Center for Embedded Computer Systems
University of California, Irvine

System Modeling
A Case Study on a Wireless Sensor Network

Gautam Sachdeva
Rainer Dömer
Pai Chou

Technical Report CECS-TR-05-12
June 15, 2005

Center for Embedded Computer System
University of California, Irvine
Irvine, California 92697-3425, USA
Email: {gsachdev,doemer,phchou}@uci.edu

System Modeling
A Case Study on a Wireless Sensor Network

Gautam Sachdeva
Rainer Dömer
Pai Chou

Technical Report CECS-TR-05-12
June 15, 2005

Center for Embedded Computer System
University of California, Irvine
Irvine, California 92697-3425, USA
Email: {gsachdev,doemer,phchou}@uci.edu

Abstract

System modeling in a large extent is a matter of handling abstract and possibly incomplete information and trying to evaluate different solutions based on the system model. The ease with which a system can be modeled depends on the semantics and syntax of the language used. At present, there is no complete language available for entire system modeling but there are few System-Level Design Language's (SLDL) that are being used for following system-level design methodology. In this project, we have modeled a wireless sensor network system in SLDL to determine if SLDL can be used for system modeling. This report gives details of the project, results & conclusions arrived during the project as well as the limitations of SLDL for entire system modeling.

Contents

List of Figures

System Modeling
A Case Study on a Wireless Sensor Network

Gautam Sachdeva, Rainer Dömer and Pai Chou
Center for Embedded Computer System
University of California, Irvine
Irvine, California 92697-3425, USA
Email: {gsachdev,doemer,phchou}@uci.edu

Abstract

System modeling in a large extent is a matter of handling abstract and possibly incomplete information and trying to evaluate different solutions based on the system model. The ease with which a system can be modeled depends on the semantics and syntax of the language used. At present, there is no complete language available for entire system modeling but there are few System-Level Design Language's (SLDL) that are being used for following system-level design methodology. In this project, we have modeled a wireless sensor network system in SLDL to determine if SLDL can be used for system modeling. This report gives details of the project, results & conclusions arrived during the project as well as the limitations of SLDL for entire system modeling.

1. Introduction

Embedded computing systems have grown tremendously in recent years, not only in popularity, but also in their complexity. With the ever increasing complexity and time-to-market pressures in the design of embedded systems, both industry and EDA vendors are trying to move the design to higher levels of abstraction, in order to cope with these issues, new methodologies that emphasize re-use at all levels of abstraction are a "must" [1]. At higher level, there is no difference between hardware and software and the complexity in terms of number of objects to be handled reduces. An embedded system at the lowest level consists millions of transistors, which reduces to only thousands of components at the register-transfer level (RTL). Furthermore, RTL components are grouped together at the algorithm level. Moving another higher level, the so-called system-level, the one system is composed of only few components which include microprocessors, special-purpose hardware units, memories and busses. Finally at the highest level, the entire system consists of many subsystems. There is always a tradeoff between the level of abstraction and accuracy. A higher abstraction level implies low accuracy, and vice versa.

A system consists of components from heterogeneous design domains which must be integrated. Multiple design domains must be considered and analog, digital, MEMs and optics represent only a few domains where embedded system originates. Multilanguage solutions are required for the design of heterogeneous systems where different parts belong to different application classes e.g. control/data or continuous/discrete. The main problem that needs to be solved when dealing with multilanguage design is the refinement of communication between heterogeneous subsystems[2]. The use of a multilanguage specification requires new validation techniques able to handle a multiparadigm model. Instead of simulation we will need cosimulation and instead of verification we will need coverification. Additionally, multilanguage specification brings about the issue of interfacing subsystems which are described in different languages.

The goal of any language designed to support system modeling must be to bring together heterogeneous information in a common language environment. Central to these problems is that different design domains employ radically different knowledge in their representation and reasoning about models. The language must provide modeling support for different design domains employing semantics and syntax appropriate for those domains. At present, there is no complete language available for system modeling but work for developing such a language is in progress. One such language being developed is system modeling language (SysML) [3]. SysML is a new general-purpose modeling language based on UML that can be used for specifying requirements, system structure and functional behavior but SysML doesn't support complete testing, comprehensive verification and validation or fully executable functional behavior. These gaps will be addressed in the future versions of SysML.

Though there is no specific C/C++ based language that supports system modeling but there are few System-Level Design Languages (SLDL) that support top-down system level design methodology. Unlike the traditional single semantics languages, the SLDL provides a collection of interacting domain theories for describing systems facets. In this project we have used SpecC (SLDL) for modeling wireless sensor network system.

1.1 SLDL

System-level specification model are written in SLDLs which can be used to model complex systems consisting of hardware and software components. Though it is possible to model designs in any of the programming languages, the choice of a good SLDL is a key in reducing the effort required in writing the specification model. A good SLDL provides native support to model both hardware and software concepts found in embedded system designs. A good SLDL provides native support to model concurrency, pipelining, structural hierarchy, interrupts and synchronization primitives. They also provide native support to implement computation models like sequential, FSM, FSMD and so on, apart from providing all the typical features provided by other programming languages. In general a SLDL requires two essential attributes: 1) it should support modeling at all levels of abstraction, from purely behavioral un-timed models to cycle accurate RTL/ISS models, and 2) the models should be executable and simulatable, so that functionality and constraints can be validated.

The following languages are popular choices for writing specification model: VHDL [4], Verilog [5], SystemC [6], and SpecC [7]. VHDL and Verilog are primarily Hardware Description Languages (HDLs) and hence are not suitable to model software components. SystemC implements system level modeling concepts in the form of C++ library. It can model both hardware and software concepts and thus is a good candidate for system level design.

SpecC is another major candidate for system design. Being a true superset of ANSI-C, it has a natural suitability to describe software components. It has added features like signals, wait, notify etc. to support hardware description.

1.1.1 SpecC

The SpecC language was specifically developed for the specification and design of digital embedded systems, including hardware and software portions. Built on the top of the ANSI-C programming language, the SpecC language supports concepts essential for embedded systems design, including behavioral and structural hierarchy, concurrency, communication, synchronization, state transitions, exception handling, and timing. Since SpecC is a true superset of ANSI-C, so every C program is also a SpecC program [7].

The SpecC language provides a minimal, well-defined set of orthogonal constructs that precisely cover the concepts identified in embedded systems in a one-to-one fashion. These constructs are defined in detail in the SpecC language reference manual [8]. It should be emphasized that the concepts supported by the SpecC language cannot be extended by the user. While this intuitively sounds like a limitation, it is actually an important feature of the language. Having a fixed and well-defined set of constructs in the language is of crucial importance for the development of CAD tools because tools need to "understand" the semantics of each construct in order to be able to support the particular concept.

Note that this is contrast to other approaches, such as SystemC, where the extendability of the language (adding user-defined classes with special functionality) is part of the methodology. While such language extendability can be easily supported in simulation (after all, it is just C++ code to be compiled), this leads to significantly higher complexity. Moreover, support of such extensions by general synthesis and verification tools becomes impossible.

2. System Modeling

The language used for system modeling must be able to support specification, analysis, design, verification and validation of a broad range of complex systems. These systems may include hardware, software, processes and subsystems. The key representation of any system is a block diagram and the ease with which a system can be represented in block diagram depends on model supported by the language.

Block diagram consists of a set of blocks and a set of interconnections between the blocks. The blocks in the diagram represent components which perform a particular function or computation. Also, the blocks can communicate with each other through interconnection. Hence, each block performs two main functions, namely computation and communication [9].

In traditional model, Figure 1 [9], such as in VHDL or Verilog the processes (represented as blocks) contain code for both communication and computation. Because communication and computation are freely intermixed in the code, they cannot be identified by any tool. As a result, it is not possible to automatically change the communication protocol when design constraints change. On the other hand, it is also impossible to automatically switch to a new algorithm to perform the computation.

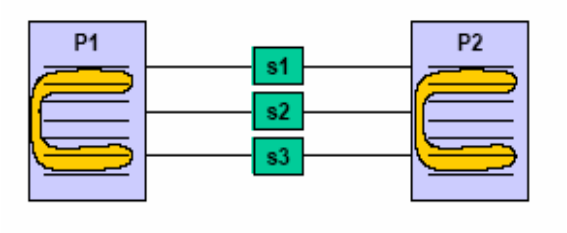

Figure 1 Traditional Model [9]

In SpecC model, as shown in Figure 2 [9], communication and computation are clearly separated. Computation is encapsulated in behaviors and communication is encapsulated in channels. Behaviors only contain computation and in order to communicate, the behaviors call the functions provided by the connected channel.

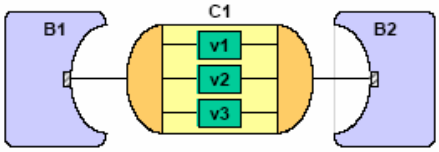

Figure 2 SpecC Model [9]

As a result of the separation of communication and computation, the SpecC model supports "plug and play". The communication protocol can be easily exchanged by the use of another channel with compatible interfaces, whenever this is desirable in the design process. In the same manner, the behaviors can also be exchanged with others, without affecting the communication protocol.

SpecC Reference Compiler (SCRC) includes a standard channel library that supports a set of standard channels for communication. These channels provide well-known mechanisms for synchronization, resource management and communication. Some of the channels included in the SpecC standard channel library are listed below. *Semaphore channel* is used for providing protected access to shared resources between different behaviors. *Queue channel* defines a type-less, fixed-size queue that can be used by any number of sender and receiver behaviors. *Handshake channel* is used for providing safe one-way synchronized communication between sender and receiver behavior. *Double Handshake Channel* provides a 2-way handshake channel for type-less data transfer between sender and receiver behaviors. SpecC standard channel library provides number of more channels, and they are listed in the SpecC language reference manual [8]. Though SpecC standard channel library includes many channels, at this time, the list is not complete and several channels are still missing from the library.

3. Case Study: Wireless Sensor Network

As mentioned before, the aim of this project was to determine whether a SLDL can be used for entire system modeling. In this project, we selected SpecC (SLDL) for system modeling for a number of reasons. First, models defined in SpecC can be easily validated through simulation. Second, it supports structural hierarchy in the style of standard block diagram. Structure is represented as a hierarchical network of behaviors and channels. Third, it also supports behavioral hierarchy by providing well defined constructs such as *par*, *fsm* and *pipe*. Hence, behaviors can be executed sequentially, concurrently or in a pipelined fashion. Fourth, SpecC Model separates the computation and communication as is the case for every real system. Moreover, SpecC provides SpecC standard channel library that includes implementation of various channels. And since SpecC is the superset of C, therefore it's easy to learn.

In order to determine if SpecC is suitable for entire system modeling, we wanted to model a system that is complex enough to determine the capabilities of a language for system modeling. We selected a wireless sensor network system, Eco System [10], for this purpose due to the fact:

1. Eco System contains many subsystems, Eco Nodes and Eco Stations
2. Subsystems being independent of each other must be simulated concurrently
3. Subsystems involves significant communication internally as well as between each other
4. Each subsystems consists of many components each performing some function
5. Each component involves reasonable amount of computation
6. Eco System is easily scalable
7. Includes analog and digital domain.

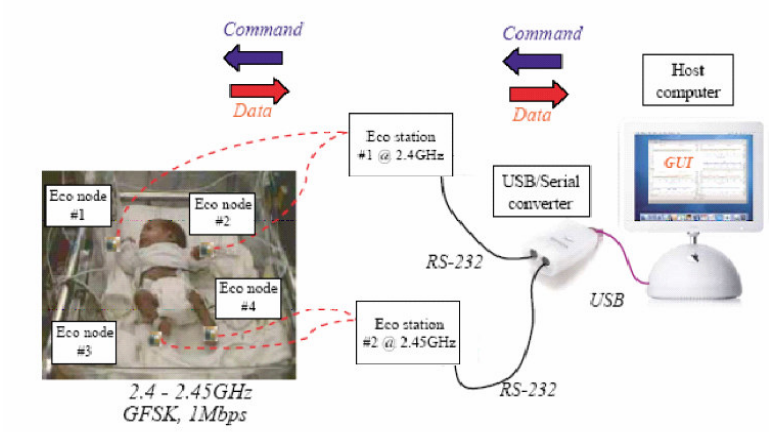

Figure 3 Application setup of the Infant Monitor [10]

Hence, Eco System was a perfect example for system modeling. Another motivation for selecting Eco System was its practical application of motion monitoring of pre-term infants. Analysis results of the model can be used in improving future versions of the Eco System.

3.1 Eco System

Eco System [10] is basically for real time motion monitoring. Eco System consists of a set of Eco nodes, Eco stations and software on the host computer. The Eco node is possibly the world's smallest, low power wireless sensor node in its class, capable of taking vibration data and transmitting wirelessly in real-time to the Eco station, which provides the up-link to the host computer. Eco nodes are generally applicable to cases where the ultra-compact form factor is essential. Most important practical application of Eco Node is monitoring motion of pre-term infants, to help the infants grow in weight and bone strength. Today's available wireless or cordless sensors are too bulky that they impede the motion of the infants.

For motion monitoring purpose of an infant we need four Eco nodes, as one is required for each limb (arms & legs) and two Eco stations are needed to monitor the four Eco nodes, see Figure 3 [10]. All the Eco nodes must operate in coordinated manner, which means they must synchronize to the same clock and take samples at the same time. In addition, they should perform communication scheduling at different times so that the nodes do not interfere with each other.

It is clear from the above diagram that four Eco Nodes and two Eco Station are being used. Each station controls or communicates with two Eco nodes and the Eco Stations are being controlled by the host computer.

3.1.1 Eco Node

Eco node performs a simple task: take an analog sample for ten times per second, and transmit the data over the wireless link to one or more receivers connected to a host computer. Eco node basically consists of two main components: Sensor and Transceiver.

ADXL202E is a dual-axis accelerometer which senses motion and give analog output for both X & Y direction. We also have an nRF24E1 transceiver with an embedded 8051-compatible microcontroller and a 10 bit 9 input ADC. The ADC converts the analog X and Y signal into digital values, the transceiver transmits the data through the wireless channel only after forming a package of the form as shown in Figure 4 [11] and the microcontroller controls the whole process by being an interface between the two.

Figure 4 nRF24E1 Package Format [11]

Preamble marks the beginning of the package, ADDR is the address of the receiving node, PAYLOAD is the data and CRC is the cyclic redundancy check. ADDR is added by the microcontroller and Preamble & CRC are added by the transceiver. All the eco nodes perform the same function but at different frequency.

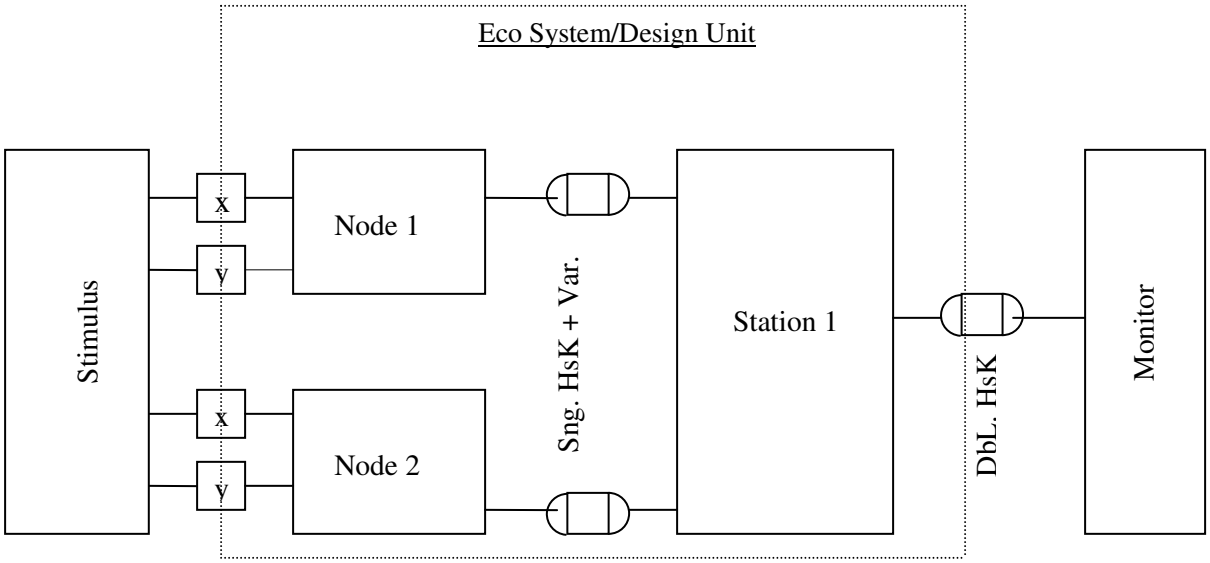

Figure 5 System Model/Test bench

3.1.2 Eco Station

An Eco station is the interface between the Eco nodes and the host computer. It receives data over the wireless link from one or more Eco nodes one at a time. Then, it sends the data to the host computer over RS-232. Once the Eco station receives the initial startup commands from the host computer, it immediately starts listening to the two Eco nodes it is assigned for a period of time. Eco Station performs the ADDR and CRC check on each package received. After having received multiple data packets from both Eco nodes, the Eco station computes the average values of the motion data on each node, and sends them in a single data packet to the host computer. The listen-package-send cycle on the second Eco station is interleaved with the same cycle of the first Eco station, such that in each sampling period the host computer can receive data from both stations sequentially.

3.1.3 Host PC

On startup, the host computer sends commands to both receivers (Eco Stations) to set the sampling rate and total duration of the experiment. Then, host starts waiting for the incoming data from the first and second Eco stations. Upon receiving the data packets, the GUI on the host computer will plot the motion data on all four Eco nodes in real-time. The same recv-recv-paint cycle repeats until the experimental duration expires.

3.2 System Modeling

System model is pure functional and abstract model which is free of any implementation details. A SpecC model is defined by a set of behaviors and channels, behaviors encapsulate computation and channels encapsulate communication and hence separate the computation from communication.

A behavior in SpecC provides structural and behavioral hierarchy and a behavior consists of hierarchy of behaviors [9]. The hierarchy of behaviors in model reflects the system functionality without implying anything much about the system architecture. At each level of hierarchy the behavior is an arbitrary serial-parallel composition of behaviors. At the lowest level of hierarchy, leaf behaviors execute the algorithms in the form of C code. Behaviors communicate either through variables and synchronize through events attached to their ports or through channels. The default execution of the behaviors is sequential but SpecC provides dedicated constructs such as par (parallel execution), pipe (pipelined execution) and fsm (FSM execution). A channel consists of set of variables and methods, methods operate on the variables and define the communication protocol.

3.2.1 Design Unit

As shown before in Figure 3, Eco System for motion monitoring of a pre-term infants requires four nodes and two stations. Each station controls and communicates with two nodes. In the model we have implemented, Figure 5, the test bench consists of *Design Unit*, *Stimulus* and *Monitor*. The design unit consists of two similar *Nodes* and one *Station* behavior executing concurrently. The test bench is implemented

in such a way that it is easily scalable by adding few lines of codes. Hence the test bench can be quickly extended for four *Nodes* and two *Stations* or even more. In fact, we simulated the test bench for four *Nodes* and two *Stations* and the results were consistent.

3.2.2 *Node* Behavior

Node behavior functionally represents the Eco node and consists of three concurrent behaviors, which are analogous to the three main components of the Eco node, namely, ADC, microcontroller, and transceiver, Figure 6.

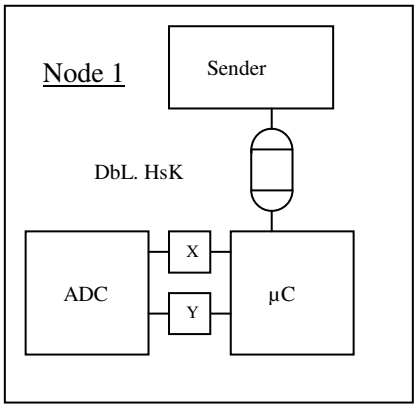

Figure 6 *Node* Behavior

Behavior *ADC* converts the analog value into 10-bit digital value. Behavior *µC* takes the data from the *ADC*, adds the address of the receiver node and provides the data payload to the *Sender* behavior. *Sender* is analogous to the transceiver, it forms the packet by adding CRC and Preamble to the payload and transmits the packet to the *Station*. In Eco System, two similar Eco nodes perform the same function and communicate to single Eco Station but at different frequency, therefore we have two similar *Node* behaviors representing the two Eco nodes and communicating with the *Station* through two different channels.

3.2.3 *Station* Behavior

Behavior *Station* is similar to the Eco station and consists of three behaviors, two *Receivers* and one *CPU*, Figure 7. Since a Station receives data from two concurrent *Node* behaviors, so we have two similar *Receiver* behaviors that can listen to each *Node* simultaneously. *Receiver* behavior performs the CRC and ADDR check on each packet received, and provides the *CPU* with the payload. Behavior *CPU* first waits for the initial startup commands from the monitor and then starts listening to the two *Receivers*. After having received multiple data packets from the *Receivers*, the *CPU* computes the average values of the motion data on each Node, and sends them in a single data packet to the *Monitor*.

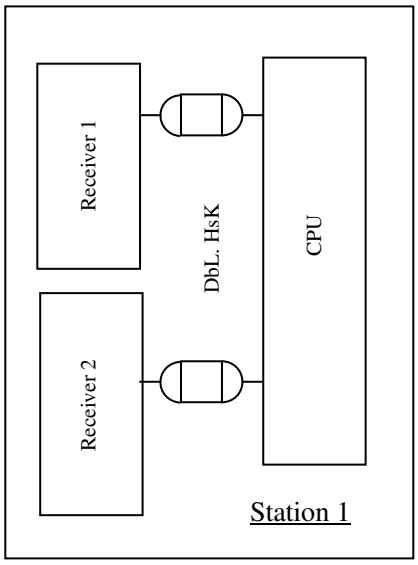

Figure 7 *Station* Behavior

3.2.4 *Monitor* & *Stimulus* Behavior

Monitor behavior performs the functions of the host computer. Initially, the *Monitor* sends initial startup commands to the *Station* that sets the number of samples to be sampled. Then, the *Monitor* starts waiting for the incoming data from the *Station* and after receiving the data packets it plots the motion data of each *Node*. Behavior *Stimulus* basically provides the test bench with the sample input data and performs the function of the sensor. Since the *Stimulus* should provide the continuous varying analog input to the two *Nodes* continuously, hence the *Stimulus* behavior consists of two leaf behaviors, *Stimulus 1* & *Stimulus 2* each executing concurrently and providing data to each node, as shown in Figure 8. To provide analog input to the test bench, we continuously vary a floating-point variable with time, though this is not an analog signal but it acts similar to an analog signal.

3.2.5 Communication

Behaviors communicate with each other either via variables or channels. The choice of the channel for communication between the behaviors depends on two main factors: 1) Required characteristics of the channel 2) No. of behaviors involved. The characteristics of the

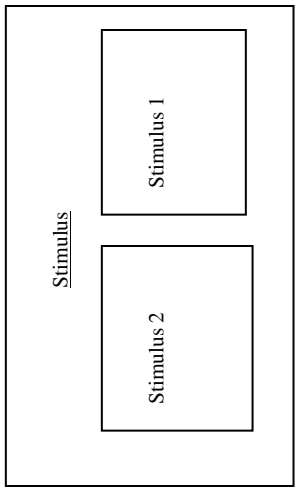

Figure 8 *Stimulus* Behavior

channel can be classified as:
1) Reliable/Unreliable
2) Blocking/Non-Blocking
3) Synchronized/Unsynchronized
4) No-Handshake/One-Way Handshake/Two-Way Handshake
5) Lossy/Non-Lossy
6) Simplex/Half-Duplex/Duplex
7) Stack/Queue/Variable and so on.

Depending on the number of behaviors involved communication channels can be classified as:
1) one-to-one
2) 1-to-N (Multicast)
3) N-to-1
4) 1-to-All (Broadcast)
5) All-to-1
6) N-to-M.

We have basically used three types of channel from the SpecC standard channel library for the communication between the behaviors, depending on the characteristics of the channel and number of behaviors involved:
1) Zero Handshake/Variable
2) Handshake plus Variable
3) Double Handshake.

For instance, to model the wireless communication channel between the Eco Node and Eco Station, we have used Handshake plus Variable channel due to the following characteristic of communication required:
1) Simplex
2) Non-Blocking Sender
3) Unreliable
4) Lossy
5) Synchronized
6) One-Way Handshake (as a particular frequency is used).

Similarly, channels between the other behaviors were selected based on the characteristic of the communication required between the behaviors.

3.2.6 Model Scalability

As mentioned before, the test bench was implemented in such a way to support scalability. In case of 4-*Nodes* and 2-*Stations*, the whole test bench was duplicated by adding few lines of code. Hence the new test bench obtained consisted of two similar *Stimulus*, *Design Unit* and *Monitor* (same as mentioned in previous section). The results obtained on simulation of this test bench were consistent. Therefore, the model can be easily extended for any number of nodes and stations, thus is fully scalable.

3.2.7 Results

Eco system, wireless sensor network was simulated on a Linux, 2.4 GHz x86 Intel® Pentium® 4 processor. Simulation time for the system model was 20ms in the case when system model contained 2000 time unit delay. In the above simulation, the CPU utilization for the task was 5% and CPU-seconds that the process spend in kernel mode was 10ms.

Figure 9, shows the input signal (x) provided to the *Node* behavior by the *Stimulus* behavior. It is clear from the figure that the input (x) to the node behavior is a sine wave, that changes its value every 100 time units.

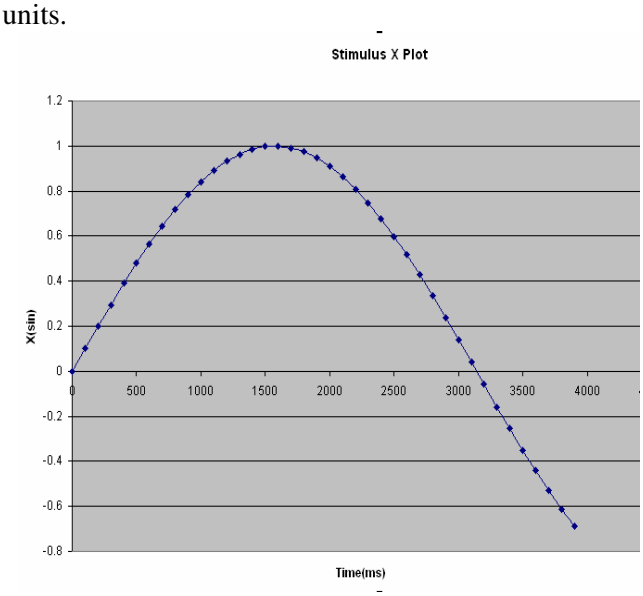

Figure 9 Test bench Input: Node 1(x)

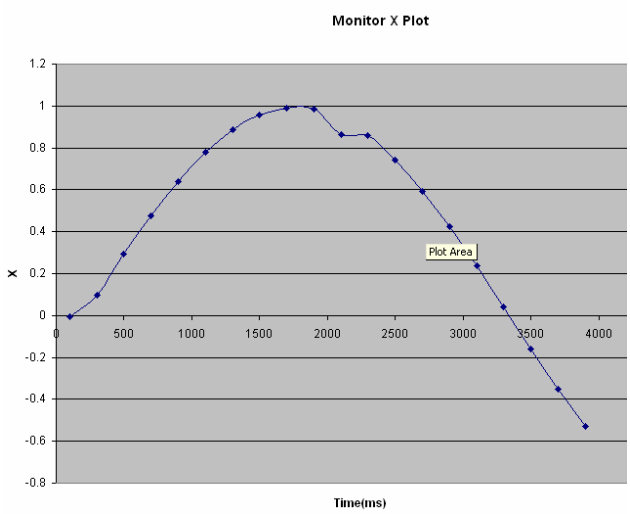

Figure 10 Test bench Output: Node 1(x)

The above figure, Figure 10, shows the output received in the *Monitor* behavior for the input sine wave and two things can be easily noted above figure. First, number of samples received in *Monitor* are half the number of samples provided by the *Stimulus*. This is due to the fact that for every two samples received by the *Station* from the *Node*, *Station* calculates the average and transmits the result to the *Monitor*. Second, the output wave is somewhat distorted as compared to the input wave due to the following reasons:

1. Analog Value is converted in to a digital value and back to analog value, adding twice the analog-to-digital error.
2. Every two digital values are averaged in the station resulting in lack of accuracy.
3. Analog input is modeled with a floating-point variable that is changed after every particular time units. If the same value of the variable is read twice due to the concurrency/parallelism of behaviors in the specC, it can result in more distorted output. This is the reason for the distortion of the output wave between time units 2000 and 2500 in the Figure 10.

The above results show that Eco system was successfully modeled in SpecC and the simulation results were consistent. Any time varying input signal provided by the sensor (*Stimulus*) can be easily recovered and monitored in the *Monitor*. Hence, SLDL can be used for entire system modeling.

4. Discussion

The various conclusions derived from this project and the future scopes of this project are listed below.

4.1 Conclusion

In this project we were able to model a wireless sensor network, Eco System, in SLDL successfully, even though SLDL have not been developed to support entire system modeling. Hence, we can claim that SLDL's such as SpecC can be used for system modeling but it needs to overcome the below mentioned limitation before it can be employed extensively for system modeling.

4.2 SpecC Limitation for System Modeling

SpecC model separates the communication and computation by encapsulating communication in channels and computation in behaviors. Though this separation of communication and computation is a useful concept and makes the life of the programmer easy, the SpecC standard channel library is very limited to utilize this feature completely. Even the basic channel such as type-less one-way handshake is still missing from the channel library. We had to use the Handshake channel (SpecC Channel library) along with a variable for this purpose, in the case of communication between the Eco node and Eco station.

Though in this project, we modeled a wireless network in SpecC but there is no support for the wireless channel in the SpecC standard channel library. Wireless channel has features such as lossy, synchronized, unreliable, non-blocking, simplex/half-duplex/duplex and so on. Not only wireless channel, but also many other kinds of channels such as broadcast, multicast, etc. need to be implemented, if SpecC is to be used for system modeling. In fact, at present there is no channel in the SpecC standard library that supports communication between more than two behaviors. For instance, in the Eco Model we mentioned before, in *Station* behavior, we have two *Receiver* behaviors that communicate with the CPU behavior. But there is no channel in the library that supports communication between three behaviors. Hence, we had to use two separate channels to overcome this limitation.

SpecC increased the flexibility by providing explicit support for the bit vectors (digital domain) and various operators on bit vectors. But SpecC doesn't provide any support for analog domain. Hardware languages such as VHDL and Verilog support analog and mixed-signal. Moreover SpecC doesn't provide any library for the stimulus block, which can be used again and again. Instead, the Stimulus block needs to be implemented every time by the programmer manually.

4.3 Future Scope

The model that has been implemented is a very basic and generic model of the Eco System. The model contains one station and two nodes, but it is scalable for N stations and 2N nodes. Model was also easily simulated and verified for two stations and four nodes by just adding few lines of code. In stead of changing the code for adding more nodes and stations, there should be script that asks for number of nodes and stations as input and generates the modified code.

In future, the model can be extended to include more complex features, such as lossy or faulty communication between nodes and station (wireless link). By introducing additional parameter in the communication channel, we can easily simulate lossy or faulty environment, and the affect of such environment on the performance of the whole system can be easily studied and analyzed. We can also come with some approximate performance and fault tolerance figures for the whole system. These results can be utilized in the future version of the Eco system. Moreover, a more extensive CRC check algorithm can be included to detect any kind of error in the packets received at the station and these packets can them be discarded.

The model can also be made customizable, where the user can specify the number of nodes and stations in the system, percentage of faulty communication and many other features. As mentioned before we basically need a script that asks the users for these inputs. Different simulation environment can be easily simulated, studied and analyzed to come up with better performance situation.

This model can also serve as a functional model to some system design tool for generating an implementation model, and various tight hardware constraints like memory size can then be applied.

5. References

[1] K. Keutzer, S. Malik, A.R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli, "System-Level Design: Orthogonalization of Concerns and Platform-Based Design", *IEEE Trans. CAD Integrated Circuits and Systems*, vol. 19, no. 12, 2000, pp. 1523-1543.

[2] P. Coste, F. Hessel, P. L. Marrec, Z. Sugar, M. Romdhani, R. Suescun, N. Zergainoh, A. A. Jerraya, "Multilanguage design of Heterogenous Systems", *TIMA Laboratory, 46 avenue Fĕlix Viallet, 38000 Grenoble France.*

[3] SysML Partners, "Systems Modeling Language (SysML) Specification, version 0.9", http://www.sysml.com

[4] Petru Eles, Krzysztof Kuchcinski, and Zebo Peng., "System Synthesis with VHDL", *Kluwer Academic Publishers, December 1997.*

[5] David J. Lilja and Sachin S. Sapatnekar, "Designing Digital Computer Systems with Verilog", *Cambridge University Press, December 2004.*

[6] Thorsten Grötker, Stan Liao, Grant Martin, and Stuart Swan, "System Design with SystemC", *Kluwer Academic Publishers, 2002.*

[7] Daniel D. Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao, "SpecC: Specification Language and Design Methodology", *Kluwer Academic Publishers, 2000.*

[8] SpecC Open Technology Consortium, "SpecC Language Reference Manual, version 2.0", 2004, http://www.specc.org.

[9] A. Gerstlauer, R. Dömer, J. Peng, D.D. Gajski, "System Design: A Practical Guide with SpecC", *Kluwer Academic Publishers, 2001.*

[10]C. Park, J. Liu and Pai. H. Chou, "Eco: an Ultra-Compact Low-Power Wireless Sensor Node for Real-Time Motion Monitoring", IPSN, 2005.

[11]nRF24E1: 2.4GHz Transmitter/MCU/ADC http://www.nvlsi.no/files/Product/data_sheet/nRF24E2rev1_2.pdf

6. Appendix

[A] Simulation Results

[B] Code

Appendix

[A] Simulation Results

```
[gsachdev@alpha eco]$ ./ECO 1000
0.00ms   StimulusID # 1 X = 0.000000  Y = 1.000000
0.00ms   StimulusID # 2 X = 0.000000  Y = 1.000000
100.00ms        StimulusID # 2 X = 0.099833  Y = 0.995004
100.00ms        StimulusID # 1 X = 0.099833  Y = 0.995004
100.00ms        Behavior # Monitor1 Payload1 = 01111100101111100110 Payload2 = 01111100101111100110
100.00ms        Behavior # Monitor1 x1 = -0.004000 y1 = 0.996000 x2 = -0.004000 y2 = 0.996000
200.00ms        StimulusID # 1 X = 0.198669  Y = 0.980067
200.00ms        StimulusID # 2 X = 0.198669  Y = 0.980067
300.00ms        StimulusID # 2 X = 0.295520  Y = 0.955336
300.00ms        StimulusID # 1 X = 0.295520  Y = 0.955336
300.00ms        Behavior # Monitor1 Payload1 = 10001001001111100010 Payload2 = 10001001001111100010
300.00ms        Behavior # Monitor1 x1 = 0.096000 y1 = 0.988000 x2 = 0.096000 y2 = 0.988000
400.00ms        StimulusID # 1 X = 0.389418  Y = 0.921061
400.00ms        StimulusID # 2 X = 0.389418  Y = 0.921061
500.00ms        StimulusID # 2 X = 0.479426  Y = 0.877583
500.00ms        StimulusID # 1 X = 0.479426  Y = 0.877583
500.00ms        Behavior # Monitor1 Payload1 = 10100001101111001111 Payload2 = 10100001101111001111
500.00ms        Behavior # Monitor1 x1 = 0.292000 y1 = 0.950000 x2 = 0.292000 y2 = 0.950000
600.00ms        StimulusID # 1 X = 0.564642  Y = 0.825336
600.00ms        StimulusID # 2 X = 0.564642  Y = 0.825336
700.00ms        StimulusID # 2 X = 0.644218  Y = 0.764842
700.00ms        StimulusID # 1 X = 0.644218  Y = 0.764842
700.00ms        Behavior # Monitor1 Payload1 = 10111000101110101000 Payload2 = 10111000101110101000
700.00ms        Behavior # Monitor1 x1 = 0.476000 y1 = 0.872000 x2 = 0.476000 y2 = 0.872000
800.00ms        StimulusID # 1 X = 0.717356  Y = 0.696707
800.00ms        StimulusID # 2 X = 0.717356  Y = 0.696707
900.00ms        StimulusID # 2 X = 0.783327  Y = 0.621610
900.00ms        StimulusID # 1 X = 0.783327  Y = 0.621610
900.00ms        Behavior # Monitor1 Payload1 = 11001101001101110000 Payload2 = 11001101001101110000
900.00ms        Behavior # Monitor1 x1 = 0.640000 y1 = 0.760000 x2 = 0.640000 y2 = 0.760000
```

```
/*****************************************************************************

*   Filename: cnst.sh

*   Description: Declares the Constant Parameters used in modelling ECO System

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************************/

/* Defines the Time Constants */
#define NS      1                   /* NS = Nano Seconds = 1 Clock Cycle */
#define MS      1000 * NS           /* MS = Micro Seconds = 1000NS */
#define mS      1000 * MS           /* mS = Milli Seconds = 1000MS */
#define S       1000 * mS           /* S = Seconds = 1000mS */

/* Defines the Delays in different  Behaviors */
#define STIMULUS_DELAY 100 * mS         /* Delay in the Stimulus */
#define ADC_DELAY 100 * mS              /* Delay in the ADC */
#define MICROCONTROLLER_DELAY 100 * mS  /* Delay in the Microcontroller */

/* Defines the Preamble and CRC */
#define preamble1 01010101ub
#define preamble2 10101010ub
#define crc 1111111111ub

/* Defines the specific bits in Packet, Package and Payload */
/* LSB = Least Significant bit */
/* MSB = Most Significant bit */
#define LSB_ADDRESS_PACKET 20

#define LSB_PREAMABLE_PACKAGE 40
#define MSB_PREAMABLE_PACKAGE 47

#define LSB_ADDRESS_PACKAGE 30
#define MSB_ADDRESS_PACKAGE 39

#define LSB_CRC_PACKAGE 0
#define MSB_CRC_PACKAGE 9

#define LSB_PAYLOAD_PACKAGE 10
#define MSB_PAYLOAD_PACKAGE 29

#define LSB_SAMPLE1_PAYLOAD 10
#define MSB_SAMPLE1_PAYLOAD 19
#define LSB_SAMPLE2_PAYLOAD 0
#define MSB_SAMPLE2_PAYLOAD 9
```

```
/*****************************************************************

*   Filename: typedef.sh

*   Description: Defines basic types used in modelling ECO System

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************/

/* Analog value is converted in to a digtal value represented by 10 bits */
typedef unsigned bit[10] SAMPLE;

/* Address used to refer each startion is 10 bi t*/
typedef unsigned bit[10] ADDR;

/* CRC used is a 10 bit vector */
typedef unsigned bit[10] CRC;

/* PAYLOAD = SAMPLE(X) @ SAMPLE(y) */
typedef unsigned bit[20] PAYLOAD;

/* PACKET = ADDR @ PAYLOAD */
typedef unsigned bit[30] PACKET;
typedef unsigned bit[8] PREAMBLE;

/* PACKAGE = PREAMBLE @ PACKET @ CRC */
typedef unsigned bit[48] PACKAGE;
```

```
/************************************************************

*   Filename: Stimulus.sc

*   Description: Provides  Analog Values to the Design Unit

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

************************************************************/

#include<stdio.h>
#include<sim.sh>
#include<stdlib.h>
#include<math.h>

#include"cnst.sh"
#include"typedef.sh"
/* SID = Stimulus ID */
behavior Stimulus( inout double X , inout double Y , inout sim_time time , in const int SID )
{
        void main(void)
        {
                sim_time runningtime;           /* Running Time for the Testbench, taken as an command line
argument */
                double a = 0.000;
                runningtime = time * mS;        /* Running Time is in milli Seconds */

                while(runningtime > now())
                {
                        X = sin(a);             /* Provides sensor Analog Output in X and Y direction using Single
Handshake Channel */
                        Y = cos(a);             /* Sin and Cos Functions are being used to make them vary with
time */
                        a = a + 0.1;

                        printf("%2.2fms\t StimulusID # %d X = %f  Y =
%f\n",(double)now()/(double)1000000.0,SID,X,Y);

                        waitfor STIMULUS_DELAY;   /* X and Y values are changed after every
STIMULUS_DELAY */

                }
                exit(0);
        }
};
```

```
/************************************************************

*   Filename: ADC.sc

*   Description: Converts an Analog Value to Digital Value

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

************************************************************/
#include<stdio.h>
#include<sim.sh>
#include"typedef.sh"
#include"cnst.sh"
/* NID = Node ID , ADCID = Analog to Digital Convertor ,SID = Station ID */
behavior ADC( in double INPUT , out SAMPLE OUTPUT , in const int NID , in const int ADCID , in const int SID )
{
        void main(void)
        {

                SAMPLE output;                  /* Output of the ADC */
                double input;           /* Input of the ADC */
                char c[11],d[11];

                while(true)
                {
                        input = INPUT;                          /* Analog value supplied by the Stimulus through
Zero Handshake Channel */
                        input = input + 1;                      /* Input value varies from -1 to 1, first add 1 to the
input value,
                                                                 * so that input value varies from 0 to 2 */
                        output = (int)(input/0.002);            /* converts a analog value to digital */
                                                                /* 10 bits are being used to represent a digital value
for 0 to 2 interval
                                                                 * hence each 0.002 interval of analog value will be
represented
                                                                 * by a digital value, hence either divide by 0.002 or
multiple by 500 */
#ifdef DEBUG
                        printf("%2.2fms\t Node ID # %d Behavior # ADC%d INPUT =
%f",(double)now()/(double)1000000.0,NID,ADCID,input);
                        printf(" OUTPUT = %s\n",ubit2str(2,&c[10],output));
#endif

                        OUTPUT = output;                        /* Provides the Digital output through Zero
Handshake Channel */
                        waitfor ADC_DELAY;                      /* ADC converts an  analog value to digital value
after every ADC_DELAY */
                }
        }
};
```

```
/*****************************************************************************************

*   Filename: Microcontroller.sc

*   Description: Takes samples from the ADC, from an Payload, adds Address and sends the Packet to Sender

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************************************/
#include<stdio.h>
#include<sim.sh>
#include"typedef.sh"
#include"cnst.sh"

import"i_sender";

/* NID = Node ID , SID = Station ID */
behavior Microcontroller( in SAMPLE datax , in SAMPLE datay ,i_sender port , in const int NID , in const int SID )
{
        void main(void)
        {

                char c[11],d[11];
                char e[31];
                SAMPLE X,Y;
                ADDR Addr;
                PAYLOAD payload;
                PACKET packet;

                while(true)
                {
                        X = datax;                              /* Input =  Digital Sample from the ADC for both X
and Y direction */
                        Y = datay;                              /* Using Zero Handshake Channel */
                        payload = X @ Y;                        /* Concatenates the Two Samples */

#ifdef DEBUG
                        printf("%2.2fms\t Node ID # %d Behavior # Microcontroller X = %s Y =
%s\n",(double)now()/(double)1000000.0,NID,ubit2str(2,&c[10],X),ubit2str(2,&d[10],Y));
#endif
                        Addr = SID;                             /* Assigns the Station ID to each payload */
                        packet = Addr @ payload;                /* Concatenates  Address of the Station to each
packet */
                        port.send(&packet,sizeof(packet));      /* Sends the packet to the Sender using double
Handsahke Channel */
                        waitfor MICROCONTROLLER_DELAY;                  /* Microcontroller sends a packet
after every MICROCONTROLLER_DELAY */
                }
        }
};
```

```
/*****************************************************************************************

*   Filename: Sender.sc

*   Description: Receives Payload from Microcontroller,adds Preamble & CRC to it and sends the package to the
Station

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************************************/

#include<stdio.h>
#include<sim.sh>

#include"typedef.sh"
#include"cnst.sh"

import"i_receiver";
import"i_send";

/* NID = Node ID , SID = Station ID */
behavior Sender(i_receiver port,i_send wirelessport,out PACKAGE package,in const int NID,in const int SID)
{
        void main(void)
        {

                char c[31];
                char d[49];
                PACKET packet;
                PACKAGE data;
                PAYLOAD payload;

                while(true)
                {
                        port.receive(&packet,sizeof(packet));     /* Receives the Packet from the MicroCotroller
using double Handshake Channel */


#ifdef DEBUG
                        printf("%2.2fms\t Node ID # %d Behavior # Sender packet =
%s\n",(double)now()/(double)1000000.0,NID,ubit2str(2,&c[30],packet));
#endif

                        /* Adds preamble and CRC to the packet and forms the Package */
                        if(packet[LSB_ADDRESS_PACKET] == 0)
                        {
                                data = preamble1 @ packet @ crc;
                        }
                        else if(packet[LSB_ADDRESS_PACKET] == 1)
                        {
```

```
                                data = preamble2 @ packet @ crc;
                        }

#ifdef DEBUG
                        printf("%2.2fms\t Node ID # %d Behavior # Sender package =
%s\n",(double)now()/(double)1000000.0,NID,ubit2str(2,&d[48],data));
#endif

                        package = data;         /* Transmits the Package to the Station */
                        wirelessport.send();    /* Using Sigle Handshake Channel */

                }
        }
};
```

```
/*****************************************************************

*   Filename: Node.sc

*   Description: Contains Two ADC , Microcontroller and Sender

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************/

#include<stdio.h>
#include<sim.sh>
#include<stdlib.h>

#include"typedef.sh"
#include"cnst.sh"

import"c_double_handshake";
import"i_send";

import"ADC";
import"Microcontroller";
import"Sender";

/* NID = Node ID , SID = Station ID */
behavior Node(in double X,in double Y,i_send wirelessport,out PACKAGE package,in int const NID,in const int
SID)
{
        c_double_handshake port;
        unsigned bit[10] datax,datay;
        ADC adc1(X,datax,NID,1,SID);                                    /* Declares ADC's */
        ADC adc2(Y,datay,NID,2,SID);
        Microcontroller microcontroller(datax,datay,port,NID,SID);      /* Declares the Microcontroller */
        Sender sender(port,wirelessport,package,NID,SID);               /* Declares the Sender */

        void main(void)
        {
                par
                {
                        adc1.main();
                        adc2.main();
                        microcontroller.main();
                        sender.main();
                }
        }
};
```

```
/*****************************************************************************************

*   Filename: Receiver.sc

*   Description: Receives Package from Node, Checks the Address, Preamble and CRC of each Package

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************************************/

#include<stdio.h>
#include<sim.sh>

#include"cnst.sh"
#include"typedef.sh"

import"i_sender";
import"i_receive";

/* SID = Station ID , RID = Receiver ID */
behavior Receiver(i_receive wirelessport,in PACKAGE package,i_sender port,in const int SID,in const int RID)
{
        void main(void)
        {

                char c[49];
                char e[9],f[11],g[21],h[11];
                PACKAGE data;
                PAYLOAD payload;
                ADDR addr1,addr;
                CRC crc1;
                PREAMBLE preamble;

                addr1 = SID;                    /* Assigns the address of the Station, each Package Address is
                                                 * checked with the Station Address */
                while(true)
                {
                        wirelessport.receive();                 /* Receives the package through Single
Handshake Channel */
                        data = package;

#ifdef DEBUG
                        printf("%2.2fms\t Station ID # %d Behavior # Receiver%d data =
%s\n",(double)now()/(double)1000000.0,SID,RID,ubit2str(2,&c[48],data));
#endif

                        preamble = data[MSB_PREAMABLE_PACKAGE:LSB_PREAMABLE_PACKAGE];   /*
Slices the Preamble from the Package */
                        if(preamble == preamble1 || preamble == preamble2)              /* Checks the Preamble */
                        {
```

```
                                addr = data[MSB_ADDRESS_PACKAGE:LSB_ADDRESS_PACKAGE];          /*
Slices the Address from the Package */
                                if(addr == addr1)                                               /* Checks the Address */
                                {
                                        crc1 = data[MSB_CRC_PACKAGE:LSB_CRC_PACKAGE];    /* Slices the
CRC */
                                        if(crc1 == crc)                                         /* Checks the CRC */
                                        {
                                                payload =
data[MSB_PAYLOAD_PACKAGE:LSB_PAYLOAD_PACKAGE];      /* Slices the Payload */
                                                port.send(&payload,sizeof(payload)); /* Sends the payload to CPU
                                                                                        * if Preamble, Address &
                                                                                        * CRC are correct, using
                                                                                        * Double Handshake Channel
*/
                                        }
                                        else
                                                printf("%2.2fms\t Station ID # %d Behavior # Receiver%d Packet
received doesn't have correct CRC\n",(double)now()/(double)1000000.0,SID,RID);
                                }
                                else
                                        printf("%2.2fms\t Station ID # %d Behavior # Receiver%d Packet received
doesn't have correct Address\n",(double)now()/(double)1000000.0,SID,RID);
                        }
                        else
                                printf("%2.2fms\t Station ID # %d Behavior # Receiver%d Packet received doesn't
have correct Preamble\n",(double)now()/(double)1000000.0,SID,RID);
                }
        }
};
```

```
/*****************************************************************************************

*   Filename: CPU.sc

*   Description: Receives Commands from Monitor, Payload from Receiver and provides the payload to the Monitor

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************************************/

#include<stdio.h>
#include<sim.sh>

#include"cnst.sh"
#include"typedef.sh"

import"i_tranceiver";
import"i_receiver";

/* SID = Station ID */
behavior CPU(i_tranceiver port,i_receiver port1,i_receiver port2,in const int SID)
{
        void main(void)
        {

                char c[21],d[21];
                PAYLOAD avgpayload1,avgpayload2;
                PAYLOAD payload1[10],payload2[10];
                int i,j;

                SAMPLE AX1,AY1,AX2,AY2;
                SAMPLE X1[10],Y1[10],X2[10],Y2[10];
                int NoOfSamples;

                port.receive(&NoOfSamples,sizeof(NoOfSamples));         /* Command Received from the
Monitor */
                                                                        /* Monitor provides the No of
Samples that need to be
                                                                         * Averaged before sending to the
MOnitor */
#ifdef DEBUG
                printf("%2.2fms\t Station ID # %d Behavior # CPU No Of Samples =
%d\n",(double)now()/(double)1000000.0,SID,NoOfSamples);
#endif

                while(true)
                {       i = 0;
                        AX1 = 0;                                        /* Contains the Average Value */
                        AY1 = 0;                                        /* Initialize then to Zero in each
Cycle */
```

```
                        AX2 = 0;
                        AY2 = 0;
                        while(i < NoOfSamples)
                        {

                        port1.receive(&payload1[i],sizeof(payload1[i])); /* Payload from  Two Receiver */
                        port2.receive(&payload2[i],sizeof(payload2[i]));

#ifdef DEBUG
                        printf("%2.2fms\t Station ID # %d Behavior # CPU Payload1[%d] = %s Payload2[%d] =
%s\n",(double)now()/(double)1000000.0,SID,i,ubit2str(2,&c[20],payload1[i]),i,ubit2str(2,&d[21],payload2[i]));
#endif

                        X1[i] = payload1[i][MSB_SAMPLE1_PAYLOAD:LSB_SAMPLE1_PAYLOAD]; /*
Separates each Sample from Payload*/
                        Y1[i] = payload1[i][MSB_SAMPLE2_PAYLOAD:LSB_SAMPLE2_PAYLOAD];
                        X2[i] = payload2[i][MSB_SAMPLE1_PAYLOAD:LSB_SAMPLE1_PAYLOAD];
                        Y2[i] = payload2[i][MSB_SAMPLE2_PAYLOAD:LSB_SAMPLE2_PAYLOAD];
                        i++;
                        }

                        /* Calculates the Average of the Received Samples */
                        for(j=0; j < i ; j++)
                        {
                        AX1 = AX1 + (X1[j]/NoOfSamples);
                        AY1 = AY1 + (Y1[j]/NoOfSamples);
                        AX2 = AX2 + (X2[j]/NoOfSamples);
                        AY2 = AY2 + (Y2[j]/NoOfSamples);
                        }

                        /* Forms the payload again */
                        avgpayload1 = AX1 @ AY1;
                        avgpayload2 = AX2 @ AY2;
                        port.send(&avgpayload1,sizeof(avgpayload1));   /* Sends the Payload to the Monitor*/
                        port.send(&avgpayload2,sizeof(avgpayload2));
                }
        }
};
```

```
/*****************************************************************

*   Filename: Station.sc

*   Description: Contains Two receiver and a CPU

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************/

#include<stdio.h>
#include<sim.sh>

#include"cnst.sh"
#include"typedef.sh"

import"c_double_handshake";
import"i_tranceiver";
import"i_receive";

import"Receiver";
import"CPU";

behavior Station(i_tranceiver port,i_receive wirelessport1,in PACKAGE package1,i_receive wirelessport2,in
PACKAGE package2,in const int SID)
{
        c_double_handshake port1,port2;
        Receiver receiver1(wirelessport1,package1,port1,SID,1);          /* Declares Two Receivers */
        Receiver receiver2(wirelessport2,package2,port2,SID,2);
        CPU cpu(port,port1,port2,SID);                                   /* Declares the CPU */

        void main(void)
        {
                par
                {
                        receiver1.main();
                        receiver2.main();
                        cpu.main();
                }
        }
};
```

```
/*****************************************************************

*   Filename: ECO.sc

*   Description: Declares the complite Testbench for ECO System

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************/

#include"cnst.sh"
#include"typedef.sh"

#include<stdio.h>
#include<sim.sh>
#include<stdlib.h>


import"c_double_handshake";
import"i_tranceiver";
import"c_handshake";
import"i_receive";

import"Stimulus";
import"Node";
import"Station";
import"Monitor";


behavior Main
{
        sim_time time;

        /* For Two Nodes and One Station */
        c_handshake wirelessport1,wirelessport2;
        c_double_handshake port1;
        PACKAGE package1,package2;
        double X1,Y1,X2,Y2;

        Stimulus stimulus1(X1,Y1,time,1);                                   /* Declares Two Stimulus */
        Stimulus stimulus2(X2,Y2,time,2);
        Node node1(X1,Y1,wirelessport1,package1,1,1);                           /* Declares Two
Nodes */
        Node node2(X2,Y2,wirelessport2,package2,2,1);
        Station station1(port1,wirelessport1,package1,wirelessport2,package2,1);/* Declares a Station */
        Monitor monitor1(port1,1);                                          /* Declares a Monitor */

        /* For Four Nodes and Two Station */
/*      c_handshake wirelessport3,wirelessport4;
        c_double_handshake port2;
```

```
PACKAGE package3,package4;
double X3,Y3,X4,Y4;

Stimulus stimulus3(X3,Y3,time,3);
Stimulus stimulus4(X4,Y4,time,4);
Node node3(X3,Y4,wirelessport3,package3,3,2);
Node node4(X4,Y4,wirelessport4,package4,4,2);
Station station2(port2,wirelessport3,package3,wirelessport3,package3,2);
Monitor monitor2(port2,2); */


int main(int argc , char *argv[])
{

        if( argc == 2)
        {
                /* Running Time for Testbench is taken as a command-line argument */
                time = atol(argv[1]);

                /* Runs the Testbench(ECO System) */
                par
                {

                        stimulus1.main();
                        stimulus2.main();
                        node1.main();
                        node2.main();
                        station1.main();
                        monitor1.main();

/*
                        stimulus3.main();
                        stimulus4.main();
                        node3.main();
                        node4.main();
                        station2.main();
                        monitor2.main();
*/

                }
        }
        else
                printf("No Argument Supplied\n");
        return(0);
}
};
```

```
/*****************************************************************************

*   Filename: Monitor.sc

*   Description: Provides Commands to Station and Receives the payload from Station

*   Author: Gautam Sachdeva

*   Last Update: 6/22/2005

*****************************************************************************/
#include<stdio.h>
#include<sim.sh>
#include"cnst.sh"
#include"typedef.sh"
import"i_tranceiver";
behavior Monitor(i_tranceiver port, in const int MID)
{
        void main(void)
        {
                double x1,y1,x2,y2;
                SAMPLE X1,Y1,X2,Y2;
                PAYLOAD payload1,payload2;
                char c[21],d[21];
                int NoOfSamples = 2;

                port.send(&NoOfSamples,sizeof(NoOfSamples));
                while(true)
                {
                        port.receive(&payload1,sizeof(payload1));                       /* Receives the payload
from the Station */
                        port.receive(&payload2,sizeof(payload2));

                        X1 = payload1[MSB_SAMPLE1_PAYLOAD:LSB_SAMPLE1_PAYLOAD];    /* Separates
each Sample */
                        Y1 = payload1[MSB_SAMPLE2_PAYLOAD:LSB_SAMPLE2_PAYLOAD];
                        X2 = payload2[MSB_SAMPLE1_PAYLOAD:LSB_SAMPLE1_PAYLOAD];
                        Y2 = payload2[MSB_SAMPLE2_PAYLOAD:LSB_SAMPLE2_PAYLOAD];

                        x1 =(((double) X1) * 0.002)-1;                                  /* Converts each Sample
back to analog*/
                        y1 =(((double) Y1) * 0.002)-1;
                        x2 =(((double) X2) * 0.002)-1;
                        y2 =(((double) Y2) * 0.002)-1;

                        printf("%2.2fms\t Behavior # Monitor%d Payload1 = %s Payload2 =
%s\n",(double)now()/(double)1000000.0,MID,ubit2str(2,&c[20],payload1),ubit2str(2,&d[20],payload2));
                printf("%2.2fms\t Behavior # Monitor%d x1 = %f y1 = %f x2 = %f y2 =
%f\n",(double)now()/(double)1000000.0,MID,x1,y1,x2,y2);
                }
        }
};
```