

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

#ifndef __ACS4_H
#define __ACS4_H

SC_MODULE (ACS4)
{
  //entity
  public:
  sc_in<bool> CLK;
  sc_in<bool> RST_n;
  sc_in<bool> CLEAR;
  sc_in< sc_int<Cbit> > C0;
  sc_in< sc_int<Cbit> > C1;
```

```
16    sc_out< sc_int<Abit> > A0;
      sc_out< sc_int<Abit> > A1;
18    sc_out< sc_int<Abit> > A2;
      sc_out< sc_int<Abit> > A3;
20
      //internal signals
22    private:
      sc_signal< sc_int<Abit> > a0_reg;
24    sc_signal< sc_int<Abit> > a1_reg;
      sc_signal< sc_int<Abit> > a2_reg;
26    sc_signal< sc_int<Abit> > a3_reg;

28    sc_signal< sc_int<Gbit> > g1;
      sc_signal< sc_int<Gbit> > g2;
30    sc_signal< sc_int<Gbit> > g3;

32    sc_signal< sc_int<Abit> > a0_a0_g0;
      sc_signal< sc_int<Abit> > a0_a2_g3;
34    sc_signal< sc_int<Abit> > a1_a2_g0;
      sc_signal< sc_int<Abit> > a1_a0_g3;
36    sc_signal< sc_int<Abit> > a2_a3_g2;
      sc_signal< sc_int<Abit> > a2_a1_g1;
38    sc_signal< sc_int<Abit> > a3_a1_g2;
      sc_signal< sc_int<Abit> > a3_a3_g1;
40
      #ifndef __CTOS__
42       sc_signal< sc_int<Abit> > a0_reg_i;
         sc_signal< sc_int<Abit> > a1_reg_i;
44       sc_signal< sc_int<Abit> > a2_reg_i;
         sc_signal< sc_int<Abit> > a3_reg_i;
46
         sc_signal< sc_int<Gbit> > g1_i;
48       sc_signal< sc_int<Gbit> > g2_i;
         sc_signal< sc_int<Gbit> > g3_i;
50
         sc_signal< sc_int<Abit> > a0_a0_g0_i;
52       sc_signal< sc_int<Abit> > a0_a2_g3_i;
         sc_signal< sc_int<Abit> > a1_a2_g0_i;
54       sc_signal< sc_int<Abit> > a1_a0_g3_i;
         sc_signal< sc_int<Abit> > a2_a3_g2_i;
56       sc_signal< sc_int<Abit> > a2_a1_g1_i;
         sc_signal< sc_int<Abit> > a3_a1_g2_i;
58       sc_signal< sc_int<Abit> > a3_a3_g1_i;

60       sc_signal< sc_int<Abit> > A0_i;
         sc_signal< sc_int<Abit> > A1_i;
62       sc_signal< sc_int<Abit> > A2_i;
         sc_signal< sc_int<Abit> > A3_i;
64
```

```
    sc_time tpd;
    sc_time tco;
    sc_event ev_tpd1;
    sc_event ev_tpd2;
    sc_event ev_tpd3;
    sc_event ev_tpd4;
    sc_event ev_tpd5;
    sc_event ev_tpd6;
    sc_event ev_tpd7;
    sc_event ev_tpd8;
    sc_event ev_tpd9;
    sc_event ev_tpd10;
    sc_event ev_tpd11;
    sc_event ev_tpd12;
    sc_event ev_tpd13;
    sc_event ev_tco;
    void ACS4_tpd1();
    void ACS4_tpd2();
    void ACS4_tpd3();
    void ACS4_tpd4();
    void ACS4_tpd5();
    void ACS4_tpd6();
    void ACS4_tpd7();
    void ACS4_tpd8();
    void ACS4_tpd9();
    void ACS4_tpd10();
    void ACS4_tpd11();
    void ACS4_tpd12();
    void ACS4_tpd13();
    void ACS4_tco();
    void as_beh_main_tpd( void );
    void as_beh_a0_tpd ( void );
    void as_beh_a1_tpd ( void );
    void as_beh_a2_tpd ( void );
    void as_beh_a3_tpd ( void );
    void as_beh_a4_tpd ( void );
    void as_beh_a5_tpd ( void );
    void ACS1_tpd (void);
    void ACS2_tpd (void);
    void ACS3_tpd (void);
    void ACS4_tpd (void);
    void sin_beh_main_tco(void);
    void as_beh_a0_a0_g0_tpd (void);
    void as_beh_a1_a2_g0_tpd (void);
#else

    void as_beh_main( void );//asynchronous behavior input function

    void as_beh_a0 ( void );//asynchronous behavior a0 function
```

```
114     void as_beh_a1 ( void );//asynchronous behavior a1 function
        void as_beh_a2 ( void );//asynchronous behavior a2 function
116     void as_beh_a3 ( void );//asynchronous behavior a3 function
        void as_beh_a4 ( void );//asynchronous behavior a4 function
118     void as_beh_a5 ( void );//asynchronous behavior a5 function

120     void ACS1 (void);
        void ACS2 (void);
122     void ACS3 (void);
        void ACS4 (void);
124   ////////////
        void sin_beh_main(void);
126   #endif


128
    public:
130   SC_HAS_PROCESS(ACS4);
    #ifndef __CTOS__
132     ACS4(sc_module_name ACS4_ctor, int ptpd, int ptco) :
        sc_module(ACS4_ctor), tpd(sc_time(ptco, SC_NS))
134   #else
        ACS4(sc_module_name ACS4_ctor) : sc_module(ACS4_ctor)
136   #endif
    {
138     #ifndef __CTOS__
          SC_METHOD(as_beh_main_tpd);
140     #else
          SC_METHOD(as_beh_main);
142     #endif
        sensitive << C1;
144     sensitive << C0;

146     #ifndef __CTOS__
          SC_METHOD(as_beh_a0_tpd); //adder A0
148     #else
          SC_METHOD(as_beh_a0);
150     #endif
        sensitive << g1;
152     sensitive << a3_reg;

154     #ifndef __CTOS__
          SC_METHOD(as_beh_a1_tpd);
156     #else
          SC_METHOD(as_beh_a1);
158     #endif
        sensitive << g2;
160     sensitive << a1_reg;

162     #ifndef __CTOS__
```

```
    SC_METHOD(as_beh_a2_tpd);
#else
    SC_METHOD(as_beh_a2);
#endif
sensitive << g1;
sensitive << a1_reg;

#ifndef __CTOS__
    SC_METHOD(as_beh_a3_tpd);
#else
    SC_METHOD(as_beh_a3);
#endif
sensitive << g2;
sensitive << a3_reg;

#ifndef __CTOS__
    SC_METHOD(as_beh_a4_tpd);
#else
    SC_METHOD(as_beh_a4);
#endif
sensitive << g3;
sensitive << a0_reg;

#ifndef __CTOS__
    SC_METHOD(as_beh_a5_tpd);
#else
    SC_METHOD(as_beh_a5);
#endif
sensitive << g3;
sensitive << a2_reg;

#ifndef __CTOS__
    SC_METHOD(as_beh_a0_a0_g0_tpd);
#else
    SC_METHOD(as_beh_a0_a0_g0);
#endif
sensitive << a0_reg;

#ifndef __CTOS__
    SC_METHOD(ACS1_tpd);
#else
    SC_METHOD(ACS1);
#endif
sensitive << a0_a2_g3;
sensitive << a0_a0_g0;

#ifndef __CTOS__
    SC_METHOD(as_beh_a1_a2_g0_tpd);
#else
```

```
212         SC_METHOD(as_beh_a1_a2_g0);
        #endif
214     sensitive << a2_reg;

216     #ifndef __CTOS__
          SC_METHOD(ACS2_tpd);
218     #else
          SC_METHOD(ACS2);
220     #endif
        sensitive << a1_a0_g3;
222     sensitive << a1_a2_g0;

224     #ifndef __CTOS__
          SC_METHOD(ACS3_tpd);
226     #else
          SC_METHOD(ACS3);
228     #endif
        sensitive << a2_a3_g2;
230     sensitive << a2_a1_g1;

232     #ifndef __CTOS__
          SC_METHOD(ACS4_tpd);
234     #else
          SC_METHOD(ACS4);
236     #endif
        sensitive << a3_a1_g2;
238     sensitive << a3_a3_g1;

240     #ifndef __CTOS__
          SC_METHOD(sin_beh_main_tco);
242     #else
          SC_METHOD(sin_beh_main);
244     #endif
        sensitive << CLK.pos();
246     sensitive << RST_n.neg();

248     #ifndef __CTOS__
          SC_METHOD(ACS4_tpd1);
250       sensitive << ev_tpd1;

252       SC_METHOD(ACS4_tpd2);
          sensitive << ev_tpd2;
254
          SC_METHOD( ACS4_tpd3);
256       sensitive << ev_tpd3;

258       SC_METHOD( ACS4_tpd4);
          sensitive << ev_tpd4;
260
```

```
        SC_METHOD(ACS4_tpd5);
        sensitive << ev_tpd5;

        SC_METHOD( ACS4_tpd6);
        sensitive << ev_tpd6;

        SC_METHOD( ACS4_tpd7);
        sensitive << ev_tpd7;

        SC_METHOD( ACS4_tpd8);
        sensitive << ev_tpd8;

        SC_METHOD( ACS4_tpd9);
        sensitive << ev_tpd9;

        SC_METHOD( ACS4_tpd10);
        sensitive << ev_tpd10;

        SC_METHOD( ACS4_tpd11);
        sensitive << ev_tpd11;

        SC_METHOD( ACS4_tpd12);
        sensitive << ev_tpd12;

        SC_METHOD( ACS4_tpd13);
        sensitive << ev_tpd13;

        SC_METHOD( ACS4_tco);
        sensitive << ev_tco;

    #endif
  }
};

#endif
```

C++ Source

```
#include "acs4.h"

#ifndef __CTOS__
/*****************************************************************/
/* TDP and TCO Version */
/*****************************************************************/

  void ACS4::ACS4_tpd1()
  {
    sc_int<Gbit> Temp =g1_i.read();
    g1.write(Temp);
    Temp =g2_i.read();
```

```
13      g2.write(Temp);
        Temp =g3_i.read();
15      g3.write(Temp);

17      ev_tpd1.cancel();
    }
19
    void ACS4::ACS4_tpd2()
21  {
        sc_int<Abit> Temp =a3_a3_g1_i.read();
23      a3_a3_g1.write(Temp);

25      ev_tpd2.cancel();
    }
27
    void ACS4::ACS4_tpd3()
29  {
        sc_int<Abit> Temp =a3_a1_g2_i.read();
31      a3_a1_g2.write(Temp);

33      ev_tpd3.cancel();
    }
35
    void ACS4::ACS4_tpd4()
37  {
        sc_int<Abit> Temp =a2_a1_g1_i.read();
39      a2_a1_g1.write(Temp);

41      ev_tpd4.cancel();
    }
43
    void ACS4::ACS4_tpd5()
45  {
        sc_int<Abit> Temp =a2_a3_g2_i.read();
47      a2_a3_g2.write(Temp);

49      ev_tpd5.cancel();
    }
51
    void ACS4::ACS4_tpd6()
53  {
        sc_int<Abit> Temp =a1_a0_g3_i.read();
55      a1_a0_g3.write(Temp);

57      ev_tpd6.cancel();
    }
59
    void ACS4::ACS4_tpd7()
61  {
```

```
    sc_int<Abit> Temp =a0_a2_g3_i.read();
63  a0_a2_g3.write(Temp);

65  ev_tpd7.cancel();
}
67
void ACS4::ACS4_tpd8()
69  {
    sc_int<Abit> Temp =A0_i.read();
71  A0.write(Temp);

73  ev_tpd8.cancel();
}
75
void ACS4::ACS4_tpd9()
77  {
    sc_int<Abit> Temp =A1_i.read();
79  A1.write(Temp);

81  ev_tpd9.cancel();
}
83
void ACS4::ACS4_tpd10()
85  {
    sc_int<Abit> Temp =A2_i.read();
87  A2.write(Temp);

89  ev_tpd10.cancel();
}
91
void ACS4::ACS4_tpd11()
93  {
    sc_int<Abit> Temp =A3_i.read();
95  A3.write(Temp);

97  ev_tpd11.cancel();
}
99
void ACS4::ACS4_tpd12()
101 {
    sc_int<Abit> Temp =a0_a0_g0_i.read();
103 a0_a0_g0.write(Temp);

105 ev_tpd12.cancel();
}
107
void ACS4::ACS4_tpd13()
109 {
    sc_int<Abit> Temp =a1_a2_g0_i.read();
```

```
111     a1_a2_g0.write(Temp);

113     ev_tpd13.cancel();
    }
115
    void ACS4::ACS4_tco()
117 {
    sc_int<Abit> Temp =a0_reg_i.read();
119 a0_reg.write(Temp);
    Temp = a1_reg_i.read();
121 a1_reg.write(Temp);
    Temp = a2_reg_i.read();
123 a2_reg.write(Temp);
    Temp = a3_reg_i.read();
125 a3_reg.write(Temp);

127 ev_tco.cancel();
    }
129


131
    void ACS4::as_beh_main_tpd( void )
133 {
    sc_int<Cbit> TempC0 = C0.read();
135 sc_int<Cbit> TempC1 = C1.read();
    g3_i.write( TempC0 + TempC1);
137 g1_i.write(TempC0);
    g2_i.write(TempC1);
139
    ev_tpd1.notify(tpd);
141 }

143 void ACS4::as_beh_a0_tpd( void )
    {
145 sc_int<Gbit> TempG1 = g1.read();
    sc_int<Abit> Temp =TempG1 + a3_reg.read();
147 a3_a3_g1_i.write(Temp);

149 ev_tpd2.notify(tpd);
    }
151
    void ACS4::as_beh_a1_tpd ( void )
153 {
    sc_int<Gbit> TempG2 = g2.read();
155 sc_int<Abit> Temp = TempG2 + a1_reg.read();
    a3_a1_g2_i.write(Temp);
157
    ev_tpd3.notify(tpd);
159 }
```

```
161  void ACS4::as_beh_a2_tpd ( void )
     {
163
       sc_int<Gbit> TempG1 = g1.read();
165    sc_int<Abit> Temp = TempG1 + a1_reg.read();
       a2_a1_g1_i.write(Temp);
167
       ev_tpd4.notify(tpd);
169  }

171  void ACS4::as_beh_a3_tpd ( void )
     {
173    sc_int<Gbit> TempG2 = g2.read();
       sc_int<Abit> Temp = TempG2 + a3_reg.read();
175    a2_a3_g2_i.write(Temp);

177    ev_tpd5.notify(tpd);
     }
179
     void ACS4::as_beh_a4_tpd ( void )
181  {
       sc_int<Gbit> TempG3 = g3.read();
183    sc_int<Abit> Temp = TempG3 + a0_reg.read();
       a1_a0_g3_i.write(Temp);
185
       ev_tpd6.notify(tpd);
187  }

189  void ACS4::as_beh_a5_tpd ( void )
     {
191    sc_int<Gbit> TempG3 = g3.read();
       sc_int<Abit> Temp = TempG3 + a2_reg.read();
193    a0_a2_g3_i.write(Temp);


195
       ev_tpd7.notify(tpd);
197  }

199  void ACS4::as_beh_a0_a0_g0_tpd (void)
     {
201    sc_int<Abit> Temp = a0_reg.read();
       a0_a0_g0_i.write(Temp);
203
       ev_tpd12.notify(tpd);
205  }

207  void ACS4::ACS1_tpd (void)
     {
```

```
209
    sc_int<Abit> Temp =0;
211 Temp = a0_a0_g0.read() — a0_a2_g3.read();
    if(Temp[Abit—1])
213 {
      Temp = a0_a2_g3.read();
215   A0_i.write(Temp);
    }
217 else
    {
219   Temp = a0_a0_g0.read();
      A0_i.write(Temp);
221 }

223 ev_tpd8.notify(tpd);
  }
225
  void ACS4::as_beh_a1_a2_g0_tpd (void)
227 {
    sc_int<Abit> Temp = a2_reg.read();
229 a1_a2_g0_i.write(Temp);

231 ev_tpd13.notify(tpd);
  }
233
  void ACS4::ACS2_tpd (void)
235 {

237 sc_int<Abit> Temp =0;
    Temp = a1_a2_g0.read() — a1_a0_g3.read();
239 if(Temp[Abit—1])
    {
241   Temp = a1_a0_g3.read();
      A1_i.write(a1_a0_g3.read());
243 }
    else
245 {
      Temp = a1_a2_g0.read();
247   A1_i.write(a1_a2_g0.read());
    }
249
    ev_tpd9.notify(tpd);
251 }

253 void ACS4::ACS3_tpd (void)
  {
255 sc_int<Abit> Temp =0;
    Temp = a2_a3_g2.read() — a2_a1_g1.read();
257 if(Temp[Abit—1])
```

```
    {
259   Temp = a2_a1_g1.read();
      A2_i.write(a2_a1_g1.read());
261 }
    else
263 {
      Temp = a2_a3_g2.read();
265   A2_i.write(a2_a3_g2.read());
    }
267
    ev_tpd10.notify(tpd);
269 }

271 void ACS4::ACS4_tpd (void)
    {
273   sc_int<Abit> Temp =0;
    Temp = a3_a1_g2.read() − a3_a3_g1.read();
275 if(Temp[Abit−1])
    {
277   Temp = a3_a3_g1.read();
      A3_i.write(a3_a3_g1.read());
279 }
    else
281 {
      Temp = a3_a1_g2.read();
283   A3_i.write(a3_a1_g2.read());
    }
285
    ev_tpd11.notify(tpd);
287 }

289 void ACS4::sin_beh_main_tco(void)
    {
291   sc_int<Abit> Temp_A0 = 0;
    sc_int<Abit> Temp_A1 = 0;
293 sc_int<Abit> Temp_A2 = 0;
    sc_int<Abit> Temp_A3 = 0;
295 //asynchronous reset
    if (RST_n.read() == 1)
297 {
      if(CLEAR.read() == 0)
299   {
        Temp_A0 = A0.read();
301     Temp_A1 = A1.read();
        Temp_A2 = A2.read();
303     Temp_A3 = A3.read();
      }
305 }
    a0_reg_i.write(Temp_A0);
```

```
307       a1_reg_i.write(Temp_A1);
          a2_reg_i.write(Temp_A2);
309       a3_reg_i.write(Temp_A3);

311       ev_tco.notify(tco);
      }
313
#else
315 /*****************************************************************/
/* Clear Version */
317 /*****************************************************************/
    void ACS4::as_beh_main( void )
319   {
        sc_int<Cbit> TempC0 = C0.read();
321     sc_int<Cbit> TempC1 = C1.read();
        g3.write( TempC0 + TempC1);
323     g1.write(TempC0);
        g2.write(TempC1);
325   }

327   void ACS4::as_beh_a0( void )
      {
329     sc_int<Gbit> TempG1 = g1.read();
        sc_int<Abit> Temp =TempG1 + a3_reg.read();
331     a3_a3_g1.write(Temp);
      }
333
      void ACS4::as_beh_a1( void )
335   {
        sc_int<Gbit> TempG2 = g2.read();
337     sc_int<Abit> Temp = TempG2 + a1_reg.read();
        a3_a1_g2.write(Temp);
339   }

341   void ACS4::as_beh_a2( void )
      {
343
        sc_int<Gbit> TempG1 = g1.read();
345     sc_int<Abit> Temp = TempG1 + a1_reg.read();
        a2_a1_g1.write(Temp);
347   }

349   void ACS4::as_beh_a3( void )
      {
351     sc_int<Gbit> TempG2 = g2.read();
        sc_int<Abit> Temp = TempG2 + a3_reg.read();
353     a2_a3_g2.write(Temp);
      }
355
```

```
void ACS4::as_beh_a4( void )
{
  sc_int<Gbit> TempG3 = g3.read();
  sc_int<Abit> Temp = TempG3 + a0_reg.read();
  a1_a0_g3.write(Temp);
}

void ACS4::as_beh_a5( void )
{
  sc_int<Gbit> TempG3 = g3.read();
  sc_int<Abit> Temp = TempG3 + a2_reg.read();
  a0_a2_g3.write(Temp);
}

void ACS4::as_beh_a0_a0_g0(void)
{
  sc_int<Abit> Temp = a0_reg.read();
  a0_a0_g0.write(Temp);
}

void ACS4::ACS1(void)
{

  sc_int<Abit> Temp =0;
  Temp = a0_a0_g0.read() − a0_a2_g3.read();
  if(Temp[Abit−1])
  {
    Temp = a0_a2_g3.read();
    A0.write(Temp);
  }
  else
  {
    Temp = a0_a0_g0.read();
    A0.write(Temp);
  }
}

void ACS4::as_beh_a1_a2_g0(void)
{
  sc_int<Abit> Temp = a2_reg.read();
  a1_a2_g0.write(Temp);
}

void ACS4::ACS2(void)
{

  sc_int<Abit> Temp =0;
  Temp = a1_a2_g0.read() − a1_a0_g3.read();
  if(Temp[Abit−1])
```

```
405     {
          Temp = a1_a0_g3.read();
407       A1.write(a1_a0_g3.read());
        }
409     else
        {
411       Temp = a1_a2_g0.read();
          A1.write(a1_a2_g0.read());
413     }
    }
415
    void ACS4::ACS3(void)
417 {
      sc_int<Abit> Temp =0;
419   Temp = a2_a3_g2.read() − a2_a1_g1.read();
      if(Temp[Abit−1])
421   {
        Temp = a2_a1_g1.read();
423     A2.write(a2_a1_g1.read());
      }
425   else
      {
427     Temp = a2_a3_g2.read();
        A2.write(a2_a3_g2.read());
429   }
    }
431
    void ACS4::ACS4(void)
433 {
      sc_int<Abit> Temp =0;
435   Temp = a3_a1_g2.read() − a3_a3_g1.read();
      if(Temp[Abit−1])
437   {
        Temp = a3_a3_g1.read();
439     A3.write(a3_a3_g1.read());
      }
441   else
      {
443     Temp = a3_a1_g2.read();
        A3.write(a3_a1_g2.read());
445   }
    }
447
    void ACS4::sin_beh_main(void)
449 {
      sc_int<Abit> Temp_A0 = 0;
451   sc_int<Abit> Temp_A1 = 0;
      sc_int<Abit> Temp_A2 = 0;
453   sc_int<Abit> Temp_A3 = 0;
```

```
    //asynchronous reset
455 if (RST_n.read() == 1)
    {
457   if(CLEAR.read() == 0)
      {
459     Temp_A0 = A0.read();
        Temp_A1 = A1.read();
461     Temp_A2 = A2.read();
        Temp_A3 = A3.read();
463   }
    }
465 a0_reg.write(Temp_A0);
    a1_reg.write(Temp_A1);
467 a2_reg.write(Temp_A2);
    a3_reg.write(Temp_A3);
469 }

471 #endif


473
#ifdef __CTOS__
475 SC_MODULE_EXPORT(ACS4);
#endif
```

A.1.3 Test-Bench

C++ Source

```
#include<stdio.h>
2
#include"constants.h"
4
#ifdef MTI_SYSTEMC
6 #define NVAL 24
#endif
8
#include"starter.h"
10 #include"data_gen.h"
#include"data_writer.h"
12
#ifdef USE_VLOG
14 #include"acs4_rtl.h"
#else
16 #include"acs4.h"
#endif
18


20 #ifdef MTI_SYSTEMC
```

```
SC_MODULE(tb_acs4)
#else
int sc_main (int argc, char **argv)
#endif
{
 #ifndef MTI_SYSTEMC
     sc_report_handler::set_actions("/IEEE_Std_1666/deprecated", SC_DO_NOTHING);
     int NVAL;

     if (argc != 2)
     {
          printf("Use: %s <N>\n", argv[0]);
          return 1;
     }

     NVAL = atoi(argv[1]);
 #endif

  sc_signal<bool> RST_n_tb; /// asynchronous reset
  sc_signal<bool> CLEAR_tb; /// synchronous clear port
  sc_signal< sc_int<Cbit> > C0_tb;
  sc_signal< sc_int<Cbit> > C1_tb;
  sc_signal< sc_int<Abit> > A0_tb;
  sc_signal< sc_int<Abit> > A1_tb;
  sc_signal< sc_int<Abit> > A2_tb;
  sc_signal< sc_int<Abit> > A3_tb;

  #ifdef MTI_SYSTEMC
  sc_clock CLK;
     starter sta;
     data_gen dg;
     data_writer dw;
  #ifdef USE_VLOG
  acs4 uut;
  #else
     ACS4 uut;
  #endif

  SC_CTOR(tb_acs4)
  : CLK("CLK", TB_CLK_PERIOD),
       sta("sta", TB_CLK_PERIOD, TB_TPD),
       dg("dg", TB_CLK_PERIOD, TB_TPD, NVAL),
     dw("dw", TB_CLK_PERIOD, NVAL),
  #ifdef USE_VLOG
  uut("uut", "acs4")
  #else
     uut("uut", TB_TPD, TB_TCO)
  #endif
  {
```

```
70  #else
      sc_clock CLK("CLK", TB_CLK_PERIOD);
72    starter sta("sta", TB_CLK_PERIOD, TB_TPD);
      data_gen dg("dg", TB_CLK_PERIOD, TB_TPD, NVAL);
74    data_writer dw("dw", TB_CLK_PERIOD, NVAL);
      ACS4 uut("uut", TB_TPD, TB_TCO);
76  #endif


78
    /// starter
80  sta.CLK(CLK);
    sta.RST_n(RST_n_tb);
82
    /// data gen
84  dg.CLK(CLK);
    dg.CLEAR(CLEAR_tb);
86  dg.C0(C0_tb);
    dg.C1(C1_tb);
88
    /// uut
90  uut.CLK(CLK);
    uut.RST_n(RST_n_tb);
92  uut.CLEAR(CLEAR_tb);
    uut.C0(C0_tb);
94  uut.C1(C1_tb);
    uut.A0(A0_tb);
96  uut.A1(A1_tb);
    uut.A2(A2_tb);
98  uut.A3(A3_tb);


100
    /// write results
102 dw.CLK(CLK);
    dw.A1(A1_tb);
104 dw.A2(A2_tb);
    dw.A3(A3_tb);
106 dw.A0(A0_tb);

108 #ifdef MTI_SYSTEMC
    }
110 #endif

112 #ifndef MTI_SYSTEMC
      sc_trace_file *tf = sc_create_vcd_trace_file("uut");
114   ((vcd_trace_file*)tf)->sc_set_vcd_time_unit(-9);

116   sc_trace(tf, CLK, "CLK");
    sc_trace(tf, RST_n_tb, "RST_n_tb");
118   sc_trace(tf, CLEAR_tb, "CLEAR_tb");
```

```
    sc_trace(tf, C0_tb, "C0_tb");
120 sc_trace(tf, C1_tb, "C1_tb");
    sc_trace(tf, A0_tb, "A0_tb");
122 sc_trace(tf, A1_tb, "A1_tb");
    sc_trace(tf, A2_tb, "A2_tb");
124 sc_trace(tf, A3_tb, "A3_tb");

126 sc_start((NVAL+SIM_CYCLES_OFFSET)*TB_CLK_PERIOD, SC_NS);

128 sc_close_vcd_trace_file(tf);

130 return 0;
  }
132 #else
  };
134 #endif
#ifdef MTI_SYSTEMC
136 SC_MODULE_EXPORT(tb_acs4);
#endif
```

A.1.4 Data Generator

Header

```
1 #include<systemc.h>
  #include"constants.h"
3
  #ifndef __DATA_GEN_H
5 #define __DATA_GEN_H

7 SC_MODULE(data_gen)
  {
9   public :
    sc_in<bool> CLK;
11  sc_out<bool> CLEAR;
    sc_out< sc_int<Cbit> > C0;
13  sc_out< sc_int<Cbit> > C1;

15  private :

17  sc_time clk_period;
    sc_time tpd;
19 unsigned int Nval;

21  void data_gen_beh();

23  public :
    SC_HAS_PROCESS(data_gen);
```

```
25    data_gen(sc_module_name data_gen_ctor,
        int pclk_period,
27      int ptpd,
          int pN) :
29      sc_module(data_gen_ctor),
        clk_period(sc_time(pclk_period,SC_NS)),
31      tpd(sc_time(ptpd,SC_NS)),
        Nval(pN)
33
    {
35      SC_THREAD(data_gen_beh);
        sensitive << CLK.pos();
37    }
};
39
#endif
```

C++ Source

```
#include "data_gen.h"
2
void data_gen::data_gen_beh()
4 {
    CLEAR.write(0);
6   C0.write(0);
    C1.write(0);
8


10  wait(2*clk_period);
    wait(tpd);
12
    CLEAR.write(1);
14
    wait(clk_period);
16
    CLEAR.write(0);
18
    unsigned int i;
20  for(i = 3; i<Nval; i++){
        C0.write(12+i-3);
22      C1.write(28+i-3);
        wait(clk_period);
24  }

26 }
```

A.1.5 Data Writer

Header

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

#ifndef __DATA_WRITER_H
#define __DATA_WRITER_H

SC_MODULE(data_writer)
{
  public :
  sc_in<bool> CLK;
  sc_in< sc_int<Abit> > A0;
  sc_in< sc_int<Abit> > A1;
  sc_in< sc_int<Abit> > A2;
  sc_in< sc_int<Abit> > A3;

  private :

  sc_time clk_period;
  unsigned int Nval;
  FILE *fp;
  unsigned int Ccnt;
  unsigned char done;

  void data_writer_beh();

  public :
  SC_HAS_PROCESS(data_writer);
  data_writer(sc_module_name data_writer_ctor,
        int pclk_period,
              int pN) :
              sc_module(data_writer_ctor),
              clk_period(sc_time(pclk_period,SC_NS)),
              Nval(pN)
  {
    Ccnt = 0;
    done = 0;
    fp = fopen("./result.txt", "w");
    SC_METHOD(data_writer_beh);
    sensitive << CLK.pos();
  }
};

#endif
```

C++ Source

```
#include"data_writer.h"

void data_writer::data_writer_beh()
```

```
{
  signed int A0val;
  signed int A1val;
  signed int A2val;
  signed int A3val;

  A0val = A0.read();
  A1val = A1.read();
  A2val = A2.read();
  A3val = A3.read();

  fprintf(fp, "%d\n", A0val);
  fprintf(fp, "%d\n", A1val);
  fprintf(fp, "%d\n", A2val);
  fprintf(fp, "%d\n", A3val);
  fprintf(fp, "\n");
  Ccnt++;
  if (Ccnt == Nval)
  {
    fclose(fp);
    done = 1;
  }
}
```

A.1.6 Starter

Header

```
#include<systemc.h>
#ifndef __STARTER_H
#define __STARTER_H

SC_MODULE(starter)
{
  public :
  sc_in<bool> CLK;
  sc_out<bool> RST_n;

  private :
  sc_time clk_period;
  sc_time tpd;
  void starter_beh();

  public :
  SC_HAS_PROCESS(starter);
  starter(sc_module_name starter_ctor,
    int pclk_period,
    int ptpd) :
```

```
    sc_module(starter_ctor),
22  clk_period(sc_time(pclk_period, SC_NS)),
    tpd(sc_time(ptpd, SC_NS))
24 {
    SC_THREAD(starter_beh);
26  sensitive << CLK.pos();
  }
28 };

30 #endif
```

C++ Source

```
#include "starter.h"
2
void starter::starter_beh()
4 {
  RST_n.write(0);
6 wait(clk_period);
  wait(tpd);
8 RST_n.write(1);
  wait(clk_period);
10 }
```

A.1.7 ACS4_rtl

Header

```
#ifndef _SCGENMOD_acs4_
2 #define _SCGENMOD_acs4_

4 #include "systemc.h"

6 class acs4 : public sc_foreign_module
  {
8 public:
    sc_in<bool> CLK;
10  sc_in<bool> RST_n;
    sc_in<bool> CLEAR;
12  sc_in< sc_int<5> > C0;
    sc_in< sc_int<5> > C1;
14  sc_out< sc_int<8> > A0;
    sc_out< sc_int<8> > A1;
16  sc_out< sc_int<8> > A2;
    sc_out< sc_int<8> > A3;
18


20  acs4(sc_module_name nm, const char* hdl_name)
     : sc_foreign_module(nm),
```

```
22          CLK("CLK"),
            RST_n("RST_n"),
24          CLEAR("CLEAR"),
            C0("C0"),
26          C1("C1"),
            A0("A0"),
28          A1("A1"),
            A2("A2"),
30          A3("A3")
    {
32          elaborate_foreign_module(hdl_name);
    }
34  ~acs4()
    {}
36
};
38
#endif
```

A.2 MATLAB

A.2.1 From two's complement to integer function

```
function [ c2 ] = complement2int( a )
2 %Function take as input a string which represent a number in two's complement
%and provides outside its value converted in integer
4 %     [ c2 ] = complement2int( a )
if a(1) == '1'
6       c2 = -2^((length(a))-1) + bin2dec(a(2:length(a)));
else
8       c2 = bin2dec(a);
end
10 end
```

A.2.2 From integer to two's complement function

```
1 function [ a_bin ] = int2complement( a, N )
%Function take as input a integer number and provides outside
3 %a string which represent the number in two's complement
% [ a_bin ] = int2complement( a, N )
5
if a < 0
7       for i = 1:100
            if abs(a) < 2^(N+i-1)
```

```
                a_bin = dec2bin(a+2^(N+i-1),N);
                break
            end

        end

else
    a_bin = dec2bin(a,N);
end


end
```

A.2.3 Input Creator

```
function [ C1, C2 ] = cCreator( Nval )
%This function creates the two vectors which contain
%the values of inputs C1 and C2
% function [ C1, C2 ] = cCreator( Nval )
C1 = zeros(Nval-3,1);
C2 = zeros(Nval-3,1);
for i=1:Nval-3
    C1(i) = 12+i-1;
    C2(i) = -4+i-1;
end

end
```

A.2.4 ACS4

```
function [ A0,A1,A2,A3 ] = acs4( Nval )
% This function is used to simulate the behavioural of ACS4
% realized in third laboratory of Integrated Systems Architecture
% [ A0,A1,A2,A3 ] = acs4( Nval )

%creating the input vector
[C1, C2] = cCreator(Nval);


%prealocation of output value
A0 = zeros(Nval-3,1);
A1 = zeros(Nval-3,1);
A2 = zeros(Nval-3,1);
A3 = zeros(Nval-3,1);

for i=1:Nval-3
```

```
17
    C1_bin = int2complement(C1(i),5);
19  C2_bin = int2complement(C2(i),5);

21  C1(i) = complement2int(C1_bin);
    C2(i) = complement2int(C2_bin);
23
    G3 = C1(i)+C2(i);
25  G2 = C2(i);
    G1 = C1(i);
27
    if i == 1
29      a0_a0_g0 = 0;
        a0_a2_g3 = G3;
31      a1_a2_g0 = 0;
        a1_a0_g3 = G3;
33      a2_a3_g2 = G2;
        a2_a1_g1 = G1;
35      a3_a1_g2 = G2;
        a3_a3_g1 = G1;
37  else
        a0_a0_g0 = A0(i-1);
39      a0_a2_g3 = A2(i-1) + G3;
        a1_a2_g0 = A2(i-1);
41      a1_a0_g3 = G3 + A0(i-1);
        a2_a3_g2 = A3(i-1) + G2;
43      a2_a1_g1 = A1(i-1) + G1;
        a3_a1_g2 = A1(i-1) + G2;
45      a3_a3_g1 = A3(i-1) + G1;
    end
47
    A0(i) = max(a0_a0_g0, a0_a2_g3);
49  A1(i) = max(a1_a2_g0, a1_a0_g3);
    A2(i) = max(a2_a3_g2, a2_a1_g1);
51  A3(i) = max(a3_a1_g2, a3_a3_g1);

53  A0_bin = int2complement(A0(i),8);
    A1_bin = int2complement(A1(i),8);
55  A2_bin = int2complement(A2(i),8);
    A3_bin = int2complement(A3(i),8);
57  A0(i) = complement2int(A0_bin);
    A1(i) = complement2int(A1_bin);
59  A2(i) = complement2int(A2_bin);
    A3(i) = complement2int(A3_bin);
61
end
63
end
```

A.2.5 Test-Bench

```
clc
clear
close all

Nval = 23;

[A0,A1,A2,A3] = acs4(Nval);

file_id= fopen('uscita.txt','w') ;

for i=1:5
fprintf(file_id,'%d\n',0);
fprintf(file_id,'%d\n',0);
fprintf(file_id,'%d\n',0);
fprintf(file_id,'%d\n',0);
fprintf(file_id,'\n');
end

for i=1:Nval-3
fprintf(file_id,'%d\n',A0(i));
fprintf(file_id,'%d\n',A1(i));
fprintf(file_id,'%d\n',A2(i));
fprintf(file_id,'%d\n',A3(i));
fprintf(file_id,'\n');
end

fprintf(file_id,'\n');
fprintf(file_id,'\n');

fclose(file_id);
```

A.3 C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 4

int main ()
{
    FILE *fptr, *fptr2, *fptr_o;
    char s1[N],s2[N];
    int riga=1, errore = 0;
```

```
12
    fptr = fopen("result.txt","r");
14  fptr2 = fopen("uscita.txt","r");
    fptr_o = fopen("esito.txt","w");
16
    if(fptr == NULL) {
18      fprintf(fptr_o, "Errore␣nell'apertura␣del␣file");
        exit(1);
20      }
    if(fptr2 == NULL){
22      fprintf(fptr_o, "Errore␣nell'apertura␣del␣file");
        exit(1);
24      }
    if(fptr_o == NULL){
26      fprintf(fptr_o, "Errore␣nell'apertura␣del␣file");
        exit(1);
28      }

30  while(fscanf(fptr,"%s",s1) != EOF){
        fscanf(fptr2,"%s",s2);
32      if (strcmp (s1,s2) != 0){
            fprintf(fptr_o, "Errore␣nella␣riga␣%d\n", riga);
34          errore = 1;
        }
36      riga++;
    }
38
    if(errore != 1){
40      fprintf(fptr_o, "Tutto␣corretto");
    }
42



44
    fclose(fptr);
46  fclose(fptr2);
    fclose(fptr_o);
48  return 0;
}
```