

Performance Modeling of the Pico-Radio Sensor Network in a VCC Environment

Tiffany Crawford
Howard University
Washington, DC 2001
tscrawford81@hotmail.com

Abstract

Imagine designing a system, choosing and defining a specific architecture, and implementing a design only to find out that the system is not fully compatible with the chosen architecture, or another architecture would have met your constraints more efficiently. Not only have you wasted time and money but also your overall performance is sub-optimal. Being able to accurately estimate how well a system will perform on a specific architecture before completing the system and/or completely defining the architecture allows a designer to make trade off decisions concerning cost, design time, and performance before reaching the stages of implementation in which a design becomes more costly and time-consuming to change. The steps described in this paper to model the Pico-Radio application layer allow designers to make trade-off and optimization decisions at a high level of abstraction.

1. Introduction

1.1 Modeling

Modeling describes a system's structure and behavior. Models have many forms and uses, such as helping scientists and engineers communicate ideas, organize thoughts, or simulate behavior and performance. The latter is one of the key components in high-level design.

1.2 Sensor Networks

Sensor networks are networks of nodes that sense data, usually environmental (ie: temperature and lighting). The nodes typically represent sensors and other behaviors. Typical sensor networks are not very dense, the nodes dissipate a moderate amount of power, and are not necessarily small.

1.3 PicoRadio

The PicoRadio system is a dense, ad hoc wireless sensor network composed of nodes that communicate across radio links. The nodes in a PicoRadio network can have one of three behaviors: controller, actuator, or sensor. These nodes are randomly located throughout a room or an entire building. They can be located in places such as within the wall, ceiling, or floor. Once installed they are expected to be self-contained for the life time of the building.

PicoRadio is a specialized form of a sensor network. An important notion is the idea of a *smart building*. The sensors sample data such as temperature, room entry and exit, or the time grass needs to be watered. Imagine walking into your office at 9:00 am on a cold winter morning to a hot cup of coffee that was poured at 8:59 and the temperature just where you like it because the heat was activated at 8:55, all due to a PicoRadio network that was expecting your arrival.

It is apparent that the PicoRadio system is complex with many design issues. How will the layers communicate with each other? How much power will be dissipated? How long will it take data packets to travel across the network? How fast will the network perform its primary functions? These are just a few examples.

1.4 Co-Verification Tools

It is important to have an idea of the answers to some of these questions in the abstract stages of design, allowing the conceptual ideas to begin to take form. Co-verification tools allow one to do this. System behaviors can be modeled without having to worry about intricate implementation details; you can build a complete virtual model of your system that can be simulated before building or buying one piece of hardware. The target architecture

e.g. processors, ASIC's, and FPGA's can also be modeled using these tools. The behavior can then be mapped to one or more target architectures and simulated for logical correctness and performance. This allows for a high level implementation that can give fairly accurate estimates of how a system will perform when implemented in hardware; design decisions can be refined as more details are added. This allows the designer to spend more time in the productive conceptual stages while significantly decreasing time for implementation, decreasing overall design time

1.5 Goal of Project

The goal of this project was to define and create a behavior model of the actuator, as well as an architecture model of the StrongARM.

This paper is organized as follows: Section two gives an in-depth description of the PicoRadio system. Section three discusses the six steps of modeling in VCC. Section four gives some insight to the actuator node behavioral model. Section five discusses a methodology used in developing a model of the software target architecture for the PicoRadio Test Bed: the StrongARM 1100.

2. PicoRadio

2.1 Behavior of the Nodes

The Pico Radio network has three nodes types: actuator, controller and sensor. System functionality occurs through a feedback loop between the three that includes the environment. The controller receives input from the user via a user interface or stored program. This data can be one of two types: a command or a request. If it is a command the controller sends it to the actuator. When the actuator receives a command it puts it in a linked list of commands to be dealt with and sends an acknowledgement. This acknowledgement lets the controller know that the packet has been received and another packet can be sent. When the actuator handles the command, another acknowledgement is sent letting the controller know that the action was completed, or started for periodic commands. This second acknowledgement contains data that the controller rearranges into a request to be sent to the sensor to make sure that "action" has occurred.

The sensor receives requests and puts them in a linked list to be handled. When the sensor receives the data it also sends an acknowledgement to the controller to let it (controller) know that it (sensor) is ready to receive more requests. When the sensor handles the requests it sends the sensed data back to the controller.

The controller receives this data and handles it accordingly. If it is data collected due to a sensor request it is sent to an output. If it is data collected due to a command the controller determines if the desired value or state has been reached and proceeds accordingly.

Command

The *command* is a message from the controller to the actuator that tells the actuator what needs to be done. It is in the form of a structure with seven fields.

The first two fields are source and destination addresses. The addresses themselves are structures with five fields; X, Y, Z, Node type, and Node Subtype. X, Y, and Z are Cartesian coordinates relative to some physical space. The node type can be actuator, controller, or sensor. The node subtype represents a refinement of type, such as temperature or humidity for sensors.

The third field represents the packet type command: data, or interest.

For actuators, the fourth and fifth, Direction and Quantity, represent the action to be performed. Direction represents the direction in which the value should be changed and the Quantity represents the absolute value of the change.

The final two fields, Period and Duration, deal with periodic requests. The first is the period i.e. how often a command should occur, and the second represents how long to continue the action.

For example, say an actuator that controlled temperature received a command with Direction = 1, representing up, Quantity = 5, Period = 5, and Duration = 20. The actuator would increase the temperature by 5 steps every five seconds for twenty seconds. Steps could be degrees in Celsius, Fahrenheit, fraction of degrees, or some other unit.

Request

The *request* is a message from controller to sensor that asks for data. It has five fields instead of seven: Source Address, Destination Address, Packet Type, App Field 1 and App Field 2.

Data

The *data* is a message from the sensor to the controller containing the requested data. Like request, it has five fields: Source Address, Destination Address, Packet Type, Value and Samples. Value contains a numeric value derived from sensor samples, and Samples indicates the number of samples summed into Value.

Acknowledgement

After every packet is received there is an acknowledgement sent from the receiver to the sender to indicate that the packet has been received. There is also a second type of acknowledgement sent from the actuator to the controller after it has done what it was told to do. Acknowledgements contains three fields: destination address, source address, and type. This allows the controller to send a request to a corresponding sensor to make sure the proper value has been achieved.

2.2 Layers

The Pico Radio consists of four layers: application, network, media access control (MAC), and physical.

The application layer includes behavior of sensors, actuators, and controllers. The application layer in a node communicates to the application layer in other nodes through the network layer.

The network layer enables nodes that are not in physical range of each other to communicate. The physical range due to requirements for low power dissipation levels is limited to about ten meters, therefore, nodes communicate through a multi-hop system; data packets are transferred from node to node until they reach their destination. The network layers job is to determine a feasible and optimal path for the data packets.

For nodes that are close to each other it is important to coordinate radio traffic in such a manner that they do not interfere with each other. The MAC layer, which is connected to the network layer, performs this function.

Finally, the physical layer actively transports data on the media. The radio transmitter and receiver are part of the physical layer.

2.3 Important Issues In Design of Pico-Radio

It is apparent that the system for the Pico-Radio is complex and involves many design issues. How will the layers communicate with each other? How much power will be dissipated? How long will data packets take to travel across the network? How fast will the network perform its primary functions? These issues and others can be addressed using VCC performance modeling.

3. Modeling in VCC

VCC is a high-level design tool from Cadence, Inc. Modeling in VCC has six basic steps:

- 1) Behavior modeling: Includes two views: a symbol description and programming code (C or C++). In its most basic form the symbol consists of a rectangle with input and output ports. The code consists of an 'init' function, which contains initialization conditions to be set at the beginning of execution, and a 'run' function which contains all functions and conditions to be performed during execution. The code is "embedded" into the symbol so a symbolic representation (a *schematic*, in this case) of a complex system with many code modules can be executable.
- 2) Functional simulation: This stage contains three views. A behavioral view, analysis view and results view. The behavioral view allows the designer to instantiate different symbols and connect them as needed through their various ports. The analysis view is where the designer can input probes, breakpoints, and other devices to debug and analyze the system while it is being executed. The results view displays the results; these can be in the form of charts, graphs, tables, etc.
- 3) Architecture Modeling: This stage includes an architecture view. In VCC there are different resource types (ie: Processor, memory, ASIC) that can be used for the designer's purposes. These resources can be described using VCC defined service declarations that are bound to VCC defined architecture services. Both include code which specifies services that the resource can handle.
- 4) Mapping: This stage includes the mapping view. In this view the designer can instantiate various behaviors and an architecture. The behaviors can communicate with each other if need be, and they are all mapped on to the chosen target architecture.
- 5) Performance simulation: This stage, the final stage discussed in this paper, is very similar to the second stage. The major difference is that performance, represented by the mapping view, is being simulated instead of functional behavior, represented by the behavior view. In order to measure performance we need to have some idea of physical architecture. This comes from the mapping view.

This stage also contains three views: mapping, analysis and results. The mapping view is what is being simulated. The analysis view is where the designer can input probes, breakpoints, and other devices to debug and analyze the system while it is being executed. The results view houses the results that can be in the form of charts graphs, tables, etc.

- 6) Links to Implementation: This stage is beyond the scope of this paper and will not be discussed.

4. Behavioral Modeling (The actuator)

As mentioned previously, the PicoRadio system is composed of three elements. The actuator is the active element. It causes an action to occur, hence the name. It is a physical entity such as a solenoid or a relay. The actuator modeled in this project is generic, that is, it takes into consideration many modes that an actuator can operate in, such as discrete analog or continuous analog.

The actuator modeled in this paper handles all signals in the same manner. The step sizes are the primary distinction between continuous and discrete analog. When the nodes are set in place and the system is initialized, all actuator nodes will send a packet of “personal” information to the closest controller so that the controller knows that the sensor exists. The packet contains the actuators address including its subtype, high and low limits, and resolution, or minimum step size. With this information, when the controller receives a command to increase temperature it knows which nodes have the ability to do so, and how to communicate with those nodes.

When the actuator receives a command, the command is put into a first in first out queue (FIFO). The actuator then handles the command appropriately. For example, if the controller wants the temperature in a certain area to be higher, the command to the actuator would be to increase the temperature by X degrees Celsius. Of course this is not what happens in real life. This command could, for instance, request that a vent be closed to 10% of full. The rising temperature is detected by the sensor. In essence, there is a feedback loop from the actuator to the sensor through the environment

5. Architectural Model of the StrongARM 1100

5.1 The StrongARM

The StrongARM microprocessor is being modeled because it is used on the PicoRadio Test Bed, which is a prototype for the eventual PicoRadio system. The StrongARM is a RISC (Reduced Instruction Set Computer) processor. It is a low-power device which makes it ideal for modeling processing engine of a PicoNode. The model in this paper can be used to design and pretest implementations for the Test Bed.

5.2 Architectural model in VCC

VCC has predefined architectures, such as an ASIC or a CPU, called resource types. These resource types come with a set of predefined services and parameter bound to the services. A VCC user has the ability to choose which services its architecture will contain. There are of course some basic services, such as a delay model for processors.

Since the StrongARM is a microprocessor, the CPU resource type is used to model it. Similar to how a behavioral model has code behind it to describe behavior the CPU model has a “processor” view that contains a virtual instruction set. The instruction syntax is simple: *inst, opcode, delay*. *Inst* represents that it is an instruction. *Opcode* represents the action of the instruction e.g. load, store, or multiply. *Delay* represents how many clock cycles it takes to perform that function. Delay is what needs to be determined in creating a processor model targeted at a specific CPU.

5.3 Methodology for finding delays

It seems as if finding the delays would be as simple as looking in the data book for delay values. There are two factors that cause this approach to be inaccurate and in some cases infeasible. For one, one to one matches between the virtual instructions and the processor instruction set do not exist in all cases. Secondly, it is difficult to model memory access. The processor does not access memory the same way in all programs, particularly between data oriented and control oriented programs.

There are six basic steps to finding delays for a processor model:

- 1) Develop or find benchmarks that test each instruction in the virtual instruction set.
- 2) Map these benchmarks on to an instance of a VCC defined CPU model.
- 3) In the analysis view of the above mapping get the amount of times **each** virtual instruction occurs in each benchmark.
- 4) Use an instruction set simulator (ISS) to measure the **total** instruction count for each benchmark.
- 5) From the data found in the steps 3 and 4, form a benchmark by solving a linear matrix of the following equations.

$$\text{Diff1} = \text{XLd} * \text{Ld1} + \text{XSt} * \text{St1} + \dots + \text{XOp.c} * \text{Op.c1} - \text{ISS cycle count1}$$

.
. .
. .
. .

$$\text{Diffn} = \text{XLd} * \text{Ldn} + \text{XSt} * \text{Stn} + \dots + \text{XOp.c} * \text{Op.cn} - \text{ISS cycle countn}$$

The instructions in the equation, i.e. Ld1, represent the number of times the instruction occurred in that benchmark. The X's are independent variables that are

being solved for. Essentially the equation represents the difference between the total number of instructions that occurred in a certain benchmark and the total number of cycle counts it took the ISS to run the same benchmark.

The Excel Solver Spreadsheet finds values for the independent variables that gives a minimum difference, or best fit. In the best-case scenario there are an equal number of benchmarks and instructions.

- 6) Refine delays. Once the delays are found using the above method they need to be refined according to the type of program you have i.e. control oriented or data oriented.

6. Conclusion

Once the architecture model with the refined delays in the processor model(s) is complete a performance simulation can be ran. In this case the programs are all control oriented so they can all be mapped onto the same processor file. This simulation can give insight to CPU load and cycle time that is pertinent data in determining power consumption. Which is a very important issue in the PicoRadio research group.

7. Acknowledgements

I would like to thank Professor Jan Rabaey and my mentor Fred Burghardt, and Marie Mayne, Monica Lin, Sheila Humphreys of the SUPERB staff. Also, I'd like to thank all the SUPERBites for a great summer!

