The ITB Journal

Volume 3 | Issue 2 Article 12

2002

Hardware and Software Codesign for Multimedia Capable Portable Devices using SystemC

Richard Gallery

Deepesh Shakya

Hardware and Software Codesign for Multimedia Capable Portable Devices using SystemC

Richard Gallery Deepesh M. Shakya

Institute of Technology, Blanchardstown
Blanchardstown, Dublin
email: {Richard.Gallery, Deepesh.Shakya}@itb.ie

Abstract

Multimedia capable portable devices such as 3G phones will host a variety of new applications. Although the underlying push for new applications in such devices is driven by the increase in bandwidth offered by 3G, it is clear that many of the "new" applications will require the provision of new and powerful graphics/video technology within the mobile device itself. Within a computing device, high bandwidth and computational cost are associated with anything but the simplest of graphics, and as a result the graphics subsystem is generally one of the most critical elements of a system, requiring particular attention in the design process. The project is examining the suitability of SystemC, a system description language, for Hardware/Software Codesign of a graphics system in a typical next generation WAP compatible device.

1. Introduction

Multimedia capable portable devices, one early example of which is the WAP compatible mobile phone, incorporate a combination of computing, wireless communications, signal processing, graphics and other technologies to provide a cost effective solution to the customer. Custom designed IC's are normally required in order to implement these demanding technologies in a cost-effective manner. Increasingly such IC's are Systems-On-A-Chip, where a variety of digital, analogue and software technologies are integrated together in the interests of cost reduction.

Hardware/software co-design plays a vital role throughout such development projects, especially during the initial stages where critical decisions regarding hardware software partitioning and system constraints are taken. However, tools to allow hardware/software co-design are not yet fully developed. In a typical hardware/software codesign, the systems level engineers would model and program in C/C++, whereas the hardware engineers would design using a hardware description language such as VHDL, leading to communication and other difficulties within a project team. Open SystemC is an industry led initiative which addresses this issue by seeking to establish a modeling platform that promotes and accelerates system

models using standard ANSI C++ with SystemC extensions (in the form of classes and data structures), which can then be proven using simulation tools before transfer to silicon.

2. Next Generation Graphics/Video requirements for Multimedia Capable Portable Devices

The next generation of mobile telecommunications is capable of providing data rates of up to 2 Megabits per second 0. This offers the prospect of broadband video and multimedia services on the move, such as Video Conferencing, on-line entertainment and Internet access. For all these features, graphics/video system of the mobile system should be designed with appropriate functionality.

A part of the project deals in the study of graphics/video requirements for next generation mobile phones. Some of the applications to be made available in next generation mobile phones include Video Conferencing, Video Streaming, Multimedia Messaging Service, Gaming, Integrated Digital Camera and Camcorder, Video Clips play back etc. In order to support these applications, the devices must be enabled with appropriate graphics/video functionalities. For e.g. Video Conferencing and Video Streaming demand good image quality. To ensure this an appropriate encoding/decoding standards must be adopted.

To support fully fledged 3D gaming, the device must be enhanced with a 3D graphics engine which supports perspective correct texture mapping, bilinear, MIP-mapping, Gouraud Shading, alpha-blending, Stippling, anti-aliasing, fogging and Z-buffering.

The device must also support other graphics functionalities like scaling, scrolling, vertical and horizontal filtering, multiple video overlays etc. Also, the device must be equipped with image grabbing functionality to enable it serve as a digital camera or camcorder.

3. An Overview of video mixing

Graphics/video systems are often capable of generating overlayed or composite images (which enables menus, graphic overlays, picture in picture etc.). The video/graphics subsystem capable of performing this task is usually termed the mixer. The simulation of a mixer is carried out as an initial aspect of the overall project to gain familiarity with the issues arising in graphics/video systems.

Mixing in a video/graphics system may be described as the combination of two images in which one image (the overlay image) is overlayed on another image (the primary image) according to an *alpha* value[4].

For each pixel in an image the output pixel that results from the mixer is governed by the following expression:

$$\text{pixel}_{\text{output image}} = \alpha \; x \; \text{pixel}_{\text{primary image}} + (1.0 - \alpha) \; x \; \text{pixel}_{\text{overlay image}}$$

Normalisation of the output value is assured through the use of *alpha* and (*1.0-alpha*). The value of alpha varies from 0.0 and 1.0.

The *alpha* value may be fixed for the entire image, in which case the images are combined in a similar manner at each pixel. In practice a system that can only achieve this would be of little value. A more useful mixer results if the alpha value may be allowed to vary such that there is a separate alpha value at each pixel. This approach allows the manner in which images are combined to be varied from one pixel to the next. Other approaches could involve fixing the alpha value for some portions of the image, and varying it for others.

Although the concept above is illustrated using floating-point calculations, in practice floating point arithmetic is not required to perform these calculations. Instead the *alpha* values may be represented as integers in the range 0..255[5]. In this case integer arithmetic may be used. Each multiplication of 8 bit numbers will produce a 16-bit result, but the lower 8 bits may be disregarded, to produce an 8-bit output.

$$\text{pixel}_{\text{output image}} = \alpha \; x \; \text{pixel}_{\text{primary image}} + (255 - \alpha) \; x \; \text{pixel}_{\text{overlay image}}$$

Figure 3-3 shows the output image obtained after mixing images in Figure 3-1 and Figure 3-2. The value of alpha taken for this mixing operation is 127 (applied across the entire image).

[4] In our research a base image format of RGB 8:8:8 with an additional 8 bit alpha value is assumed.

[5] Or other ranges if appropriate to the system requirements, for example some systems might find a more restricted alpha range sufficient.

Figure 3-1 Primary Image

Figure 3-2 Overlay Image

In order to gain familiarity with the techniques involved, an algorithmic simulation of the mixer was carried out. This algorithmic simulation is done without taking into account many of the hardware aspects required by a real mixer, and serves to provide a general understanding of the working mechanisms of the mixer. In addition the results obtained from the algorithmic simulation can be used to provide a test bench against which further, more involved, simulations can be verified[6].

Figure 3-3 Mixed Output Image

Once the algorithmic simulation has been developed to a sufficient level a hardware simulation can also be developed. Thus the mixer is developed for both algorithmic simulation and hardware simulation.

[6] An advantage of working with both the hardware and algorithmic simulations at the same time is that the output obtained from the hardware simulation can be compared with the output obtained from the algorithmic simulation. The results obtained from both simulations should be same. To check this, a test class is developed to compare the output of two simulations. This can be done by comparing the corresponding pixels for both the outputs images. If they are all the same then there are no errors in the hardware implementation.

4 System Architecture for the Graphics/Video Subsystem

Our initial hardware simulation of the mixer serves to illustrate the manner in which hardware simulation may be carried out in a programming language like C++. Amongst issues that arise in the hardware simulation in C++ are those of synchronisation and parallelism, for which there is no inherent support in C++. These are some of the areas in which System C[7] offers a solution for the development of hardware simulations in a high level language.

The image data for the overlay image and the principal image, alpha value and output image data are stored in the main system memory (RAM). These data are transferred into the mixer. The two images that are subjected for mixing operation have been named as the overlay image and the primary image here. The overlay image is overlayed on the principal image.

Rather than mixing one pixel at a time the images are mixed an image block at a time (e.g. 8 pixels at a time, or 64 pixels at a time). Data is transferred from the RAM to the mixer in data blocks. The minimum size of a data block depends upon the size of the data bus (for example, a 64 bit data bus would allow a minimum size of 8 bytes for data transfer blocks. An image block will consist of one or more data blocks. For example, the alpha image has one data block (the 8 bit alpha values) per image block, whereas the primary image has three data blocks per image block. The three data block accounts for the Red, Green and the Blue components of the primary image.

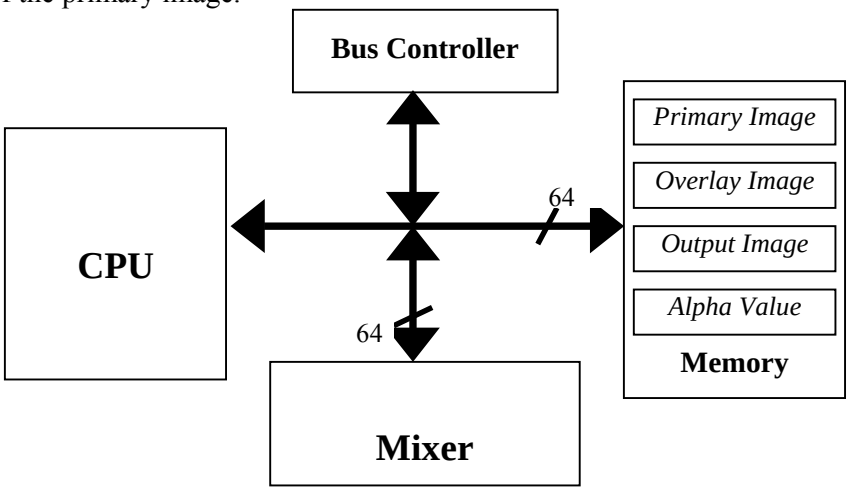

Figure 4-1 System Architecture of the Mixer

7 SystemC is a modeling platform consisting of C++ class libraries and a simulation kernel for design at the system-behavioral and register-transfer-levels.

The data bus may be occupied by memory transfer associated with devices other than the mixer (for example there may be a processor on the bus), so prior to the transfer of data into the mixer, the mixer should take control of the data bus (become bus master). The top-level system architecture of the hardware simulation of the mixer is shown in Figure 4-1.

4.1 Mixer Library

The classes that have been used for the hardware simulation of the mixer are categorised as follows:

Utility

Utility classes consist of the classes relating to the allocation of memory, generation of alpha values, reading and writing bitmap images, conversion of image data from its interleaved[8] form to the deinterleaved[9] form and vice versa.

Interface Classes

A number of interface classes have been defined in order to link two different classes that may be a class from the Hardware Simulation classes and a class from the Utility classes. If a class in the Hardware Simulation classes needs information from the Utility classes, it shouldn't receive this information directly. It should get it through an interface class. This makes the classes completely independent of each other.

Hardware Simulation

Various classes have been defined to simulate the different hardware aspects of the mixer.

Algorithmic Simulation

A separate class has been defined for the algorithmic simulation. This class performs mixing operation irrespective of the hardware contents. The main objective of this class is to generate the reference output data to check the correctness of the output obtained from the hardware simulation.

Test Routines

It is necessary to check the correctness of the output obtained from the hardware simulation of the mixer. Therefore, a class has been created that takes the output data

[8] Image data with all image components i.e. red, green and blue.

obtained from the hardware and algorithmic simulation of the mixer and then compares these two outputs.

5 RTL Hardware Model of the Mixer

Having performed an algorithmic simulation, followed by a high level architectural simulation, the mixer was next simulated in C++ at the RTL[10] level. Here, the mixer hardware is controlled by a micro-programmed control unit [8], where the micro-program consists of a series of instructions that correspond to the control signals (for the hardware) required to carry out the mixing operation.

5.1 System Architecture

The architecture of the hardware is designed on the principle that registers are connected to other registers via gates. In practice a real hardware implementation would be implemented slightly differently, gates would be connected via a bus and data would be transferred between them by latch enabling the register (that the data would be transferred to). The gating model introduces an additional element to achieve the same effect, but it is a conceptual abstraction, which may be useful in the modeling of the hardware, without adding any additional requirements to the hardware implementation [9].

The model presented in this document consists of three main units: the Control Unit (CU), the Data Addressing Unit (DAU) and the Arithmetic Unit (AU) of the mixer. The CU generates a series of control signals, which are provided to the DAU and the AU of the mixer, and dictate the tasks to be carried out in these modules.

Figure 5-1 shows a high-level system architectural model of the hardware mixer.

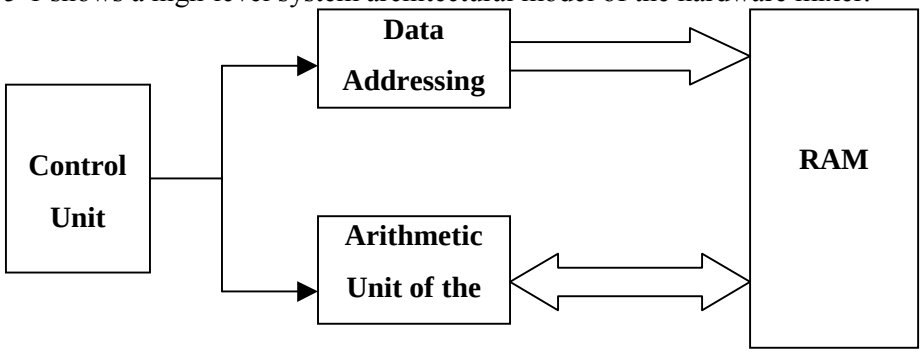

Figure 5-1 System Architecture of RTL Hardware model of the Mixer

[9] Image data that consists of specific component only red, green or blue.
[10] Register Transfer Level. In the RTL level model, the complete system state is separated into groups of bits (called registers) and considers the flow of information from one register to the next in each clock cycle.

The DAU generates the addresses of the data (to be fetched from the memory) as indicated by the microprogram. The AU on the other hand carries out the mixing operation (after receiving the required data i.e. primary image data, overlay image data and alpha value).

The CU consists of an Instruction Store (IS), Instruction Register (IR) and the Program Counter (PC). All instructions are stored in the IS. Each (microprogram) instruction consists of the following: -

- Control signals, to control the hardware
- A flag (BR) to indicate whether the instruction is branchable or not
- A field (BRT) to indicate the type of branching
- The branch address, if any

The output of the CU is the control signals required for each instruction to be executed. Control signals are connected to the controlling inputs on the hardware models for e.g. a gate is opened only when the control signal connected to it is enabled. This is shown in Figure 5-2.

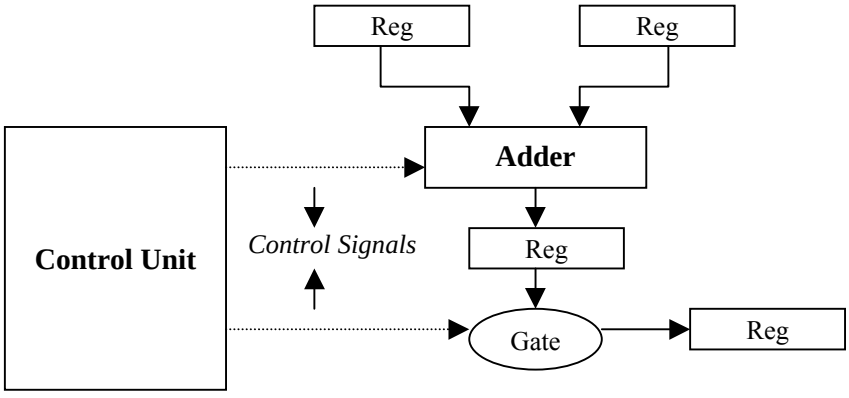

Figure 5-2 Control of an Adder and a Gate through control signals

5.2 Branching in the CU under direction of micro-instructions and signals

Execution of the instructions stored in the IS begins with the initialization of the PC with the address (within the IS) of the initial instruction. The instruction corresponding to the address contained in the PC is loaded from the IS to the IR. If the BR flag is not set, the PC is incremented i.e. ready for the next instruction in the IS to be fetched. If the BR flag is set, the type of branching is first checked. If the branch is of type BR_JUMP_COUNT_NZ, then the input control signal IP_COUNT_Z is checked. If this signal is not set then the next instruction will be the instruction contained in the address of the previous instruction and this

is loaded into the PC. If the input control signal IP_COUNT_Z is set, which means all the pixels have been processed, the program counter is incremented by one. The next instruction is I_STOP, which would terminate the mixing process. If the branch is of type BR_NO_BRANCH, then the PC is incremented.

Figure 5-3 shows the flowchart of the pixels being processed.

Figure 5-4 gives a model of the CU.

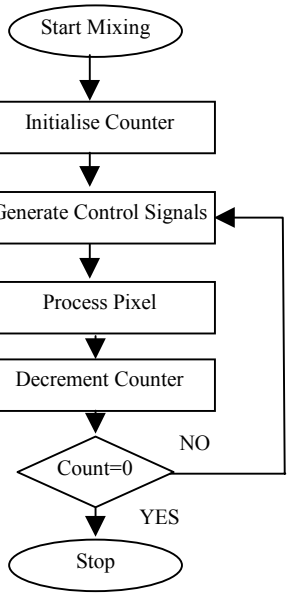

Figure 5-3 Flowchart of the mixer operation

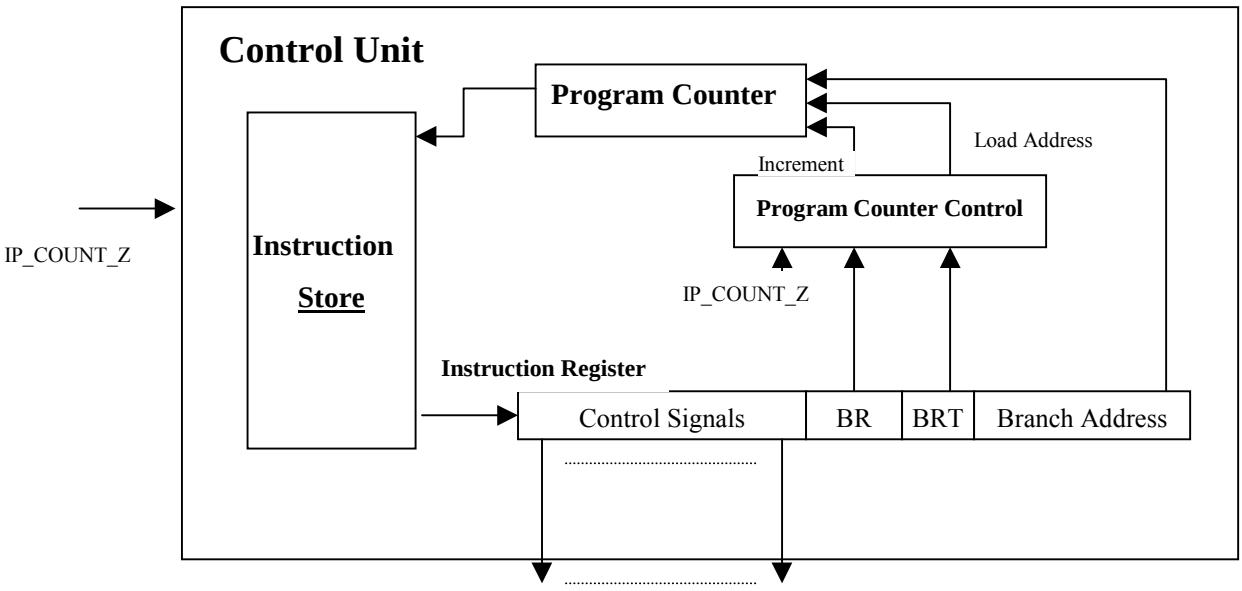

Figure 5-4 The Control unit of the mixer

5.3 Conclusion on this Hardware Model

This approach worked as expected. Two bitmap images of size 640x480 were mixed to get mixed image output. The output obtained was compared with the output obtained from the algorithmic simulation. The corresponding RGB components for every corresponding pixel of the two output images were subtracted, and the result obtained for all the pixels was found to be zero, indicating the desired output has been obtained.

Additionally this model also supports synchronisation and parallelism. For this a hardware specific simulation kernel has been developed. In each cycle, an instruction is executed and then all the registers are updated.

6. SystemC

SystemC [1][2][3] is a modeling platform consisting of C++ class libraries and a simulation kernel for design at the system-behavioral and register-transfer-levels.

SoC (system on chip) designs are a combination of hardware and software - not just hardware only as in ASIC design. In fact, there is more software than hardware in most designs. Therefore, there is a need for a language that describes both the functionality of the software and the hardware. SystemC is one solution, which satisfies this requirement.

SystemC has different features to assist in the system level design. It consists of *Modules.* These are the basic building blocks for partitioning a design. They allow a design to be separated into more manageable pieces and to hide internal data representation and algorithms from the other modules. A typical module consists of *ports* for the module to communicate with the environment, *processes* that describe the functionality of the module, *internal data* and *channels* for communication among the module's processes. The design, which maintains hierarchy, contains modules within modules. The other features in SystemC include a rich set of signal types, data types, clocks, reactivity, multiple abstraction levels, functional models, fixed-point data types, and communication protocols [1].

7. Hardware Model of the Mixer using SystemC

The problem in C++ is that it cannot easily be used to describe hardware as it doesn't have a natural way to represent constrained data types, concurrency and clocks 0. This problem is

solved using SystemC. SystemC has tools such as concurrency, reactivity[11], data-types required for modeling the hardware. SystemC also supports hierarchy of the modules. The concept of modules in SystemC allows us to build separate entities[12] and the communication between these entities is carried out through channels. The channel we have used in the hardware model of the mixer is *sc_signal<T>*. This channel implements both the *in*- and *inout*-interfaces[13].

In the current implementation of the mixer in SystemC, a separate module has been created for each entity for e.g. registers, adder, multiplexer, gates of the DAU and subtractor, adder of the AU. A better approach would have been to maintain the hierarchy of modules by constructing modules for the DAU, AU and CU. Then building different modules (for e.g. adder, register, multiplexer, gates etc) inside these modules. This approach will be taken in later simulation process of the project.

One advantage in SystemC is that it allows the user to make use of a user defined *Packet Type* which is similar to the data type. The input and output ports of the modules and channel (*sc_signal*<T> in our case) could be defined as the packet type. The packet type is defined by a structure. SystemC allows a user to pass this packet type from module to module. The advantage of the packet type is that an entire structure of data can be transferred from one module to another through a single port.

The control unit (CU) part of the mixer consists of an instruction store (IS) and the program counter (PC). The initial task in designing the CU for the mixer was to define the packet type for the instruction. This packet type consists of the information on the instruction type ID, branch type, branch address and the array of control signals generated by the instruction, as shown below.

```
struct instruction_type
{
        instruction_type_E ID;
        branch_type_E br_typ;
        int branchable;
        int br_addr;
        int op_ctrl_lines[C_NO_CONTROL_SIGNALS];
        ……..
};
```

[11] Hardware can be taken as a set of non-terminating process that reacts continuously to events in their environment 0.
[12] Entity in this document means the hardware abstraction of the digital system.
[13] An interface that lets data in and out of the module through its ports.

As it is seen in the above code, the control signals have been defined as *int* type. In fact, it should be *bool* type. Due to some reasons, the simulation was not working properly with the *bool* type. This problem will be resolved as we take further steps in SystemC simulation process.

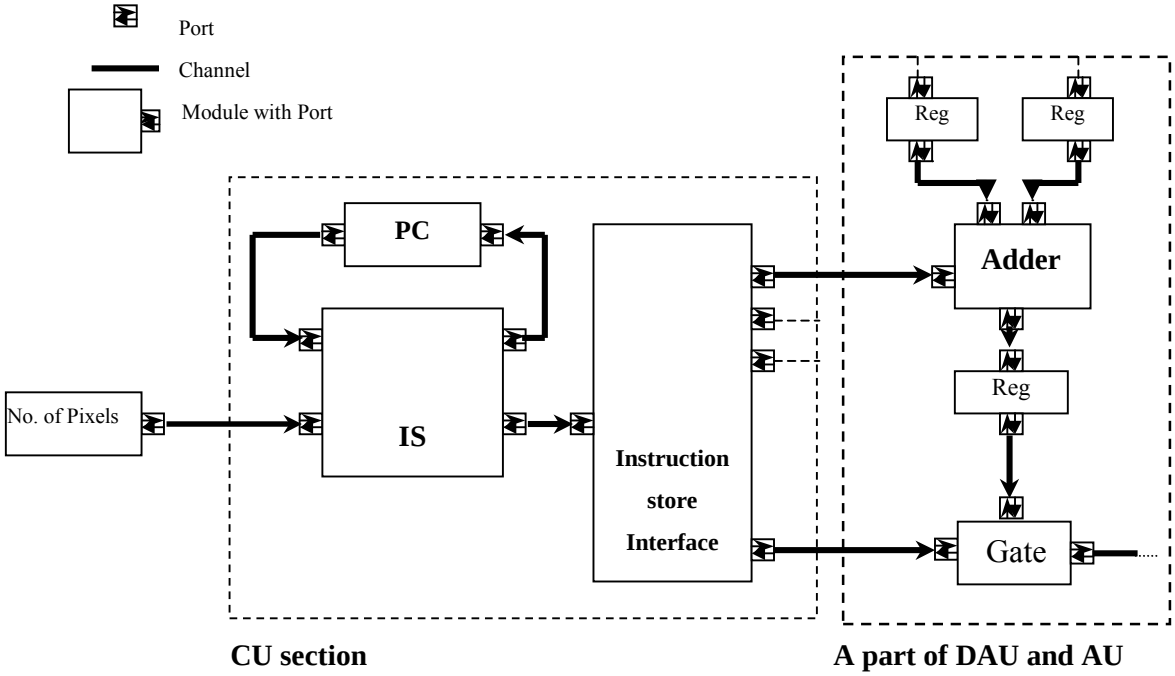

Figure 7-1 Hardware Mixer Model in SystemC

The next step was to define a module for the IS. The IS holds the microprogram that is required to control the hardware to carry out the mixing operation. This module consists of two output ports and three input ports. The first output port is defined as the packet type for instructions. The second output port is for delivering the address of the next instruction if the current instruction is *branchable*. The first input port takes the input from the PC that informs the instruction store which instruction to fetch. The second input port takes information of the number of lines processed. This is shown in Figure 7-1. The third input port is for the clock. The port for the clock is not shown in the figure. When all the lines in the image have been processed then the IS module ceases operation.

The ports for the IS module are defined as follows: -

```
sc_out<instruction_type> inst_out_opin;
sc_out<int> next_address_opin; //to be supplied to the PC

sc_in<int> PC_ipin;  //program counter input
sc_in<int> no_lines_scanned_ipin;
sc_in<bool> clk;
```

In the code given above *instruction_type* is the packet type. The IS module outputs the instruction to its output port as the packet type. The control signals are embedded inside this packet type.

The input ports of the other modules in the mixer (for e.g. gates, adder etc) take control signals as the input (Figure 7-1). Thus, to establish communication between the IS module and the other modules in the mixer, the input port of the latter module should be of the same type as the output port of the former module. For this a separate module is created called an *instruction interface* module. This module takes input of the packet type and delivers separate outputs for each control signals. Thus an interface is required between the IS module and the other modules to establish proper communication between them. In Figure 7-1, the IS and the instruction store interface have been shown as separate modules. The better approach would have been to define these modules in a hierarchical manner by having these modules defined inside another module. This module would then act as the IS module. As was said earlier, the module hierarchy approach has not been observed in this SystemC module. This will be corrected later in the project.

```
SC_MODULE (instruction_interface_mod)
{
        sc_in<instruction_type> inst_str_ipin; // packet type input

        sc_out<int> ID_V;             // defined only for debugging purpose
        sc_out<int> br_typ_V;      // not required for interface
        sc_out<int> branchable_V;
        sc_out<int> br_addr_V;

        sc_out<int> C_INIT_REG_BASE_ADD_ALPHA_V;  //separate output for
        sc_out<int> C_INIT_REG_BASE_ADD_RED1_V;  //each control signals
        ……;
}
```

The PC module takes input from the IS module. If the current instruction is not *branchable* then the PC is simply incremented. If the current instruction is *branchable* then the IS module provides the PC with the address of the next instruction to be fetched.

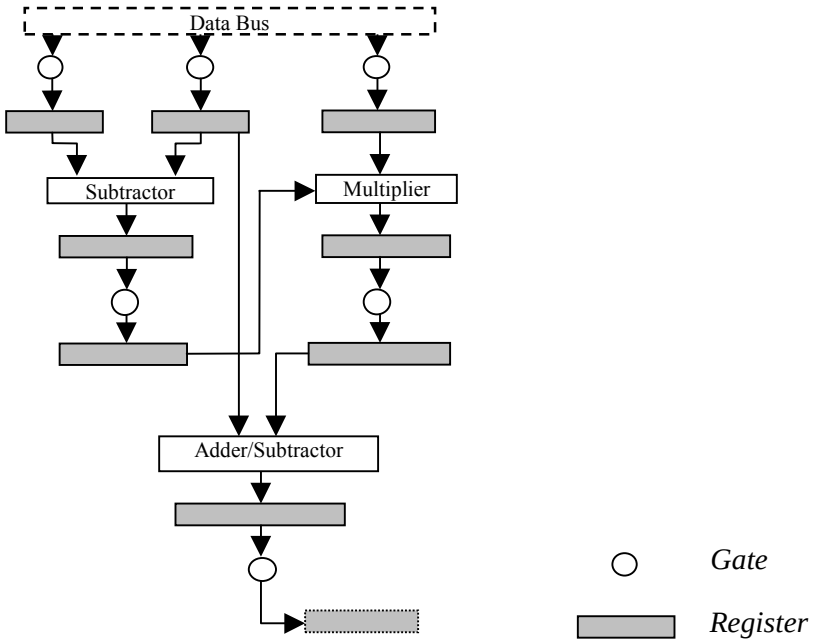

Figure 7-2 Arithmetic Unit organization

Having implemented the CU, the next step was to create modules for different components of DAU and AU.

The modules for the DAU are multiplexer, adder, registers, multiplier and gates. The modules for the AU are subtractor, adder/subtractor, multiplier, registers and gates. The organization of the AU is shown in Figure 7-2. Each module is provided with its own sensitivity list[14]. A module is activated if an event occurs in any one of the members of its sensitivity list. Registers are made sensitive to events in the input port. Whenever there is a change in a register input then the input of the register is transferred to its output. Gates are made sensitive to the control signals which come with the instructions. Other modules such as adder, subtractor, multiplier, and adder/subtractor are also made sensitive to the control signals (Figure 7-1). These modules are triggered only when they are told to do so by the instruction.

7.1 Conclusion on SystemC approach

SystemC provides the tools required for the hardware modeling. The provision of modules and ports support hierarchy of modules whereas the provision of channels provides an

[14] Sensitivity is a list of variables for a module. If event occurs in any one of them, the module reports corresponding changes. The module will not invoke unless an event occurs in the member of the sensitivity list.

abstraction for communication. These features are not available in C++ based hardware modeling.

The model is working as expected. The method of testing the outputs in this approach is not as straightforward as in the earlier approach. The outputs are first stored in the file. The output data in the file is later on compared with the output obtained from the algorithmic simulation. There might be some efficient way of testing but it remains as a work to be explored in the later part of the project.

8 Further work

Having completed the current work, a system requirements specification will be produced for a graphics/video system.

A graphics/video system will be designed that complies with the SRS earlier generated for the graphics/video system. The graphics/video system will then be implemented using SystemC.

After completing this, a research will be conducted into the hardware/software partitioning of the hardware/software Codesign and techniques for facilitating this within SystemC. A report will be produced highlighting the suitability of SystemC for the design.

References

[1] Joachim Gerlach, Wolfgang Rosenstiel, *System Level Design Using the SystemC Modeling Platform,* University of Tübingen, Germany, Workshop on System Design Automation (SDA'00), Rathen, Germany, March 2000, pp. 185-189.
[2] Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, *System Design with SystemC,* Design, Automation, and Test in Europe (DATE '01) **.**
[3] *SystemC Version 2.0, User's Guide,* www. SystemC.org.
[4] Stuart Swan, *An introduction to System Level Modeling SystemC 2.0,* Cadence Design Systems, Inc. May 2001.
[5] *Official Website of Nokia mobile phones,* www. nokia.com.
[6] G. Economakos, P. Oikonomakos, I. Panagopoulos, I. Poulakis, and G. Papakonstantinou,
[7] *Behavioral Synthesis with SystemC,* Design, Automation, and Test in Europe (DATE '01)**.**
[8] William Stallings, *Computer Organization and Architecture*, pp. 578-617, Prentice Hall.
[9] William Stallings, *Computer Organization and Architecture*, pp. 554-576, Prentice Hall.
[10] Dr. Guido Arnout, *SystemC Standard,* Proceedings on the 2000 conference on Asia and South Pacific design automation January 2000.
Stan Y. Liao, *Towards a new standard in System Level Design*, Advanced Technology Group, Synopsys, Inc.