

SUNSHINE: A Multi-Domain Sensor Network Simulator

Jingyao Zhang

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Yaling Yang, Chair
Y.Thomas Hou
Danfeng Yao

August 27, 2010
Blacksburg, Virginia

Keywords: Sensor networks, cross-domain simulator,
hardware-software co-simulation, performance
Copyright 2010, Jingyao Zhang

SUNSHINE: A Multi-Domain Sensor Network Simulator

Jingyao Zhang

(ABSTRACT)

Simulators are important tools for analyzing and evaluating different design options for wireless sensor networks (sensornets) and hence, have been intensively studied in the past decades. However, existing simulators only support evaluations of protocols and software aspects of sensornet design. They cannot accurately capture the significant impacts of various hardware designs on sensornet performance. As a result, the performance/energy benefits of customized hardware designs are difficult to be evaluated in sensornet research. To fill in this technical void, in this thesis, we describe the design and implementation of SUNSHINE, a scalable hardware-software cross-domain simulator for sensornet applications. SUNSHINE is the first sensornet simulator that effectively supports joint evaluation and design of sensor hardware and software performance in a networked context. SUNSHINE captures the performance of network protocols, software and hardware up to cycle-level accuracy through its seamless integration of three existing sensornet simulators: a network simulator TOSSIM [1], an instruction-set simulator SimulAVR [2] and a hardware simulator GEZEL [3]. SUNSHINE solves challenging design problems, including data exchanges and time synchronizations across different simulation domains and simulation accuracy levels. SUNSHINE also provides hardware specification scheme for simulating flexible and customized hardware designs. Several experiments are given to illustrate SUNSHINE's cross-domain simulation capability, demonstrating that SUNSHINE is an efficient tool for software-hardware codesign in sensornet research.

Acknowledgments

I would like to express my deep and sincere gratitude to my advisor Dr. Yaling Yang for giving me an opportunity to work with her. I am extremely privileged to be a student under Dr. Yang's guidance. Her extensive knowledge, rigorous research attitude and creative thinking have been a source of inspiration for me throughout the years. Dr. Yang is always accessible and willing to help her students not only on research, but also on life. I have learned a lot from her about research methods, life attitude and most importantly, how to interact with others.

I would like to express my warm and sincere thanks to Dr. Patrick Schaumont who gives me a lot of help and advice on the project with immense patience. His broad vision and keen insight on the technical issues have greatly inspired me during my stay at Virginia Tech.

It is my pleasure to work in SHINE group. Everyone in this group are nice and diligent. I am so glad that I will continue my Ph.D. study in the same group.

I would like to thank my friends in Blacksburg and my roommates Yue Meng and Hengheng Chen who have made the past two years unforgettable.

I want to thank my boyfriend Bin Gu. I really appreciate everything he has done for me. His company and his encouragement have made my life meaningful.

I would like to express my thanks to my aunts, my uncles and my cousins who have been very supportive of my studies and graduate education.

My deepest gratitude goes to my parents for their unconditional love, care and support for my study.

Grant Information

This thesis is supported by the National Science Foundation under Grant No. CCF-0916763. Any opinions, results and conclusions or recommendations expressed in this material and related work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Contents

1	INTRODUCTION	1
1.1	Motivation	1
1.2	Related Articles	2
1.3	Thesis Organization	3
2	BACKGROUND	4
2.1	Related Work	4
2.1.1	NS-2	4
2.1.2	TOSSIM	5
	TOSSIM architecture	5
	TOSSIM characteristics	6
	TOSSIM drawbacks	7
2.1.3	ATEMU	8
2.1.4	Avrora	9
2.2	Comparisons of SUNSHINE with Existing Simulators	10
3	SYSTEM DESCRIPTION	12
3.1	System Components	12
3.1.1	SimulAVR	12
3.1.2	GEZEL	13
3.2	System Design Flow	13
3.3	System Architecture	14

4	CC2420 IPBLOCK	18
4.1	SPI Communication	19
4.2	Finite State Machine of CC2420ipblock	21
5	CROSS-DOMAIN INTERFACE	24
5.1	Timing Synchronization	24
5.2	Cross-Domain Data Exchange	26
6	HARDWARE SIMULATION SUPPORT	30
6.1	Hardware Specification Scheme	30
6.2	Hardware Behavior	33
7	EXPERIMENTS	35
7.1	Basic 2-node Sensor Network Simulation Experiments	35
7.2	Tradeoff between Simulation Accuracy and Wall Clock Time	36
7.3	Effect of Computation Load on Sensor Nodes	39
8	CONCLUSION AND FUTURE WORK	42
A	HARDWARE SPECIFICATION FILE	46
B	SUNSHINE MAIN SIMULATION FILE	51

List of Figures

2.1	TOSSIM architecture	6
2.2	ATEMU components architecture	8
2.3	Avrora software architecture	9
3.1	Network Design Flow using SUNSHINE	14
3.2	Simulation domains	15
3.3	Software architecture	16
4.1	Integration of different domain simulators	19
4.2	SPI connection	20
4.3	SPI timing diagram	20
4.4	Interaction between CC2420ipblock and TOSSIM	22
4.5	CC2420 Radio Control States	23
5.1	Simulation time in different domains	25
5.2	Synchronization Algorithm	25
5.3	The synchronized simulation time in SUNSHINE	27
5.4	Converting a functional-level event to cycle-level events	28
5.5	Event conversion process	29
6.1	Hardware specification for a single node (Multiple nodes can be captured by instantiating multiple AVR and multiple radio chip modules.)	32
6.2	Traces for TinyOS Reception application	33

7.1	Simulation configurations. Co-sim nodes are marked with shades.	37
7.2	Simulated time in terms of number of co-sim nodes	38
7.3	Wall clock time in terms of number of co-sim nodes	39
7.4	Simulated time in terms of computation intensity	40
7.5	Wall clock time in terms of computation intensity	40

List of Tables

2.1	Comparison between simulators	11
7.1	Simulation cases	36

Chapter 1

INTRODUCTION

1.1 Motivation

Over the past few years, we have witnessed an impressive growth of sensor network (sensor-net) applications, ranging from environmental monitoring, to health care and home entertainment. A remaining roadblock to the success of sensornets is the constrained processing-power and energy-budget of existing sensor node platforms. This prevents many interesting candidate applications, whose software implementations are prohibitively slow and energy-wise impractical over these platforms. On the other hand, in the hardware community, it is well-known that the specialized hardware implementation of demanding sensor tasks can outperform equivalent software implementations by orders of magnitude. In addition, recent advances in low-power programmable hardware chips (Field-Programmable Gate Arrays) have made flexible and efficient hardware implementations achievable for sensor node architectures [4]. As a result, the joint software-and-hardware design of a sensornet application is a very appealing approach to support sensornets.

Unfortunately, joint software-and-hardware designs of sensornet applications remain largely unexplored since there is no effective simulation tool for these designs. Due to the distributed

nature of sensor networks, simulators are necessary tools to help sensor network researchers develop and analyze new designs. Developing hardware-and-software codesigned sensor network applications would have been an extremely difficult job without the help of a good simulation and analysis instrument. While a great effort has been invested in developing sensor network simulators, these existing sensor network simulators, such as TOSSIM [1], ATEMU [5], and Avrora [6] focus on evaluating the designs of communication protocols and application software. They all assume a fixed hardware platform and their over-simplified models of hardware cannot accurately capture the impact of alternative hardware designs on the performance of network applications. As a result, sensor network researchers cannot easily configure and evaluate various joint software-and-hardware designs and are forced to fit into the constraints of existing fixed sensor hardware platforms. This lack of simulator support also makes it difficult for the sensor network research community to develop a clear direction on improving the sensor hardware platforms. The performance/energy benefits that are available to the hardware community therefore remain hard to reach.

To address this critical problem, we developed a new sensor network simulator, named SUNSHINE (Sensor Unified Analyzer for Software and Hardware in Networked Environments), to support hardware-software co-design in sensor networks. By the integration of a network simulator TOSSIM, an instruction-set simulator SimulAVR, and a hardware simulator GEZEL, SUNSHINE can simulate the impact of various hardware designs on sensor networks at cycle-level accuracy. The performance of software network protocols and applications under realistic hardware constraints and network settings can be captured by SUNSHINE.

1.2 Related Articles

The SUNSHINE extension is described in:

1. Jingyao Zhang, Yi Tang, Sachin Hirve, Patrick Schaumont and Yaling Yang, “SUNSHINE: A Multi-Domain Sensor Network Simulator”, ACM Mobicom 2010 (poster).
2. Jingyao Zhang, Yi Tang, Sachin Hirve, Patrick Schaumont and Yaling Yang, “Cross-Domain Design Tools for Sensor Network and Architecture”, US-Korea Conference on Science, Technology, and Entrepreneurship (UKC) 2010 (poster).
3. Jingyao Zhang, Yi Tang, Sachin Hirve, Srikrishna Iyer, Patrick Schaumont and Yaling Yang, “SUNSHINE: A Multi-Domain Sensor Network Simulator”, IEEE INFOCOM 2011 (submitted).

1.3 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 first introduces a few widely used sensornet simulators, and then makes comparisons between SUNSHINE and the existing network simulators. Chapter 3 provides a description of SUNSHINE’s architecture. Chapter 4 gives details on implementing interfaces between different domain simulators. Chapter 5 discusses cross-domain techniques used in SUNSHINE. Chapter 6 introduces SUNSHINE’s hardware simulation behavior. Chapter 7 gives several experimental results using SUNSHINE. Finally, Chapter 8 provides some conclusions and future works.

Chapter 2

BACKGROUND

Due to the difficulties in setting up sensor network testbeds, sensor network researchers prefer to simulate and validate their applications and protocols before experimenting in real networks. This makes sensor network simulators an important tool in sensor network research. A number of wireless network simulators have been proposed including NS-2 [7], TOSSIM [1], ATEMU [5] and Avrora [6] to name a few.

In this section, we first briefly discuss the designs and limitations of a few widely used wireless sensor network simulators. Then, we make comparisons between SUNSHINE and the existing network simulators.

2.1 Related Work

2.1.1 NS-2

NS-2 [7] is the classical network simulation framework that is used in the context of wired and wireless networks. NS-2 is a discrete event based simulator that simulates networks at packet level. It is widely used in wireless network area to evaluate lower layer communication

algorithms. Even though NS-2 is a useful network simulation framework, it is not suitable for wireless sensor networks for several reasons.

Firstly, NS-2 lacks an appropriate radio module that fits for sensor networks. In addition, NS-2 is focused on evaluating network protocols using general models, such as routing model, mobility model and mac layer model, etc. It fails to model application behaviors which can have a greater impact on sensor's performance and life estimation.

2.1.2 TOSSIM

TOSSIM [1] is a discrete event simulator for wireless sensor networks. Each sensor node platform (e.g. mote) in the networks uses TinyOS as its operating system. TOSSIM is able to simulate a complete sensor network as well as capture the network's behaviors and interactions. Therefore, users are able to analyze TinyOS applications in TOSSIM simulation before testing and verifying the applications over real motes. TOSSIM also provides debugger tools for users to examine their TinyOS codes that can help users debug programs more efficiently.

TOSSIM architecture

Figure 2.1 [1] shows TOSSIM's architecture. TOSSIM consists of an Event Queue, Components Graphs, Radio Model, Communication Services, ADC Event, ADC Model and etc. In the event-based network domain simulator, every sensor node's behavior can be regarded as a functional-level event. These events are kept in the simulator's event queue in sequence according to their timestamps. These events are processed in ascending order of their timestamps. When the simulation time arrives at one event's timestamp, that event is executed by the simulator. The Radio Model, Communication Services, ADC Event and ADC Model are software programs that simulate the real life's corresponding modules.

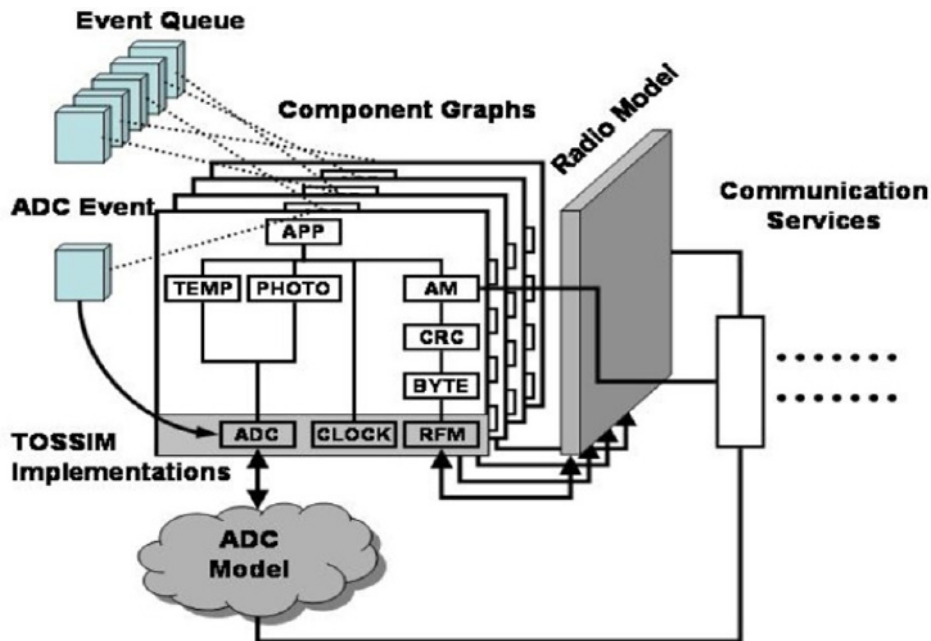


Figure 2.1: TOSSIM architecture

TOSSIM characteristics

As an event based sensor network simulator, TOSSIM has following characteristics [8]:

- Fidelity

TOSSIM aims to provide a high fidelity simulation of TinyOS applications. The simulator is able to simulate packet transmission/reception and packet losses in the simulation. Furthermore, TOSSIM simulates communications at bit level that is more accurate than ns-2 which simulates communications at packet level.

- Imperfections

TOSSIM cannot model interrupts correctly. On a real mote, an interrupt can fire no matter other codes are running or not. However, as an event-driven simulator, an interrupt in TOSSIM simulation cannot fire until current running codes finish executing.

- Time

As a discrete event-driven simulator, TOSSIM only models event arrival time. It does not model event's execution time. This disables users so they cannot estimate and analyze sensor motes applications' real execution time.

- Building

TOSSIM modified the nesC compiler (ncc) to support the TinyOS application to be compiled either for TOSSIM simulation or for running on the real hardware platform.

- Networking

With continuous development of TinyOS and TOSSIM, so far TOSSIM is able to simulate mica, micaz networking stack, including the MAC, encoding, timing and synchronous acknowledgements.

TOSSIM drawbacks

TOSSIM is a widely used simulator in sensornet research community due to its higher scalability and more accurate representation of sensornet than NS-2 [1]. Even though TOSSIM is able to capture network behaviors and interactions, for example packet transmission, reception and packet losses at a high fidelity, it does not provide enough details at cycle-level. Therefore, TOSSIM cannot capture and compare the performance of various hardware designs and the software implementations of sensornet applications.

In addition, TOSSIM simulation results cannot be considered authoritative because TOSSIM does not consider several factors that should be considered in real system. For example, event's execution time and correct hardware interrupt behavior as discussed above.

2.1.3 ATEMU

ATEMU, the first instruction-level simulator for sensor network, is a fine-grained tool written in C computer language. ATEMU is able to emulate the operation of each individual sensor node in the whole sensor network.

As shown in Figure 2.2, ATEMU consists of an AVR Emulator, a graphical debugger tool (XATDB), a configuration specification File and several peripheral devices. AVR Emulator is in charge of executing instructions running on AVR. XATDB allows user to debug application programs on the ATEMU emulator. The configuration specification File specifies the hardware platform. Peripheral devices are linked and communicated with the AVR Emulator.

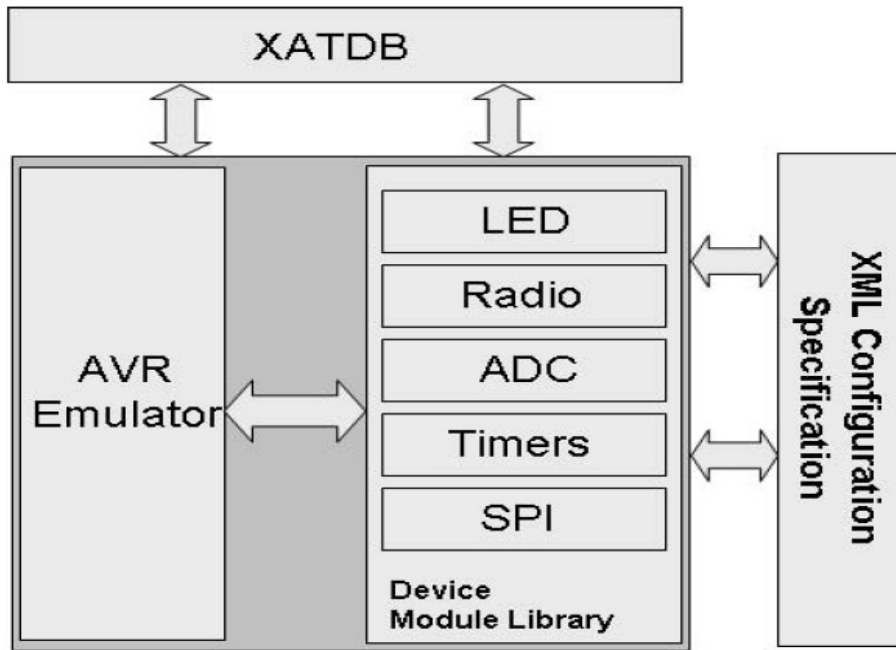


Figure 2.2: ATEMU components architecture

Even though ATEMU is able to simulate a whole sensor network, it executes slowly when simulating large scale sensor networks.

2.1.4 Avrora

Avrora is also an instruction-level sensor network simulator which is written in Java computer language. Avrora simulates a network of motes with cycle accuracy.

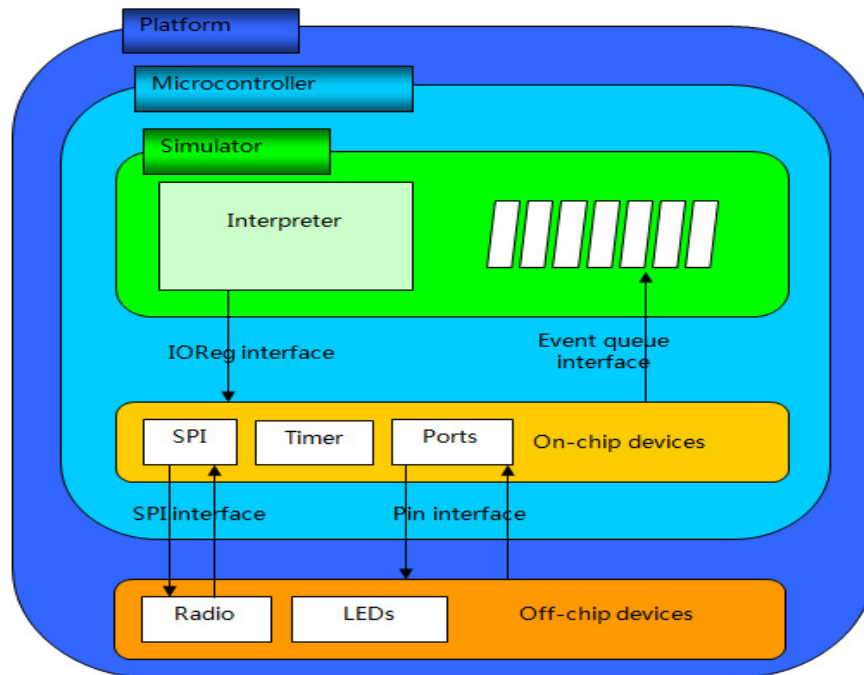


Figure 2.3: Avrora software architecture

As shown in Figure 2.3 [9], Avrora consists of an Interpreter, an Event Queue, several on-chip devices and several off-chip devices. The on-chip devices are communicated with the Interpreter through Input/Output Register's interfaces, while the off-chip devices are controlled through hardware components' pins or through Serial Peripheral Interface Bus (SPI). The Event Queue, which stores time-triggered events, is in charge of interpreting sensor nodes' behaviors.

Avrora uses multi-threading techniques with an efficient synchronization schemes to guarantee different sensor nodes running on different threads can interact with each other based on a correct causal relationship. Avrora achieves better scalability and faster simulation speed than ATEMU.

ATEMU [5] and Avrora [6] are the two existing sensornet simulators that venture out of the event-based simulations in network domain. They provide cycle-accurate software domain simulation to evaluate the fine-grained behaviors of software over AVR controllers of MICA2 sensor boards.

Though ATEMU and Avrora are cycle-level sensornet simulators, they can only simulate Crossbow AVR/MICA2 sensor boards. They cannot accurately capture the impact of alternative hardware designs on the performance of sensornet applications. In other words, they do not support flexibility and extensibility in hardware beyond very simple parameter settings.

2.2 Comparisons of SUNSHINE with Existing Simulators

In this part, I made several comparisons between SUNSHINE and other existing network simulators.

SUNSHINE provides true hardware flexibility where a user can make changes in hardware design of sensor node's platforms and verify his/her sensornet application's feasibility. SUNSHINE is able to simulate different potential hardware architectures. For example, SUNSHINE can simulate a sensor board with an FPGA to handle heavy computational intensive tasks, such as advanced data packets encryption/decryption and data packets compression. This provides a new direction to sensornet design and enables network researchers to evaluate their designs under different hardware platforms. SUNSHINE provides a valuable instrument to both sensornet community and hardware development community.

Further, each existing simulator can only work in one domain. For example, NS-2 and TOSSIM only work in event-based network simulation domain while ATEMU and Avrora

Table 2.1: Comparison between simulators

Aspect \ Name	TOSSIM	Avrora/ATEMU	SUNSHINE
HW Flexibility & Extensibility	No	No	Yes
Hardware behavior	No	No	Yes
User-defined Platform Architecture	No	No	Yes
User-defined Application	Yes	Yes	Yes
User-defined Network Topology	Yes	Yes	Yes
Applications	1	≥ 1	≥ 1
Cycle Accuracy	No	Yes	Yes
Transition between event-based and cycle-accurate simulator	No	No	Yes

can only execute cycle-accurate simulations. While TOSSIM and NS-2 lose their simulation fidelity due to the coarse simulation granularity, the all cycle-accurate simulations of ATEMU and Avrora require long execution time. Different from these existing simulators, SUNSHINE offers its user flexible middle ground between cycle-accurate and event-based simulations. It can combine a variety of nodes that simulated at coarse event-level and nodes that are simulated at fine cycle-level.

Finally, SUNSHINE offers ability to capture hardware behavior of sensor nodes. This unique capability of SUNSHINE can get the finer details of interactions among hardware components at even bit level, which is not explored in Avrora, ATEMU or TOSSIM.

Table 2.1 summarizes the differences between TOSSIM, Avrora, ATEMU and SUNSHINE. As shown in Table 2.1, hardware flexibility is one of the most significant advantages of SUNSHINE. Also, SUNSHINE’s ability of capturing hardware behavior is another improvement for sensornet simulators.

Chapter 3

SYSTEM DESCRIPTION

SUNSHINE is a scalable and accurate cross-domain simulator which seamlessly combines three existing simulators: network domain simulator TOSSIM [1], software domain simulator SimulAVR [2], and hardware domain simulator GEZEL [3]. The details of TOSSIM are introduced in Chapter 2.1.2. In the following, I first briefly introduce SimulAVR and GEZEL simulators. Then, I introduce SUNSHINE's system architecture as well as its simulation process.

3.1 System Components

3.1.1 SimulAVR

SimulAVR [2] is an instruction set simulator that supports software domain simulation for the Atmel AVR family of microcontrollers. The hardware interface part of SimulAVR is written in C++ computer language which can be easily interconnected with GEZEL. SimulAVR provides accurate timing of software execution and can simulate multiple AVR devices in one simulation. However, current SimulAVR does not support sleep mode or wakeup mode.

We have added sleep mode and wakeup mode schemes in order to help emulate energy saving schemes.

3.1.2 GEZEL

GEZEL [3] is a hardware domain simulator that includes a simulation kernel and a hardware description language for custom hardware development. It can simulate various hardware platforms at cycle-level and has been used for hardware-software co-design of crypto-processors [10–12], cryptographic hashing modules [13], high-level estimation of chip power consumption [14], and formal verification of security properties of hardware modules [15], [16]. GEZEL models can be automatically translated into a hardware implementation that enables a user to create his/her own hardware, to determine the functional correctness of the custom hardware within actual system context and to monitor cycle-accurate performance metrics for the design.

GEZEL is the key technology to enable a user to optimize the partition between hardware and software and to optimize the sensor-node architecture. With the support of GEZEL, the cross-domain cosimulator can capture the software and hardware interactions and their individual performance in cycle-level in a networked context. SUNSHINE users can also use GEZEL to simulate different designs of hardware modules and sensor platforms.

3.2 System Design Flow

Figure 3.1 shows the sensornet’s design flow using SUNSHINE. SUNSHINE is composed of a simulation kernel and a user interface to configure the network. In the configuration step, the users first need to configure the network topology and select a subset of nodes (co-simulated nodes) to be simulated by SimulAVR and GEZEL at cycle-level accuracy.

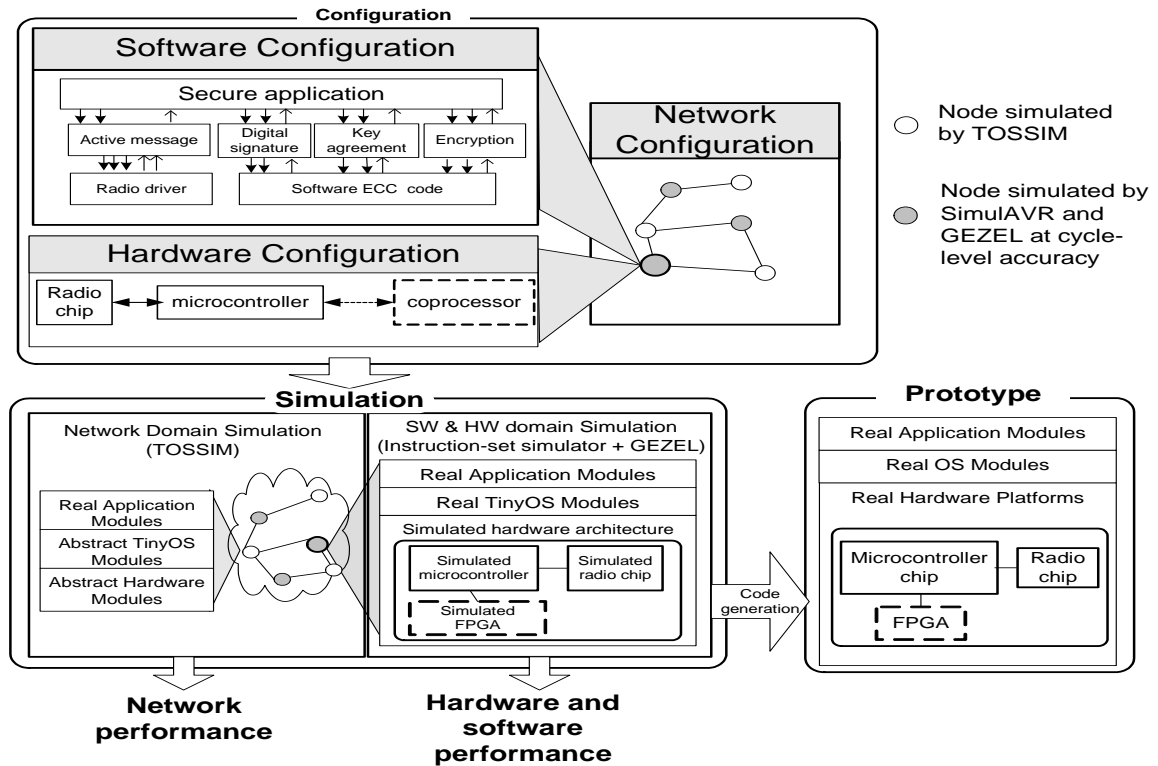


Figure 3.1: Network Design Flow using SUNSHINE

The other remaining nodes are simulated by TOSSIM and only the high-level functional behaviors of these nodes (TOSSIM nodes) are simulated. For the cycle-level co-sim nodes, the users should configure the co-sim nodes' software modules and hardware platforms. After configuration, the users can run simulation. SUNSHINE emulates the AVR microcontroller, and the TinyOS application executing on top of it, with cycle-level timing accuracy.

After getting satisfactory simulation results, the users can directly load the binaries executed on cycle-level co-sim nodes on to real sensor boards and run the application on real testbeds.

3.3 System Architecture

SUNSHINE integrates TOSSIM, SimulAVR and GEZEL to simulate sensor networks in network, software, and hardware domains. A user of SUNSHINE can select a subset of sensor nodes to be simulated in hardware and software domains. These nodes are called cycle-level hardware-

software co-simulated (co-sim) nodes and their cycle-level behaviors are accurately captured by the SimulAVR and GEZEL components in SUNSHINE. The other remaining sensor nodes are simulated in network domain by TOSSIM and only the high-level functional behaviors of these nodes are simulated. These nodes are named TOSSIM nodes for short.

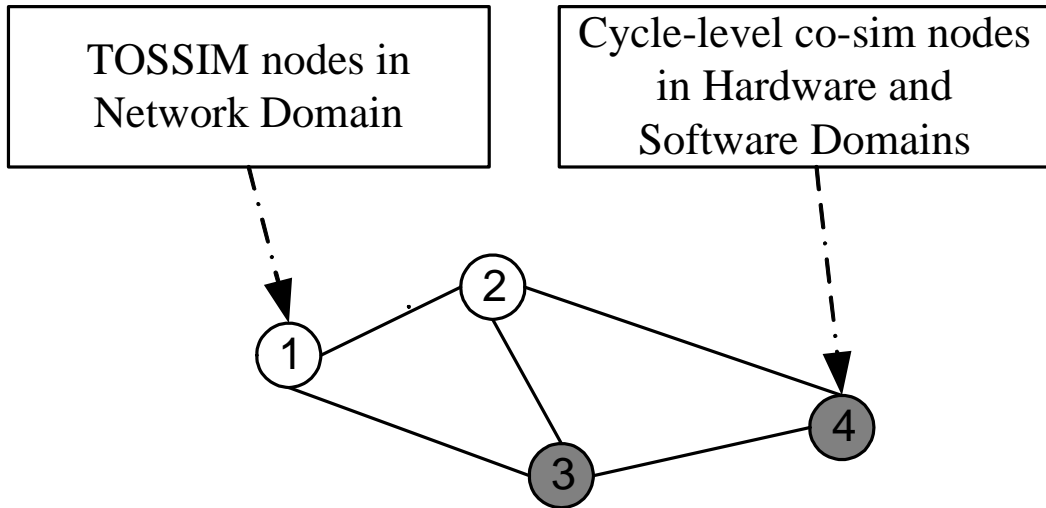


Figure 3.2: Simulation domains

SUNSHINE is able to run multiple cycle-level co-sim nodes with TOSSIM nodes in one simulation. Figure 3.2 illustrates the basic idea of SUNSHINE. Node 1 and 2 are TOSSIM nodes which are simulated in network domain, while node 3 and 4 are co-sim nodes which are simulated in cycle-level software and hardware domains. When running simulation, these TOSSIM nodes and co-sim nodes interact with each other according to the network configuration and sensornet applications. Cycle-level co-sim nodes can show details of sensor nodes' behaviors, such as hardware behavior, but simulate at a relatively slow speed. TOSSIM nodes do not simulate much details of the sensor nodes but simulate at a much faster speed. The mix of cycle-level simulation with event-based simulation ensures that SUNSHINE can leverage the fidelity of cycle-accurate simulation, while still benefitting from the scalability of event-driven simulation.

The simulation process in SUNSHINE is illustrated by Figure 3.3. First, for cycle-level

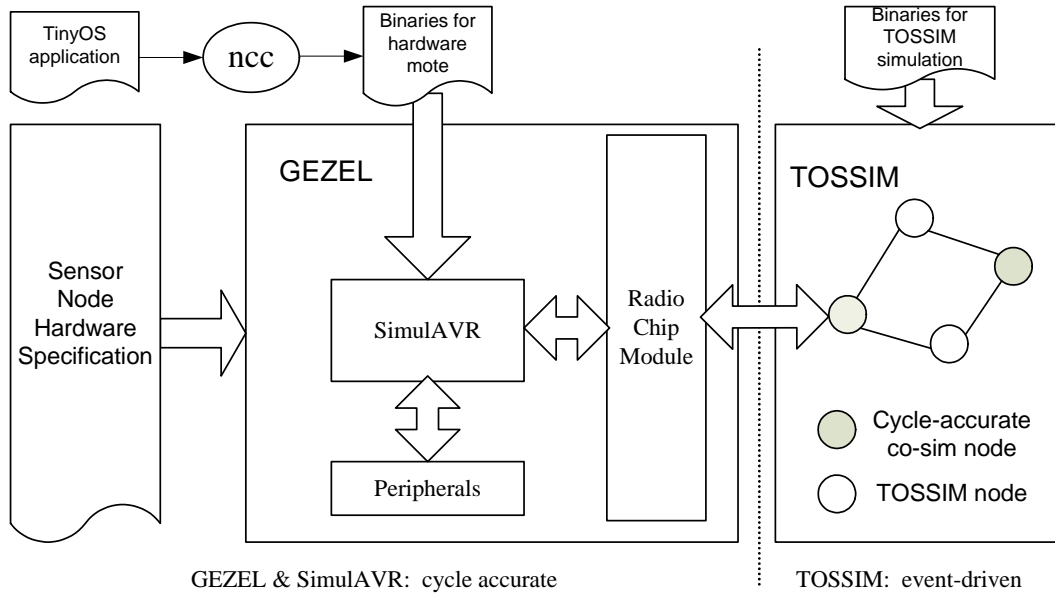


Figure 3.3: Software architecture

hardware-software co-sim sensor nodes (e.g. Node 3 and 4 in Figure 3.2), executable binaries for real sensor node’s platforms are compiled from TinyOS applications using NesC compiler (ncc). Since SUNSHINE can simulate hardware platform that is compatible with the series of MICA boards, TinyOS executable binaries can be executed directly in SUNSHINE simulation. These binaries are interpreted by SimulAVR, component of SUNSHINE. SimulAVR captures the behavior of these binaries under the constraints of different AVRs and memory space. At the same time, GEZEL interprets the sensor node’s hardware specification, and simulates AVR’s interfaces and interactions with other hardware modules and peripherals. One such peripheral that GEZEL simulates is the radio chip module on a sensor node. GEZEL’s simulation of this radio chip module also includes an interface to the TOSSIM simulator, which creates the wireless communication channels. Through these wireless communication channels, a cycle-level co-sim node interacts with other sensor nodes, which are simulated either as cycle-level co-sim nodes by GEZEL and SimulAVR, or are simulated as functional-level nodes by TOSSIM (e.g. Node 1 and 2 in Figure 3.2). Sensornet applications that need to run on TOSSIM nodes are compiled through ncc. TOSSIM nodes

interact with cycle-level co-sim nodes using the timing synchronization and the cross-domain data exchange techniques to maintain correct causal relationship. These two methods will be introduced in the following chapters.

Chapter 4

CC2420 IPBLOCK

GEZEL is an extensible simulator based on building GEZEL library blocks. The library block named `ipblock` allows adding new cosimulation interface between GEZEL and other simulators and building new models' behaviors. The behavior of the library block is written in C++ computer language and compiled directly to the GEZEL kernel. In SUNSHINE, we built an `ipblock` named `CC2420ipblock` which simulates the behavior of the Radio Chip CC2420. The `CC2420ipblock` acts as a bridge that links hardware-software domain simulator GEZEL + SimulAVR with event-based simulator TOSSIM.

In this chapter, the functions of `CC2420ipblock` are discussed.

Figure 4.1 shows an interconnection between GEZEL + SimulAVR and TOSSIM. `CC2420ipblock` consists of SPI communication interface and Finite State Machine (FSM) of the radio chip module. The SPI communication interface in `CC2420ipblock` is used to exchange data with GEZEL+SimulAVR, while FSM is designed to simulate the radio chip's behavior according to the CC2420 datasheet [17].

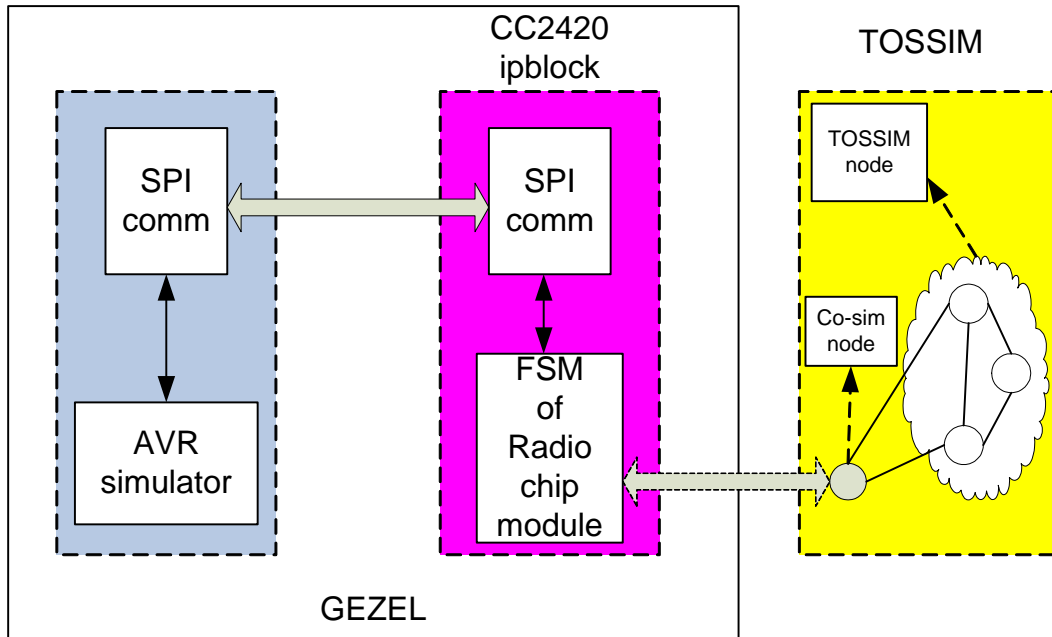


Figure 4.1: Integration of different domain simulators

4.1 SPI Communication

SPI is a synchronous serial standard that data exchanges between two devices at bit level every clock cycle. Two devices are distinguished as master and slave device separately. Master device sets the slave device's clock cycle and the selective pin (CS_n). If the selective pin is active, the SPI is set up between the master and the slave.

Figure 4.2 illustrates the connection between microcontroller ATmega128 with radio chip CC2420. Four pins are in charge of SPI communication. In CC2420, the selective pin CS_n is active low. $SCLK$ is the CC2420's clock period set by ATmega128. As SPI operates in full duplex mode, two pins are responsible for exchanging data. AVR transmits one byte length data frame to CC2420 via the slave output pin SO while receives the one byte length data frame from CC2420 via the slave input pin SI .

Figure 4.3 [17] shows the SPI timing diagram between the AVR and the radio chip CC2420.

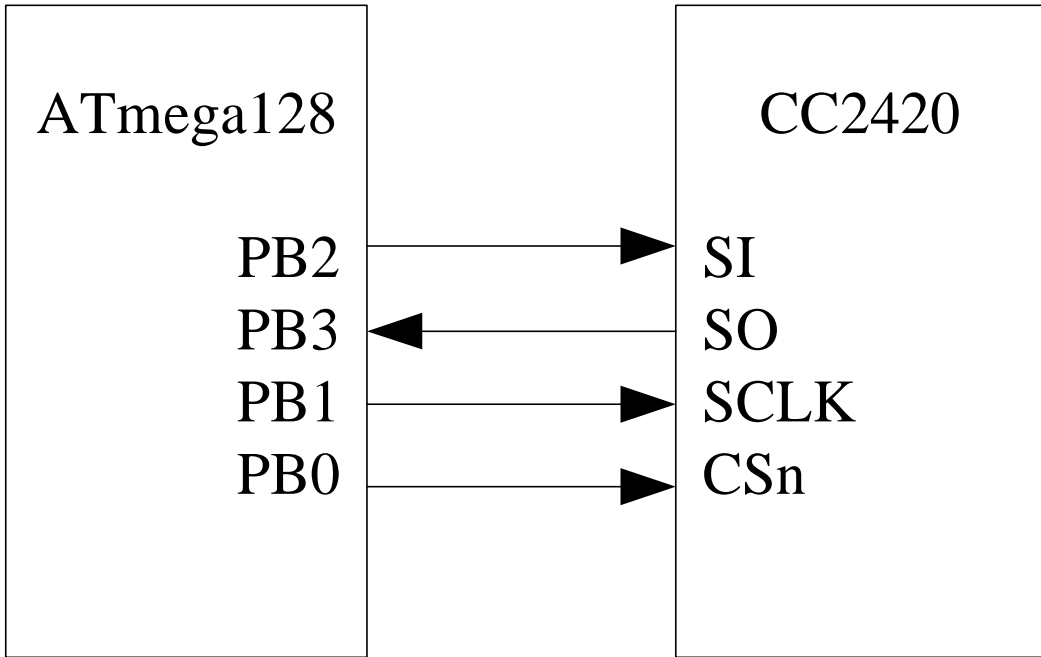


Figure 4.2: SPI connection

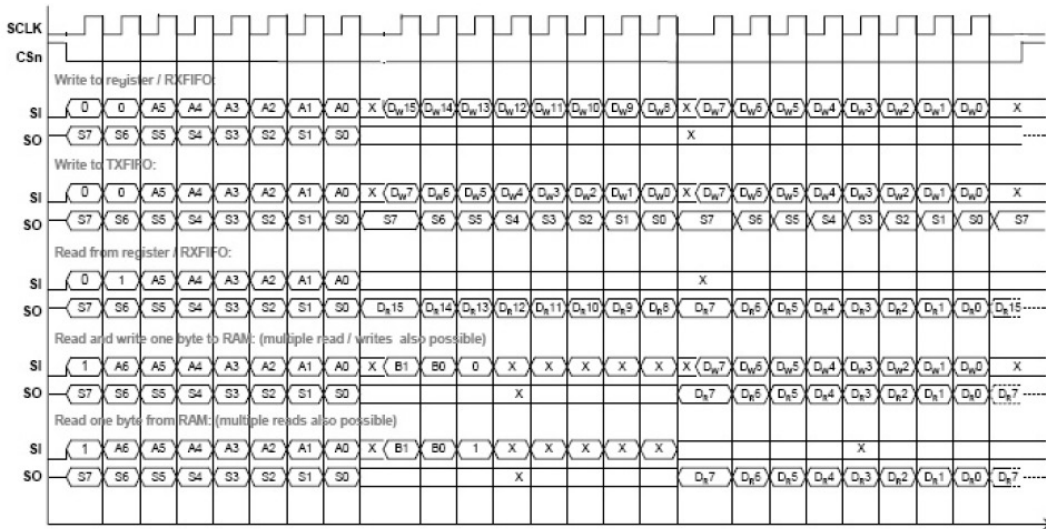


Figure 4.3: SPI timing diagram

The AVR can use SPI interface to write or read data to or from radio chip’s registers and RAM.

The most significant bit (bit 7) of the first byte’s data frame is RAM/Register bit (0 for

register access, 1 for RAM access). If it is register access, the following one should be the R/W bit (0 for write, 1 for read). Then, the rest of the bits belonging to the first byte's data frame are the address-bits (A5:0). (D15:8) and (D7:0) represent data frames exchanged between AVR and CC2420's registers.

If it is RAM access, bit 7 of the first byte data frame is set to 1, and the following seven bits are the address-bits (A6:0). (B1:0) of the second byte are set to access three memory banks: TXFIFO (bank 0), RXFIFO (bank 1) and security (bank 2). Status byte (D7:0) represents data frames exchanged between AVR and CC2420's RAM.

Status byte (S7:0) is used to inform AVR CC2420's status, for example, whether Radio Frequency (RF) transmission is active, whether an FIFO (First Input First Output RAM) underflow has occurred during transmission, etc.

AVR is able to configure registers, set command strobes, read status and access RAM of CC2420 via SPI. In summary, AVR interacts with CC2420 via SPI.

4.2 Finite State Machine of CC2420ipblock

As shown in Figure 4.1, CC2420ipblock contains an FSM of the radio chip module. The FSM describes the radio chip's behavior according to Figure 4.5 [17].

The change of states is done either by using command strobes (e.g. SXOSCOFF command strobe) or by internal events (e.g. Transmission completed as shown in Figure 4.5).

The command strobes are received from AVR via SPI, while the internal events are received from TOSSIM. If the radio chip module receives SXOSCOFF command strobe from AVR, it will turn off the crystal oscillator and RF, and jump to the Power Down state.

For example, if TOSSIM receives a transmission event: after transmitting one data packet,

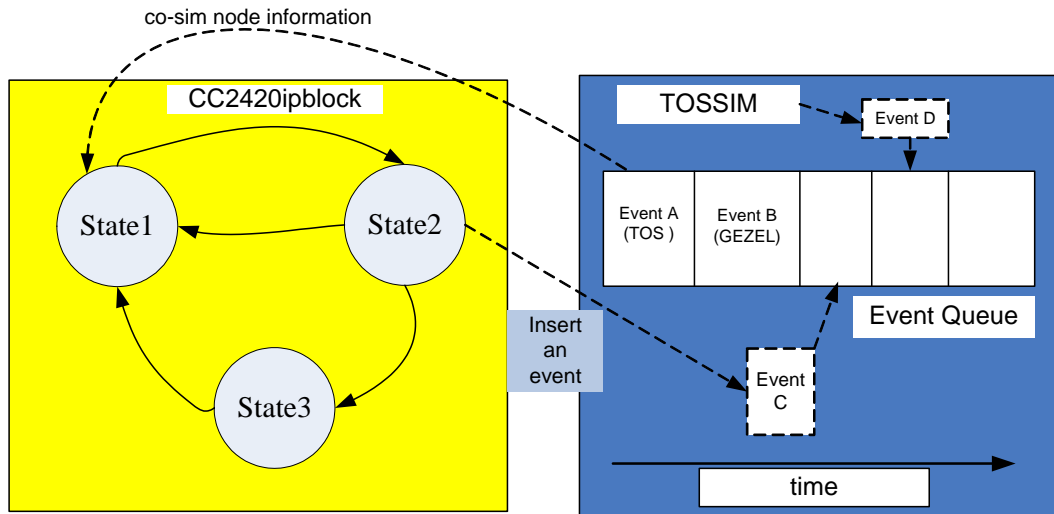


Figure 4.4: Interaction between CC2420ipblock and TOSSIM

TOSSIM will insert an transmission completed event. As soon as detecting this event, the radio chip’s state will switch from “TX_FRAME” to “RX_CALIBRATE”.

Figure 4.4 illustrates how CC2420ipblock interacts with TOSSIM. At any states of the FSM, the radio chip module can generate a TOSSIM event and insert it to the event queue. For example, at any reception states, if the radio chip module receives a STXON command strobe from AVR, it will generate a “send” event into the event queue. Also, TOSSIM itself is able to insert an event. The event inserted by TOSSIM can be either a co-sim node’s event or a TOSSIM node’s event. If the co-sim node’s event inserted by TOSSIM is fired, the radio chip module will detect the event, change its radio state and execute corresponding behaviors.

In the following chapter, Section 5.2 describes the details of cross-domain data exchange technique.

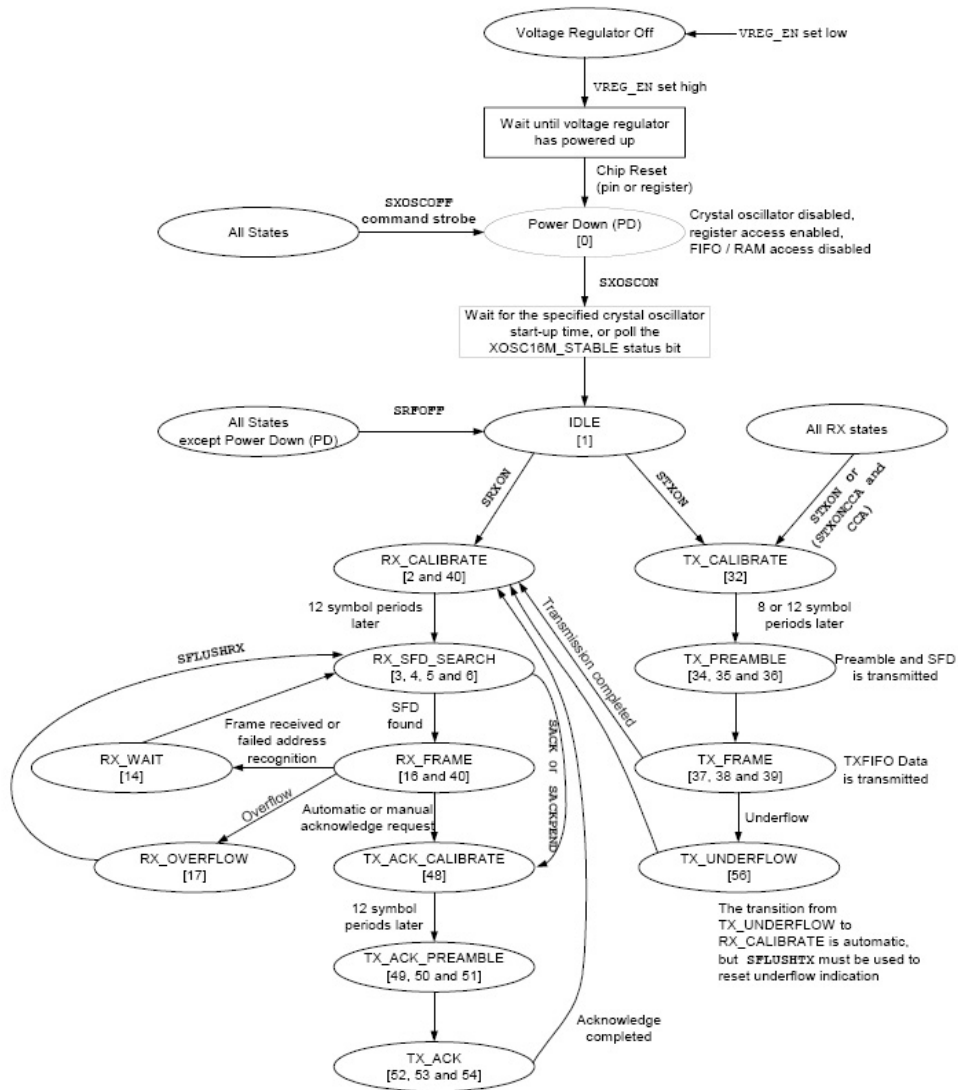


Figure 4.5: CC2420 Radio Control States

Chapter 5

CROSS-DOMAIN INTERFACE

In this section, we will discuss how we interface the three components of SUNSHINE, each working in a different domain of simulation.

5.1 Timing Synchronization

SUNSHINE integrates network domain simulator TOSSIM, cycle-level hardware domain simulator GEZEL and software domain simulator SimulAVR for the purpose of scalability. However, simulations in these three domains run at different step sizes. Figure 5.1 illustrates mismatches in simulation time between event-driven simulation and cycle-level simulation. As shown in Figure 5.1, SimulAVR and GEZEL run at cycle-level steps, where each simulation step captures the behaviors of an AVR or a hardware component at one clock cycle. Therefore, the simulation time is gradually increasing. However, in TOSSIM, which is a discrete event simulator, each simulation step captures the occurrence and handling of a network event. As the time durations between events are irregular, the simulation time in TOSSIM also increases at irregular steps. This difference in simulation time may cause potential violations in causal relationship among different sensor nodes in simulation.

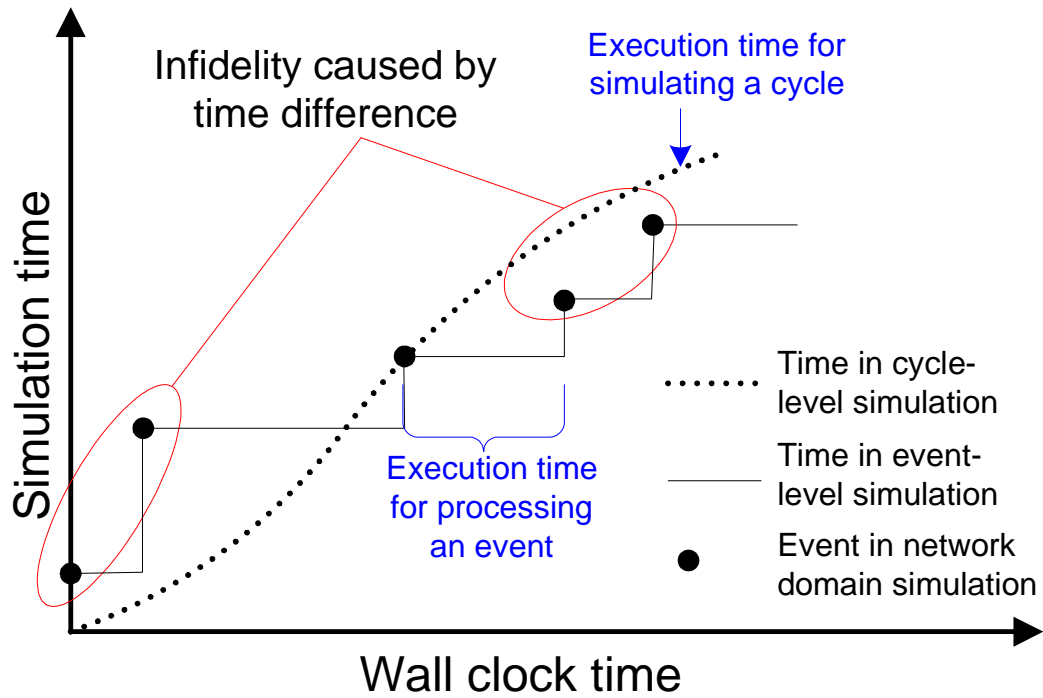


Figure 5.1: Simulation time in different domains

To solve this issue, SUNSHINE includes an efficient time synchronization algorithm to ensure evaluation fidelity and maintain the correct causal relationship between simulated modules and nodes. The graphic illustration of this algorithm is depicted in Figure 5.2.

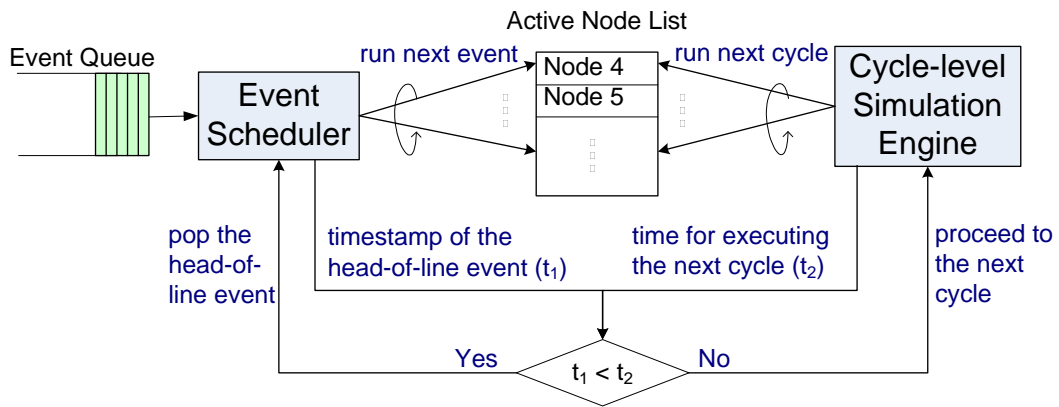


Figure 5.2: Synchronization Algorithm

In this design, TOSSIM uses the Event Scheduler to handle all discrete network events

while GEZEL uses the Cycle-level Simulation Engine to control the simulation of hardware modules and the AVR every clock cycle. All network events are in the Event Queue and are sorted according to their timestamps that record their occurrence time.

The Event Scheduler processes the head-of-line (HOL) event in the Event Queue only when the Cycle-level Simulation Engine has progressed to the event's time stamp. By selecting either an event or a cycle-level simulation to be simulated next, SUNSHINE will maintain the correct causality between different simulation schemes in the whole network.

The design in Figure 5.2 also provides synchronization supports for cycle-level sensor nodes in sleep mode by maintaining an Active Node List. This list holds the active nodes that need to be simulated with cycle-level accuracy. The Event Scheduler adds or removes nodes from the list upon node wakeup or node sleep events. At each cycle-level simulation step, the Cycle-level Simulation Engine only processes a clock cycle for the nodes of the Active Node List. As a result, a node's sleep or wakeup state does not need to be checked every clock cycle. Given the fact that in sensor networks, a sensor node spends most of its time in sleep mode, this design will greatly accelerate SUNSHINE's simulations.

Based on the synchronization algorithm, the desired behavior of a synchronized simulation can be achieved. As shown in Figure 5.3, the simulation time of the cycle-level simulation closely follows the simulation time of the event-based simulation. Events in the network domain are processed with the correct causal order compared to the cycle-level simulation, and the SUNSHINE simulator correctly interleaves cycle-level processing with event-driven processing.

5.2 Cross-Domain Data Exchange

Since SUNSHINE integrates simulation engines working in three different domains, it is necessary to implement interfaces for cross-domain data exchange so that software running

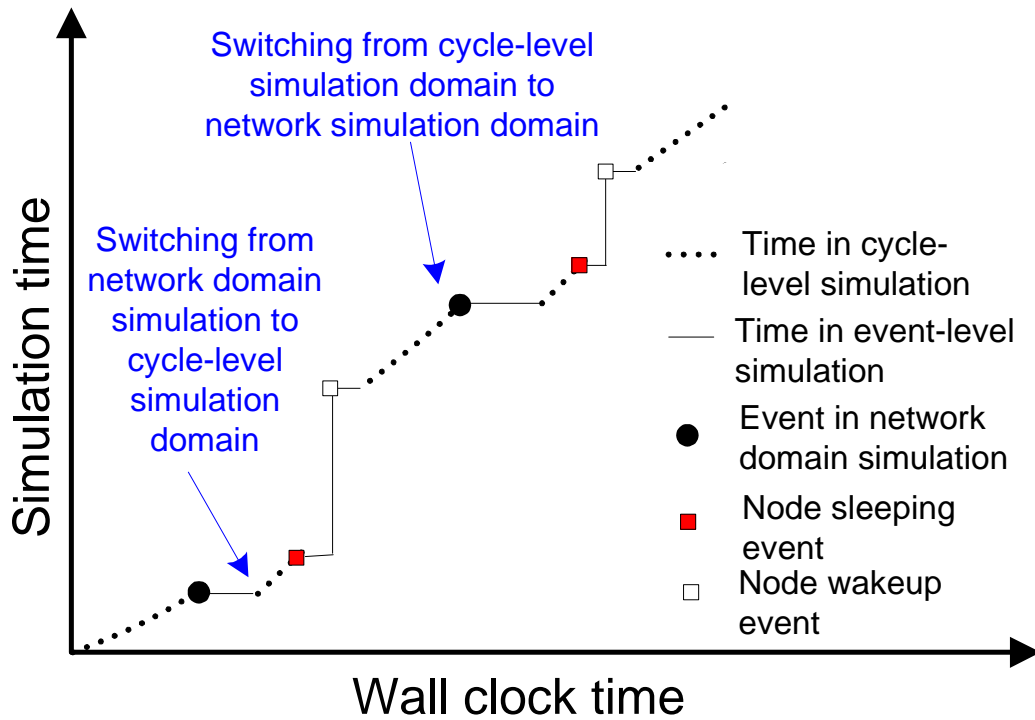


Figure 5.3: The synchronized simulation time in SUNSHINE

on simulated AVR can not only read or write data from or to simulated hardware modules but exchange messages with other simulated nodes through wireless channels.

The interface between GEZEL simulated models in hardware domain and SimulAVR simulated modules in software domain is realized by GEZEL's existing cosimulation interface mechanism. This mechanism enables SUNSHINE to create abstracted and cycle-true models of real hardware-software interfaces between AVR, simulated by SimulAVR, and peripherals, simulated by GEZEL. The details of GEZEL's cosimulation interface can be found in GEZEL's documentation [18].

Sensor nodes in network domain simulated by TOSSIM also need to exchange data with nodes simulated in software-hardware domain by GEZEL + SimulAVR through wireless channels. However, network domain simulation and hardware-software domain simulation have different simulation abstractions. For TOSSIM, it abstracts the functions and interac-

tions among network components as high-level abstract events. For example, as shown in figure 5.4, the reception of an entire packet may be regarded as a single event. However, in hardware and software domain simulation by GEZEL and SimulAVR, the functions and interactions among components are simulated at much finer granularity. For example, in figure 5.4, to simulate the reception of a data packet, the bits that are received and read from the radio chip at every clock cycle have to be captured.

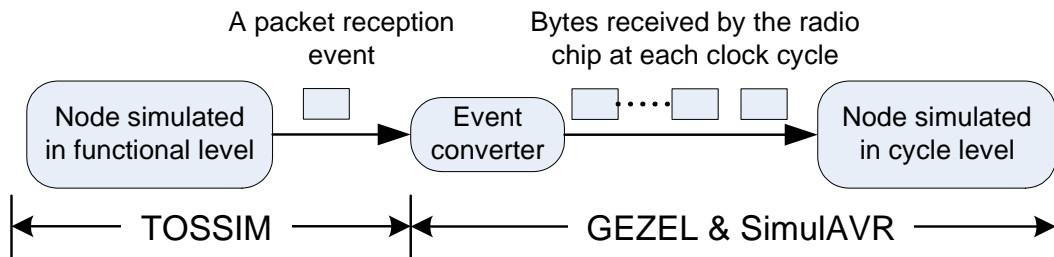


Figure 5.4: Converting a functional-level event to cycle-level events

To bridge this gap between abstraction levels, SUNSHINE modifies TOSSIM’s event queue as well as creates an event converter that consists of a packet converter and a time converter. The event in TOSSIM’s event queue can be either TOSSIM’s original event or the event that is used for cycle-level hardware-software co-sim node’s event generated by GEZEL and simulated AVR. The packet converter supports packet’s format conversion between TOSSIM and GEZEL + SimulAVR. The time converter converts TOSSIM’s current time to detailed simulation time.

Figure 5.5 illustrates the event conversion process. If a cycle-level co-sim node transmits a data packet, it should follow several steps in simulation. The Simulated AVR first sends the data packet to the radio chip module via Serial Peripheral Interface Bus (SPI) with cycle accuracy. The radio chip module stores data packets in TXFIFO. After receiving a transmission command strobe from the simulated AVR, the radio chip module inserts an event to TOSSIM’s event queue. When the event is fired, TOSSIM gets the data packet

from the TXFIFO, uses the packet converter to change the original data packet's format to TOSSIM packet's format and then sends the packet to the radio channel.

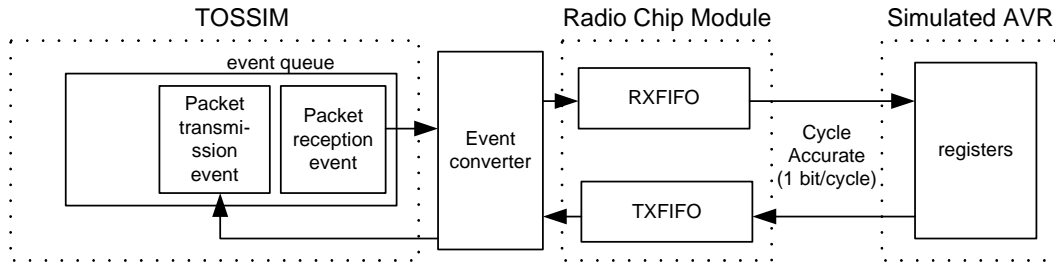


Figure 5.5: Event conversion process

If an event that indicates a co-sim node to receive a packet from the radio channel is fired from the Event Queue, the packet converter modifies the data packet to the right format and puts the packet into the RXFIFO of the radio chip module. In addition, time converter converts TOSSIM's current event time to several detailed simulated time, such as the time that the start of frame delimiter (SFD) field of received packet has been completely received by the cycle-level co-sim node and the time that the length field of received packet has been received by the co-sim node, etc. Some of these timing information is used for generating interrupts to the simulated AVR in the co-sim node. After receiving interrupt signals from the radio chip module, the simulated AVR begins reading the bits of the data packet from the radio chip module through SPI over multiple cycles.

Using the event converter, SUNSHINE is able to convert coarse packet communication events to the cycle-level packet reception and transmission behaviors and vice versa. Based on this mechanism, SUNSHINE satisfies both simulated AVR's cycle-level and TOSSIM's event-level accurate requirement.

Chapter 6

HARDWARE SIMULATION SUPPORT

As SUNSHINE is able to simulate hardware behavior, in this chapter, we discuss SUNSHINE's support for hardware simulation.

6.1 Hardware Specification Scheme

One of the primary contributions of SUNSHINE is to support hardware flexibility and extensibility. SUNSHINE describes sensor node's platform hardware architecture at simulator configuration level using GEZEL-based hardware specification files. Users of SUNSHINE can make various modifications to a sensor node's platform architecture, such as using different microcontrollers, adopting multiple microcontrollers, adding hardware coprocessors, connecting with new peripherals and performing other customizations on the platform. The syntax of a valid hardware specification file based on GEZEL is relatively simple. Users can write their own specification files according to GEZEL semantics[19].

To demonstrate this point, Figure 6.1 shows specific details of how hardware architecture of

a MICAz platform is described in SUNSHINE. We listed a snippet of the hardware specification file in Figure 6.1. The file is divided into three pieces, each of which is dedicated to a relevant hardware part. From the code snippet, we can see that users can pick hardware components using “iptype” statements to configure a sensor node’s hardware platform. In this specific example, microcontroller Atmega128 and radio chip CC2420 are chosen to form the MICAz hardware platform. The components’ corresponding ports are interconnected through virtual wires that are also described in the specification file. For example, “Atm128sinkpin” wires the input pin B3 of the AVR core to the output pin MISO of the CC2420 chip, while “Atm128sourcepin” wires the output pin B0 of the AVR core to the SS input pin of the CC2420 chip. While our example shows MICAz platform, a user can also pick other components to form a different hardware platform in their sensornet simulation. For example, one can use ARM instead of Atmega128 by modifying the hardware specification file. Based on this mechanism, SUNSHINE can easily combine different hardware components to form different hardware platforms for sensornet simulation. In other words, SUNSHINE supports running a network simulation over flexible hardware platforms that are created based on either commercial off-the-shelf sensor boards or the user’s own customized platform designs.

The example in Figure 6.1 also shows how SUNSHINE enables different co-sim nodes to run different software applications through the use of “ipparm” statements. The “ipparm” can also be used to set parameters for hardware components. In Figure 6.1, the statement *ipparm*“*exec = app*” means the simulated AVR will execute the executable binary named “app”, which is compiled from a software application using *ncc*. Users can also configure the simulated AVR to execute other binaries in a co-sim node through *ipparm* statements. (For more details, please see Appendix A.) By configuring different co-sim nodes to execute different software binaries, SUNSHINE can simulate a sensornet that has multiple different applications. This is a significant improvement over TOSSIM, which can only run one application in a whole network.

Essentially, SUNSHINE’s simulation configuration steps are as follows. First, the executable binaries of applications are compiled from their source codes. Then, as shown in Figure 6.1, a Hardware Specification file is created to describe how hardware components form the hardware platforms in the sensornet. The Hardware Specification file also links the generated executable binaries to the corresponding hardware platforms. After the configuration, SUNSHINE simulation can start.

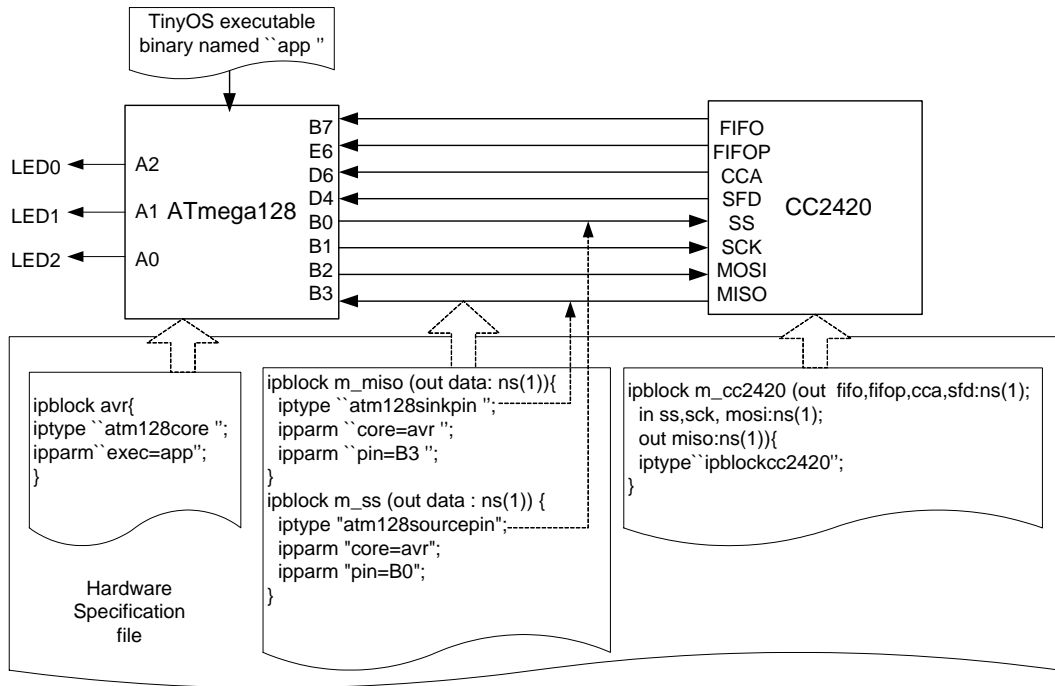


Figure 6.1: Hardware specification for a single node (Multiple nodes can be captured by instantiating multiple AVR and multiple radio chip modules.)

From the above description, one can easily see that SUNSHINE can be used to simulate various hardware platform designs to find the most suitable hardware module for a given network environment and a given set of application requirements. Therefore, SUNSHINE is an efficient tool to help hardware designers develop better sensor node platforms. In addition, researchers in the field of software can also use SUNSHINE to easily configure novel hardware architectures and then evaluate their sensornet applications and protocols over these customized architectures. Because SUNSHINE can change hardware components easily

at simulator configuration level, even software researchers with little hardware knowledge can configure sensornet hardware platforms.

6.2 Hardware Behavior

Unlike other sensornet simulators, SUNSHINE can accurately capture hardware’s behavior. Users can determine whether the microcontroller is in sleep mode or active mode as well as identify the radio chip’s current state. In addition, through simulating GEZEL code, which is a hardware description language, SUNSHINE can display cycle-level behavior of hardware components when applications are running on co-sim nodes. This would help hardware designers know how hardware module behaves in sensornet applications.



Figure 6.2: Traces for TinyOS Reception application

For example, users can track hardware pins’ activities when running a sensornet application in SUNSHINE by doing the following. The signal tracing mechanism of SUNSHINE records stimuli files when the simulation is set in debug mode. These stimuli files, named Value-Change Dump (VCD) files, can be read by digital waveform viewing tools, such as GTKWave, to produce graphic illustrations of hardware pin values. As an example, Figure 6.2 is obtained by running an application named “Reception” on a sensor node in SUNSHINE. In this

simulation, a TinyOS application named TxThroughput runs on one co-sim node and the application named Reception runs on another co-sim node. In TxThroughput, the sensor node keeps sending packets to the radio channel using the largest message payload size. The Reception application listens to the channel and receives packets from the channel. Figure 6.2 shows how the AVR microcontroller interacts with the CC2420 radio chip in the Reception application. As shown, very detailed activities of hardware pins are captured by SUNSHINE.

Chapter 7

EXPERIMENTS

I performed the experiments on a Dell laptop that has Intel (R) Core (TM) 2 Duo CPU T5750 @ 2.00GHz, 3G RAM and runs Linux 2.6.32-23-generic. SUNSHINE integrates TinyOS version 2.1.1, an improved version of SimulAVR and GEZEL version 2.5. I used the latest version of the simulators available at the time of performing the experiments. The hardware platform configured in these simulations is MICAz.

7.1 Basic 2-node Sensor Network Simulation Experiments

In the experiment in Section 6.2, both the Reception and TxThroughput applications are executed in co-sim nodes. In this experiment, I demonstrate that SUNSHINE can simulate a sensornet with a mixture of co-sim nodes and TOSSIM nodes. In the experiment, I set up a sensornet with two sensor nodes. I performed each of the configuration shown in Table 7.1. In the first two cases, I used one cycle-level co-sim node and one TOSSIM node. In the third case, I used two cycle-level co-sim nodes to run the simulation. The first case's co-sim node's

hardware specification file and the application’s simulation script are listed in Appendix A and Appendix B respectively.

By analyzing the traces generated in the experiment, I find that in case 2 and 3, the behaviors of the hardware pins on the Reception side are the same (i.e. they all look like Fig. 6.2) The traces of the hardware pins on the TxThroughput side are also the same for case 1 and case 3. I do not include all these trace figures as they are fairly similar.

Essentially, this experiment shows that while SUNSHINE can mix co-sim nodes with TOSSIM nodes, such mixture will not affect SUNSHINE’s capability in analyzing hardware pin’s behaviors in cycle-accurate level.

Table 7.1: Simulation cases

Cases	TxThroughput Application	Reception Application
Case 1	Cycle-level co-sim node	TOSSIM node
Case 2	TOSSIM node	Cycle-level co-sim node
Case 3	Cycle-level co-sim node	Cycle-level co-sim node

7.2 Tradeoff between Simulation Accuracy and Wall Clock Time

In the following experiments, I analyzed the wall clock time as well as the simulated time. The wall clock time is the time required by SUNSHINE to complete a simulation. The simulated time is SUNSHINE’s prediction of the execution time of a sensornet application based on the simulation of the sensornet.

To analyze the characteristics of SUNSHINE, I simulated a packet relaying application. I used a ring network topology consisting of 8 nodes, as shown in Figure 7.1 (a). The first node sends a packet to the second. As soon as the second node receives the packet, it forwards the same packet to the third node, and so on. Finally, the eighth node sends the packet

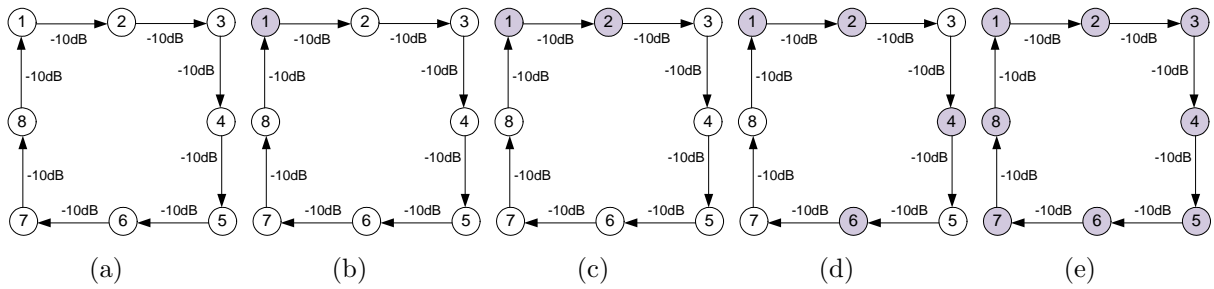


Figure 7.1: Simulation configurations. Co-sim nodes are marked with shades.

back to the first one, signalling the completion of a round trip. The channel attenuation gain between any two successive nodes is set to $-10dB$.

I performed this experiment under two different scenarios. Each scenario consists of 5 stages. In the first stage, I used TOSSIM to simulate the whole sensor network as shown in Figure 7.1 (a). In the subsequent stages, I gradually increased the number of co-sim nodes in the network as shown in Figure 7.1 (b) through Figure 7.1 (e). For each stage, the simulated time and the wall clock time were recorded in order to compare TOSSIM’s simulation with SUNSHINE’s simulation performance.

In the first scenario, called “simple relaying”, the application just performs the basic packet forwarding. In the second scenario, called “advanced relaying”, I modified the basic packet forward application by letting the sensor node perform a computation intensive task before it relays the packet to the next node. In this experiment, I let each sensor node execute 10000 loops before it relays the packet to the next node. In a real application, this computation-intensive task can be packet encryption/decryption or compression algorithms.

The experiment results are shown in Figure 7.2 and Figure 7.3. Figure 7.2 depicts the simulated time for a packet to make a round trip in the simulation. It shows that as the number of co-sim nodes increase, the simulated time also increases. This is because co-sim nodes can capture the execution time of applications over AVR, while TOSSIM nodes omit this and only count the communication delay. Hence, as the number of co-sim nodes

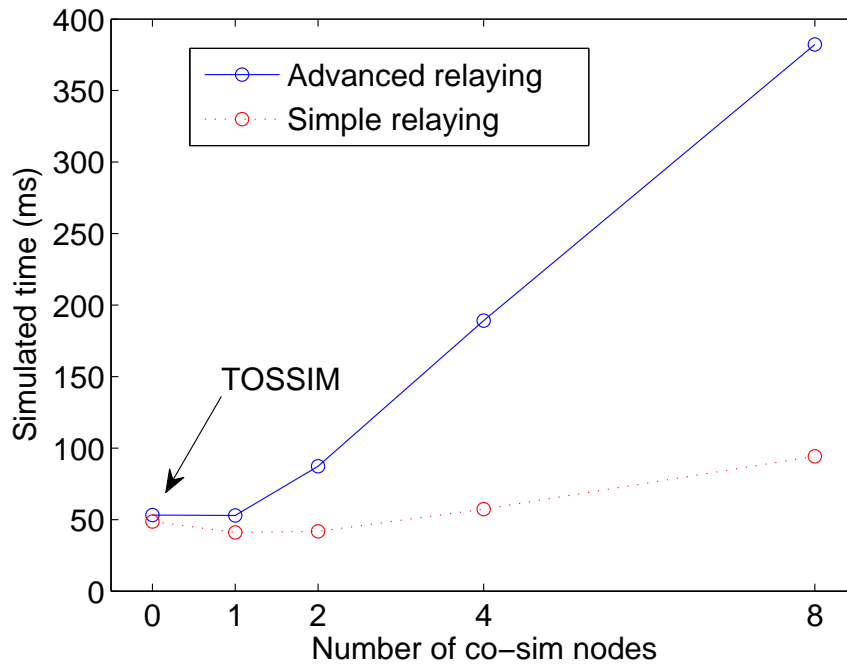


Figure 7.2: Simulated time in terms of number of co-sim nodes

increases, more execution time will be captured and hence the simulated time increases. This increase in simulated time actually demonstrates the benefit of having cycle-accurate simulation in SUNSHINE.

Figure 7.3 shows the wall clock time for SUNSHINE to simulate a packet making a round trip in the ring topology. It can be seen that there is a tradeoff between simulation accuracy and wall clock time. The wall clock time will raise with the increase of the number of co-sim nodes in both simple relaying and advanced relaying scenarios. This is the cost that a user needs to pay if he/she wants to capture more detailed behaviors of sensor nodes by simulating more co-sim nodes. Users can choose any combination of co-sim nodes and TOSSIM nodes based on their needs. SUNSHINE provides a good flexibility on simulating sensornet applications with a different number of co-sim nodes and TOSSIM nodes. In addition, the experiment shows that multiple cycle-level sensor nodes and event-based sensor nodes are able to seamlessly

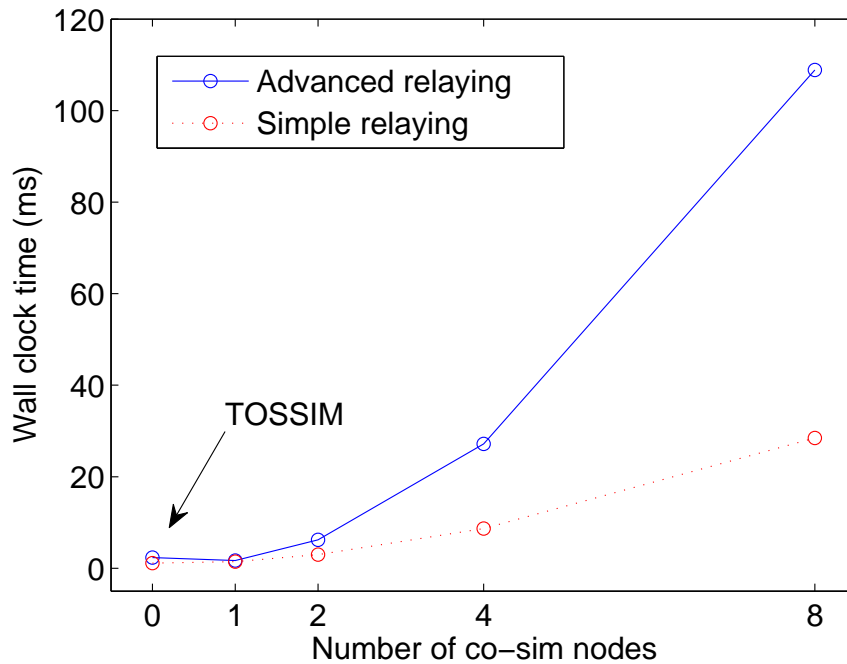


Figure 7.3: Wall clock time in terms of number of co-sim nodes

interact with each other while maintaining the correct causal relationship in SUNSHINE simulation.

7.3 Effect of Computation Load on Sensor Nodes

In this experiment, I used the same simulation setting, shown in Figure 7.1 (e), and load all nodes with the advanced relaying applications. I varied the level of computation intensity of the advanced relaying application in terms of the number of loops used in the intensive computation.

As we can see from Figure 7.4 and Figure 7.5, with the increase of computation intensity, SUNSHINE's simulated time and wall clock time increase correspondingly. While in TOSSIM, the simulated and wall clock time are independent of computation load on sensor nodes.

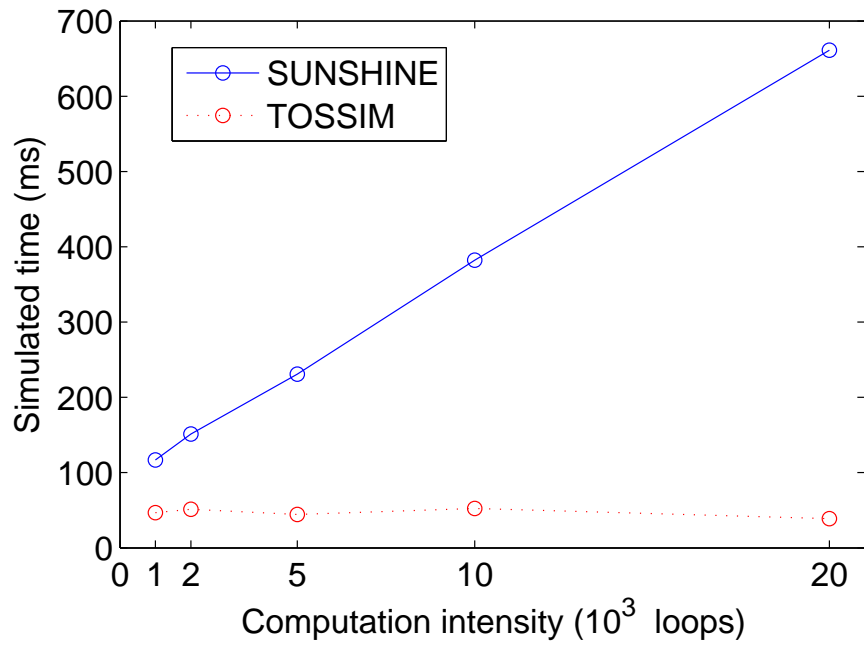


Figure 7.4: Simulated time in terms of computation intensity

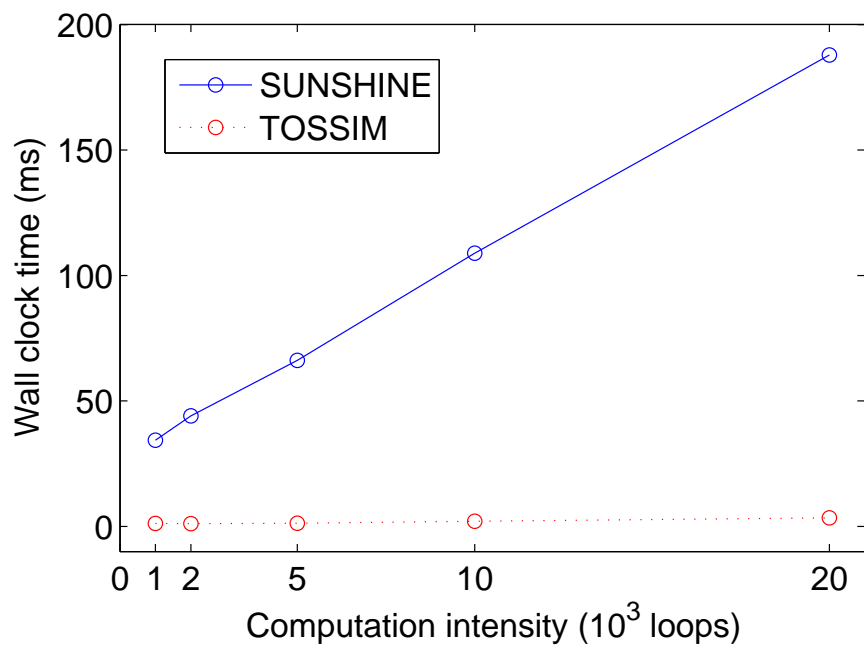


Figure 7.5: Wall clock time in terms of computation intensity

From these figures I can see that although TOSSIM can run faster than SUNSHINE as shown in Figure 7.5, the simulated time in TOSSIM does not change while the computation intensity of applications increases (See Figure 7.4). This means that TOSSIM's fast simulation speed is achieved at the cost of losing the ability to capture the execution time of applications on the AVR microcontroller processor. This limits the applicability of TOSSIM to time-sensitive protocols. For example, some security protocols, such as distance-bounding protocol [20], require precise time-out behavior to thwart physical man-in-the-middle attacks. SUNSHINE is able to test and verify such protocols since it correctly captures the impact of computation intensity on sensornet performance (See Figure 7.4).

Chapter 8

CONCLUSION AND FUTURE WORK

In this thesis, I have presented SUNSHINE, a cross-domain simulator for the design, development and implementation of wireless sensor network applications. SUNSHINE is realized by the seamless integration of a network-oriented simulation engine, an instruction-set simulation engine and a hardware domain simulation engine. By the seamless integration of the simulators in different domains, the performance of network protocols and software applications under realistic hardware constraints and network settings can be captured by SUNSHINE with network-event, instruction-level, and cycle-level accuracy. SUNSHINE outperforms other existing simulation engines because it can support user-defined sensor node platform architecture, which is a significant improvement for sensornet simulators. SUNSHINE can also capture hardware behavior which is the unique feature of sensornet simulators. SUNSHINE can serve as an efficient tool for both software and hardware researchers to design sensor node's platform architectures as well as develop sensornet applications.

In the next step, we will transit SUNSHINE simulation models into real sensor prototypes. To do so, we will create a design flow to map simulations of software and hardware designs

to real software and hardware implementations. SUNSHINE supports simulation reusability since it can map simulations of software and hardware designs to real software and hardware implementations on sensor prototypes. This can reduce the development time for sensor networks. We consider developing a reconfigurable hardware platform based on low-power FPGAs to demonstrate this design flow. This hardware platform can also serve directly as a novel flexible platform that enables the sensor network community to quickly prototype various sensor architectures.

Bibliography

- [1] P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: accurate and scalable simulation of entire tinyos applications,” in *Computer Communications and Networks, International Conference on Embedded networked sensor systems*, pp. 126–137, 2003.
- [2] “Simulavr: an avr simulator.” <http://www.nongnu.org/simulavr/>.
- [3] P. Schaumont, D. Ching, and I. Verbauwhede, “An interactive codesign environment for domain-specific coprocessors,” *ACM Transactions on Design Automation for Embedded Systems*, vol. 11, no. 1, pp. 70–87, 2006.
- [4] S. Ohara, M. Suzuki, S. Saruwatari, and H. Morikawa, “A prototype of a multi-core wireless sensor node for reducing power consumption,” in *International Symposium on Applications and the Internet*, July 2008.
- [5] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. Baras, “Atemu: a fine-grained sensor network simulator,” *Sensor and Ad Hoc Communications and Networks*, pp. 145–152, Oct. 2004.
- [6] B. L. Titzer, K. D. Lee, and J. Palsberg, “Aurora: Scalable sensor network simulation with precise timing,” in *In Proc. of the 4th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, pp. 477–482, 2005.
- [7] *The Network Simulator-ns-2*. <http://www.isi.edu/nsnam/ns/>.
- [8] P. Levis and N. Lee, *TOSSIM: A simulator for TinyOS Networks*. <http://www.cs.berkeley.edu/pal/pubs/nido.pdf>.
- [9] B. Titzer, “Aurora: Scalable sensor simulation with precise timing,” tech. rep., 4760 Boelter Hall, UCLA, Feb. 2005.
- [10] P. Schaumont and I. Verbauwhede, “A component-based design environment for electronic system-level design,” in *IEEE Design and Test of Computers Magazine, special issue on Electronic System-Level Design*, Sep. – Oct. 2006.
- [11] A. Hodjat, L. Batina, D. Hwang, and I. Verbauwhede, “A hyperelliptic curve crypto coprocessor for an 8051 microcontroller,” in *In IEEE Workshop on Signal Processing Systems (SIPS)*, Nov. 2005.

- [12] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, “Multi-core curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over $gf(2n)$,” *IEEE Transactions on Computers*, vol. 56, no. 9, pp. 1269–1282, 2007.
- [13] M. Knezevic, K. Sakiyama, Y. Lee, and I. Verbauwhede, “On the high-throughput implementation of ripemd-160 hash algorithm,” in *In Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP’08)*, pp. 85–90, July 2008.
- [14] S. Ahuja, D. A. Mathaikutty, and S. Shukla, “Model-based power estimation using least squares regression on fsmd models,” tech. rep., 2008.
- [15] B. Kopf and D. Basin, “An information-theoretic model for adaptive side-channel attacks,” in *In CCS ’07: Proceedings of the 14th ACM conference on Computer and communications security*, pp. 286–296, 2007.
- [16] M. R. Hansen, J. Madsen, and A. W. Brekling, “Semantics and verification of a language for modeling hardware architectures,” *In Formal Methods and Hybrid Real-Time Systems 2007*, vol. 4700 of Lecture Notes on Computer Science, pp. 300–319, 2007.
- [17] *2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B)* .
<http://focus.ti.com/docs/prod/folders/print/cc2420.html>.
- [18] *GEZEL Library blocks*. http://rijndael.ece.vt.edu/gezel2/index.php-/GEZEL_Library_Blocks.
- [19] *GEZEL Language Reference*. http://rijndael.ece.vt.edu/gezel2/index.php-/GEZEL_Language_Reference.
- [20] S. Capkun and J. P. Hubaux, “Secure positioning in wireless networks,” *IEEE Journal of Selected Areas in Communications*, vol. 24, Feb. 2006.

Appendix A

HARDWARE SPECIFICATION FILE

```
//--- AVR
ipblock avr {
    iptype "atm128core";
    ipparm "exec=app";
}

//--- LEDS
ipblock m_led0(out data : ns(1)) {
    iptype "atm128sourcepin";
    ipparm "core=avr";
    ipparm "pin=A2";
}

ipblock m_led1(out data : ns(1)) {
```

```
    iptype "atm128sourcepin";
    ipparm "core=avr";
    ipparm "pin=A1";
}

ipblock m_led2(out data : ns(1)) {
    iptype "atm128sourcepin";
    ipparm "core=avr";
    ipparm "pin=A0";
}

//--- CC2420 Status
ipblock m_fifo (in data : ns(1)) {
    iptype "atm128sinkpin";
    ipparm "core=avr";
    ipparm "pin=B7";
}

ipblock m_fifop(in data : ns(1)) {
    iptype "atm128sinkpin";
    ipparm "core=avr";
    ipparm "pin=E6";
}

ipblock m_cca (in data : ns(1)) {
    iptype "atm128sinkpin";
    ipparm "core=avr";
```



```
    ipparm "pin=D6";
}

ipblock m_sfd (in data : ns(1)) {
    iptype "atm128sinkpin";
    ipparm "core=avr";
    ipparm "pin=D4";
}

//--- CC2420 SPI Control
ipblock m_ss (out data : ns(1)) {
    iptype "atm128sourcepin";
    ipparm "core=avr";
    ipparm "pin=B0";
}

ipblock m_sck (out data : ns(1)) {
    iptype "atm128sourcepin";
    ipparm "core=avr";
    ipparm "pin=B1";
}

ipblock m_mosi (out data : ns(1)) {
    iptype "atm128sourcepin";
    ipparm "core=avr";
    ipparm "pin=B2";
}
```

```

ipblock m_miso (in data : ns(1)) {
    iptype "atm128sinkpin";
    ipparm "core=avr";
    ipparm "pin=B3";
}

//--- CC2420
ipblock m_cc2420(out fifo, fifop, cca, sfd : ns(1);
                in ss, sck, mosi : ns(1);
                out miso : ns(1)) {
    iptype "ipblockcc2420";
    ipparm "node_id = 2";
}

//--- Interconnect
dp top {
    sig w_led0, w_led1, w_led2 : ns(1);
    sig w_fifo, w_fifop, w_cca, w_sfd : ns(1);
    sig w_ss, w_sck, w_mosi, w_miso : ns(1);

    use avr;
    use m_led0 (w_led0);
    use m_led1 (w_led1);
    use m_led2 (w_led2);
    use m_fifo (w_fifo);
    use m_fifop(w_fifop);
}

```

```
use m_cca (w_cca);
use m_sfd (w_sfd);
use m_ss (w_ss);
use m_sck (w_sck);
use m_mosi (w_mosi);
use m_miso (w_miso);
use m_cc2420(w_fifo, w_fifop, w_cca, w_sfd, w_ss, w_sck, w_mosi, w_miso);
always {
    $display($cycle, " CCA ", w_cca, " SS ", w_ss, " SCK ", w_sck,
              " MOSI ", w_mosi, " MISO ", w_miso, " L0 ", w_led0,
              " L1 ", w_led1, " L2 ", w_led2);
}
}

system S {
    top;
}
```

Appendix B

SUNSHINE MAIN SIMULATION FILE

```
#include "runcosim.h"

aipblock * gplatform_ipblockcreate(char *instname, char *tname);

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

/*=====co-sim node initialization=====*/
    unsigned long id_list [] = {1}; // co-sim node id number
    cosim_node_init(id_list,1); //the sum of cosim nodes
/*=====*/

/*=====TOSSIM Initialization=====*/
    Tossim* t=new Tossim(NULL);
    Radio* r=t->radio ();
    char line [128];
```

```

FILE *fp1; FILE *fp2;
int d1,d2;
double d3;
int *str;
int val;

t->addChannel("RadioCountToLedsC", fdopen(1, "w"));
t->addChannel("Boot", fdopen(1, "w"));

for (int i = 0; i < 2; i++) {
    Mote* m = t->getNode(i);
    m->bootAtTime(2* i + 1);
    for (int j = 0; j < 2; j++) {
        if (i != j) {
            r->add(i, j, -10.0);
        }
    }
    for (int j = 0; j < 500; j++) {
        m->addNoiseTraceReading((char)(drand48() * 20) - 70);
    }
    m->createNoiseModel();
}

/*=====*/

run_cosim(argc,argv,envp,t); //run SUNSHINE simulation
return 0;
}

```