SIE 4094: ANALOG AND MIXED-SIGNAL DESIGN, SPECIALIZATION

PROGRAMMABLE ANALOG INTEGRATED CIRCUIT W/TABLE OF CONTENT

BY
CARSTEN WULFF

SUBJECT TEACHER: TROND YTTERDAL

TEACHING SUPERVISOR: TROND YTTERDAL

FACULTY OF ELECTRICAL ENGINEERING AND TELECOMMUNICATIONS
DEPARTEMENT OF PHYSICAL ELECTRONICS
NORVEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

PREFACE

This report describes the PAIC (Programmable Analog Integrated Circuit) project. The project has been carried out at Norwegian University of Science and Technology (NTNU), Department of Physical Electronics, Mixed signal design.

The project has been carried out by Carsten Wulff in a 5vt (5 point weighting) project, which amounts to 24 hours a week. The purpose of the project is stated below:

Programmable Analog Integrated Circuit w/Table of content

Field programmable gate arrays (FPGA) are becoming an alternative to application specific circuits (ASIC) for some applications. FPGA are heavily used in education because of their great flexibility. We wish to develop a similar technology for analog circuits, which will be used in an Internet laboratory. The Internet laboratory is under development at the department of physical electronics.

The assignment is to develop a concept for programmable analog integrated circuit, which contains information about the cells in the circuit and how these cells must be connected to perform a specific function. The concept is to be verified by defining a specific circuit, which is to be modeled and simulated on a functional level.

I would like to thank my teaching supervisor, Trond Ytterdal, support during the project, for always being available for questions and for getting the idea for PAIC.

Carsten Wulff, carsten@wulff.no, November 2001

ABSTRACT

A concept for programmable analog integrated circuits is presented. The simulations show that the PAIC can supply a user with information about its contents and how it should be programmed to perform a specific function. To verify the PAIC concept a test circuit is built from the PAIC's analog cells. PAIC is shown to be a viable design for a programmable analog integrated circuit, and a foundation for further work.

TABLE OF CONTENTS:

TABLE OF FIGURES:

TABLE OF TABLES:

1 INTRODUCTION

This chapter explains the motivation for the project and provide an introduction to programmable analog integrated circuits and the PAIC project.

1.1 MOTIVATION

When educating analog designers, there is a need for real world experience on integrated circuits. The ability to enable a student to make real measurements on analog circuits provides a large advantage over simulations. It provides an intuitive understanding of how analog circuits i.e. operational transconductors work. The test equipment for analog integrated circuits is often expensive, and to equip a lab to serve 30 students is impossible for most universities. The ability to remotely conduct experiments has been explored [1-5]. Already there are Internet labs that enable a student to make measurements on integrated circuits over the Internet. These labs often have a limitation on what types of integrated circuits the student has access to. Some labs [15] have large switching matrixes so the student can select a circuit measure, others have a single IC with some tunable parameters [16, 17]. The PAIC project takes a different approach to solve this limitation. Instead of using expensive switching matrixes, it aims to provide a lab with circuit programmability.

1.2 PROGRAMMABLE ANALOG CIRCUITS

The concept of a programmable analog circuit is to have an integrated circuit with "standard" cells, which can be wired into an analog circuit i.e. a filter or an amplifier. Figure 1 show a very simple example of a programmable analog circuit. By controlling a routing network, which can connect the analog cells to each other, we can "build" analog circuits. This routing network is sometimes called "signal highway".

Programmable analog devices have been reported for the last decade. The earliest reference at IEEE is from 1991 [6]. Several manufactures have made programmable analog circuits, among these are Motorola, IPM Inc, Lattice [13] and Anadigm [14]. Several designs of Field Programmable Analog Arrays (FPAA) have been reported [6,9,10,11,13,14], but these are often aimed at a commercial market as an analog counterpart to FPGA for rapid prototyping of analog circuits. The marked for these FPAA have not gained the same momentum as FPGA, this because of the much greater challenges involved in creating a programmable analog integrated circuit. One of the main challenges in creating FPAA is the fact that analog circuits do not have a smallest common denominator. Digital circuits can (in theory) be created from NAND gates no matter the complexity of the circuit. One approach to circumvent this obstacle is to create expert cells [10,11], where each analog cell has a set of tunable parameters i.e. a filter with tunable cut-off frequency. These expert cells are designed by analog designers and are guaranteed to operate within specification no matter how they are connected to other cells. It is a modification of this approach that PAIC has taken.

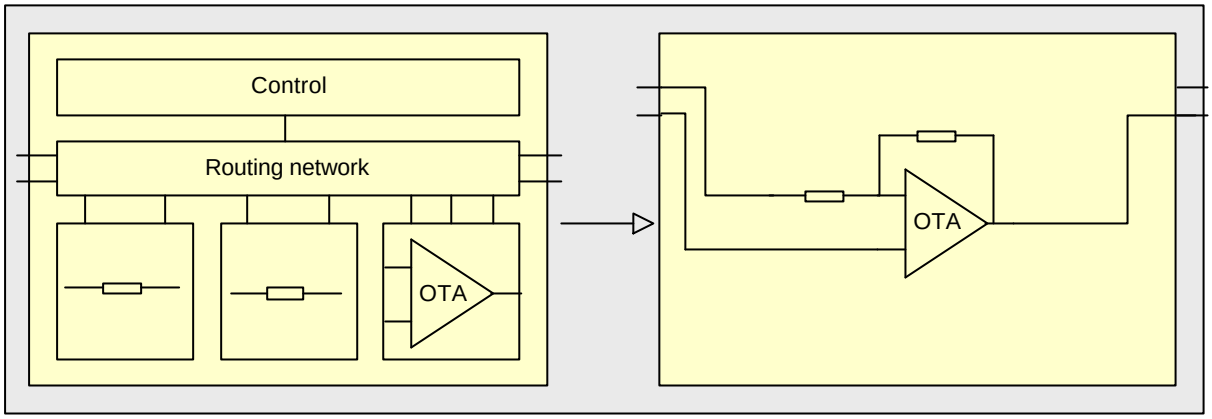

Figure 1 Programmable analog circuit concept

1.3 MODELLING LANGUAGE

The hardware modelling languages (HDL) available to the PAIC project were VHDL, Verilog, SPICE and SystemC. VHDL and Verilog are traditionally digital modelling languages, although there is a possibility to model simple analog circuits. SPICE is a simulation language for analog circuits, which can model digital circuits very accurately, but simulation time can be in the order of days (or worse) for digital circuits. SystemC is a system level modelling language that provides great flexibility since it is built as an extension to C++. Of these languages SystemC was chosen for the PAIC for two reasons: its status as an emerging new technology with great momentum and unsurpassed flexibility.

1.4 TABLE OF CONTENT (TOC)

The vision is that PAIC will be the first in a long line of programmable analog integrated circuits, each year adding to the test pool at NTNU. Figure 2 shows how such a test pool might be organized. Several PAICs are mounted on a printed circuit board (PCB) sharing all digital signals except the enable signal. A micro-controller selects which PAIC to program and programs it according to the instructions given by the PC. The PC is connected to a web server that provides a interface to client PC's over the internet. The users will be able to "draw" a representation of the circuit they want to test. To make this possible the web server has to have information on which PAIC contains the analog cell the user wants, and if the circuit the user has drawn is possible to route using the available PAICs. The TOC provides the outside world with information on what analog cells the PAIC contains. The web server can ask the micro-controller (through the PC) to retrieve a list of the analog cells and their respective addresses. It can store this information locally on disc or other storage. When a new PAIC is inserted into the test pool the software retrieves the new list of analog cells. It can thus provide the user with an almost unlimited number of different tests to run. The content of the TOC is an 8-bit number, which defines the cell type, and an 8-bit sub number that defines a specialization of the cell type. The web server must thus contain a look-up table where a complete definition of the analog cell resides.

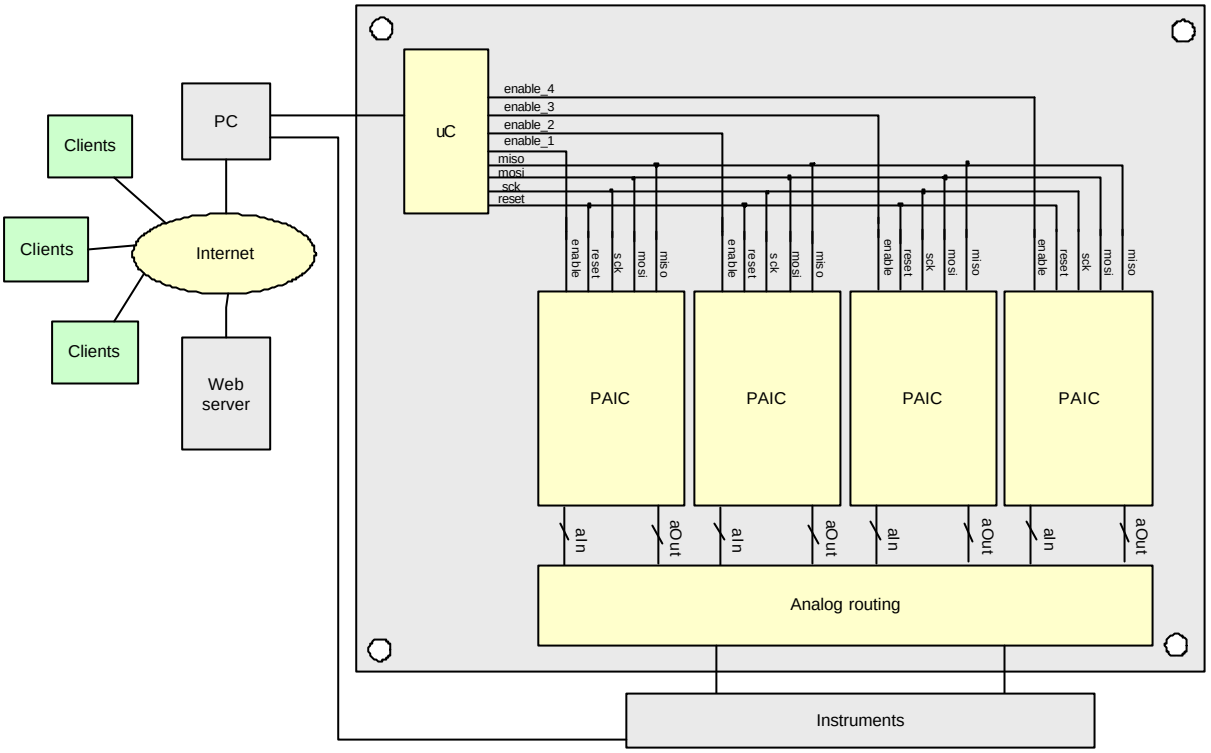

Figure 2 Analog test pool with PAIC foundation

1.5 READBACK

As mentioned in the previous chapter the web server needs information on how to connect the analog cells in the way a user wants. To do this it needs to know what connections are possible and which connections are impossible. The PAIC provides a function that allows the micro-controller to read the physical connections between the analog cells. Normally the analog cell will be connected through an output register & switch (ORS) to an input register & switch (IRS) of another analog cell in the PAIC. A simple algorithm in the micro-controller can read this connection.

Pseudo code for reading back physical connections:

```
reset PAIC

foreach module{
   foreach ORS{
       Set output low
   }
}

foreach module{
   foreach IRS{
       foreach module{
          foreach ORS{
              Set ouput high
              Read from IRS
              if(data from IRS conatins a one){
                store output
              }
              Set output low
          }
       }
   }
}
```

This algorithm is time consuming because it has to iterate through each IRS and ORS on the chip, but with minimal programming in the micro-controller it can provide the web server with information on all connections inside the PAIC. After reading the list the web server can store the information locally for faster access.

1.6 ANALOG CELLS

The analog cells in the PAIC will be designed by 4th year students in the course Analog CMOS 2, spring 2002. The analog cells are listed in Table 1. The first column is the abbreviation used for the different cells. Coloumn 2 gives the number of cells of a type. Coloumn 3 is the PAIC address for the cell (hexadecimal). The digital to analog converter (DAC) and the differential comparator were the only analog cells that were modeled with functionality. The other cells were modeled with an interface so they could be connected inside the PAIC in the correct manner. The DAC and the differential comparator were modeled as ideal circuits.

Cell	Nr	Module Address	Description
DAC	2	0x0,0xD	8-bit DAC
DiffOTA	1	0x1	Differential Operational Amplifier
DiffComp	1	0x2	Differential Comparator
Bandgap	1	0x3	Low voltage bandgap reference
ResArray	4	0x5,0x6,0x7,0x8	Resistor array
CapArray	4	0x9,0xA,0xB,0xC	Capasitor array

Table 1 Analog cells in the PAIC design

1.7 DEFENITIONS

"Analog cell" is used to describe the analog blocks i.e. the differential comparator or the DAC. An "analog module" is the analog cell plus a framework that, which we will see, consists of several blocks that provide control and input/output switching for the analog cell. "Analog framework" is the "analog module" without the "analog cell".

2 SPECIFICATION

The goal of the PAIC project is to create a concept for a framework that can provide analog cells with circuit programmability. In addition to providing an effective routing network, the PAIC must be simple to use. Because of a short timeframe and limited man-hours, the complexity of the PAIC must be kept within reasonable limits. As mentioned, PAIC must be able to supply the user with information on the analog cells and the routing network with minimal external decoding. It must be possible to interface the PAIC with a PC through a micro-controller. It should also be designed with a routing network that does not introduce noise, because of its use in laboratory measurements.

For verifying the PAIC concept, an 8-bit successive approximation analog to digital converter (ADC) was chosen as the test circuit. The ADC is built up of one digital to analog converter (DAC), one differential comparator and a successive approximation register modeled in software. The software routine performs a binary search for the correct digital word. A flowchart for the successive approximation ADC architecture is provided in Figure 4. A block diagram of this ADC is shown in Figure 3.

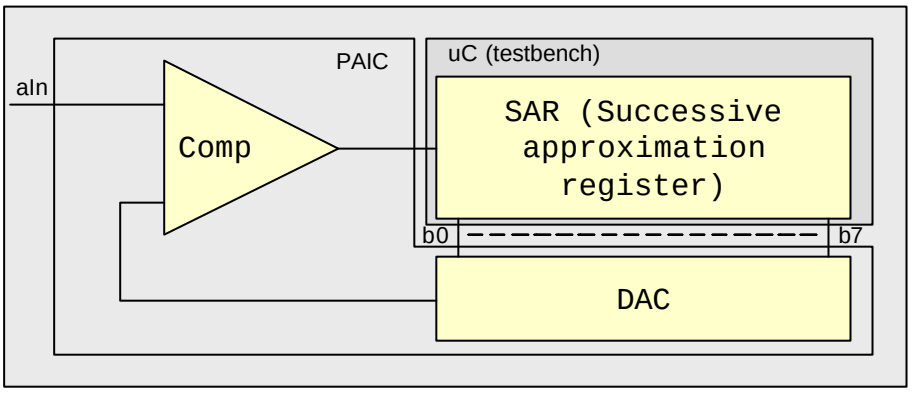

Figure 3 ADC block diagram

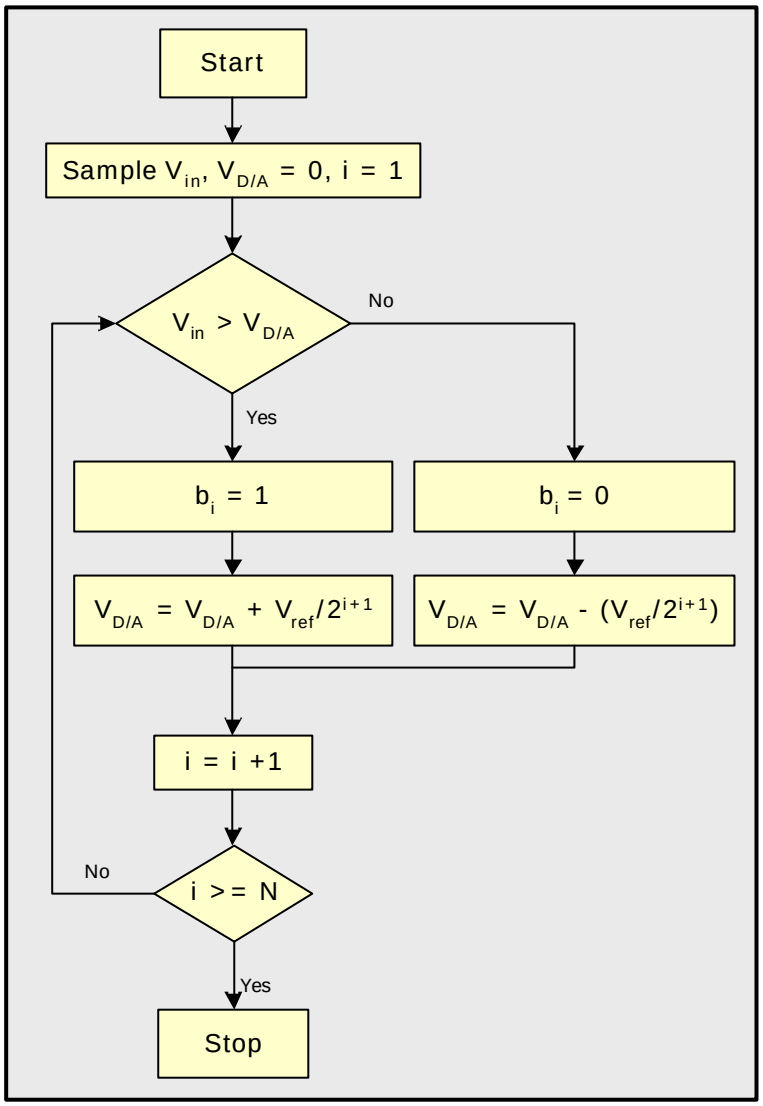

Figure 4 Successive approximation ADC flowchart

3 DESIGN

This chapter will explain the system level design of the PAIC and how the final concept came to life.

3.1 INTERFACE

When choosing the data interface for PAIC there were two key considerations: it should be simple to use and have minimal impact on the number of pins. The second consideration resulted in choosing a serial interface. There are a number of serial interfaces available, i.e. Universal Asynchronous Receiver and Transmitter (UART) or Serial Peripheral Interface (SPI). Both UART and SPI are widely supported in micro-controllers on the market, but SPI simpler to implement. Therefore SPI was chosen as the PAIC data interface. An explanation of the SPI interface is given in Appendix I.

3.2 VERSION 1

The first version of the PAIC routing network is based on the assumption that there will be a limited number of nodes in an analog circuit. Therefore, there is no need to provide all possible connections of analog cells. It is also based on the assumption that the analog cells will have different number of ports, and thus it does not distinguish between inputs and outputs. An overview is provided in Figure 5, we can see the control and routing blocks from Figure 1. An example of a signal highway is provided in Figure 6.

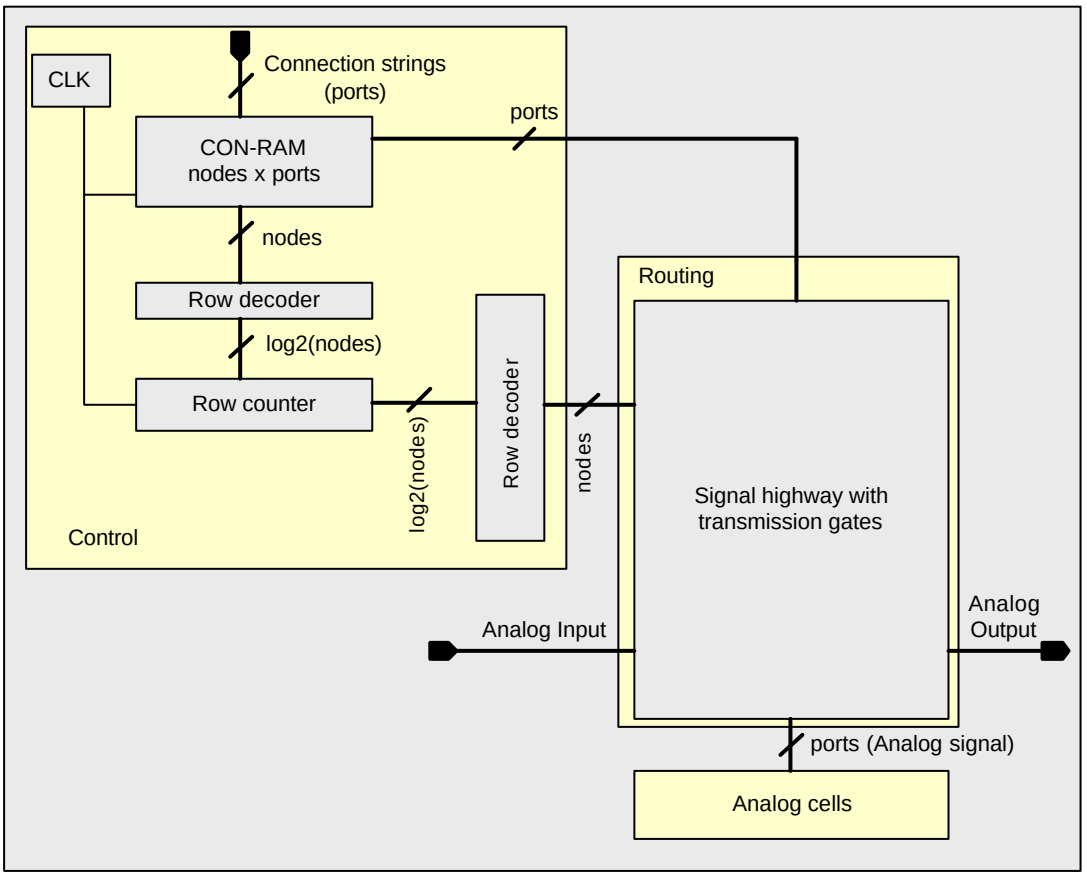

Figure 5. PAIC version 1

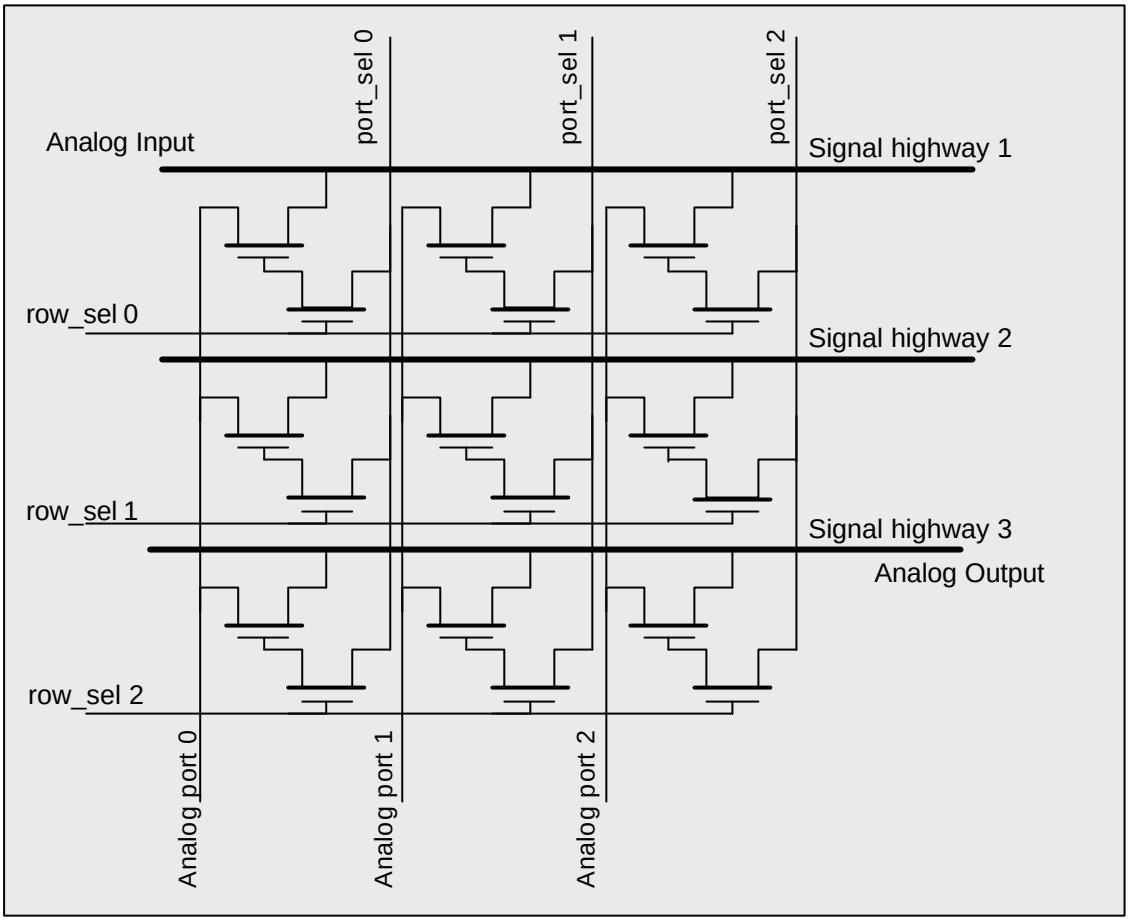

Figure 6. Signal highway example

Each of the analog ports (inputs and outputs of the analog cells) are connected to transmission gates which in turn are connected to the signal highway. Controlling the transmission gate is a second transmission gate with connections to the row decoder and the connection RAM. Cycling the RAM and the signal highway simultaneously updates the signal highway connections. The RAM size is given by the number of ports times the number of nodes (signal highways). In a theoretical circuit this could amount to 16x30 bit RAM, this is based on 10 analog cells with three ports each and a maximum of 16 nodes in a circuit. This results in 480 bits to reprogram the circuit. An example of a circuit is given in Figure 7, it uses 4 nodes and 8 ports giving a total 32 bits to reprogram the circuit. Figure 8 shows the equivalent circuit of Figure 7. The advantage of version 1 is high routing capability, only limited by number of nodes in the analog circuit. The disadvantages are: Complex control of routing network, no readback support, introduces noise because of frequent transitions close to signal highway (row_select transmission gates), analog cell output load dependent on routing network and a need for on-chip RAM

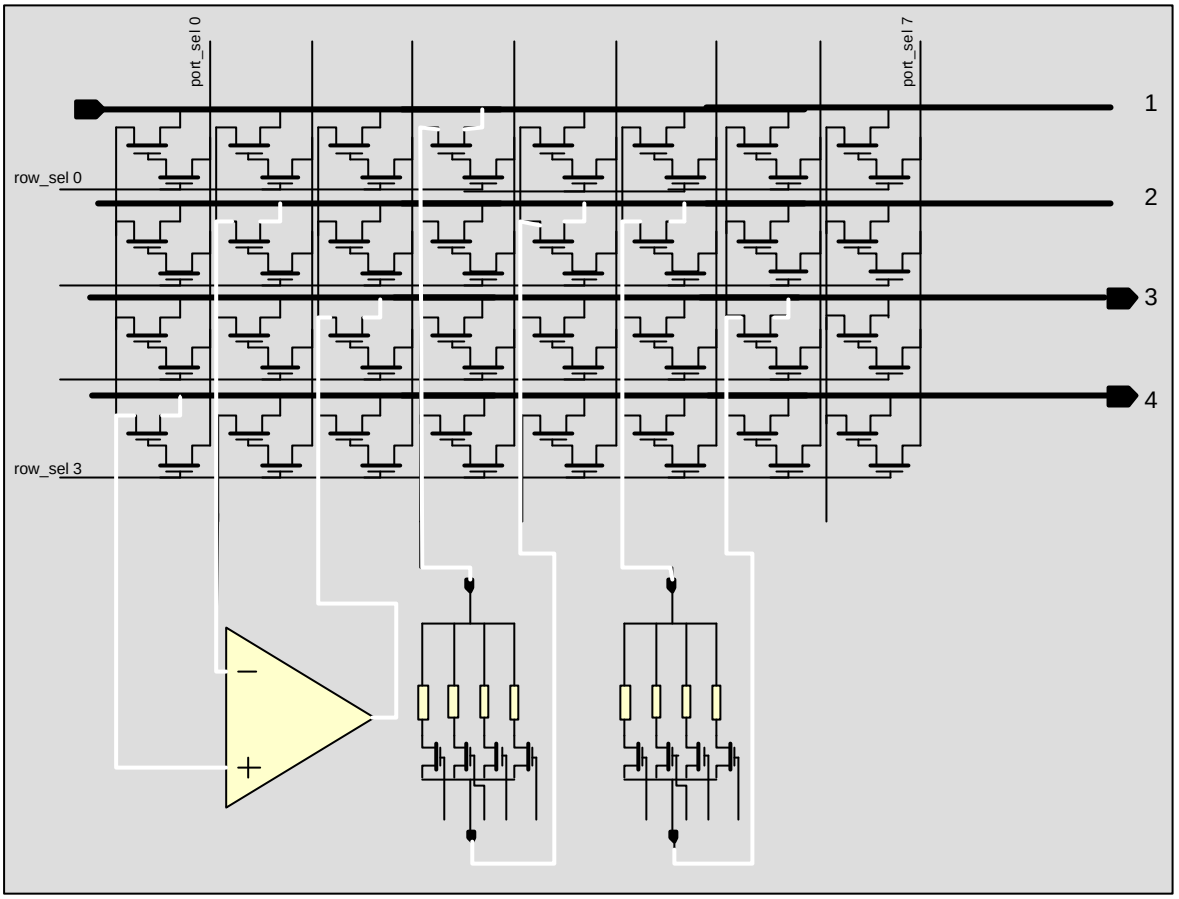

Figure 7. Example circuit

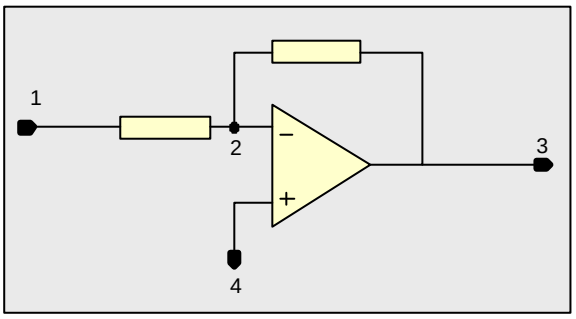

Figure 8. Equivalent circuit

For the connection in Figure 7 the RAM would contain the following bits:

	P0	P1	P2	P3	P4	P5	P6	P7
R0	0	0	0	1	0	0	0	0
R1	0	1	0	0	1	1	0	0
R2	0	0	1	0	0	0	1	0
R3	1	0	0	0	0	0	0	0

Table 2 RAM content for Figure 7

3.3 VERSION 2

Instead of using a single large signal highway it was decided to use one smaller signal highway for each input. This reduces the number of possible connections but it also reduces the complexity of the control circuitry. A block diagram of version 2 is shown in Figure 10. Version 2 uses a string (14 bits) for each input, which is loaded into a shift register. This string is from now on called a module packet. The module packet consists of three main fields. The first 4 bits contain the module address, the next 2 bits contain the line address and the last 8 bits contain the highway payload. The highway payload loaded into the input register & switch (IRS) that controls transmission gates, which in turn controls which analog signal should be connected to the respective output of the IRS as pictured in Figure 9. Both the module address and the line address increase with log2(n), hence the number of analog cells and number of input/outputs has little effect on the length of the module packet. The highway payload, on the other hand, has significant impact on the length of the module packet.

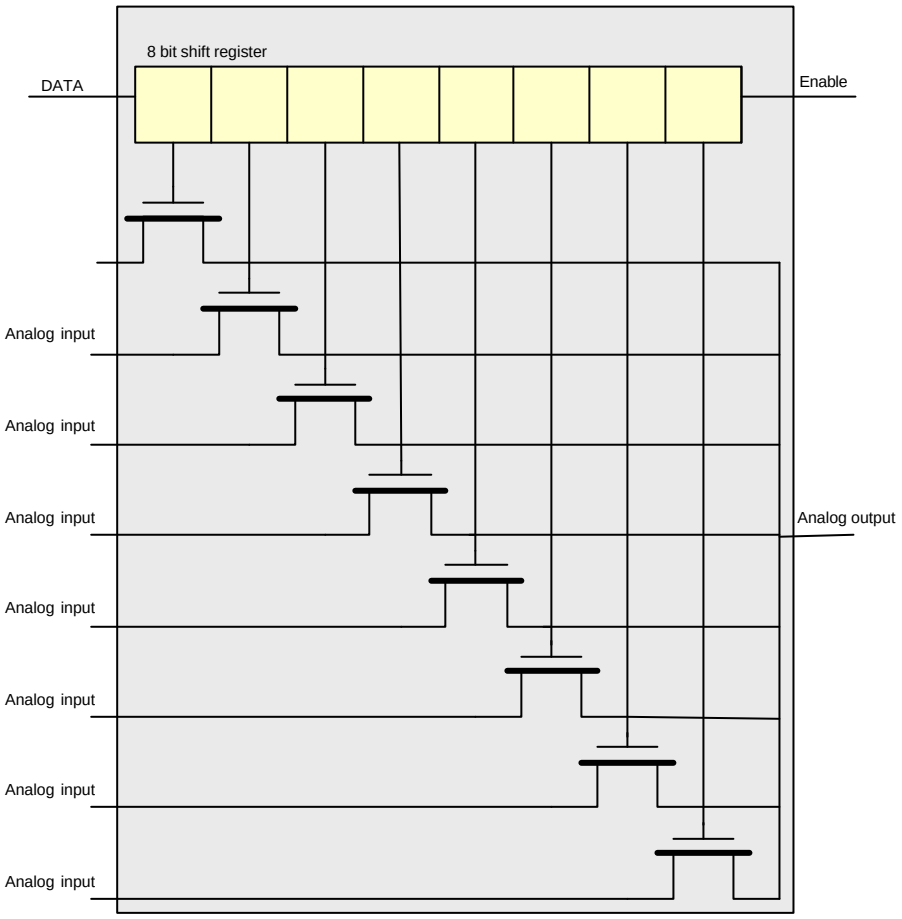

Figure 9. Input register & switch

The signal highway in version 2 limits the complexity of the circuits that can be connected. With one IRS per analog cell there is an 8 input limit. One way to circumvent this problem is to use two or more IRS per analog cell. This does not increase the highway payload (HWP) but will increase the line address since the analog framework will contain

more than 4 IRS. If 3 bits (8 IRS) where used for the line address the number of possible connections would double (as would the number of clock cycles needed to program the registers). In mathematical terms, n bit increase will give 2^(2+n)*(HWP length) possible connections where as an increase of highway payload will give 2^(2)*(HWP length +HWP length increase). The input to PAIC is a small control circuitry with a SPI interface to the outside world, it controls loading and reading back from the main register and controls a counter which iterates through the highway payload. The highway payload is shifted into the IRS via the counter/mux connection. The module address is connected to a decoder that controls the various framework register enable signals. Version 2 contains an output switch for routing the output signal off chip. The output switch contains a buffer to supply the analog cell with a constant load and to drive the output signals. Version 2 has one global analog input and one global analog output. The advantages of version 2 are: Reduced control complexity from version 1, no need for on-chip RAM and less noise introduced since analog signals can be better separated from digital logic. The disadvantages are: No readback support, limited input signals to analog cells, few global input/output signals, some noise from the digital portions will still be introduced, longer programming time than version 1.

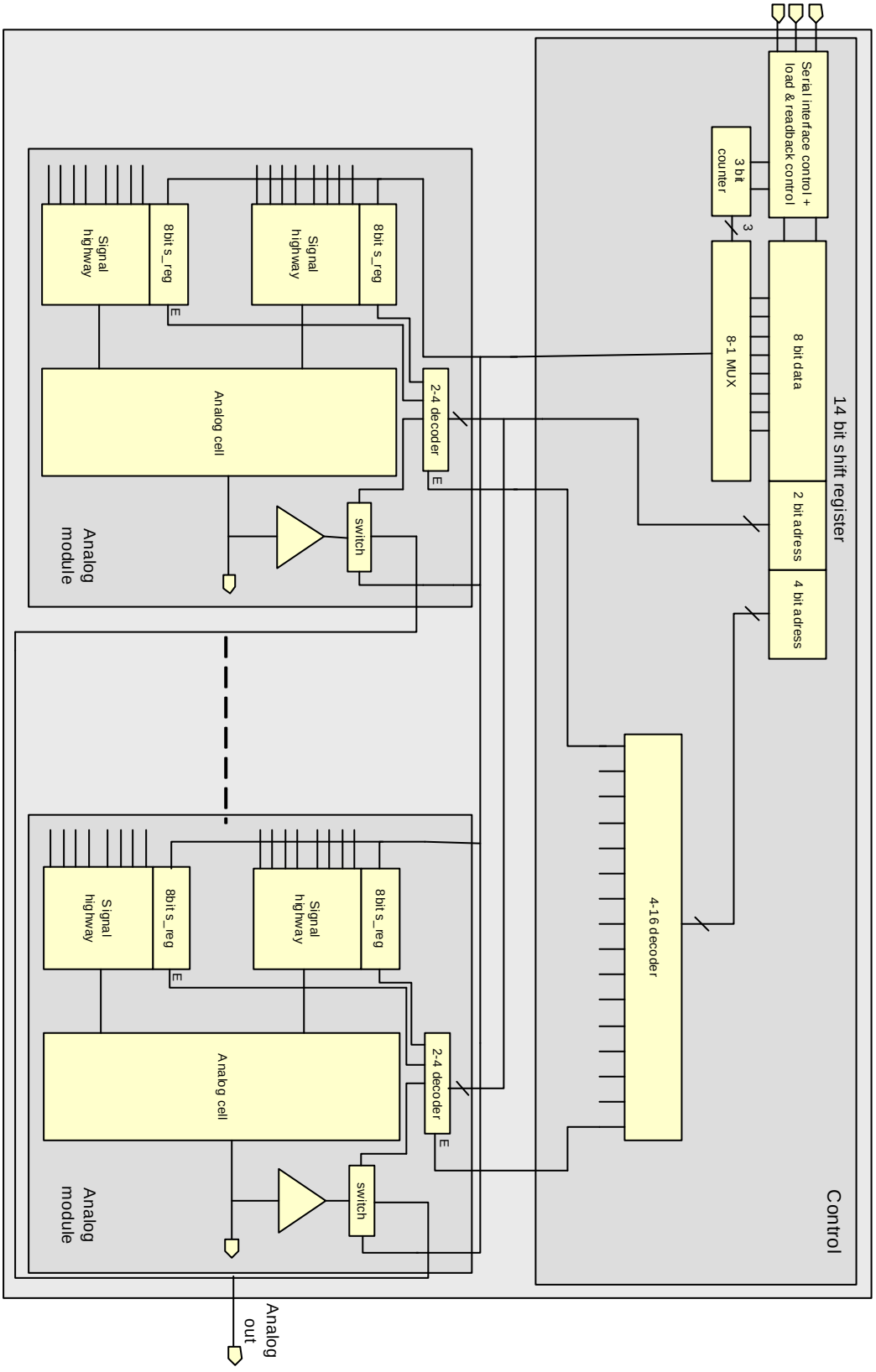

Figure 10 Version 2 block diagram

3.4 VERSION 2 VS VERSION 1

An advantage of version 2 over version 1 is the reduction of noise from the digital modules. In version 2 the analog signal paths can be better separated from noisy digital paths and the input control can open the clock line to the framework registers so the digital circuits close to the analog module are in sleep mode. In version 1 this is not possible since the RAM must be cycled to update the routing network. The greatest advantage of version 2 vs version 1 is the reduction of digital complexity, as always a reduction comes with a cost, but in the version 2 the cost is programming time that can be alleviated by a higher clock rate. Version 2 was chosen as a basis for further research

3.5 VERSION 3

Version 3 is a modification of version 2 to make it a viable architecture. It has increased the line address to 3 bits and the framework around each analog cell now consists of; a line decoder, five IRS, one internal module register (ModReg) and two output register & switch (ORS). Figure 11 shows an overview of version 3. The IRS is the same as in version 2 with some modification to allow reading the routing network. The module register serves as a control register or a digital input to the analog cell. The ORS is a modification of the output switch in version 2. It can switch the output signal from the analog cell to one of four global outputs and it is modified to allow reading of the routing network. The main register was removed and replaced by an address register (AddrReg). Instead of the shift registers in version 2 all registers in version 3 are parallel load, except a register in the SPI block. Addressing the modules is done through a module address and a line address; both are stored in the 8-bit address register. The module address is connected to a 4-16 decoder (ModDec) that provides an enable signal for the line decoders and the line address is connected to the line decoder (LineDec) that provides an enable signal for IRS, ORS and a module register. An 8-bit bi-directional bus is used for data transfer to and from the registers. Programming and controlling version 3 is done through a SPI for data input/output and a 3-bit bus for control signals. Version 3 also has a table of content, which contains a definition of the analog cells. The master clock was removed from version 3 [12], and the architecture was made semi-asynchronous. Semi-asynchronous involves the use of synchronous registers but not a master clock. The clock signals for the synchronous registers are supplied from the control logic. A thorough explanation of the design will be given later under the implementation section. The advantages of version 3 are: readback support included, more input signals to analog cells than version 2 (line address increase), four global input/output signals, less introduced noise due to no master clock, reduced programming time through the introduction of a global reset signal (no need to program analog modules that will not be used) and an 8-bit digital input to analog cells.

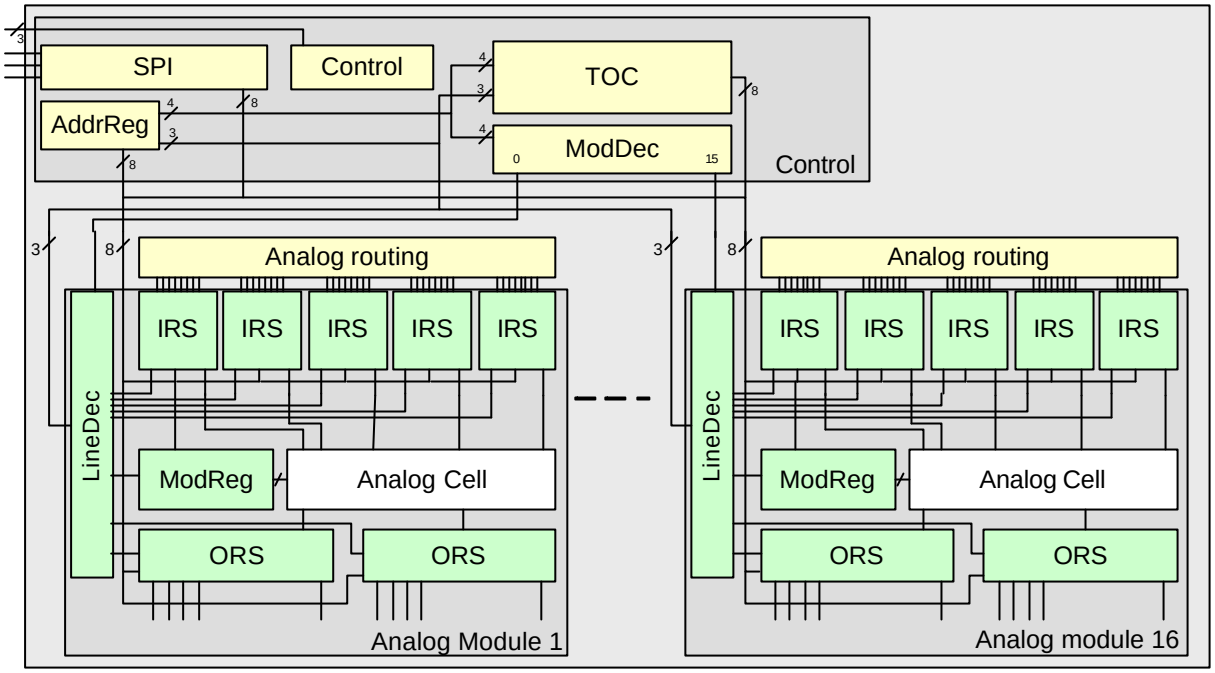

Figure 11 Overview of Version 3

3.6 VERSION 3 VS VERSION 2

Most of the modifications in version 3 were made to make version 2 into a viable architecture i.e. modification of the IRS and ORS to support readback of the routing network. The removal of the master clock signal gives version 3 better noise performance than version 2. Without the master clock there will be no transitions in the digital logic unless data is written to the PAIC, therefore the digital logic will be more quiet than with a master clock.

4 IMPLEMENTATION

This chapter will explain the SystemC implementation of the PAIC design, it is a detailed explanation of version 3.

4.1 TOP LEVEL

The top level consists of a SPI, control logic, address register, the TOC and the PAIC_AnalogModules (PAM). Figure 12 shows the top-level design. Table 3 describes the PAIC interface. Types in the interface table are SystemC/C++ types.

Port	Direction	Type	Description
reset	in	bool	Global reset signal
enable	in	bool	Global enable signal
miso	out	bool	SPI, master in slave out
mosi	in	bool	SPI, master out slave in
sck	in	bool	SPI, master driven clock
Cntr(0,2)	in	sc_uint<3>	Control signals
aIn(0,3)	inout	voltage	Global analog inputs
aOut(0,3)	inout	voltage	Global analog outputs

Table 3 PAIC Interface

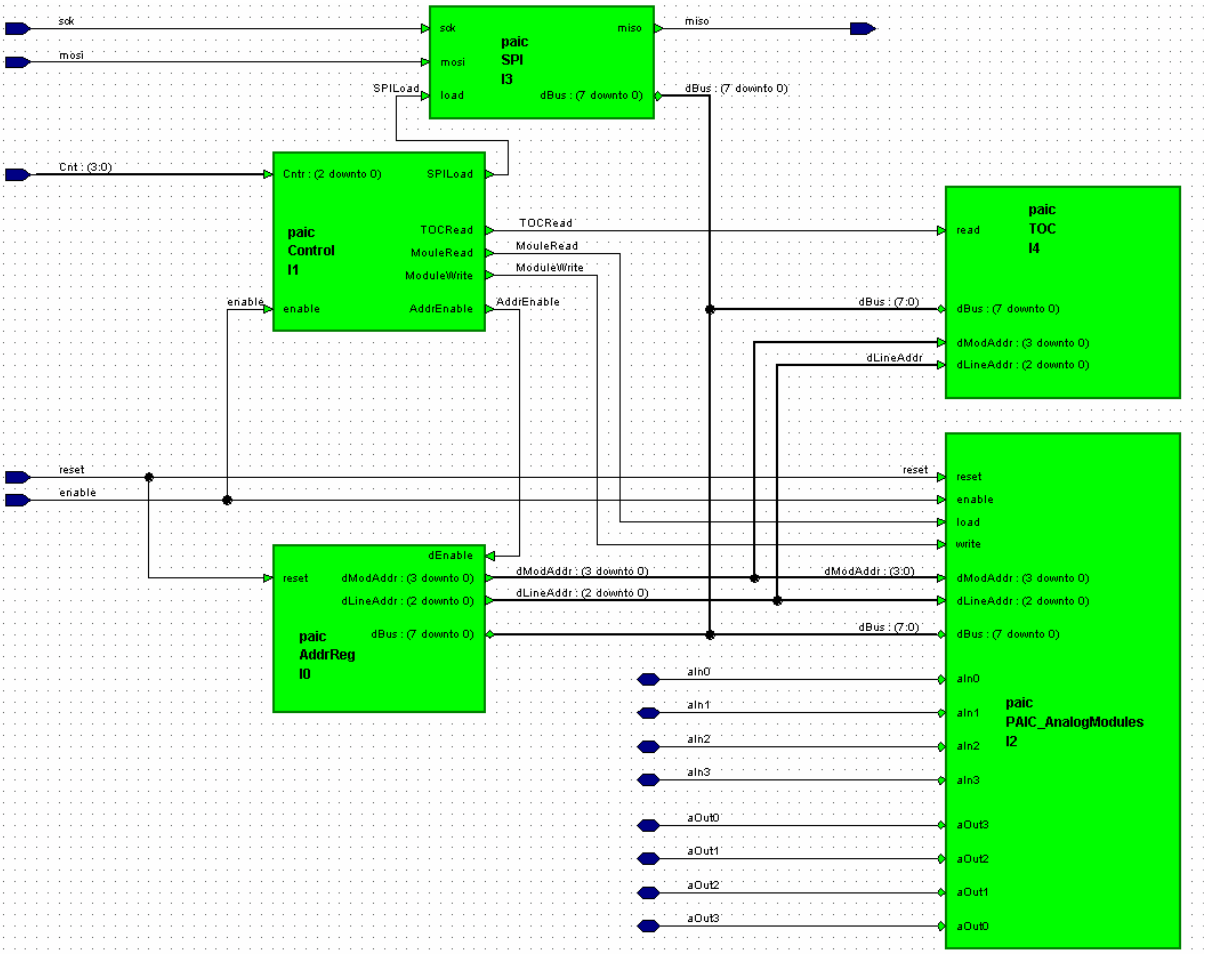

Figure 12 PAIC top-level

4.1.1 SPI

The SPI in the PAIC is an 8-bit shift-register with parallel load. A 0-1 transition on the `sck` writes the most significant bit to the `miso` signal, the shift register shifts one position, the least significant bit is read from the `mosi` signal and the `dBus` is updated. A positive transition on `load` reads the current value of the `dBus` and writes the shift register in parallel. SPI interface is given in Table 4.

Port	Direction	Type	Description
sck	in	bool	clock for the spi register
miso	out	bool	master in slave out
mosi	in	bool	master out slave in
reset	in	bool	resets SPI register
dBus	inout	sc_lv<8>	Bidirectional 8-bit bus
load	in	bool	Loads the value of dBus into the SPI register

Table 4 SPI Interface

4.1.2 CONTROL LOGIC (CONTROL)

The PAIC control is a 3-8 decoder. The enable signal is connected to the global enable signal and thus the control logic will always provide an idle state if enable is low. Control signals and interface is summarized in Table 5.

Port	Direction	Type	Description
Cntr	in	sc_lv<3>	input control signals
enable	in	bool	enables the control decoder
dCntOut[8]	out	bool	output control signals
Cntr	Control signal		
000	None (dCntOut[0])		
001	AddrWrite (dCntOut[1])		
010	ModuleWrite (dCntOut[2])		
011	ModuleReadBack (dCntOut[3])		
100	TOCRead (dCntOut[4])		
101	SPILoad (dCntOut[5])		
110	Not used (dCntOut[6])		
111	Not used (dCntOut[7])		

Table 5 Control interface and Cntr mapping

4.1.3 ADDRESS REGISTER (ADDRREG)

The address register holds the module and line address. It is an 8-bit synchronous register with `dEnable` connected to the clock input. The three least significant bits are connected to the `dLineAddr` and bits 3 to 6 connected to `dModAddr` (bit 7 is not used). A positive transition on `dEnable` will load the register with the value on `dBus`. Setting `reset` signal high will reset the register to 0x00.

Port	Direction	Type	Description
reset	in	bool	reset register
dEnable	in	bool	enables address register
dIn	inout	sc_lv<8>	address register input
dLineAddr	out	sc_uint<3>	line address out
dModAddr	out	sc_uint<4>	module address out

Table 6 AddrReg interface

4.1.4 TABLE OF CONTENT (TOC)

The table of content is a read only memory (ROM) with a 16 x (2 x 8) structure, in other words the `dModAddr` selects a (2 x 8) ROM and `dLineAddr` selects the byte to be read. This way there is no need for an internal address register in the ROM, it can use the existing address register. The TOC stores the cell number at line address 0 and the sub number at line address 1. It is possible to expand the ROM up to a 16 x (8 x 8) structure if more data is needed to specify an analog cell.

Port	Direction	Type	Description
read	in	bool	writes register value to dBus
dLineAddr	in	sc_uint<3>	line address
dModAddr	in	sc_uin<4>	module address
dBus	inout	sc_lv<8>	ROM out

Table 7 TOC Interface

4.1.5 PAIC_ANALOGMODULES :

PAIC_AnalogModules (PAM) serves as the connector of all the analog modules, thus the routing network of PAM is quite complex. In addition to the analog modules the PAM contains a module decoder for decoding the module address. The interface for the PAM and the module decoder is provided in Table 8 and Table 9 respectively. A diagram for the PAM is not included in this report because of its complexity, but is included on the PAIC CD-ROM.

Port	Direction	Type	Description
reset	in	bool	reset for AnModFramework
enable	in	bool	enable signal for the ModDec
load	in	bool	load signal for AnModFramework, connected to ModuleRead
write	in	bool	write signal for the AnModFramework, connected to ModuleWrite
dModAddr	in	sc_uint<4>	Module address, connected to the ModDec
dLineAddr	in	sc_uint<3>	Line address, connected to the AnModFramework
aPAICIn[4]	inout	voltage	Global input signals
aPAICOut[4]	inout	voltage	Global output signals

Table 8 PAIC_AnalogModules interface

Port	Direction	Type	Description
dModAddr	in	sc_uint<4>	Module Address
dModDecOut[16]	out	bool	Enable signals for AnModFramework
dEnable	in	bool	Enable signal

Table 9 ModDec Interface

To avoid manually instantiating each analog framework and connecting it to the analog cell, PAM relies on polymorphism (objects ability to conserve the specialization of a child object). A generic analog cell was created in SystemC which all analog cells inherit, the module definition is repeated below.

```
struct GenericCell : sc_module{

	sc_inout<voltage> aModOut0;
	sc_inout<voltage> aModOut1;

	sc_inout<voltage> aModIn0;
	sc_inout<voltage> aModIn1;
	sc_inout<voltage> aModIn2;
	sc_inout<voltage> aModIn3;
	sc_inout<voltage> aModIn4;
	sc_in<voltage> aVdd;
	sc_in<voltage> aVss;

	sc_in<sc_uint<8> > dModIn;
	sc_in<bool> reset;
};
```

The generic cell definition contains the ports needed to connect it to the analog framework. This way the programming load of connecting all analog cells and modules greatly decreases. The analog framework, which will be explained in detail later, has an instance of this object and when an analog framework is instantiated it is passed an analog cell casted to the generic cell, thus avoiding the manual connection of the analog cell. An example of this process is shown below.

Example on instantiating an analog framework with a 8-bit DAC analog cell:

```
aMod[0] = new AnModFramework("aMod0");
aMod[0]->ConnectAnalogCell( new DAC_8("DAC_8"));
```

The ConnectAnalogCell method:

```
GenericCell *da;
.
.
//-----------------------------------------------------------------------
void ConnectAnalogCell(GenericCell *ac){
//-----------------------------------------------------------------------
       da  = ac;
       da->aModIn0(aModIn0);
       da->aModIn1(aModIn1);
       da->aModIn2(aModIn2);
       da->aModIn3(aModIn3);
       da->aModIn4(aModIn4);
       da->aModOut0(aModOut0);
       da->aModOut1(aModOut1);
       da->dModIn(dModIn);
       da->aVdd(s_aVdd);
       da->aVss(s_aVss);
       da->reset(reset);
}
```

4.2 ANALOG FRAMEWORK

The analog framework (AnModFramework) is the heart of the system and includes the logic necessary to allow flexible connection of the analog cell. A block diagram is shown in Figure 13. It contains one to five IRS (depending on how many is needed for the analog cell), two ORS, a line decoder and an 8-bit module register. A generic analog cell in Figure 13 represents how the respective analog cells will be connected. The interface for AnModFrameWork is shown in Table 10.

Port	Direction	Type	Description
dEnable	in	bool	enable signal from ModDec
dLineAddr	in	sc_uint<3>	Line address
dWrite	in	bool	write signal for IRS, ModReg and ORS
dLoad	in	bool	load signal for IRS
reset	in	bool	reset signal for IRS, ModReg and ORS
dBus	inout	sc_lv<8>	data bus for IRS, ModReg and ORS
aIn0[7]	inout	voltage	Analog input signal for IRS0
aIn1[7]	inout	voltage	Analog input signal for IRS1
aIn2[7]	inout	voltage	Analog input signal for IRS2
aIn3[7]	inout	voltage	Analog input signal for IRS3
aIn4[7]	inout	voltage	Analog input signal for IRS4
aOut0	inout	voltage	Analog output signal from ORS0
aOut1	inout	voltage	Analog output signal from ORS1
aGlobalOut[4]	inout	voltage	Global analog ouput signals from ORS0 and ORS1

Table 10 Analog framework interface

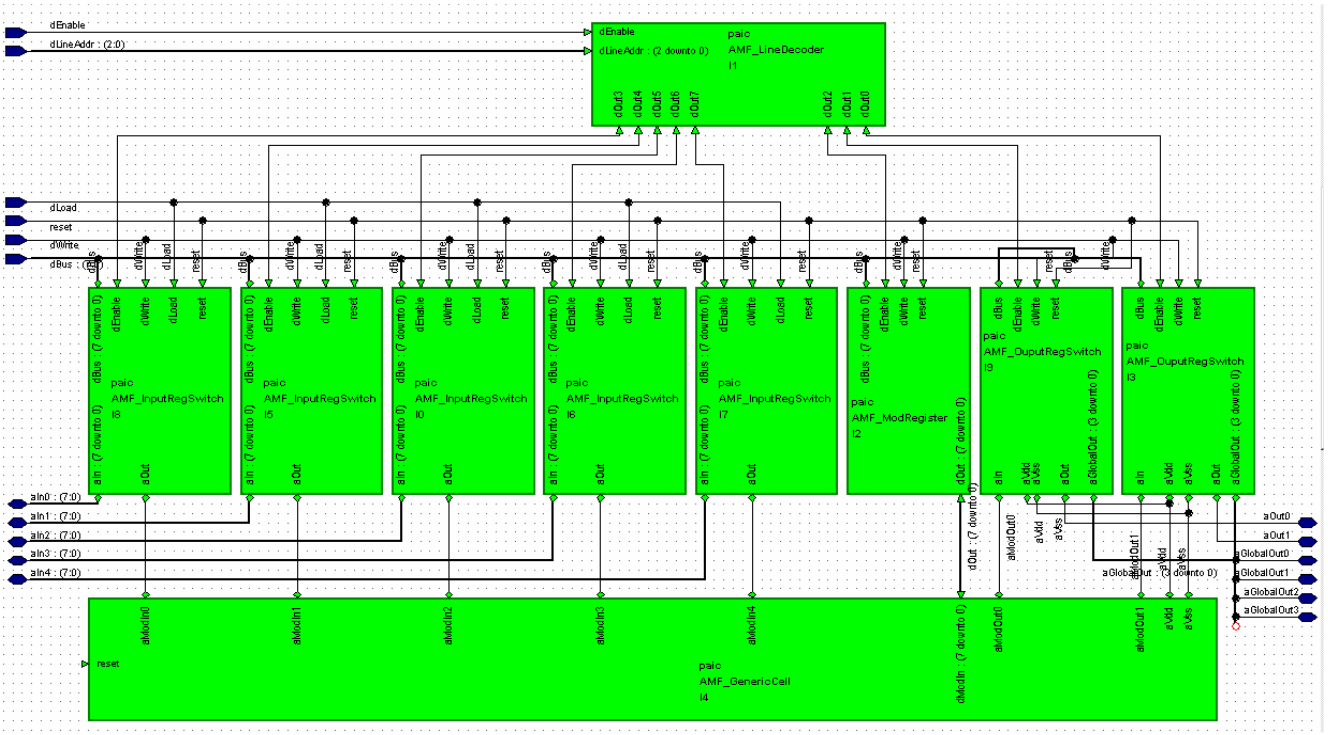

Figure 13 Analog Framework block diagram

4.2.1 LINE DECODER (LINEDEC)

The line decoder is a 3-8 decoder with enable, it selects which register in the framework to enable for reading or writing. The registers are selected according to Table 12. Interface of LineDec is shown in Table 11.

Port	Direction	Type	Description
dEnable	in	bool	enable signal
dLineAddr	in	bool	Line Address
DOUT[8]	out	bool	Output signals to registers

Table 11 LineDec interface

dLineAddr	Register enabled
000	ORS0
001	ORS1
010	ModReg
011	IRS0
100	IRS1
101	IRS2
110	IRS3
111	IRS4

Table 12 Line addresses

4.2.2 INPUT REGISTER & SWITCH (IRS)

The input register & switch has two functions, to switch analog signals according to an 8-bit word and to "transmit" the analog inputs over on the data bus during readback. The IRS consists of eight IRS cells (Figure 14). The IRS cell consists of a master-slave flip-flop, a CMOS transmission gate and control logic. The states of the IRS are summarized in Table 14. The interface of the IRS is shown in Table 13. The block diagram IRS is shown in Figure 15.

Port	Direction	Type	Description
reset	in	bool	reset signal
dBus	in	sc_lv<8>	data bus
dLoad	in	bool	writes the value of analog inputs to dBus
dWrite	in	bool	writes the value of dBus to the register
dEnable	in	bool	enables the register
aIn[8]	inout	voltage	analog input to the signal highway
aOut	inout	voltage	Ouput from the signal highway

Table 13 IRS interface

reset	dEnable	dWrite	dLoad	State
0	0	X	X	No change
0	1	1	0	*dOut* <= *dBus*
0	1	0	1	*dBus* <= Analog inputs
0	1	1	1	Not allowed
1	X	X	X	*dOut* <= 0x00

Table 14 State diagram for IRS

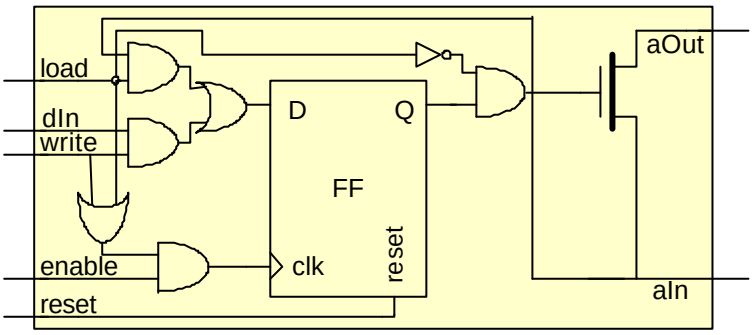

Figure 14 IRS cell

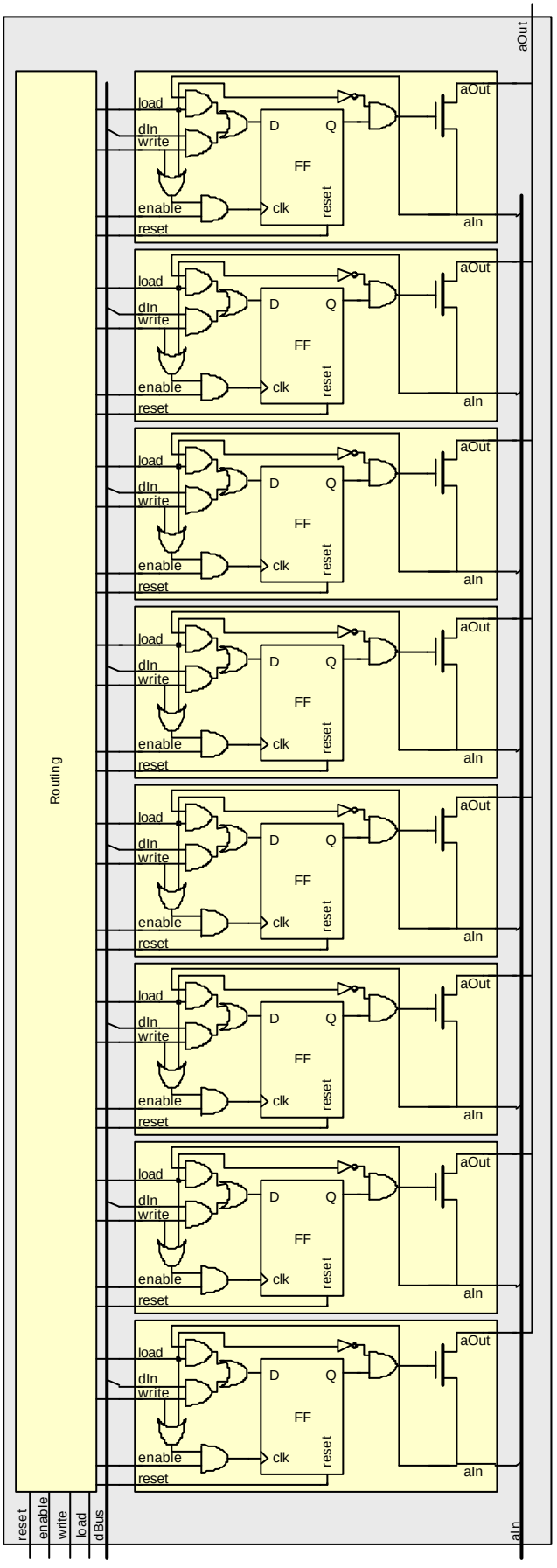

Figure 15 IRS block diagram

4.2.3 OUTPUT REGISTER & SWITCH (ORS)

The output switch has three purposes: To provide a constant load for the analog cell invariant of the connection made within the PAIC, switch the analog output to one of the global outputs if needed and provide digital high and low states during readback. A buffer provides a constant load to the analog cell at the same time providing enough power to drive the signal lines invariant of how the analog modules are connected. The ORS has seven states, summarized in Table 16. The interface is given in Table 15. ORS block diagram is shown in Figure 16.

Port	Direction	Type	Description
reset	in	bool	reset register
dEnable	in	bool	enable register, from ModDec
dWrite	in	bool	writes to register from dBus, connected to clock
dBus	inout	sc_lv<8>	data bus
Vdd	inout	voltage	For readback high feature
Vss	inout	voltage	For readback low feature
aIn	inout	voltage	analog cell output
aOut	inout	voltage	analog output
aGlobalOut[4]	inout	voltage	global analog output

Table 15 ORS interface

Register value	State	Description
000	Normal	*aOut <= aIn*
001	GlobalOut1	*aOut <= aIn* *aGlobalOut1 <= aIn*
010	GlobalOut2	*aOut <= aIn* *aGlobalOut2 <= aIn*
011	GlobalOut3	*aOut <= aIn* *aGlobalOut3 <= aIn*
100	GlobalOut4	*aOut <= aIn* *aGlobalOut4 <= aIn*
101	Readback_low	*aOut <= 0*
110	Readback_high	*aOut <= 1*

Table 16 ORS states

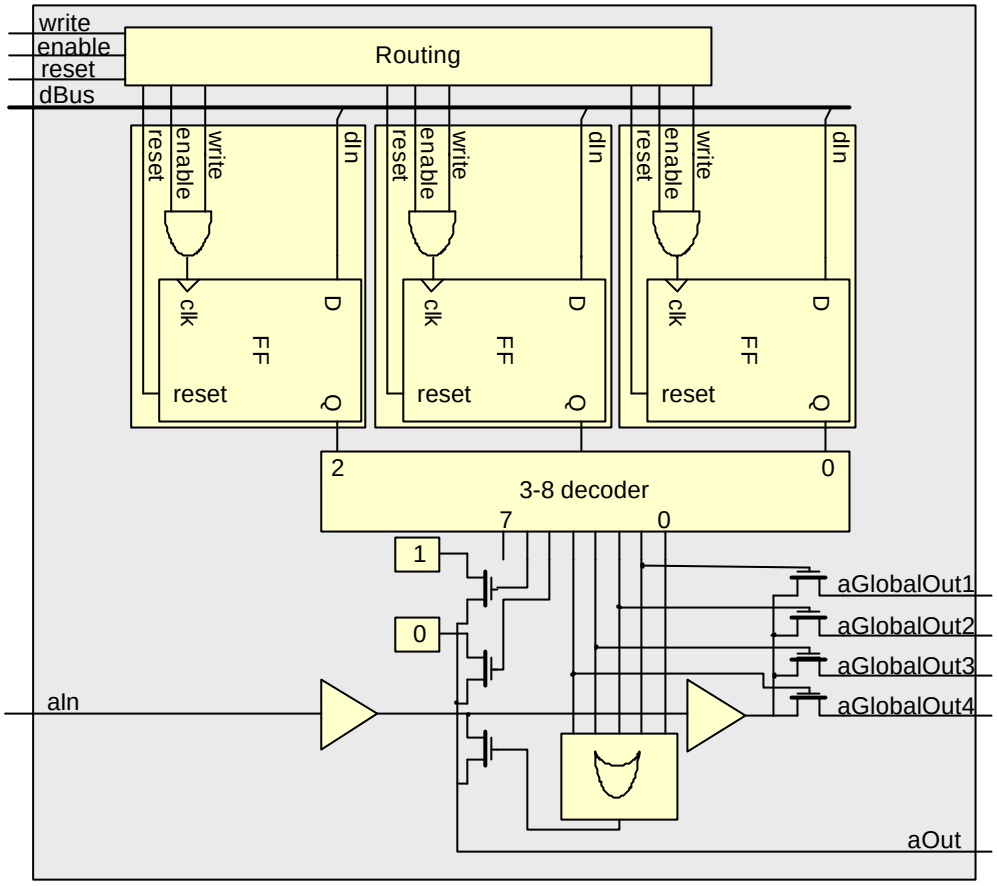

Figure 16 ORS block diagram

4.2.4 INTERNAL MODULE REGISTER (MODREG)

The internal module register is a synchronous 8-bit register with `clk = dWrite`. It functions as a digital input or control register for the analog cell. ModReg interface is shown in Table 17.

Port	Direction	Type	Description
reset	in	bool	reset register
dEnable	in	bool	enable register
dWrite	in	bool	writes value on dBus to regsiter, connected to clock
dBus	inout	sc_lv<8>	payload bus
dOut	out	sc_uint<8>	digital input for analog cell

Table 17 ModReg interface

5 SIMULATION RESULTS

This chapter will provide an introduction into how simulations were preformed and simulation results from four different simulations.

5.1 INTRODUCTION

PAIC was simulated using SystemC and the built-in trace function, the trace function writes the output as a `.vcd` file. The waveform viewer used to show the `.vcd` file was the freeware GKTWave. An example of creating a trace file is shown below.

```
trace_file = sc_create_vcd_trace_file("TestPAIC");  //Creates a trace file
sc_trace(trace_file,Cntr,"Cntr"); // Adds Cntr to the trace list
sc_trace(trace_file,miso,"miso"); // Adds mosi to the trace list
sc_trace(trace_file,mosi,"mosi"); // Adds miso to the trace list
sc_trace(trace_file,sck,"sck");   // Adds sck to the trace list
sc_trace(trace_file,paic->dBus,"paic_dBus"); // Adds dBus to the trace list
```

To introduce the PAIC simulation a simple example follows. Figure 17 shows an example of the output waveforms, in this simulation the SPI register is loaded with two 8-bit strings `"00010011"` and `"00000100"`. The control signals applied after loading the string are respectively `"001"` and `"010"`. We can see from Table 5 that `Cntr="001"` loads the value on the bus into the address register and that `Cntr="010"` loads the value on the bus into the register given by the module and line address. The module address is given by bits 3-6. From Table 1 we can see that this is the differential comparator. The line address `"011"` is IRS0 (Table 12). This results in the sixth analog input of IRS0 being connected to the first input on the differential comparator (Figure 18).

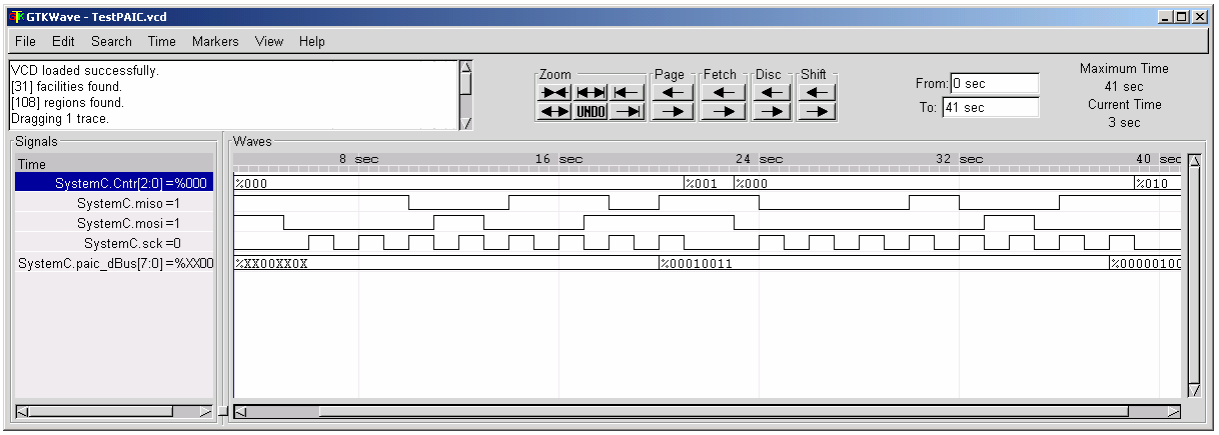

Figure 17 Example of output waveform

Testing of the different modules was done during modelling of the PAIC. Most of these simple tests involved checking that the module worked the way it was intended. These tests will not be reproduced here since they are implicit part of the later functional tests. We will start with an example on connecting the differential comparator to analog inputs and outputs to introduce the syntax of the testbenches and to explain how to use the SPI bus and control signals.

5.2 CONNECTION OF THE DIFFERENTIAL COMPARATOR

Connecting the DiffComp to analog inputs and outputs involve sending strings to the IRS and ORS registers. All analog modules in PAIC have the global inputs [1-4] connected to the first four inputs of the first IRS on each input on the analog cell. Figure 18 shows how the global inputs are connected. The connection and the corresponding bit strings are shown in Table 18.

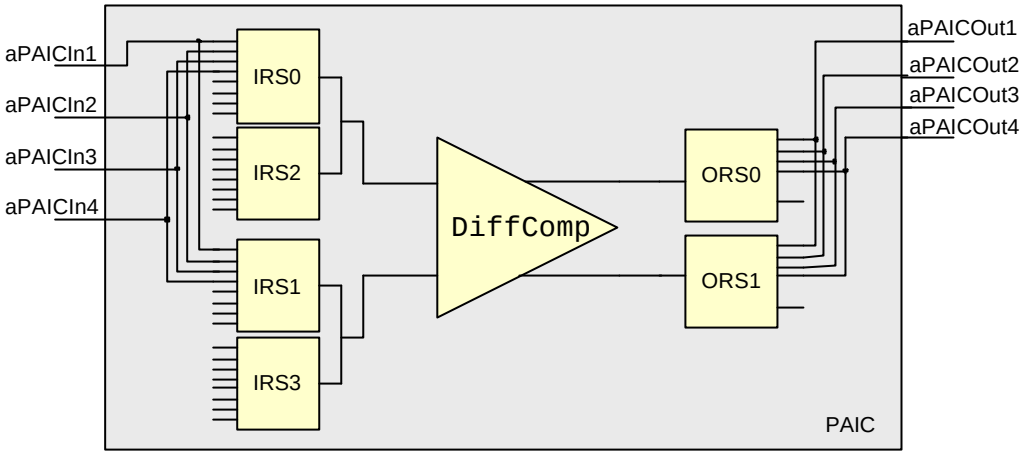

Figure 18 DiffComp connection to global signals

Global signal	DiffComp signal	Address string	Payload
aPAICIn1	input one (IRS0)	00010011	00000001
aPAICIn2	input two (IRS2)	00010100	00000010
aPAICOut0	output one (ORS0)	00010000	00000001 (Table 16)
aPAICOut1	output two (ORS1)	00010001	00000010 (Table 16)

Table 18 Connections and bit string for DiffComp Example

As the only control of the PAIC is through the `Cntr` signal and the SPI bus two C++ functions were made to read and write these signals.

wCntr method:

```
//---------------------------------------------------------------------------
void wCntr(sc_lv<3> cnt){
//---------------------------------------------------------------------------
       Cntr.write(cnt);    //updates the Cntr signal
       sc_cycle(1)         //runs one simulation cycle
       Cntr.write("000");//resets the Cntr signal
       sc_cycle(1);  //runs one simulation cycle
}
```

wSPI method:

```
//---------------------------------------------------------------------------
void wSPI(sc_lv<8> input){
//---------------------------------------------------------------------------
       sc_lv<8> sreg;      //variable for the master register
       int i;
       bool LSB;
       bool MSB;
       sreg = input;//loads the master register
       for(i=0;i<8;i++){
              MSB = sreg[7].to_int();
              mosi.write(MSB);           //Writes MSB to mosi signal
              sreg.range(1,7) = sreg.range(0,6); //Shifts master register
```

```
                sck = false;sc_cycle(1);
                sck=true;sc_cycle(1);      //clocks the transfer

                LSB  = miso.read(); //reads the miso signal
                sreg[0] = LSB;              //writes the miso signal to LSB of the
                                            //master register
                dOut = sreg;        //uploads the global variable that
                                            //holds the output 8-bit value
        }
        sck = false;
}
```

Using these two methods we can easily write the string in Table 18 to PAIC. Code for this test is provided below. The results are show in Figure 19 and Figure 20, from this figures we can clearly see how the comparator toggles when the analog input toggles.

Connect DiffComp testbench:

```
wCntr("000");
sck = false;
enable = true;

//Reset Chip
sc_cycle(1);
reset = true;
sc_cycle(1);
reset = false;

//Connect aPAICIn0 to IRS0 out
wSPI("00010011");
wCntr("001");

wSPI("00000001");
wCntr("010");

//Connect aPAICIn1 to IRS1 out
wSPI("00010100");
wCntr("001");

wSPI("00000010");
wCntr("010");

//Connect OSR0 out to aPAICOut0
wSPI("00010000");
wCntr("001");

wSPI("00000001");
wCntr("010");

//Connect OSR1 out to aPAICOut1
wSPI("00010001");
wCntr("001");

wSPI("00000010");
wCntr("010");

//Toggle analog value
bool toggle = true;
for(i=0;i<10;i++){
        aPAICIn[0] = 0.5;
        if(toggle){
                aPAICIn[1] =0;
                toggle = false;
        }else{
                aPAICIn[1] =1;
                toggle = true;
        }
```

```
        sc_cycle(1);
}
```

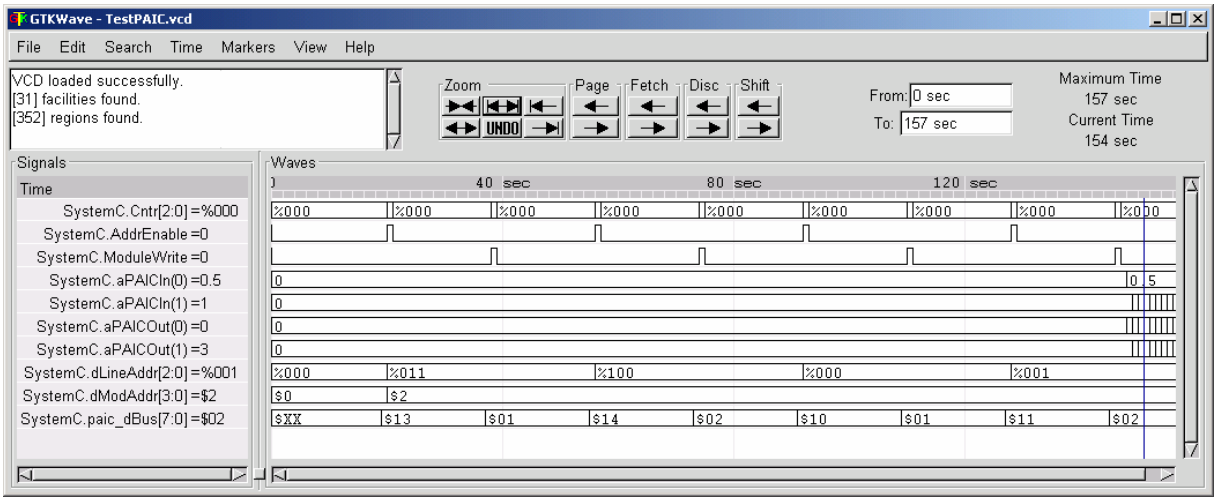

Figure 19 DiffComp results

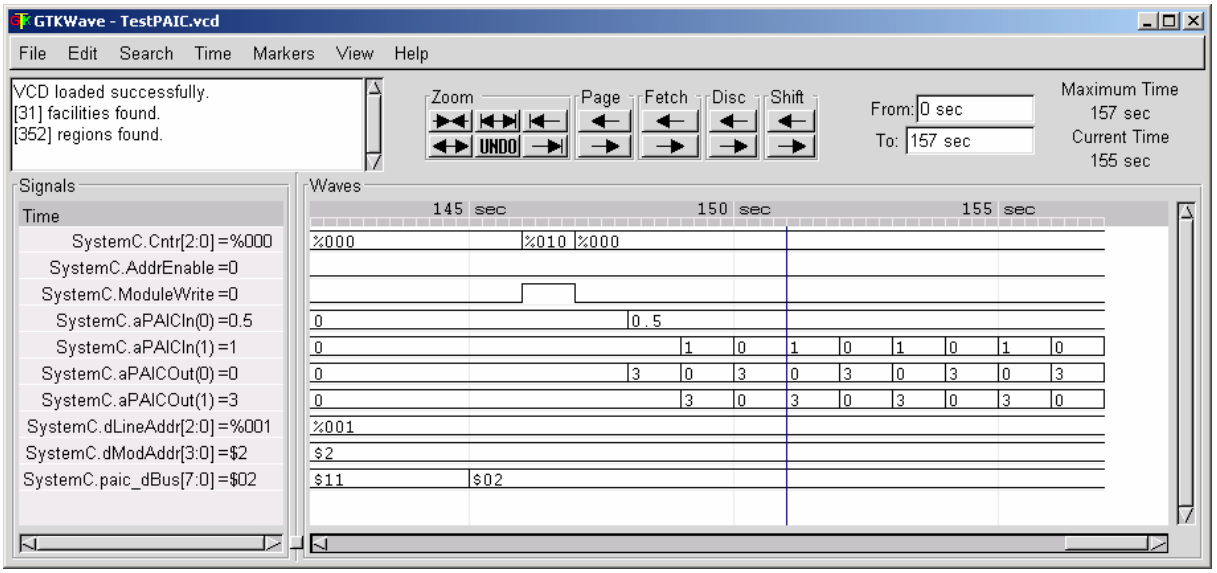

Figure 20 DiffComp results zoomed on output toggle

5.3 ADC

The simulation of the ADC is a functional analysis. The testbench starts by routing the DiffComp and the DAC into an ADC, it then performs a loop through 256 analog values with 1LSB steps starting at ½ LSB. The testbench is included in Appendix V page XXXII. The ADC output was confirmed visually since the output counts from 0 to 255 and any missing codes are easy to detect. The test output is supplied in Appendix III. Figure 21 shows an excerpt from the ADC simulation. It shows how the software routine finds the correct digitial word for the analog value 1.14453125. The correct digital word for this value is 146 (using an 8-bit converter), and we can see how the software routine does a binary search via the words: 128 (to low), 196 (to high), 160 (to high), 144 (to low), 152 (to high), 148 (to high), 146 (to low) and 147 (to high). The parentheses is a translation of the comparator output (aPAICOut(0)). The comparator tells us that 146 is to high and 147 is to low, which means that the analog value lies between 146 and 147. This is correct since we started looping ½ LSB and the analog value for this excerpt is 146*2/256 + ½ LSB = 1,140625 + 0,00390625 = 1,14453125.

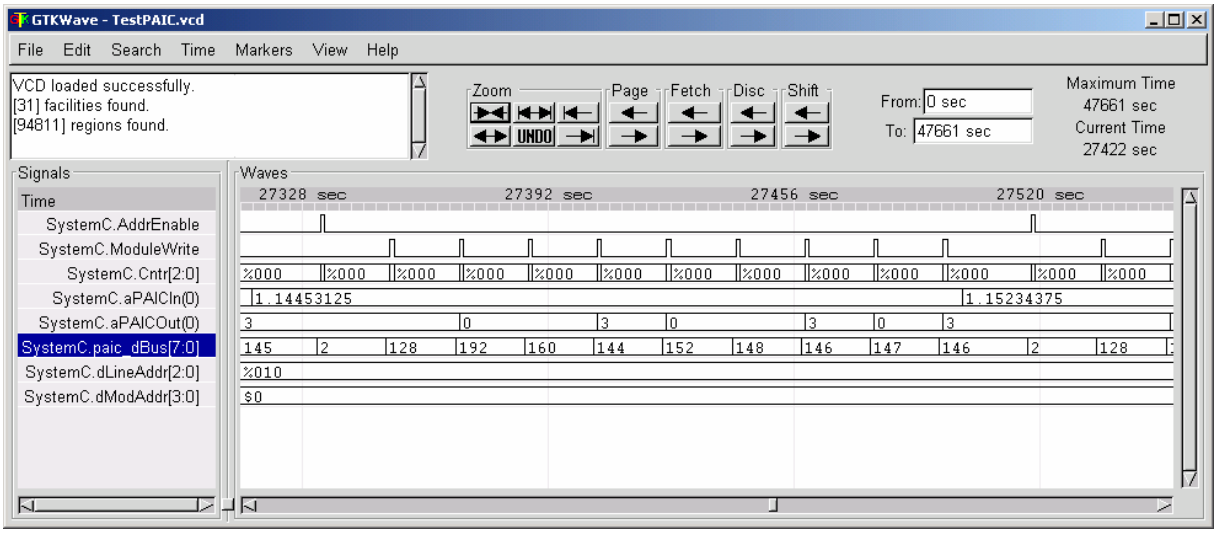

Figure 21 Excerpt from ADC simulation

5.4 READBACK OF ROUTING

The testbench for readback (provided below) was written from the readback pseudocode. It begins by resetting all ORS to low output. The variables `mod` and `line` holds the module address and line address for the IRS we currently are investigating. The variables `i` and `z` holds the address for the current ORS. We start by setting the output of the ORS high, then we read the analog inputs of the current IRS by setting the control signal "011". This sends the load signal (Table 5) to the IRS, from Figure 15 we can see that the load signal reads the value on the analog inputs into the register "converting" them to digital signals. The control signal "101" loads the value of the `dBus` into the SPI register. The readout of the readback data is simultaneous to the loading of the address register with the address for the current ORS. To decide if the is a connection we do an or_reduce (`b0 OR b1 OR - - OR b7 = result`) of the readback data. If this produces a 1 we save the data to a file. The final task is to set the output of the current ORS low. The result is show in Table 19. As we can see from the results the outputs from modules 5-9 and 10 –12, respectively ResArrays and CapArrays, are connected to IRS0 and IRS1 on all modules except the ResArray and

CapArray. The only deviation from this is the connection needed to connect the ADC. The marked line shows this connection from `DAC->ORS0` to `DiffComp->IRS1`.

Readback Testbench:

```
//-------------------------------------------------------------------------
void ReadbackTest(){
//-------------------------------------------------------------------------
	int mod = 0;
	int line = 3;
	int z;

	sc_lv<5> modAddr;
	sc_lv<3> lineAddr;
	sc_lv<8> t_addr;

	for(i=0;i<13;i++){
		//Create address string for ORS0
		modAddr = i;
		t_addr.range(2,0) = "000";
		t_addr.range(7,3) = modAddr;

		//Set ORS0 to low
		wSPI(t_addr);
		wCntr("001");
		wSPI("00000101");
		wCntr("010");

		//Create address string for ORS1
		modAddr = i;
		t_addr.range(2,0) = "001";
		t_addr.range(7,3) = modAddr;

		//Set ORS1 to low
		wSPI(t_addr);
		wCntr("001");
		wSPI("00000101");
		wCntr("010");
	}

	for(mod=0;mod<13;mod++){
		for(line = 3;line<8;line++){
			for(i=0;i<13;i++){
				for(z=0;z<2;z++){

					//Create address string for ORS(z)
					lineAddr = z;
					modAddr = i;
					t_addr.range(2,0) = lineAddr;
					t_addr.range(7,3) = modAddr;

					//Set ORS(z) high
					wSPI(t_addr);
					wCntr("001");
					wSPI("00000110");
					wCntr("010");

					//Create address string for IRS(line)
					lineAddr = line;
					modAddr = mod;
					t_addr.range(2,0) = lineAddr;
					t_addr.range(7,3) = modAddr;

					//Load SPI from the IRS(line) register
					wSPI(t_addr);
					wCntr("001"); //load addr
					wCntr("011"); //read analog input
					wCntr("101"); //load paic spi register
```

```
                        //Create address string for ORS(z)
                        lineAddr = z;
                        modAddr = i;
                        t_addr.range(2,0) = lineAddr;
                        t_addr.range(7,3) = modAddr;

                        //Load addr and readback data
                      wSPI(t_addr);
                        wCntr("001");

                        //SAVE RESULTS
                        if(dOut.or_reduce()){
                              //save readback data if there is a connection
                              readbackSave(mod,line,i,z,dOut);
                        }

                        //Set ORS(z) low
                        wSPI("0000101");
                        wCntr("010");
                        }//output end
                  }
            }//line end
      }//mod end
}//ReadbacktTest end
```

Module addr	IRS addr	Module addr	ORS addr	Connection string
0	3	5	0	00010000
0	3	6	0	00100000
0	3	7	0	01000000
0	3	8	0	10000000
0	5	9	0	00000001
0	5	10	0	00000010
0	5	11	0	00000100
0	5	12	0	00001000
1	3	5	0	00010000
1	3	6	0	00100000
1	3	7	0	01000000
1	3	8	0	10000000
1	5	9	0	00000001
1	5	10	0	00000010
1	5	11	0	00000100
1	5	12	0	00001000
2	3	5	0	00010000
2	3	6	0	00100000
2	3	7	0	01000000
2	3	8	0	10000000
2	4	0	0	00010000
2	5	9	0	00000001
2	5	10	0	00000010
2	5	11	0	00000100
2	5	12	0	00001000
3	3	5	0	00010000
3	3	6	0	00100000
3	3	7	0	01000000
3	3	8	0	10000000
3	5	9	0	00000001

3	5	10	0	00000010
3	5	11	0	00000100
3	5	12	0	00001000
4	3	5	0	00010000
4	3	6	0	00100000
4	3	7	0	01000000
4	3	8	0	10000000
4	5	9	0	00000001
4	5	10	0	00000010
4	5	11	0	00000100
4	5	12	0	00001000

Table 19 Result from readback data

5.5 READING TABLE OF CONTENT

Reading the TOC is done by counting through the module address and line address. The code for reading the TOC is shown below. It iterates through the module address and line address and reads the TOC. The TOC data is then written to a file, the contents is shown in Table 20. The description field gives analog cell that corresponds to the data at line address 0. The sub number at line address 1 is "00000001" for all analog cells since this is the first PAIC to be made and the sub number defines a specialisation of the analog cell type.

```
//-------------------------------------------------------------------------------
void TOCTest(){
//-------------------------------------------------------------------------------
	int mod = 0;
	int line = 3;
	int z;

	sc_lv<5> modAddr;
	sc_lv<3> lineAddr;
	sc_lv<8> t_addr;

	for(mod=0;mod<16;mod++){
		for(line=0;line<2;line++){
			//Create address string
			lineAddr = line;
			modAddr = mod;
			t_addr.range(2,0) = lineAddr;
			t_addr.range(7,3) = modAddr;

			//Write address
			wSPI(t_addr);
			wCntr("001");

			//Read TOC
			wCntr("100");

			//Load SPI register
			wCntr("101");

			//Read data from PAIC
			wSPI("01010101");

			//Save TOC data
			TOCSave(mod,line,dOut);
		}
	}
}
```

Mod addr	Line addr	Data	Desc
0	0	00000001	DAC
0	1	00000001	
1	0	00000010	DiffOTA
1	1	00000001	
2	0	00000011	DiffComp
2	1	00000001	
3	0	00000100	Bandgap
3	1	00000001	
4	0	00000000	
4	1	00000000	
5	0	00000110	ResArray
5	1	00000001	
6	0	00000110	ResArray
6	1	00000001	
7	0	00000110	ResArray
7	1	00000001	
8	0	00000110	ResArray
8	1	00000001	
9	0	00000111	CapArray
9	1	00000001	
10	0	00000111	CapArray
10	1	00000001	
11	0	00000111	CapArray
11	1	00000001	
12	0	00000111	CapArray
12	1	00000001	
13	0	00000001	DAC
13	1	00000001	
14	0	00000000	
14	1	00000000	
15	0	00000000	
15	1	00000000	

Table 20 TOC data

6 CONCLUSION

A concept for programmable analog integrated circuits has been presented. The simulations have shown that the PAIC can supply a user with information about its contents and how it should be programmed to perform a specific function. The PAIC concept have been verified through "building" and simulating an ADC from PAIC's analog cells. It has shown it self to be a viable design for a programmable analog integrated circuit, and a foundation for further work.

7 FUTURE WORK

The goal of PAIC is to tape out summer 2002. To achieve this goal there is a lot to be done. The digital portions of the PAIC have to be modeled in a language (i.e. VHDL) that can synthesize the design. The analog cells have to be designed, simulated and laid out. Since the PAIC is a mixed-signal and non standard design, much of the layout of the digital blocks have to be done by hand, especially in the analog framework. Probably one of the greatest challenges is to route the analog signals between analog modules in a way that provide shielding from noise sources, uses minimal area and does not limit the bandwidth below the desired test bandwidth. Which analog circuits that should be possible to "build" from the PAIC's analog cells, in addition to the ADC, needs to be investigated. Making sure that the user cannot "build" a circuit that can destroy the chip is a challenge that must be solved.

8 REFERENCES

[1] H. Shen, Z. Xu, B. Dalager, V. Kristiansen, Ø. Strøm, M.S. Shur, T.A. Fjeldly, J. Lü, T. Ytterdal , "Conducting Laboratory Experiments over the Internet", IEEE Trans. on Education, 42, No. 3, pp. 180-185 (1999).

[2] T.A. Fjeldly, M. S. Shur, H. Shen and T. Ytterdal, "Automated Internet Measurement Laboratory (AIM-Lab) for Engineering Education", Proceedings of 1999 Frontiers in Education Conference (FIE'99), San Juan, Puerto Rico, IEEE Catalog No. 99CH37011(C), 12a2 (1999).

[3] M. Shur, T. A. Fjeldly and H. Shen, "AIM-Lab - A System for Conducting Semiconductor Device Characterization via the Internet", late news paper at 1999 International Conference on Microelectronic Test Structures (ICMTS 1999), Gothenburg, Sweden (1999).

[4] T.A. Fjeldly, M.S. Shur, H. Shen, and T. Ytterdal, "AIM-Lab: A System for -Remote Characterization of Electronic Devices and Circuits over the Internet", Proc. 3rd IEEE Int. Caracas Conf. on Devices, Circuits and Systems (ICCDCS-2000), Cancun, Mexico, IEEE Catalog No. 00TH8474C, pp. I43.1 -I43.6 (2000)

[5] K. Smith, J.O. Strandman, R. Berntzen, T.A. Fjeldly, M.S. Shur, "Advanced Internet Technology in Laboratory Modules for Distance-Learning," accepted for presentation at the American Society for Engineering Education Annual Conference & Exposition 2001, ASEE'01".

[6] Lee, E.K.F.; Gulak, P.G. "A CMOS field-programmable analog array ". Solid-State Circuits, IEEE Journal of , Volume: 26 Issue: 12 , Dec. 1991. Page(s): 1860 –1867

[7] Stephan Ohr, EE Times. "Programmable analog- here we go again" http://www.planetanalog.com/ Nov 15, 1999

[8] Peter Clarke, EE Times. "Startup rolls switched-capacitor analog array". http://www.planetanalog.com/ Jul 31, 2000

[9] P.G. Gulak "Field-Programmable Analog Arrays: Past, present and future perspectives". 1995. IEEE.

[10] Hans W. Klein "Introductory EPAC: an Analog FPGA", IMP Inc.
ISBN# 0-7803-2636-9

[11] Hans W. Klein "The EPAC Architecture: An Expert Cell Approach to Field Programmable Analog Devices" Lattice Semiconductor Corp. Analog Integrated Circuits and Signal Processing 17, pages 91-103, 1998.

[12] Claire Tristram. "It's time for Clockless Chips" Technology Review, October 2001. http://www.technologyreview.com

[13] Lattice Semiconductor Corporation. "ispPAC 20 (In-System Programmable Analog Circuit)", May 2001, http://www.latticesemi.com

[14] Anadigm. “AN10E40 Data manual” http://www.anadigm.com

[15] Tor A. Fjeldly, Raymond Berntzen Jan O. Strandman, Michael S. Shur & Kjell Jeppson. “LAB-on-WEB” http://www.lab-on-web.com/

[16] Trond Ytterdal, Carsten Wulff, Thomas A Sæthre & Arne Skjevlan “Next Generation Lab” http://ngl.fysel.ntnu.no

[17] Carsten Wulff, Trond Ytterdal, Thomas A. Sæthre, Arne Skjevlan, Tor A. Fjeldy, Michael S. Shur “Next Generation Lab – A solution for remote characterization of analog integrated circuits”. Paper submitted to International Caracas Conference on Devices, Circuits and Systems (ICCDCS) April 2002.

APPENDIX

APPENDIX I: SERIAL PERIPHERAL INTERFACE

SPI is a widely used interface, and is featured in many micro-controllers and IC's. The general idea behind a SPI is to have two shift registers connected together with the `miso` (master in slave out) and `mosi` (master out slave in) signals as show in Figure 22. The master controls the transfer through the `sck` signal. When `sck` goes high, the master writes its MSB to the `mosi` signal and shifts one position, the slave writes its MSB to the `miso` signal and shifts one position. Both master and slave write to their LSB from their respective input signals. This way the data is shifted from master to slave and from slave to master, thus providing bi-directional data transfer.

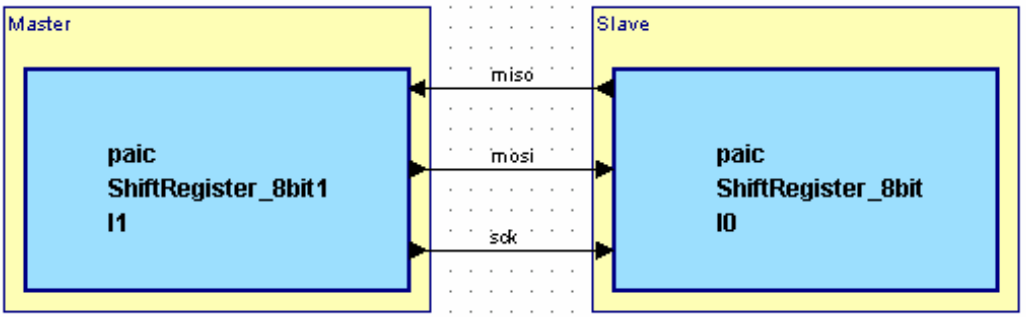

Figure 22 SPI bloc diagram

APPENDIX II: SYSTEMC

ABOUT

SystemC is a system level modelling language. Unlike any other HDL language it creates an executable file that contains the system design and the testbench. It has built-in templates for modules and processes like a standard HDL language and the possibility to use the power of C++. Although SystemC provides great flexibility it does not include any analog libraries. Therefore, the act of connecting more than two analog signals together becomes a challenge and simulation of circuits like Figure 8 becomes exceedingly difficult. Creating an analog library for modelling such circuits was considered beyond the scope of this project. The SystemC 1.0.2 library was used together with Mircosoft Visual Studio .NET to create the model used in simulation of the PAIC.

SYNTAX

This chapter is an introduction into the basic syntax of SystemC. A flip-flop example with testbench and simulation output is provided in appendix 0. The structure of SystemC is similar to what we find in other HDL, there is a concept of modules, processes, ports and signals. To define a module we write `struct something: sc_module`. This informs the compiler to create an object of `something` inheriting `sc_module`. To create ports on the module one of the following is used; `sc_in<type>`, `sc_out<type>` or `sc_inout<type>`. The `type` can be any of the standard C++ types or the SystemC types, there is also the possibility

to create user defined types i.e `voltage` or `current`. Using `sc_signal<type>` makes signal declarations. The objects inheriting `sc_module` must contain a constructor, which is defined using `SC_CTOR(something)`, the constructor contains a definition of child modules, methods, sensitivities and port/signal bindings. To define a method as a SystemC method we use `SC_METHOD(method_name)`. The method is made sensitive signals or ports (invariant of the type) by following the method declaration by i.e. `sensitive_pos(clk)`. Ports and signal bindings are made using the syntax `module->clock(clock)`, this binds the `clock` of the module to a `clock` signal. We can declare a variable of the SystemC types for temporary storage, the scope of the variables are the same as in C++.

To simulate a SystemC design we write a testbench that is incorporated into the executable. The simulation output can be written to screen, to file or traced using built-in trace functions. The output can be processed visually in a terminal, in matlab, waveform viewers or any other data processing software. The trace functions in SystemC can write a number of different formats, in PAIC the .vcd format was used. All SystemC executables must include a `sc_main()` function similar to C++ `main()`. Often when programming a design in SystemC you want several testbenches to simulate different portions of your design. Writing the testbench as an object it is simpler to swap testbenches without having to rewrite the `sc_main()` method.

APPENDIX III: ADC OUTPUT

aPAICin0	**Out**	**DiffCompOut**	0.300781	38	0.000000
0.003906	0	0.000000	0.308594	39	3.000000
0.011719	1	3.000000	0.316406	40	0.000000
0.019531	2	0.000000	0.324219	41	3.000000
0.027344	3	3.000000	0.332031	42	0.000000
0.035156	4	0.000000	0.339844	43	3.000000
0.042969	5	3.000000	0.347656	44	0.000000
0.050781	6	0.000000	0.355469	45	3.000000
0.058594	7	3.000000	0.363281	46	0.000000
0.066406	8	0.000000	0.371094	47	3.000000
0.074219	9	3.000000	0.378906	48	0.000000
0.082031	10	0.000000	0.386719	49	3.000000
0.089844	11	3.000000	0.394531	50	0.000000
0.097656	12	0.000000	0.402344	51	3.000000
0.105469	13	3.000000	0.410156	52	0.000000
0.113281	14	0.000000	0.417969	53	3.000000
0.121094	15	3.000000	0.425781	54	0.000000
0.128906	16	0.000000	0.433594	55	3.000000
0.136719	17	3.000000	0.441406	56	0.000000
0.144531	18	0.000000	0.449219	57	3.000000
0.152344	19	3.000000	0.457031	58	0.000000
0.160156	20	0.000000	0.464844	59	3.000000
0.167969	21	3.000000	0.472656	60	0.000000
0.175781	22	0.000000	0.480469	61	3.000000
0.183594	23	3.000000	0.488281	62	0.000000
0.191406	24	0.000000	0.496094	63	3.000000
0.199219	25	3.000000	0.503906	64	0.000000
0.207031	26	0.000000	0.511719	65	3.000000
0.214844	27	3.000000	0.519531	66	0.000000
0.222656	28	0.000000	0.527344	67	3.000000
0.230469	29	3.000000	0.535156	68	0.000000
0.238281	30	0.000000	0.542969	69	3.000000
0.246094	31	3.000000	0.550781	70	0.000000
0.253906	32	0.000000	0.558594	71	3.000000
0.261719	33	3.000000	0.566406	72	0.000000
0.269531	34	0.000000	0.574219	73	3.000000
0.277344	35	3.000000	0.582031	74	0.000000
0.285156	36	0.000000	0.589844	75	3.000000
0.292969	37	3.000000	0.597656	76	0.000000

0.605469	77	3.000000	1.136719	145	3.000000
0.613281	78	0.000000	1.144531	146	0.000000
0.621094	79	3.000000	1.152344	147	3.000000
0.628906	80	0.000000	1.160156	148	0.000000
0.636719	81	3.000000	1.167969	149	3.000000
0.644531	82	0.000000	1.175781	150	0.000000
0.652344	83	3.000000	1.183594	151	3.000000
0.660156	84	0.000000	1.191406	152	0.000000
0.667969	85	3.000000	1.199219	153	3.000000
0.675781	86	0.000000	1.207031	154	0.000000
0.683594	87	3.000000	1.214844	155	3.000000
0.691406	88	0.000000	1.222656	156	0.000000
0.699219	89	3.000000	1.230469	157	3.000000
0.707031	90	0.000000	1.238281	158	0.000000
0.714844	91	3.000000	1.246094	159	3.000000
0.722656	92	0.000000	1.253906	160	0.000000
0.730469	93	3.000000	1.261719	161	3.000000
0.738281	94	0.000000	1.269531	162	0.000000
0.746094	95	3.000000	1.277344	163	3.000000
0.753906	96	0.000000	1.285156	164	0.000000
0.761719	97	3.000000	1.292969	165	3.000000
0.769531	98	0.000000	1.300781	166	0.000000
0.777344	99	3.000000	1.308594	167	3.000000
0.785156	100	0.000000	1.316406	168	0.000000
0.792969	101	3.000000	1.324219	169	3.000000
0.800781	102	0.000000	1.332031	170	0.000000
0.808594	103	3.000000	1.339844	171	3.000000
0.816406	104	0.000000	1.347656	172	0.000000
0.824219	105	3.000000	1.355469	173	3.000000
0.832031	106	0.000000	1.363281	174	0.000000
0.839844	107	3.000000	1.371094	175	3.000000
0.847656	108	0.000000	1.378906	176	0.000000
0.855469	109	3.000000	1.386719	177	3.000000
0.863281	110	0.000000	1.394531	178	0.000000
0.871094	111	3.000000	1.402344	179	3.000000
0.878906	112	0.000000	1.410156	180	0.000000
0.886719	113	3.000000	1.417969	181	3.000000
0.894531	114	0.000000	1.425781	182	0.000000
0.902344	115	3.000000	1.433594	183	3.000000
0.910156	116	0.000000	1.441406	184	0.000000
0.917969	117	3.000000	1.449219	185	3.000000
0.925781	118	0.000000	1.457031	186	0.000000
0.933594	119	3.000000	1.464844	187	3.000000
0.941406	120	0.000000	1.472656	188	0.000000
0.949219	121	3.000000	1.480469	189	3.000000
0.957031	122	0.000000	1.488281	190	0.000000
0.964844	123	3.000000	1.496094	191	3.000000
0.972656	124	0.000000	1.503906	192	0.000000
0.980469	125	3.000000	1.511719	193	3.000000
0.988281	126	0.000000	1.519531	194	0.000000
0.996094	127	3.000000	1.527344	195	3.000000
1.003906	128	0.000000	1.535156	196	0.000000
1.011719	129	3.000000	1.542969	197	3.000000
1.019531	130	0.000000	1.550781	198	0.000000
1.027344	131	3.000000	1.558594	199	3.000000
1.035156	132	0.000000	1.566406	200	0.000000
1.042969	133	3.000000	1.574219	201	3.000000
1.050781	134	0.000000	1.582031	202	0.000000
1.058594	135	3.000000	1.589844	203	3.000000
1.066406	136	0.000000	1.597656	204	0.000000
1.074219	137	3.000000	1.605469	205	3.000000
1.082031	138	0.000000	1.613281	206	0.000000
1.089844	139	3.000000	1.621094	207	3.000000
1.097656	140	0.000000	1.628906	208	0.000000
1.105469	141	3.000000	1.636719	209	3.000000
1.113281	142	0.000000	1.644531	210	0.000000
1.121094	143	3.000000	1.652344	211	3.000000
1.128906	144	0.000000	1.660156	212	0.000000

```
1.667969      213    3.000000                1.839844      235    3.000000
1.675781      214    0.000000                1.847656      236    0.000000
1.683594      215    3.000000                1.855469      237    3.000000
1.691406      216    0.000000                1.863281      238    0.000000
1.699219      217    3.000000                1.871094      239    3.000000
1.707031      218    0.000000                1.878906      240    0.000000
1.714844      219    3.000000                1.886719      241    3.000000
1.722656      220    0.000000                1.894531      242    0.000000
1.730469      221    3.000000                1.902344      243    3.000000
1.738281      222    0.000000                1.910156      244    0.000000
1.746094      223    3.000000                1.917969      245    3.000000
1.753906      224    0.000000                1.925781      246    0.000000
1.761719      225    3.000000                1.933594      247    3.000000
1.769531      226    0.000000                1.941406      248    0.000000
1.777344      227    3.000000                1.949219      249    3.000000
1.785156      228    0.000000                1.957031      250    0.000000
1.792969      229    3.000000                1.964844      251    3.000000
1.800781      230    0.000000                1.972656      252    0.000000
1.808594      231    3.000000                1.980469      253    3.000000
1.816406      232    0.000000                1.988281      254    0.000000
1.824219      233    3.000000                1.996094      255    3.000000
1.832031      234    0.000000
```

APPENDIX IV: FLIP-FLOP EXAMPLE

FLIP-FLOP EXAMPLE:

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

#ifndef _FLIPFLOP
#define _FLIPFLOP

struct FLIP_FLOP: sc_module{

sc_in<bool> clock;
sc_in<bool> reset;
sc_in<bool> D;
sc_out<bool> Q;

bool temp;
//------------------------------------------------------------
SC_CTOR(FLIP_FLOP){
//------------------------------------------------------------
       SC_METHOD(FF);
       sensitive_pos(clock);
}

//------------------------------------------------------------
void FF(){
//------------------------------------------------------------
       if(!reset.read()){
              temp = D.read();
              Q.write(temp);
       }else{
              Q.write(false);
       }

}

};
#endif
```

FLIP-FLOP TESTBENCH:

```
struct REG8_Testbench{
```

```
sc_signal<bool> clock;
sc_signal<bool> reset;
sc_signal<bool> dIn[8];
sc_signal<bool> dOut[8];

FLIP_FLOP *ff[8];
sc_lv<8> out;

//------------------------------------------------------------
void run(){
//------------------------------------------------------------
int i = 0 ;
char si[10];
char str[20];
for(i=0;i<8;i++){
	strcpy(str,"ff");strcat(str,itoa(i,si,10));
	ff[i] = new FLIP_FLOP(str);
	ff[i]->clock(clock);
	ff[i]->reset(reset);
	ff[i]->D(dIn[i]);
	ff[i]->Q(dOut[i]);
}

	sc_initialize();//calls SC_CTOR() and initsializes the simulation

	reset = false;
	dIn_write("11111111"); //writes a value to the register
	clock=false;sc_cycle(1);clock=true;sc_cycle(1);//clocks a period
	dOut_read(); //writes result to the output stream


	reset = true; //resets the register
	clock = false;sc_cycle(1);clock = true;sc_cycle(2); //clocks a period
	dOut_read(); //writes result to the output stream
	reset = false;

	dIn_write("10101010"); //writes a value to the register
	clock = false;sc_cycle(1);clock = true;sc_cycle(2); //clocks a period
	dOut_read(); //writes result to the output stream

	sc_stop(); //stops the simulation
}

//------------------------------------------------------------
void dIn_write(sc_lv<8> val){
//------------------------------------------------------------
	int i=0;
	for(i=0;i<8;i++){
		dIn[i].write(val[i].to_int());
	}
}

//------------------------------------------------------------
void dOut_read(){
//------------------------------------------------------------
	int i=0;
	for(i=0;i<8;i++){
		out[i] = dOut[i].read();
	}
	cout << out << "\n";
}

}; //end REG8_Testbench

//------------------------------------------------------------
int sc_main(int ac, char *av[])
//------------------------------------------------------------
```

```
{
	REG8_Testbench *t1 = new REG8_Testbench();
	t1->run();
	return 0;
}
```

OUTPUT FROM REG8_TESTBENCH:

```
c:\>reg8
               SystemC Version 1.0.2  --- Mar  1 2001 18:27:58
                                ALL RIGHTS RESERVED
                    Copyright (c) 1988-2001 by Synopsys, Inc.
11111111
00000000
10101010
c:\>
```

APPENDIX V: SYSTEMC SOURCE CODE

IRS:

```
/*!
	\file		IRS.h
	\brief		Input register and switch for analog framework
	\author			Carsten Wulff
	\started	2001.10.4
	\modified	2001.11.19

*/
#include "MsmlV2.h"

#ifndef _IRS
#define _IRS

struct IRS: sc_module{
	//Digital
	sc_in<bool> reset;
	sc_inout<sc_lv<8> > dIn;
	sc_in<bool> dLoad;
	sc_in<bool> dWrite;
	sc_in<bool> dEnable;

	//Analog
	sc_in<voltage> aIn[8];
	sc_out<voltage> aOut;

	//Signals
	sc_signal<sc_uint<8> > dControl;

	//Variables
	double aOutputTemp;
	sc_uint<8> t_LoadBus;

	//------------------------------------------------------------------
	SC_CTOR(IRS){
	//------------------------------------------------------------------
		int i;
		SC_METHOD(SCM_ControlRegister);
		sensitive_pos(dWrite);
		sensitive_pos(dLoad);
```

```
		sensitive(reset);

		SC_METHOD(SCM_SignalHighway);
		for(i=0;i<8;i++){
		sensitive(aIn[i]);
		}
		sensitive(dControl);

		aOutputTemp = 0;
	}

	//------------------------------------------------------------------------
	void SCM_ControlRegister(){
	//------------------------------------------------------------------------
		int i;
		if(!reset.read()){
			if(dWrite.read() & dEnable.read() & !dLoad.read()){
				dControl.write(dIn.read());
			}//end if
			if(dLoad.read() & dEnable.read() & !dWrite.read()){
				dControl.write(0);
				for(i=0;i<8;i++){
					if(aIn[i].read() > 1){
						t_LoadBus[i] = 1;
					}
					else{
						t_LoadBus[i] =0;
					}
				}
				dIn.write(t_LoadBus);
			}//end if
		}else{
			dControl.write(0);
		}//end if
	}//end method

	//------------------------------------------------------------------------
	void SCM_SignalHighway(){
	//------------------------------------------------------------------------
		int i;
		//Normal Operation
		aOutputTemp = 0;
		for(i=0;i<8;i++){
			if(dControl.read()[i]){
				aOutputTemp = aOutputTemp + aIn[i].read() ;
			}
		}
		aOut.write(aOutputTemp);
	}//end method
};
#endif
```

ORS:

```
/*!
	\file		ORS.h
	\brief		Output register and switch in analog framework
	\author			Carsten Wulff
	\started	2001.10.7
	\modified	2001.11.19

*/

#include "MsmlV2.h"

#ifndef _ORS
#define _ORS
```

```
enum   vm_state   {loadlow_st,   loadhigh_st,   norm_st,   global0_st,   global1_st,
global2_st, global3_st};

struct ORS: sc_module{

	static const int nrGlobal = 4;

	//Digital Ports
	sc_in<bool> reset;
	sc_in<bool> dEnable;
	sc_in<bool> dWrite;
	sc_inout<sc_lv<8> > dIn;

	//Analog Ports
	sc_in<voltage> aVdd;
	sc_in<voltage> aVss;
	sc_in<voltage> aIn;
	sc_inout<voltage> aOut;
	sc_inout<voltage> aGlobalOut[nrGlobal];

	//Signals
	sc_signal<vm_state> current_state;

	//Variables
	sc_uint<8> t_dIn;

	//---------------------------------------------------------------------
	SC_CTOR(ORS){
	//---------------------------------------------------------------------
		current_state = norm_st;
		SC_METHOD(Control);
		sensitive_pos(dWrite);
		sensitive(reset);

		SC_METHOD(Analog);
		sensitive(aIn);
		sensitive(current_state);
	}

	//---------------------------------------------------------------------
	void Control(){
	//---------------------------------------------------------------------
		/* ===========STATES=========
			dIn                  OPERATION                        State
			0000 0000    Normal                          norm_st
			0000 0001    GlobalOut0                      global0_st
			0000 0010    GlobalOut1                      global1_st
			0000 0011    GlobalOut2                      global2_st
			0000 0100    GlobalOut3                      global3_st
			0000 0101    Load w/ aOut LOW          loadlow_st
			0000 0110    Load w/ aOut HIGH         loadhigh_st

		*/
		if(!reset.read()){
			if(dEnable.read() && dWrite.read()){
				t_dIn = dIn;
				switch(t_dIn){
				case 0:
					current_state = norm_st;
					break;
				case 1:
					current_state = global0_st;
					break;
				case 2:
					current_state = global1_st;
					break;
				case 3:
```

```
                                current_state = global2_st;
                                break;
                        case 4:
                                current_state = global3_st;
                                break;
                        case 5:
                                current_state = loadlow_st;
                                break;
                        case 6:
                                current_state = loadhigh_st;
                                break;
                        default:
                                current_state  = norm_st;
                                break;
                        }//end switch

                }else{
                        current_state = current_state;
                }
        }else{
                current_state = norm_st;
        }
}

//---------------------------------------------------------------------
void Analog(){
//---------------------------------------------------------------------
        switch(current_state){
                case norm_st:
                        aOut.write(aIn.read());
                        break;
                case global0_st:
                        aOut.write(aIn.read());
                        aGlobalOut[0].write(aIn.read());
                        break;
                case global1_st:
                        aOut.write(aIn.read());
                        aGlobalOut[1].write(aIn.read());
                        break;
                case global2_st:
                        aOut.write(aIn.read());
                        aGlobalOut[2].write(aIn.read());
                        break;
                case global3_st:
                        aOut.write(aIn.read());
                        aGlobalOut[3].write(aIn.read());
                        break;
                case loadlow_st:
                        aOut.write(aVss);
                        break;
                case loadhigh_st:
                        aOut.write(aVdd);
                        break;
                default:
                        aOut.write(aIn.read());
                        break;

        }//end switch
}//end analog
};
#endif
```

MODREG:

```
/*!
        \file           ModReg.h
        \brief          A Register for the analog module
```

```
	\author			Carsten Wulff
	\started	2001.10.16
	\modified	2001.11.19

*/

#include "MsmlV2.h"

#ifndef _MODREG
#define _MODREG

struct ModReg: sc_module{

	//Digital Ports
	sc_in<bool> reset;
	sc_in<bool> dEnable;
	sc_in<bool> dWrite;
	sc_inout<sc_lv<8> > dIn;
	sc_out<sc_uint<8> > dOut;

	//Variables
	sc_uint<8> t_dIn;

	//------------------------------------------------------------------------
	SC_CTOR(ModReg){
	//------------------------------------------------------------------------
		SC_METHOD(SCM_Reg);
		//sensitive(dEnable);
		sensitive_pos(dWrite);
		sensitive(reset);
	}

	//------------------------------------------------------------------------
	void SCM_Reg(){
	//------------------------------------------------------------------------
		if(!reset.read()){
			if(dEnable.read() && dWrite.read()){
				t_dIn = dIn.read();
				dOut.write(t_dIn);
			}else{
				dOut.write(t_dIn);
			}
		}else{
			dOut.write(0);
		}
	}
};
#endif
```

LINEDEC:

```
/*!
	\file		LineDec.h
	\brief		Select which register to enable within the module
	\author			Carsten Wulff
	\started	2001.10.7
	\modified	2001.10.16

*/

#include "MsmlV2.h"

#ifndef _LINEDEC
#define _LINEDEC

struct LineDec: sc_module{
	//Digital
```

```
sc_in<bool> dEnable;
sc_in<sc_uint<3> > dLineAddr;
sc_out<bool > DOUT[8];

//Variables
sc_uint<3> t_dLineAddr;

//------------------------------------------------------------------------
SC_CTOR(LineDec){
//------------------------------------------------------------------------
	SC_METHOD(decode);
	sensitive(dEnable);
	sensitive(dLineAddr);
}

//------------------------------------------------------------------------
void decode(){
//------------------------------------------------------------------------
	t_dLineAddr = dLineAddr;
	if(dEnable.read()){

		switch(t_dLineAddr){
			case 0:
				DOUT[0] = 1;
				DOUT[1] = 0;
				DOUT[2] = 0;
				DOUT[3] = 0;
				DOUT[4] = 0;
				DOUT[5] = 0;
				DOUT[6] = 0;
				DOUT[7] = 0;
				break;
			case 1:
				DOUT[0] = 0;
				DOUT[1] = 1;
				DOUT[2] = 0;
				DOUT[3] = 0;
				DOUT[4] = 0;
				DOUT[5] = 0;
				DOUT[6] = 0;
				DOUT[7] = 0;
				break;
			case 2:
				DOUT[0] = 0;
				DOUT[1] = 0;
				DOUT[2] = 1;
				DOUT[3] = 0;
				DOUT[4] = 0;
				DOUT[5] = 0;
				DOUT[6] = 0;
				DOUT[7] = 0;
				break;
			case 3:
				DOUT[0] = 0;
				DOUT[1] = 0;
				DOUT[2] = 0;
				DOUT[3] = 1;
				DOUT[4] = 0;
				DOUT[5] = 0;
				DOUT[6] = 0;
				DOUT[7] = 0;
				break;
			case 4:
				DOUT[0] = 0;
				DOUT[1] = 0;
				DOUT[2] = 0;
				DOUT[3] = 0;
				DOUT[4] = 1;
```

```
                                        DOUT[5] = 0;
                                        DOUT[6] = 0;
                                        DOUT[7] = 0;
                                        break;
                                case 5:
                                        DOUT[0] = 0;
                                        DOUT[1] = 0;
                                        DOUT[2] = 0;
                                        DOUT[3] = 0;
                                        DOUT[4] = 0;
                                        DOUT[5] = 1;
                                        DOUT[6] = 0;
                                        DOUT[7] = 0;
                                        break;
                                case 6:
                                        DOUT[0] = 0;
                                        DOUT[1] = 0;
                                        DOUT[2] = 0;
                                        DOUT[3] = 0;
                                        DOUT[4] = 0;
                                        DOUT[5] = 0;
                                        DOUT[6] = 1;
                                        DOUT[7] = 0;
                                        break;
                                case 7:
                                        DOUT[0] = 0;
                                        DOUT[1] = 0;
                                        DOUT[2] = 0;
                                        DOUT[3] = 0;
                                        DOUT[4] = 0;
                                        DOUT[5] = 0;
                                        DOUT[6] = 0;
                                        DOUT[7] = 1;
                                        break;
                                default:
                                        break;
                        }
                }
                if(!dEnable.read()){
                        DOUT[0] = 0;
                        DOUT[1] = 0;
                        DOUT[2] = 0;
                        DOUT[3] = 0;
                        DOUT[4] = 0;
                        DOUT[5] = 0;
                        DOUT[6] = 0;
                        DOUT[7] = 0;
                }
        }
};
#endif
```

ANMODFRAMEWORK:

```
/*!
        \file           AnModFrameWork.h
        \brief          Analog Module Framework
        \author                 Carsten Wulff
        \started        2001.10.4
        \modified       2001.11.19

*/
#include "MsmlV2.h";
#include "ORS.h"
#include "IRS.h"
#include "LineDec.h"
#include "ModReg.h"
```

```
#include "GenericCell.h"

#ifndef _ANMODFRAMEWORK
#define _ANMODFRAMEWORK

struct AnModFramework: sc_module{

		//Digital Ports
		sc_in<bool> reset;
		sc_in<bool> dEnable;
		sc_in<bool> dWrite;
		sc_in<bool> dLoad;
		sc_in<sc_uint<3> > dLineAddr;
		sc_inout<sc_lv<8> > dIn;
		sc_signal<sc_uint<8> > dModIn;


		//Analog Ports
		sc_in<voltage> aIn0[8];
		sc_in<voltage> aIn1[8];
		sc_in<voltage> aIn2[8];
		sc_in<voltage> aIn3[8];
		sc_in<voltage> aIn4[8];

		sc_signal<voltage> aModOut0;
		sc_signal<voltage> aModOut1;

		sc_signal<voltage> aModIn0;
		sc_signal<voltage> aModIn1;
		sc_signal<voltage> aModIn2;
		sc_signal<voltage> aModIn3;
		sc_signal<voltage> aModIn4;

		sc_inout<voltage> aGlobalOut[4];
		sc_inout<voltage> aOut0;
		sc_inout<voltage> aOut1;

		//============== Signals ==============
		sc_signal<bool> dDecOut[8];
		sc_signal<voltage> s_aGlobalOut[4];
		sc_signal<voltage> s_aVss;
		sc_signal<voltage> s_aVdd;


		//======== OBJECTS =======
		GenericCell *da;
		LineDec *dec;
		ORS *ors0;
		ORS *ors1;
		ModReg *modreg;
		IRS *irs0;
		IRS *irs1;
		IRS *irs2;
		IRS *irs3;
		IRS *irs4;

		//---------------------------------------------------------------
		SC_CTOR(AnModFramework){
		//---------------------------------------------------------------
			int i;
			dec = new LineDec("dec");
			ors0 = new ORS("ors0");
			ors1 = new ORS("ors1");
			modreg = new ModReg("modreg");
			irs0 = new IRS("irs0");
			irs1 = new IRS("irs1");
			irs2 = new IRS("irs2");
			irs3 = new IRS("irs3");
```

```
irs4 = new IRS("irs4");

//Reset
ors0->reset(reset);
ors1->reset(reset);
modreg->reset(reset);
irs0->reset(reset);
irs1->reset(reset);
irs2->reset(reset);
irs3->reset(reset);
irs4->reset(reset);

dec->dEnable(dEnable);
//Enable Signals

for(i=0;i<8;i++){
dec->DOUT[i](dDecOut[i]);
}

ors0->dEnable(dDecOut[0]);
ors1->dEnable(dDecOut[1]);
modreg->dEnable(dDecOut[2]);
irs0->dEnable(dDecOut[3]);
irs1->dEnable(dDecOut[4]);
irs2->dEnable(dDecOut[5]);
irs3->dEnable(dDecOut[6]);
irs4->dEnable(dDecOut[7]);

//Line Address
dec->dLineAddr(dLineAddr);

//Data bus
ors0->dIn(dIn);
ors1->dIn(dIn);
modreg->dIn(dIn);
irs0->dIn(dIn);
irs1->dIn(dIn);
irs2->dIn(dIn);
irs3->dIn(dIn);
irs4->dIn(dIn);


//Load signals to signal registers
irs0->dLoad(dLoad);
irs1->dLoad(dLoad);
irs2->dLoad(dLoad);
irs3->dLoad(dLoad);
irs4->dLoad(dLoad);

//Write signals to signal registers
irs0->dWrite(dWrite);
irs1->dWrite(dWrite);
irs2->dWrite(dWrite);
irs3->dWrite(dWrite);
irs4->dWrite(dWrite);
ors0->dWrite(dWrite);
ors1->dWrite(dWrite);
modreg->dWrite(dWrite);

//Module register out
modreg->dOut(dModIn);

//======= Analog Port/Signal Bindings =======
//Avdd, Avss, global output
ors0->aVdd(s_aVdd);
ors0->aVss(s_aVss);
ors0->aGlobalOut[0](aGlobalOut[0]);
ors0->aGlobalOut[1](aGlobalOut[1]);
```

```
                ors0->aGlobalOut[2](aGlobalOut[2]);
                ors0->aGlobalOut[3](aGlobalOut[3]);
                ors0->aOut(aOut0);
                ors0->aIn(aModOut0);

                ors1->aVdd(s_aVdd);
                ors1->aVss(s_aVss);
                ors1->aGlobalOut[0](aGlobalOut[0]);
                ors1->aGlobalOut[1](aGlobalOut[1]);
                ors1->aGlobalOut[2](aGlobalOut[2]);
                ors1->aGlobalOut[3](aGlobalOut[3]);
                ors1->aOut(aOut1);
                ors1->aIn(aModOut1);

                //Signal highway inputs
                for(i=0;i<8;i++){ irs0->aIn[i](aIn0[i]); }
                for(i=0;i<8;i++){ irs1->aIn[i](aIn1[i]); }
                for(i=0;i<8;i++){ irs2->aIn[i](aIn2[i]); }
                for(i=0;i<8;i++){ irs3->aIn[i](aIn3[i]); }
                for(i=0;i<8;i++){ irs4->aIn[i](aIn4[i]); }

                //Signal highway outputs
                irs0->aOut(aModIn0);
                irs1->aOut(aModIn1);
                irs2->aOut(aModIn2);
                irs3->aOut(aModIn3);
                irs4->aOut(aModIn4);

                //============= METHODS ===============
                SC_METHOD(bias);
                sensitive(reset);
        }

        //---------------------------------------------------------------
        void bias(){
        //---------------------------------------------------------------
                s_aVss.write(0);
                s_aVdd.write(3);
        }

        //---------------------------------------------------------------
        void ConnectAnalogCell(GenericCell *ac){
        //---------------------------------------------------------------
                da  = ac;
                da->aModIn0(aModIn0);
                da->aModIn1(aModIn1);
                da->aModIn2(aModIn2);
                da->aModIn3(aModIn3);
                da->aModIn4(aModIn4);
                da->aModOut0(aModOut0);
                da->aModOut1(aModOut1);
                da->dModIn(dModIn);
                da->aVdd(s_aVdd);
                da->aVss(s_aVss);
                da->reset(reset);
        }
};
#endif
```

SPI:

```
/*!
	\file		SPI.h
	\brief		Serial Peripheral Interface
	\author			Carsten Wulff
	\started	2001.10.4
	\modified	2001.11.19

*/

#include "msmlV2.h";

#ifndef _SPI
#define _SPI

struct SPI: sc_module{

	sc_in<bool> clk;
	sc_inout<bool> miso;
	sc_inout<bool> mosi;
	sc_in<bool> load;
	sc_inout<sc_lv<8> > dOut;
	sc_in<bool> reset;

	sc_lv<8> sreg;
	bool MSB;
	bool LSB;
	int count;

	//------------------------------------------------------------------------
	SC_CTOR(SPI){
	//------------------------------------------------------------------------
		SC_METHOD(state);
		sensitive_pos(clk);
		count = 0;

		SC_METHOD(loadReg);
		sensitive_pos(load);

		SC_METHOD(res);
		sensitive(reset);
		sensitive_pos(clk);
	}

	//------------------------------------------------------------------------
	void state(){
	//------------------------------------------------------------------------
		MSB = sreg[7].to_int();
		miso.write(MSB);
		sreg.range(1,7) = sreg.range(0,6);
		LSB = mosi.read();
		sreg[0] = LSB;
		dOut.write(sreg);

	}//end count

	//------------------------------------------------------------------------
	void loadReg(){
	//------------------------------------------------------------------------
		sreg = dOut.read();

	}

	//------------------------------------------------------------------------
	void res(){
	//------------------------------------------------------------------------
```

```
            if(reset.read()){
            sreg = "00000000";
            }
      }
};
#endif
```

ADDRREG:

```
/*!
      \file       AddrRegister.h
      \brief      Address Register, holds line and Module address
      \author           Carsten Wulff
      \started    2001.10.28
      \modified   2001.11.19

*/

#include "MsmlV2.h"

#ifndef _AddrReg
#define _AddrReg

struct AddrReg: sc_module{

      //Digital Ports
      sc_in<bool> reset;
      sc_in<bool> dEnable;
      sc_inout<sc_lv<8> > dIn;
      sc_out<sc_uint<3> > dLineAddr;
      sc_out<sc_uint<4> > dModAddr;

      //Variables
      sc_uint<3> t_LineAddr;
      sc_uint<4> t_ModAddr;

      //---------------------------------------------------------------------
      SC_CTOR(AddrReg){
      //---------------------------------------------------------------------
            SC_METHOD(SCM_Reg);
            sensitive(dEnable);
            sensitive(reset);
      }

      //---------------------------------------------------------------------
      void SCM_Reg(){
      //---------------------------------------------------------------------

            if(!reset.read()){
                  if(dEnable.read()){
                        t_LineAddr[0] = dIn.read()[0];
                        t_LineAddr[1] = dIn.read()[1];
                        t_LineAddr[2] = dIn.read()[2];
                        t_ModAddr[0] = dIn.read()[3];
                        t_ModAddr[1] = dIn.read()[4];
                        t_ModAddr[2] = dIn.read()[5];
                        t_ModAddr[3] = dIn.read()[6];
                        dLineAddr.write(t_LineAddr);
                        dModAddr.write(t_ModAddr);
                  }else{
                        dLineAddr.write(t_LineAddr);
                        dModAddr.write(t_ModAddr);
                  }
            }else{
                  dLineAddr.write(0);
                  dModAddr.write(0);
            }
```

```
	}
};
#endif
```

MODDEC:

```
/*!
	\file		ModDec.h
	\brief		Module decoder, decodes the module address
	\author			Carsten Wulff
	\started	2001.10.4
	\modified	2001.11.19

*/


#include "MsmlV2.h";


struct ModDec: sc_module{

	sc_in<sc_uint<4> > dModAddr;
	sc_out<bool> dModDecOut[16];
	sc_in<bool> dEnable;

	//------------------------------------------------------------------------
	SC_CTOR(ModDec){
	//------------------------------------------------------------------------
		SC_METHOD(decode);
		sensitive(dModAddr);
		sensitive(dEnable);
	}

	//------------------------------------------------------------------------
	void decode(){
	//------------------------------------------------------------------------

		if(dEnable.read()){

			switch(dModAddr.read()){
				case 0:
					dModDecOut[0] = 1;
					dModDecOut[1] = 0;
					dModDecOut[2] = 0;
					dModDecOut[3] = 0;
					dModDecOut[4] = 0;
					dModDecOut[5] = 0;
					dModDecOut[6] = 0;
					dModDecOut[7] = 0;
					dModDecOut[8] = 0;
					dModDecOut[9] = 0;
					dModDecOut[10] = 0;
					dModDecOut[11] = 0;
					dModDecOut[12] = 0;
					dModDecOut[13] = 0;
					dModDecOut[14] = 0;
					dModDecOut[15] = 0;
					break;
				case 1:
					dModDecOut[0] = 0;
					dModDecOut[1] = 1;
					dModDecOut[2] = 0;
					dModDecOut[3] = 0;
					dModDecOut[4] = 0;
					dModDecOut[5] = 0;
					dModDecOut[6] = 0;
					dModDecOut[7] = 0;
```

```
            dModDecOut[8] = 0;
            dModDecOut[9] = 0;
            dModDecOut[10] = 0;
            dModDecOut[11] = 0;
            dModDecOut[12] = 0;
            dModDecOut[13] = 0;
            dModDecOut[14] = 0;
            dModDecOut[15] = 0;
            break;
        case 2:
            dModDecOut[0] = 0;
            dModDecOut[1] = 0;
            dModDecOut[2] = 1;
            dModDecOut[3] = 0;
            dModDecOut[4] = 0;
            dModDecOut[5] = 0;
            dModDecOut[6] = 0;
            dModDecOut[7] = 0;
            dModDecOut[8] = 0;
            dModDecOut[9] = 0;
            dModDecOut[10] = 0;
            dModDecOut[11] = 0;
            dModDecOut[12] = 0;
            dModDecOut[13] = 0;
            dModDecOut[14] = 0;
            dModDecOut[15] = 0;
            break;
        case 3:
            dModDecOut[0] = 0;
            dModDecOut[1] = 0;
            dModDecOut[2] = 0;
            dModDecOut[3] = 1;
            dModDecOut[4] = 0;
            dModDecOut[5] = 0;
            dModDecOut[6] = 0;
            dModDecOut[7] = 0;
            dModDecOut[8] = 0;
            dModDecOut[9] = 0;
            dModDecOut[10] = 0;
            dModDecOut[11] = 0;
            dModDecOut[12] = 0;
            dModDecOut[13] = 0;
            dModDecOut[14] = 0;
            dModDecOut[15] = 0;
            break;
        case 4:
            dModDecOut[0] = 0;
            dModDecOut[1] = 0;
            dModDecOut[2] = 0;
            dModDecOut[3] = 0;
            dModDecOut[4] = 1;
            dModDecOut[5] = 0;
            dModDecOut[6] = 0;
            dModDecOut[7] = 0;
            dModDecOut[8] = 0;
            dModDecOut[9] = 0;
            dModDecOut[10] = 0;
            dModDecOut[11] = 0;
            dModDecOut[12] = 0;
            dModDecOut[13] = 0;
            dModDecOut[14] = 0;
            dModDecOut[15] = 0;
            break;
        case 5:
            dModDecOut[0] = 0;
            dModDecOut[1] = 0;
            dModDecOut[2] = 0;
            dModDecOut[3] = 0;
```

```
            dModDecOut[4] = 0;
            dModDecOut[5] = 1;
            dModDecOut[6] = 0;
            dModDecOut[7] = 0;
            dModDecOut[8] = 0;
            dModDecOut[9] = 0;
            dModDecOut[10] = 0;
            dModDecOut[11] = 0;
            dModDecOut[12] = 0;
            dModDecOut[13] = 0;
            dModDecOut[14] = 0;
            dModDecOut[15] = 0;
            break;
        case 6:
            dModDecOut[0] = 0;
            dModDecOut[1] = 0;
            dModDecOut[2] = 0;
            dModDecOut[3] = 0;
            dModDecOut[4] = 0;
            dModDecOut[5] = 0;
            dModDecOut[6] = 1;
            dModDecOut[7] = 0;
            dModDecOut[8] = 0;
            dModDecOut[9] = 0;
            dModDecOut[10] = 0;
            dModDecOut[11] = 0;
            dModDecOut[12] = 0;
            dModDecOut[13] = 0;
            dModDecOut[14] = 0;
            dModDecOut[15] = 0;
            break;
        case 7:
            dModDecOut[0] = 0;
            dModDecOut[1] = 0;
            dModDecOut[2] = 0;
            dModDecOut[3] = 0;
            dModDecOut[4] = 0;
            dModDecOut[5] = 0;
            dModDecOut[6] = 0;
            dModDecOut[7] = 1;
            dModDecOut[8] = 0;
            dModDecOut[9] = 0;
            dModDecOut[10] = 0;
            dModDecOut[11] = 0;
            dModDecOut[12] = 0;
            dModDecOut[13] = 0;
            dModDecOut[14] = 0;
            dModDecOut[15] = 0;

            break;

        case 8:
            dModDecOut[0] = 0;
            dModDecOut[1] = 0;
            dModDecOut[2] = 0;
            dModDecOut[3] = 0;
            dModDecOut[4] = 0;
            dModDecOut[5] = 0;
            dModDecOut[6] = 0;
            dModDecOut[7] = 0;
            dModDecOut[8] = 1;
            dModDecOut[9] = 0;
            dModDecOut[10] = 0;
            dModDecOut[11] = 0;
            dModDecOut[12] = 0;
            dModDecOut[13] = 0;
            dModDecOut[14] = 0;
            dModDecOut[15] = 0;
```

```
        break;
    case 9:
        dModDecOut[0] = 0;
        dModDecOut[1] = 0;
        dModDecOut[2] = 0;
        dModDecOut[3] = 0;
        dModDecOut[4] = 0;
        dModDecOut[5] = 0;
        dModDecOut[6] = 0;
        dModDecOut[7] = 0;
        dModDecOut[8] = 0;
        dModDecOut[9] = 1;
        dModDecOut[10] = 0;
        dModDecOut[11] = 0;
        dModDecOut[12] = 0;
        dModDecOut[13] = 0;
        dModDecOut[14] = 0;
        dModDecOut[15] = 0;
        break;
    case 10:
        dModDecOut[0] = 0;
        dModDecOut[1] = 0;
        dModDecOut[2] = 0;
        dModDecOut[3] = 0;
        dModDecOut[4] = 0;
        dModDecOut[5] = 0;
        dModDecOut[6] = 0;
        dModDecOut[7] = 0;
        dModDecOut[8] = 0;
        dModDecOut[9] = 0;
        dModDecOut[10] = 1;
        dModDecOut[11] = 0;
        dModDecOut[12] = 0;
        dModDecOut[13] = 0;
        dModDecOut[14] = 0;
        dModDecOut[15] = 0;
        break;
    case 11:
        dModDecOut[0] = 0;
        dModDecOut[1] = 0;
        dModDecOut[2] = 0;
        dModDecOut[3] = 0;
        dModDecOut[4] = 0;
        dModDecOut[5] = 0;
        dModDecOut[6] = 0;
        dModDecOut[7] = 0;
        dModDecOut[8] = 0;
        dModDecOut[9] = 0;
        dModDecOut[10] = 0;
        dModDecOut[11] = 1;
        dModDecOut[12] = 0;
        dModDecOut[13] = 0;
        dModDecOut[14] = 0;
        dModDecOut[15] = 0;
        break;
    case 12:
        dModDecOut[0] = 0;
        dModDecOut[1] = 0;
        dModDecOut[2] = 0;
        dModDecOut[3] = 0;
        dModDecOut[4] = 0;
        dModDecOut[5] = 0;
        dModDecOut[6] = 0;
        dModDecOut[7] = 0;
        dModDecOut[8] = 0;
        dModDecOut[9] = 0;
        dModDecOut[10] = 0;
        dModDecOut[11] = 0;
```

```
                        dModDecOut[12] = 1;
                        dModDecOut[13] = 0;
                        dModDecOut[14] = 0;
                        dModDecOut[15] = 0;
                        break;
                case 13:
                        dModDecOut[0] = 0;
                        dModDecOut[1] = 0;
                        dModDecOut[2] = 0;
                        dModDecOut[3] = 0;
                        dModDecOut[4] = 0;
                        dModDecOut[5] = 0;
                        dModDecOut[6] = 0;
                        dModDecOut[7] = 0;
                        dModDecOut[8] = 0;
                        dModDecOut[9] = 0;
                        dModDecOut[10] = 0;
                        dModDecOut[11] = 0;
                        dModDecOut[12] = 0;
                        dModDecOut[13] = 1;
                        dModDecOut[14] = 0;
                        dModDecOut[15] = 0;
                        break;
                case 14:
                        dModDecOut[0] = 0;
                        dModDecOut[1] = 0;
                        dModDecOut[2] = 0;
                        dModDecOut[3] = 0;
                        dModDecOut[4] = 0;
                        dModDecOut[5] = 0;
                        dModDecOut[6] = 0;
                        dModDecOut[7] = 0;
                        dModDecOut[8] = 0;
                        dModDecOut[9] = 0;
                        dModDecOut[10] = 0;
                        dModDecOut[11] = 0;
                        dModDecOut[12] = 0;
                        dModDecOut[13] = 0;
                        dModDecOut[14] = 1;
                        dModDecOut[15] = 0;
                        break;
                case 15:
                        dModDecOut[0] = 0;
                        dModDecOut[1] = 0;
                        dModDecOut[2] = 0;
                        dModDecOut[3] = 0;
                        dModDecOut[4] = 0;
                        dModDecOut[5] = 0;
                        dModDecOut[6] = 0;
                        dModDecOut[7] = 0;
                        dModDecOut[8] = 0;
                        dModDecOut[9] = 0;
                        dModDecOut[10] = 0;
                        dModDecOut[11] = 0;
                        dModDecOut[12] = 0;
                        dModDecOut[13] = 0;
                        dModDecOut[14] = 0;
                        dModDecOut[15] = 1;
                        break;
                default:
                        break;
        }
}
if(!dEnable.read()){
        dModDecOut[0] = 0;
        dModDecOut[1] = 0;
        dModDecOut[2] = 0;
        dModDecOut[3] = 0;
```

```
                        dModDecOut[4] = 0;
                        dModDecOut[5] = 0;
                        dModDecOut[6] = 0;
                        dModDecOut[7] = 0;
                        dModDecOut[8] = 0;
                        dModDecOut[9] = 0;
                        dModDecOut[10] = 0;
                        dModDecOut[11] = 0;
                        dModDecOut[12] = 0;
                        dModDecOut[13] = 0;
                        dModDecOut[14] = 0;
                        dModDecOut[15] = 0;
                }
        }
};
```

TOC:

```
/*!
        \file           TOC.h
        \brief          Table of Content
        \author                 Carsten Wulff
        \started        2001.11.1
        \modified       2001.11.19

*/


#include "msmlV2.h";

#ifndef _TOC
#define _TOC

struct TOC: sc_module{

        sc_in<bool> read;
        sc_in<sc_uint<4> > dModAddr;
        sc_in<sc_uint<3> > dLineAddr;

        sc_inout<sc_lv<8> > dBus;

        sc_uint<4> addrReg;
        sc_lv<8> data;
        sc_uint<8> t_addr;

        //------------------------------------------------------------------------
        SC_CTOR(TOC){
        //------------------------------------------------------------------------
                SC_METHOD(ROMCntr);
                sensitive(read);
        }

        //------------------------------------------------------------------------
        void ROMCntr(){
        //------------------------------------------------------------------------
                if(read.read()){
                        ROM(dModAddr.read(),dLineAddr.read());
                        dBus.write(data);
                }
        }

        //------------------------------------------------------------------------
        void ROM(sc_uint<4> dModAddr,sc_uint<3> dLineAddr){
        //------------------------------------------------------------------------
                switch(dModAddr){
```

```
case 0:
       switch(dLineAddr){
              case 0:              data = "00000001"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 1:
       switch(dLineAddr){
              case 0:              data = "00000010"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 2:
       switch(dLineAddr){
              case 0:              data = "00000011"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 3:
       switch(dLineAddr){
              case 0:              data = "00000100"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 4:
       switch(dLineAddr){
              case 0:              data = "00000101"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 5:
       switch(dLineAddr){
              case 0:              data = "00000110"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 6:
       switch(dLineAddr){
              case 0:              data = "00000110"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 7:
       switch(dLineAddr){
              case 0:              data = "00000110"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 8:
       switch(dLineAddr){
              case 0:              data = "00000110"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
       }
       break;
case 9:
       switch(dLineAddr){
              case 0:              data = "00000111"; break;
              case 1:              data = "00000001"; break;
              default:      data = "00000000"; break;
```

```
                        }
                        break;
                case 10:
                        switch(dLineAddr){
                                case 0:             data = "00000111"; break;
                                case 1:             data = "00000001"; break;
                                default:    data = "00000000"; break;
                        }
                        break;
                case 11:
                        switch(dLineAddr){
                                case 0:             data = "00000111"; break;
                                case 1:             data = "00000001"; break;
                                default:    data = "00000000"; break;
                        }
                        break;
                case 12:
                        switch(dLineAddr){
                                case 0:             data = "00000111"; break;
                                case 1:             data = "00000001"; break;
                                default:    data = "00000000"; break;
                        }
                        break;
                case 13:
                        switch(dLineAddr){
                                case 0:             data = "00000000"; break;
                                case 1:             data = "00000000"; break;
                                default:    data = "00000000"; break;
                        }
                        break;
                case 14:
                        switch(dLineAddr){
                                case 0:             data = "00000000"; break;
                                case 1:             data = "00000000"; break;
                                default:    data = "00000000"; break;
                        }
                        break;
                case 15:
                        switch(dLineAddr){
                                case 0:             data = "00000000"; break;
                                case 1:             data = "00000000"; break;
                                default:    data = "00000000"; break;
                        }
                        break;
                default: data = "00000000"; break;
        }
    }
};
#endif
```

CONTROL:

```
/*!
        \file           Control.h
        \brief          Control signal decoder
        \author                 Carsten Wulff
        \started        2001.10.4
        \modified       2001.11.19

*/

#include "MsmlV2.h"

#ifndef _Control
#define _Control

struct Control : sc_module{
```

```
sc_in<sc_lv<3> > Cntr;
sc_in<bool> enable;

sc_out<bool> dCntOut[8];

sc_uint<3> t_cntr;

//------------------------------------------------------------------------
SC_CTOR(Control){
//------------------------------------------------------------------------

	SC_METHOD(Decide);
	sensitive(Cntr);
}

//------------------------------------------------------------------------
void Decide(){
//------------------------------------------------------------------------
	if(enable.read()){
	t_cntr = Cntr.read();
		switch(t_cntr){
			case 0:
				dCntOut[0] = 1;
				dCntOut[1] = 0;
				dCntOut[2] = 0;
				dCntOut[3] = 0;
				dCntOut[4] = 0;
				dCntOut[5] = 0;
				dCntOut[6] = 0;
				dCntOut[7] = 0;
				break;
			case 1:
				dCntOut[0] = 0;
				dCntOut[1] = 1;
				dCntOut[2] = 0;
				dCntOut[3] = 0;
				dCntOut[4] = 0;
				dCntOut[5] = 0;
				dCntOut[6] = 0;
				dCntOut[7] = 0;
				break;
			case 2:
				dCntOut[0] = 0;
				dCntOut[1] = 0;
				dCntOut[2] = 1;
				dCntOut[3] = 0;
				dCntOut[4] = 0;
				dCntOut[5] = 0;
				dCntOut[6] = 0;
				dCntOut[7] = 0;
				break;
			case 3:
				dCntOut[0] = 0;
				dCntOut[1] = 0;
				dCntOut[2] = 0;
				dCntOut[3] = 1;
				dCntOut[4] = 0;
				dCntOut[5] = 0;
				dCntOut[6] = 0;
				dCntOut[7] = 0;
				break;
			case 4:
				dCntOut[0] = 0;
				dCntOut[1] = 0;
				dCntOut[2] = 0;
				dCntOut[3] = 0;
				dCntOut[4] = 1;
```

```
                                        dCntOut[5] = 0;
                                        dCntOut[6] = 0;
                                        dCntOut[7] = 0;
                                        break;
                                case 5:
                                        dCntOut[0] = 0;
                                        dCntOut[1] = 0;
                                        dCntOut[2] = 0;
                                        dCntOut[3] = 0;
                                        dCntOut[4] = 0;
                                        dCntOut[5] = 1;
                                        dCntOut[6] = 0;
                                        dCntOut[7] = 0;
                                        break;
                                case 6:
                                        dCntOut[0] = 0;
                                        dCntOut[1] = 0;
                                        dCntOut[2] = 0;
                                        dCntOut[3] = 0;
                                        dCntOut[4] = 0;
                                        dCntOut[5] = 0;
                                        dCntOut[6] = 1;
                                        dCntOut[7] = 0;
                                        break;
                                case 7:
                                        dCntOut[0] = 0;
                                        dCntOut[1] = 0;
                                        dCntOut[2] = 0;
                                        dCntOut[3] = 0;
                                        dCntOut[4] = 0;
                                        dCntOut[5] = 0;
                                        dCntOut[6] = 0;
                                        dCntOut[7] = 1;
                                        break;
                                default:

                                        break;
                                }
                        }else{
                                dCntOut[0] = 0;
                                dCntOut[1] = 0;
                                dCntOut[2] = 0;
                                dCntOut[3] = 0;
                                dCntOut[4] = 0;
                                dCntOut[5] = 0;
                                dCntOut[6] = 0;
                                dCntOut[7] = 0;
                        }

        }
};

#endif
```

PAIC_ANALOGMODULES:

```
/*!
        \file           PAIC_AnalogModules.h
        \brief          All Analog modules in paic and routing network
        \author                 Carsten Wulff
        \started        2001.10.19
        \modified       2001.11.19

*/

#include "MsmlV2.h"
#include "ModDec.h"
```

```
#include "AnModFramework.h"
#include "analog_cells.h"

struct PAIC_AnalogModules: sc_module{

	sc_in<bool> reset;
	sc_in<bool> enable;
	sc_in<bool> load;
	sc_in<bool> write;

	sc_in<sc_uint<4> > dModAddr;
	sc_in<sc_uint<3> > dLineAddr;
	sc_inout<sc_lv<8> > dIn;

	sc_inout<voltage> aPAICIn[4];
	sc_inout<voltage> aPAICOut[4];


	sc_signal<bool> ModDecOut[16];

	sc_signal<voltage> aOut0[16];
	sc_signal<voltage> aOut1[16];

	AnModFramework *aMod[16];
	ModDec *modDec;

	//---------------------------------------------------------------------
	SC_CTOR(PAIC_AnalogModules){
	//---------------------------------------------------------------------
		int i;
		int z;
		//Module Mux
		modDec = new ModDec("modDec");

		modDec->dEnable(enable);
		modDec->dModAddr(dModAddr);
		for(i=0;i<16;i++){
			modDec->dModDecOut[i](ModDecOut[i]);
		}

		//================ Analog Framework and cells ==================
		int nrAnalogModules = 13;

		//aMod[0] == 8 bit DAC
		aMod[0] = new AnModFramework("aMod0");
		aMod[0]->ConnectAnalogCell( new DAC_8("DAC_8"));

		//aMod[1] == BandGap
		aMod[1] = new AnModFramework("aMod1");
		aMod[1]->ConnectAnalogCell( new Vreference("Vreference"));

		//aMod[2] == DiffComp
		aMod[2] = new AnModFramework("aMod2");
		aMod[2]->ConnectAnalogCell( new DiffComp("DiffComp"));

		//aMod[3] == DiffOta
		aMod[3] = new AnModFramework("aMod3");
		aMod[3]->ConnectAnalogCell( new DiffOta("DiffOta"));

		//aMod[4] == SC_Integrator
		aMod[4] = new AnModFramework("aMod4");
		aMod[4]->ConnectAnalogCell( new SwitCapIntegrator("SCIntegrator"));


		//aMod[5] == Res_Array1
		aMod[5] = new AnModFramework("aMod5");
		aMod[5]->ConnectAnalogCell( new ResArray("Res_Array1"));
```

```
//aMod[6] == Res_Array2
aMod[6] = new AnModFramework("aMod6");
aMod[6]->ConnectAnalogCell( new ResArray("Res_Array2"));

//aMod[7] == Res_Array3
aMod[7] = new AnModFramework("aMod7");
aMod[7]->ConnectAnalogCell( new ResArray("Res_Array3"));

//aMod[8] == Res_Array4
aMod[8] = new AnModFramework("aMod8");
aMod[8]->ConnectAnalogCell( new ResArray("Res_Array4"));


//aMod[9] == CapArray1
aMod[9] = new AnModFramework("aMod9");
aMod[9]->ConnectAnalogCell( new CapArray("CapArray1"));

//aMod[10] == CapArray2
aMod[10] = new AnModFramework("aMod10");
aMod[10]->ConnectAnalogCell( new CapArray("CapArray2"));

//aMod[11] == CapArray3
aMod[11] = new AnModFramework("aMod11");
aMod[11]->ConnectAnalogCell( new CapArray("CapArray3"));

//aMod[12] == CapArray4
aMod[12] = new AnModFramework("aMod12");
aMod[12]->ConnectAnalogCell( new CapArray("CapArray4"));

for(i=0;i<nrAnalogModules;i++){
       //aMod[i]->clock(clock);
       aMod[i]->reset(reset);
       aMod[i]->dLoad(load);
       aMod[i]->dWrite(write);
       aMod[i]->dEnable(ModDecOut[i]);
       aMod[i]->dLineAddr(dLineAddr);
       aMod[i]->dIn(dIn);

       for(z=0;z<4;z++){
       aMod[i]->aGlobalOut[z](aPAICOut[z]);
       }

       for(z=0;z<4;z++){
              aMod[i]->aIn0[z](aPAICIn[z]);
              aMod[i]->aIn1[z](aPAICIn[z]);
       }

       aMod[i]->aOut0(aOut0[i]);
       aMod[i]->aOut1(aOut1[i]);


       if(i < 5){
       //Signal Routing


              aMod[i]->aIn0[4](aOut0[5]);
              aMod[i]->aIn0[5](aOut0[6]);
              aMod[i]->aIn0[6](aOut0[7]);
              aMod[i]->aIn0[7](aOut0[8]);

              aMod[i]->aIn2[0](aOut0[9]);
              aMod[i]->aIn2[1](aOut0[10]);
              aMod[i]->aIn2[2](aOut0[11]);
              aMod[i]->aIn2[3](aOut0[12]);

       }
}
```

```
            //Signal Routing
            aMod[2]->aIn1[4](aOut0[0]);
      }

};
```

PAICWTOC:

```
/*!
      \file         PaicwTOC.h
      \brief        Top level PAIC
      \author             Carsten Wulff
      \started      2001.10.4
      \modified     2001.11.19

*/

#include "MsmlV2.h"
#include "Control.h"
#include "SPI.h"
#include "AddrRegister.h"
#include "PAIC_AnalogModules.h"
#include "TOC.h"

struct paicwtoc: sc_module{

      int i;
      //sc_in<bool> clock;
      sc_in<bool> reset;
      sc_inout<bool> miso;
      sc_inout<bool> mosi;
      sc_in<bool> sck;
      sc_in<bool> enable;
      sc_in<sc_lv<3> > Cntr;

      sc_inout<voltage> aPAICIn[4];
      sc_inout<voltage> aPAICOut[4];


      //Signals
      sc_signal<bool> IDLE;
      sc_signal<bool> ModuleWrite;
      sc_signal<bool> ModuleReadback;
      sc_signal<bool> AddrEnable;
      sc_signal<bool> Cnt6;
      sc_signal<bool> TOCread;
      sc_signal<bool> SPIload;
      sc_signal<bool> Cnt7;
      sc_signal<sc_uint<4> > dModAddr;
      sc_signal<sc_uint<3> > dLineAddr;
      sc_signal<sc_lv<8> > dBus;



      sc_signal<bool> readback;

      //Objects
      PAIC_AnalogModules *paic;
      Control *cnt;
      SPI *spi;
      AddrReg *ad;
      TOC *toc;


      //------------------------------------------------------------------------
      SC_CTOR(paicwtoc){
      //------------------------------------------------------------------------
```

```
		spi = new SPI("spi");
		spi->mosi(mosi);
		spi->miso(miso);
		spi->clk(sck);
		spi->dOut(dBus);
		spi->reset(reset);
		spi->load(SPIload);


		ad = new AddrReg("ad");
		//ad->clock(clock);
		ad->reset(reset);
		ad->dEnable(AddrEnable);
		ad->dIn(dBus);
		ad->dModAddr(dModAddr);
		ad->dLineAddr(dLineAddr);

		toc = new TOC("toc");
		//toc->clock(clock);
		toc->read(TOCread);
		toc->dBus(dBus);
		toc->dModAddr(dModAddr);
		toc->dLineAddr(dLineAddr);

		cnt = new Control("cnt");
		cnt->Cntr(Cntr);
		cnt->enable(enable);
		cnt->dCntOut[0](IDLE);
		cnt->dCntOut[1](AddrEnable);
		cnt->dCntOut[2](ModuleWrite);
		cnt->dCntOut[3](ModuleReadback);
		cnt->dCntOut[4](TOCread);
		cnt->dCntOut[5](SPIload);
		cnt->dCntOut[6](Cnt6);
		cnt->dCntOut[7](Cnt7);

		paic = new PAIC_AnalogModules("paic");
		//paic->clock(clock);
		paic->write(ModuleWrite);
		paic->reset(reset);
		paic->load(ModuleReadback);
		paic->enable(enable);

		paic->dModAddr(dModAddr);
		paic->dLineAddr(dLineAddr);
		paic->dIn(dBus);

		for(i=0;i<4;i++){
			paic->aPAICIn[i](aPAICIn[i]);
			paic->aPAICOut[i](aPAICOut[i]);
		}


	}

};
```

GENERIC CELL:

```
/*!
	\file		Generic cell.h
	\brief		Interconnect parent of all analog modules.
	\author			Carsten Wulff
	\started	2001.10.18
```

```
       \modified    2001.11.19

*/
#include "MsmlV2.h"

#ifndef _GENERICCELL
#define _GENERICCELL

struct GenericCell : sc_module{
       sc_inout<voltage> aModOut0;
       sc_inout<voltage> aModOut1;

       sc_inout<voltage> aModIn0;
       sc_inout<voltage> aModIn1;
       sc_inout<voltage> aModIn2;
       sc_inout<voltage> aModIn3;
       sc_inout<voltage> aModIn4;
       sc_in<voltage> aVdd;
       sc_in<voltage> aVss;

       sc_in<sc_uint<8> > dModIn;
       sc_in<bool> reset;
};
#endif
```

TESTBENCH:

```
//TestPaicwTOC.h


#include "MsmlV2.h"

#include "PaicwTOC.h"


struct TestPAICwTOC{

       int i;
       sc_signal<bool> reset;
       sc_signal<bool> enable;
       sc_signal<sc_lv<3> > Cntr;

       sc_signal<bool> sck;
       sc_signal<bool> miso;
       sc_signal<bool> mosi;

       sc_signal<voltage> aPAICIn[4];
       sc_signal<voltage> aPAICOut[4];

       sc_lv<8> dOut;

       paicwtoc *paic;
       sc_trace_file * trace_file;

       //---------------------------------------------------------------------
       void run(){
       //---------------------------------------------------------------------


              paic = new paicwtoc("paic");
              paic->reset(reset);
              paic->miso(miso);
              paic->mosi(mosi);
              paic->sck(sck);
              paic->enable(enable);
              paic->Cntr(Cntr);
```

```
        for(i=0;i<4;i++){
              paic->aPAICIn[i](aPAICIn[i]);
              paic->aPAICOut[i](aPAICOut[i]);
        }

        sc_initialize();
        trace_file = sc_create_vcd_trace_file("TestPAIC");

        sc_trace(trace_file,reset, "reset");
        sc_trace(trace_file,Cntr,"Cntr");
        sc_trace(trace_file,miso,"miso");
        sc_trace(trace_file,mosi,"mosi");
        sc_trace(trace_file,sck,"sck");
        sc_trace(trace_file,dOut,"dOut");

        //=== INTERFACE TEST ===
        InterfaceTest();

        //=== PAIC Tests ===
        //ConfigDiffOta();
        ADC();
        //TOCTest();
        //ReadbackTest();

        sc_stop();
        sc_close_vcd_trace_file(trace_file);
}



//=== INTERFACE TEST ===
//-----------------------------------------------------------------------
void InterfaceTest(){
//-----------------------------------------------------------------------

        sc_lv<8> t_output;

        sc_trace(trace_file,paic->dBus,"paic_dBus");
        sc_trace(trace_file,paic->dLineAddr,"dLineAddr");
        sc_trace(trace_file,paic->dModAddr,"dModAddr");

        sc_trace(trace_file,paic->AddrEnable,"AddrEnable");
        sc_trace(trace_file,paic->ModuleWrite,"ModuleWrite");
        sc_trace(trace_file,paic->ModuleReadback,"ModuleReadBack");
        sc_trace(trace_file,paic->TOCread,"TOCread");
        sc_trace(trace_file,paic->SPIload,"SPILoad");

        sc_trace(trace_file,aPAICOut[0].read(),"aPAICOut[0]");
        sc_trace(trace_file,aPAICOut[1].read(),"aPAICOut[1]");
        sc_trace(trace_file,aPAICOut[2].read(),"aPAICOut[2]");
        sc_trace(trace_file,aPAICOut[3].read(),"aPAICOut[3]");

        sc_trace(trace_file,aPAICIn[0].read(),"aPAICIn[0]");
        sc_trace(trace_file,aPAICIn[1].read(),"aPAICIn[1]");
        sc_trace(trace_file,aPAICIn[2].read(),"aPAICIn[2]");
        sc_trace(trace_file,aPAICIn[3].read(),"aPAICIn[3]");

        wCntr("000");
        sck = false;
        enable = true;

        //Reset Chip
        sc_cycle(1);
        reset = true;
        sc_cycle(1);
        reset = false;

        //Connect aPAICIn0 to IRS0 out
```

```
        wSPI("00010011");
        wCntr("001");

        wSPI("00000001");
        wCntr("010");

        //Connect aPAICIn1 to IRS1 out
        wSPI("00010100");
        wCntr("001");

        wSPI("00000010");
        wCntr("010");

        //Connect OSR0 out to aPAICOut0
        wSPI("00010000");
        wCntr("001");

        wSPI("00000001");
        wCntr("010");

        //Connect OSR1 out to aPAICOut1
        wSPI("00010001");
        wCntr("001");

        wSPI("00000010");
        wCntr("010");

        bool toggle = true;
        for(i=0;i<10;i++){
               aPAICIn[0] = 0.5;
               if(toggle){
                      aPAICIn[1] =0;
                      toggle = false;
               }else{
                      aPAICIn[1] =1;
                      toggle = true;
               }
               sc_cycle(1);
        }


        sc_cycle(1);
        sc_cycle(1);
        sc_cycle(1);



}

//=== READBACK TEST ====
//----------------------------------------------------------------------
void ReadbackTest(){
//----------------------------------------------------------------------

        int mod = 0;
        int line = 3;
        int z;

        sc_lv<5> modAddr;
        sc_lv<3> lineAddr;
        sc_lv<8> t_addr;

        for(i=0;i<13;i++){
               //Create address string for ORS0
               modAddr = i;
               t_addr.range(2,0) = "000";
               t_addr.range(7,3) = modAddr;
```

```
        //Set ORS0 to low
        wSPI(t_addr);
        wCntr("001");
        wSPI("00000101");
        wCntr("010");

        //Create address string for ORS1
        modAddr = i;
        t_addr.range(2,0) = "001";
        t_addr.range(7,3) = modAddr;

        //Set ORS1 to low
        wSPI(t_addr);
        wCntr("001");
        wSPI("00000101");
        wCntr("010");
}

for(mod=0;mod<13;mod++){
        for(line = 3;line<8;line++){

                for(i=0;i<13;i++){
                        for(z=0;z<2;z++){

                        //Create address string for ORS(z)
                        lineAddr = z;
                        modAddr = i;
                        t_addr.range(2,0) = lineAddr;
                        t_addr.range(7,3) = modAddr;

                        //Set ORS(z) high
                        wSPI(t_addr);
                        wCntr("001");
                        wSPI("00000110");
                        wCntr("010");

                        //Create address string for IRS(line)
                        lineAddr = line;
                        modAddr = mod;
                        t_addr.range(2,0) = lineAddr;
                        t_addr.range(7,3) = modAddr;

                        //Load SPI from the IRS(line) register
                        wSPI(t_addr);
                        wCntr("001"); //load addr
                        wCntr("011"); //read analog input
                        wCntr("101"); //load paic spi register

                        //Create address string for ORS(z)
                        lineAddr = z;
                        modAddr = i;
                        t_addr.range(2,0) = lineAddr;
                        t_addr.range(7,3) = modAddr;

                        //Load addr and readback data
        wSPI(t_addr);
                        wCntr("001");

                        //SAVE RESULTS
                        if(dOut.or_reduce()){
                                //save readback if connection
                                readbackSave(mod,line,i,z,dOut);
        }

                        //Set ORS(z) low
                        wSPI("0000101");
                        wCntr("010");
                        }//output end
```

```
                        }
                }//line end
        }//mod end

}//ReadbacktTest end

//=== TOC TEST ===
//-----------------------------------------------------------------------
void TOCTest(){
//-----------------------------------------------------------------------
        int mod = 0;
        int line = 3;
        int z;

        sc_lv<5> modAddr;
        sc_lv<3> lineAddr;
        sc_lv<8> t_addr;

        for(mod=0;mod<16;mod++){
                for(line=0;line<8;line++){
                        lineAddr = line;
                        modAddr = mod;
                        t_addr.range(2,0) = lineAddr;
                        t_addr.range(7,3) = modAddr;
                        wSPI(t_addr);
                        wCntr("001");
                        wCntr("100");
                        wCntr("101");
                        wSPI("01010101");
                        TOCSave(mod,line,dOut);
                }

        }



}

//=== ADC TEST ===
//-----------------------------------------------------------------------
void ADC(){
//-----------------------------------------------------------------------
        sc_lv<8> tout;

        //=== Connect Comparator and DAC ===
        //aPAICIn0 => DiffComp->IRS0->aIn0
        wSPI("00010011");
        wCntr("001");
        wSPI("00000001");
        wCntr("010");

        //DAC->ORS0->aOut => DiffComp->IRS1->aIn4
        wSPI("00010100");
        wCntr("001");
        wSPI("00010000");
        wCntr("010");

        //DiffComp->ORS0->aOut => aPAICOut0
        wSPI("00010000");
        wCntr("001");
        wSPI("00000001");
        wCntr("010");

        //DiffComp->ORS1->aOut => aPAICOut1
        wSPI("00010001");
        wCntr("001");
        wSPI("00000010");
```

```
        wCntr("010");


        double analogVal = 0.00390625;
        double increment;
        increment = 0.0078125;
        int z;

        int valsub[8];
        valsub[0] = 64;
        valsub[1] = 32;
        valsub[2] = 16;
        valsub[3] = 8;
        valsub[4] = 4;
        valsub[5] = 2;
        valsub[6] = 1;
        valsub[7] = 1;

        for(z=0;z<256;z++){

               //Update analog input value
               aPAICIn[0].write(analogVal);
               sc_cycle(1);sc_cycle(1);

               //== DAC ==
               int data = 128;
               sc_uint<8> result;

               //Write DAC address into AddrReg
               wSPI("00000010");
               wCntr("001");

               //Convert analog value
               for(i=0;i<8;i++){
                      wSPI(data);
                      wCntr("010");

                      if(aPAICOut[0].read() == 3){
                             data = data + valsub[i] ;
                             result[7-i] = 1;
                      }else{
                             data = data - valsub[i] ;
                             result[7-i] = 0;
                      }
               }

               //Save result to file
               AdcSave(analogVal,result,aPAICOut[0].read());

               //Show the final result in the waveform viewer
               wSPI(result);
               wCntr("010");

               //Increment analog value
               analogVal += increment;

               sc_cycle(1);sc_cycle(1);sc_cycle(1);
        }//end for
}

//---------------------------------------------------------------------
void wCntr(sc_lv<3> cnt){
//---------------------------------------------------------------------
        Cntr.write(cnt);
        sc_cycle(1);
        Cntr.write("000");
        sc_cycle(1);
```

```
	}


	//---------------------------------------------------------------------
	void wSPI(sc_lv<8> input){
	//---------------------------------------------------------------------
		sc_lv<8> sreg;
		int i;
		bool LSB;
		bool MSB;
		sreg = input;
		for(i=0;i<8;i++){
			MSB = sreg[7].to_int();
			mosi.write(MSB);
			sreg.range(1,7) = sreg.range(0,6);
			sck = false;sc_cycle(1);sck=true;sc_cycle(1);
			LSB  = miso.read();
			sreg[0] = LSB;
			dOut = sreg;
		}
		sck = false;
	}



	//---------------------------------------------------------------------
	void  readbackSave(sc_uint<4>  curMod,  sc_uint<3>  curLA,  sc_uint<4>  conMod,
sc_uint<3> conLA, sc_lv<8> data){
	//---------------------------------------------------------------------
		FILE *out = fopen("readbackPAIC.dat","a+");
		fprintf(out,"%d\t",curMod);
		fprintf(out,"%d\t",curLA);
		fprintf(out,"%d\t",conMod);
		fprintf(out,"%d\t",conLA);

	fprintf(out,"%d%d%d%d%d%d%d%d\n",data[7].to_int(),data[6].to_int(),data[5].to
_int(),data[4].to_int(),data[3].to_int(),data[2].to_int(),data[1].to_int(),data[0].
to_int());
		fclose(out);

	}

	//---------------------------------------------------------------------
	void TOCSave(sc_uint<4> curMod, sc_uint<3> curLA, sc_lv<8> data){
	//---------------------------------------------------------------------
		FILE *out = fopen("TOCReadDATA.dat","a+");
		fprintf(out,"%d\t",curMod);
		fprintf(out,"%d\t",curLA);

	fprintf(out,"%d%d%d%d%d%d%d%d\n",data[7].to_int(),data[6].to_int(),data[5].to
_int(),data[4].to_int(),data[3].to_int(),data[2].to_int(),data[1].to_int(),data[0].
to_int());
		fclose(out);

	}

	//---------------------------------------------------------------------
	void AdcSave(double f, sc_uint<8> i, double compval){
	//---------------------------------------------------------------------
		FILE *out = fopen("ADCPAIC.dat","a+");
		fprintf(out,"%f\t",f);
		fprintf(out,"%d\t",i);
		fprintf(out,"%f\t",compval);
		fprintf(out,"\n");
		fclose(out);

	}
```

```
	//-------------------------------------------------------------------
	void sName(char *name, int i, char *retstr){
	//-------------------------------------------------------------------
		char ctemp[20];
		char si[10];
		strcpy(ctemp, name);
		strcat(ctemp,"(");
		strcat(ctemp, itoa(i,si,10));
		strcat(ctemp, ")");
		strcpy(retstr,ctemp);
		//return ctemp;
	}


};
```

APPENDIX VI: PAIC-CDROM

The PAIC-CDROM is included at the back of this report.