SUNSHINE: Integrate TOSSIM and P-Sim

Yi Tang

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

September 23, 2011
Blacksburg, VA

Keywords: Sensor networks, cross-domain simulator,
hardware-software co-simulation

SUNSHINE: Integrate TOSSIM and P-Sim

Yi Tang

ABSTRACT

Simulators are important tools for wireless sensor network (sensornet) design and evaluation. However, existing simulators only support evaluations of protocols and software aspects of sensornet design. Thus they cannot accurately capture the significant impacts of various hardware designs on sensornet performance.

To fill in the gap, we proposed 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 through the integration of two modules: a network simulator TOSSIM [1] and hardware-software simulator P-Sim composed of an instruction-set simulator SimulAVR [2] and a hardware simulator GEZEL [3].

This thesis focuses on the integration of TOSSIM and P-Sim. It discusses the integration design considerations and explains how to address several integration challenges: time conversion, data conversion, and time synchronization.

Some experiments are also given to demonstrate SUNSHINE’s cross-domain simulation capability, showing SUNSHINE's strength by integrating simulators from different domains.

Acknowledgements

I would like to express my gratitude to my advisor Dr. Yaling Yang, for giving me support and guidance throughout my entire graduate study. I could not have imagined having a better advisor and mentor for my master study.

I would like to thank Dr. Patrick Schaumont. He helped us with numerous questions and gave the SUNSHINE project a lot of sound advice and guidance.

Also I would like to thank my teammates: Jingyao Zhang, Sachin Hirve, and Srikrishna Iyer. Without the joint effort from them we would never have made SUNSHINE happen.

Lastly, and most importantly, I wish to thank my parents, Yunshan Tang and Jingqi Huang, my grandparents, Peiling Ma and Shichao Tang. They are supporting and loving me all the time. To them I dedicate this thesis.

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

List of Figures

Chapter 1 Introduction

1.1 Motivation

Over the past few years, we have witnessed an impressive growth of sensornet 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]. Hence, 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 sensornets, simulators are necessary tools to help sensornet researchers develop and analyze new designs. Developing hardware-and-software codesigned sensornet 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 sensornet simulators, these existing sensornet 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, sensornet 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 sensornet research community to develop a clear direction on how to improve the sensor hardware platforms. The performance/energy benefits that are available to the hardware community therefore remain hard to reach.

To address the challenge, we propose the idea of integrating simulators from three different domains: network (protocol), software, and hardware. Figure 1.1 shows this design concept. As a result, we are able to provide more complete and accurate simulation information from different levels, from network performance to hardware profiling and verification.

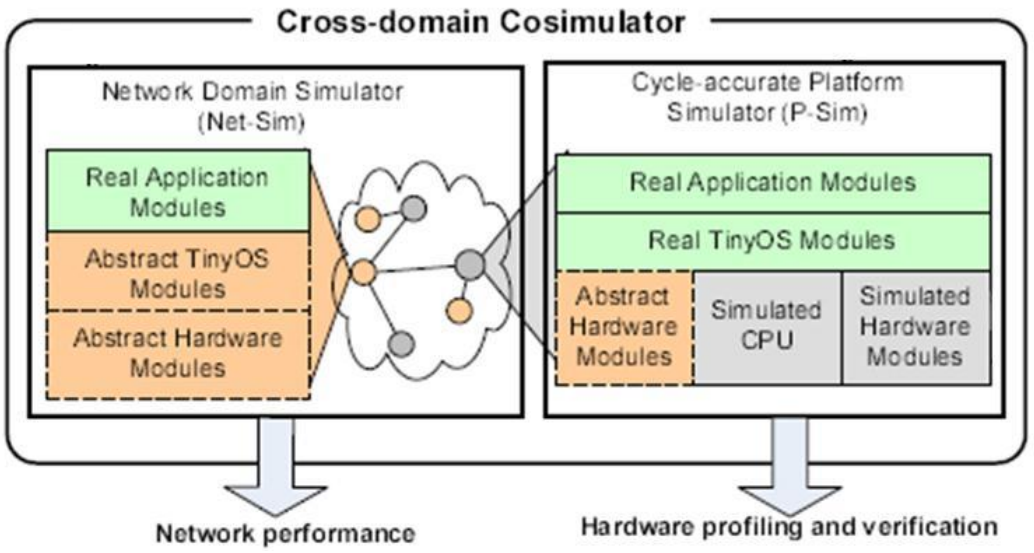
Cross-domain Cosimulator
Network Domain Simulator (Net-Sim)
Real Application Modules
Abstract TinyOS Modules
Abstract Hardware Modules
Cycle-accurate Platform Simulator (P-Sim)
Real Application Modules
Real TinyOS Modules
Abstract Hardware Modules
Simulated CPU
Simulated Hardware Modules
Network performance
Hardware profiling and verification

Figure 1.1: Design concept of cross-domain simulator

Following this concept, we designed a new sensornet simulator, SUNSHINE (Sensor Unified aNalyzer for Software and Hardware in Networked Environments), to support hardware-software co-design in sensornets. By the integration of two modules: a network simulator TOSSIM [1] and hardware-software simulator P-Sim, composed of an instruction-set simulator SimulAVR [2] and a hardware simulator GEZEL [3], SUNSHINE can capture the performance of software network protocols and applications under realistic hardware constraints and network settings.

1.2 Related Articles

The SUNSHINE extension is described in:

1. Jingyao Zhang, Yi Tang, Sachin Hirve, Srikrishna Iyer, Patrick Schaumont, and Yaling Yang, "A Software-Hardware Emulator for Sensor Networks", IEEE SECON 2011.

2. Jingyao Zhang, Yi Tang, Sachin Hirve, Patrick Schaumont and Yaling Yang, “SUNSHINE: A Platform-Agnostic Sensor Network Simulator”, ACM Mobicom 2010 (poster).

3. 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).

1.3 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 first introduces overall architecture of SUNSHINE. Chapter 3 provides in-depth description of the cross-domain interface that integrates TOSSIM and P-Sim together. Chapter 4 gives several SUNSHINE experimental results showing the result of the integration. Chapter 5 concludes the thesis and discussed the future work.

Chapter 2 Background

SUNSHINE integrates two main modules: a network simulator TOSSIM and hardware-software simulator P-Sim, composed of an instruction-set simulator SimulAVR and a hardware simulator GEZEL. This section firstly introduces the two main modules. Then we explain the overall architecture of SUNSHINE to show how two main modules integrate and work together.

2.1 TOSSIM

TOSSIM is a discrete event simulator for TinyOS sensor networks [7]. It runs real TinyOS sensornet applications on a virtual simulation environment, simulates packet transmission and reception as events.

As shown in Figure 2.1, instead of running on a real sensor node, the application runs in TOSSIM environment. The packets are passed through the TOSSIM radio model for evaluation to determine whether it is a lost packet or not. Packets sent and received of each node are captured as events. TOSSIM pops out events according to their time stamp, moving the simulation forward.

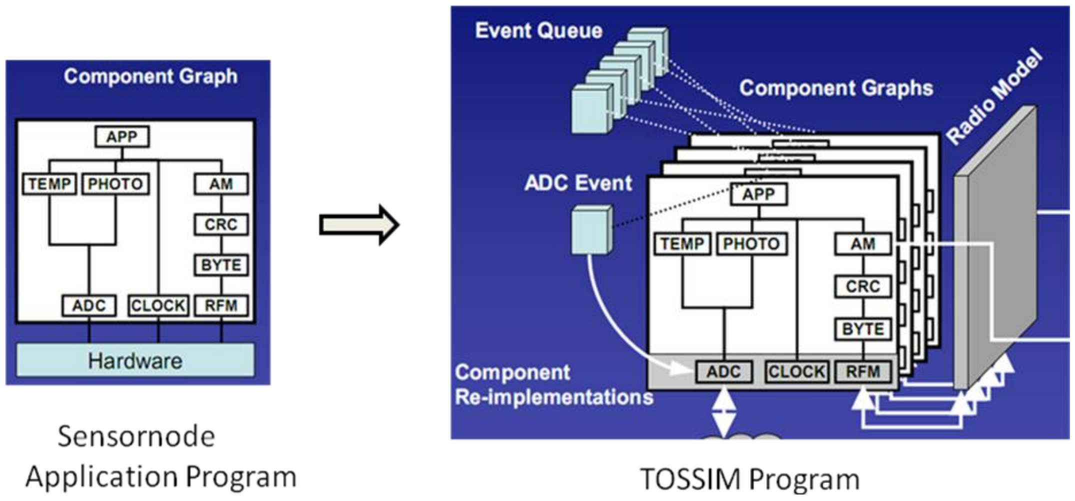

Figure 2.1: TOSSIM architecture

2.2 P-Sim

P-Sim acts as a hardware-software simulator. It captures hardware and software information of a sensor node on cycle-accurate level. P-Sim is composed of SimulAVR and GEZEL.

SimulAVR 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++ which can be easily interconnected with GEZEL. SimulAVR provides accurate timing of software execution and can simulate multiple AVR devices in one simulation.

GEZEL 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 cryptoprocessors [8–10], cryptographic hashing modules [11], high-level estimation of chip power consumption [12], and formal verification of security properties of hardware modules [13], [14]. 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 co-simulator 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.

2.3 SUNSHINE Architecture

SUNSHINE integrates TOSSIM and P-Sim to simulate sensornets in network, software, and hardware domains. Our system design supports selection of 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

P-Sim. 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.

SUNSHINE is able to co-simulate multiple cycle-level co-sim nodes with TOSSIM nodes. Figure 2.2 shows a basic example. 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 hardware and software 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 many 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 benefit from the scalability of event-driven simulation.

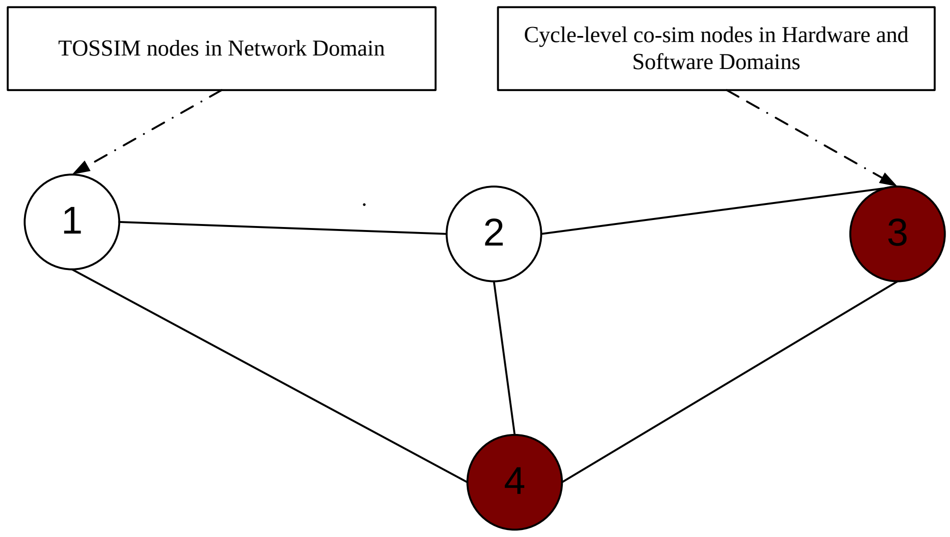
TOSSIM nodes in Network Domain
Cycle-level co-sim nodes in Hardware and Software Domains
1
2
3
4

Figure 2.2: Example of co-simulation

The software architecture is shown in Figure 2.3. First, for co-sim sensor nodes (e.g. Node 3 and 4 in Figure 2.2), executable binaries for real sensor nodes are compiled from TinyOS sensornet applications. These binaries are interpreted by SimulAVR. 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 of such peripherals 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 co-sim nodes by GEZEL and SimulAVR, or are simulated as TOSSIM nodes (e.g. Node 1 and 2 in Figure 2.2). Sensornet applications that need to run on TOSSIM nodes are compiled as special binaries for TOSSIM simulation.

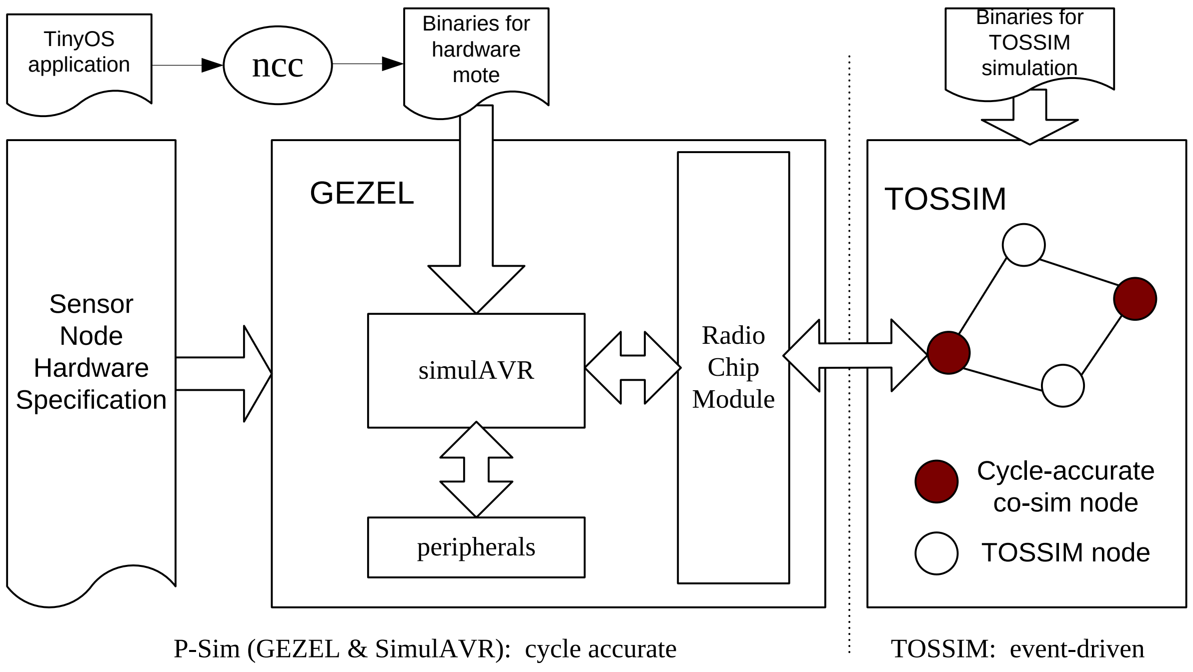
TinyOS application
ncc
Binaries for hardware mote
Binaries for TOSSIM simulation
Sensor Node Hardware Specification
GEZEL
simulAVR
peripherals
Radio Chip Module
TOSSIM
Cycle-accurate co-sim node
TOSSIM node
P-Sim (GEZEL & SimulAVR): cycle accurate
TOSSIM: event-driven

Figure 2.3: SUNSHINE software architecture

The key point discussed in this thesis is the interface between P-Sim and TOSSIM, shown as the bi-directional arrow between the P-Sim module and TOSSIM module in Figure 2.3. The interface enables data exchange and synchronization between network and hardware-software domains, thus the two modules can seamlessly work together. We will discuss the cross-domain in great details in Chapter 3.

Chapter 3 Cross-Domain Interface

This chapter explores the detailed design of the cross-domain interface.

There are several design challenges associated with the interface of TOSSIM and P-Sim. The first one is the time granularity difference. As mentioned in previous sections, TOSSIM captures the time as event time stamp while P-Sim interprets time on a more fine-grained cycle-accurate level. Therefore this causes time conversion and synchronization challenges.

Second, P-Sim acts as a real sensor node, which interprets packet data on bit level. It reads and writes in the way as a real sensor node does. While on TOSSIM side, it has its own data structure to store the packet data. Therefore how to correctly convert P-Sim packet to TOSSIM packet and vice versa will be another problem to solve.

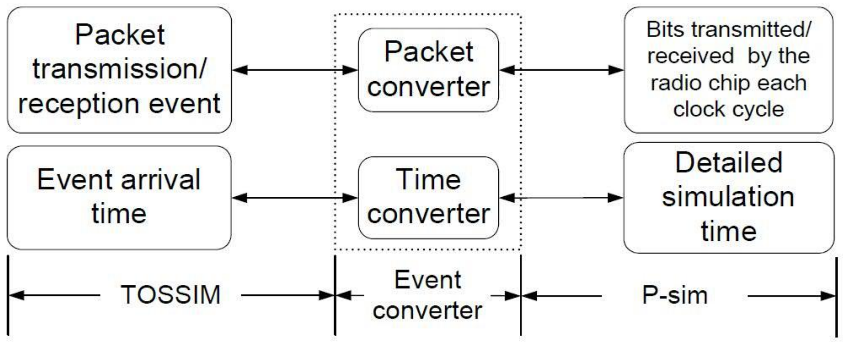

Figure 3.1: Time Converter and Packet Converter

In following sections, we explained Time Converter that solves time conversion issue and Packet Converter that solves packet data conversion issue, shown in Figure 3.1. Also we provided solution to time synchronization issue in section 3.3.

In addition, to assist the time and packet conversion, we create a data structure named sharevar_cc2420 shown below. It stores all the time and packet information on a per sensor node basis. The information is shared between TOSSIM and P-Sim.

```
typedef struct sharevar_cc2420 {
   unsigned long node_id;  //id of cosimulation node
   enum cosim_node_mode node_state;

   int threshold;
   int exceed_threshold;
   int rx_fifo_overflow;

   int tx_fifo_underflow;
        //tx_fifo_underflow==1 if underflow happens during transmittion
   int cca_tos;
        //cca_tos==0 when channel is clear
        //cca_tos==1 when channel is busy
   int tx;
              //tx==1 transmission starts
              //tx==0 transmission ends
              //tx==-1 initialization
   int tx_after_sfd;
              //tx==1 transmission starts after SFD
              //tx==0 transmission ends
   int addr_recog;
        //addr_recog==1 address recognition enabled
        //addr_recog==0 address recognition disabled
   int addr_recog_status;
        //addr_recog_status==1 address recognition successful
        //addr_recog_status==0 address recognition failed
       //addr_recog_status==-1 initialization
   int rx_after_sfd;
            //rx_after_sfd==1 reception starts after SFD
            //rx_after_sfd==0 reception ends
   int rx_after_length;
            //rx_after_length==1 reception starts after length
            //rx_after_length==0 reception ends
   int rx;
        //rx==1 reception starts
        //rx==0 reception ends
        //rx=-1 initialization
   //int txack;    //1:reception of tx_ack starts, else 0
   int txack_after_sfd;
        //txack_after_sfd==1 ack transmission starts after SFD
        //txack_after_sfd==0 ack transmission ends
        //txack_after_sfd==-1 idle

   fifo_t rx_fifo;  // RX FIFO
   fifo_t tx_fifo;  // TX FIFO
   unsigned char sec_ram[SEC_RAM_LENGTH];  //SEC RAM 112 Bytes
} sharevar_cc2420_t;
```

3.1 Time Converter

3.1.1 Overview

The design of time converter follows the CC2420 Radio control states shown in Figure 3.2, according to the CC2420 datasheet [15]. The RX and TX states play a key role in sensornet simulation.

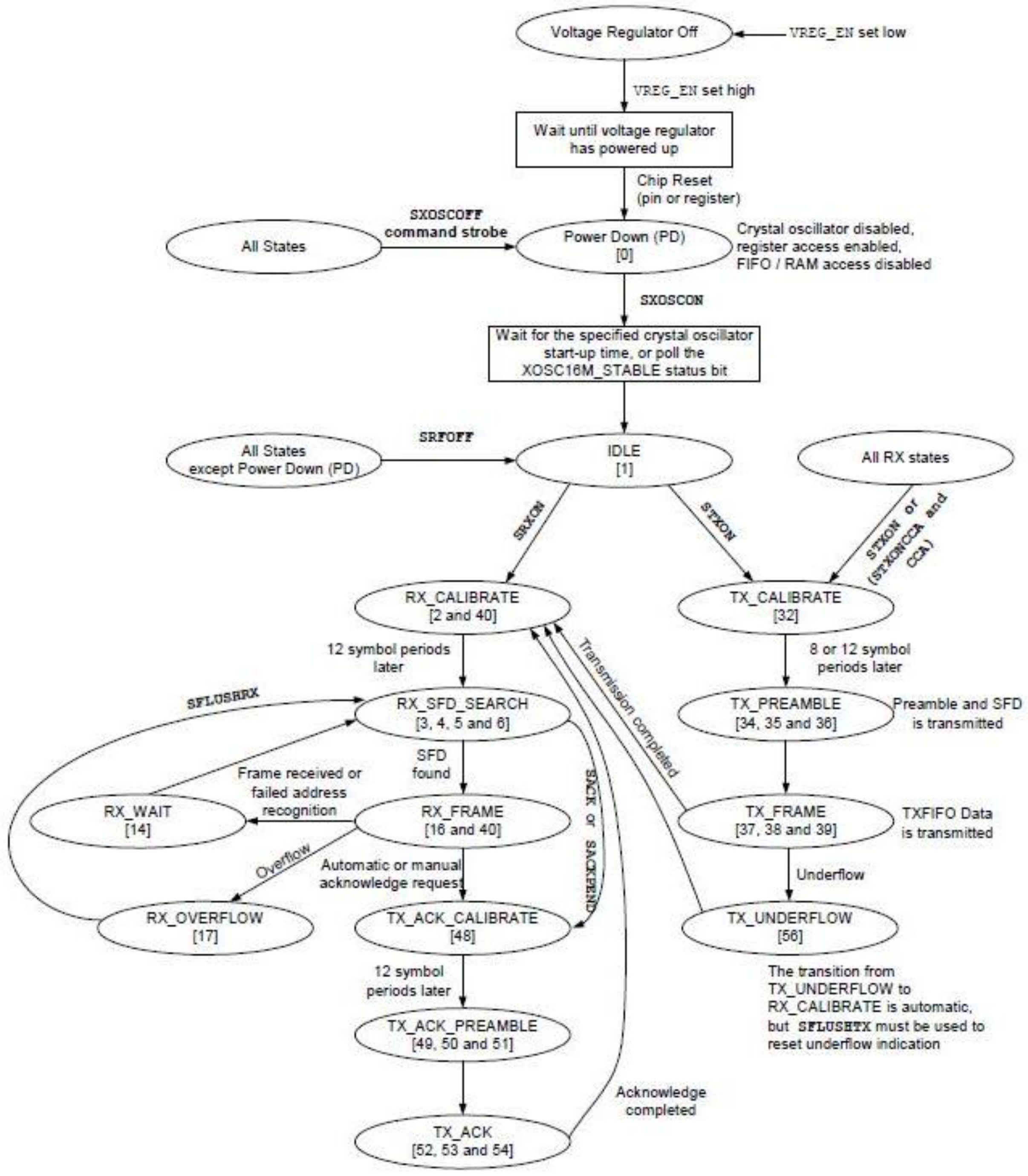
Voltage Regulator Off
VREG_EN set low
VREG_EN set high
Wait until voltage regulator has powered up
Chip Reset (pin or register)
All States
SXOSCOFF command strobe
Power Down (PD) [0]
Crystal oscillator disabled, register access enabled, FIFO / RAM access disabled
SXOSCON
Wait for the specified crystal oscillator start-up time, or poll the XOSC16M_STABLE status bit
All States except Power Down (PD)
SRFOFF
IDLE [1]
All RX states
SRXON
STXON
STXON or (STXONCCA and CCA)
RX_CALIBRATE [2 and 40]
TX_CALIBRATE [32]
12 symbol periods later
8 or 12 symbol periods later
Transmission completed
SFLUSHRX
RX_SFD_SEARCH [3, 4, 5 and 6]
TX_PREAMBLE [34, 35 and 36]
Preamble and SFD is transmitted
SFD found
Frame received or failed address recognition
SACK or SACKPEND
RX_WAIT [14]
RX_FRAME [16 and 40]
TX_FRAME [37, 38 and 39]
TXFIFO Data is transmitted
Overflow
Automatic or manual acknowledge request
Underflow
RX_OVERFLOW [17]
TX_ACK_CALIBRATE [48]
TX_UNDERFLOW [56]
12 symbol periods later
The transition from TX_UNDERFLOW to RX_CALIBRATE is automatic, but SFLUSHTX must be used to reset underflow indication
TX_ACK_PREAMBLE [49, 50 and 51]
Acknowledge completed
TX_ACK [52, 53 and 54]

Figure 3.2: CC2420 State Machine

On P-Sim side, the radio chip is modeled as a state machine that covers all the real CC2420 radio states shown in Figure 3.2. While on TOSSIM side, there is only a coarse-gained abstraction of TX, RX (Two states). Thus the converter needs to generate more fine-grained signals based on TOSSIM's simulation results, and feed them into P-sim. Specifically, TOSSIM only cares the start and end time of a TX event, while the converter computes a few more extra timing points such as when the transmission proceeds to after the preamble point.

We use a series of shared variables, as shown in the above sharevar_cc2420 data structure, to synchronize the states on P-Sim and TOSSIM side. P-Sim monitors value change of these variables every circle, so once there is a value change, P-Sim will get a specific signal and change its state accordingly.

Then the question will be how to change the value of these variables in the correct time on TOSSIM side. We created a series of additional Co-sim events (events used for co-sim nodes) in addition to normal TOSSIM events. These events are generated with the correct time stamp and inserted to the event queue. And when different Co-sim events pop out, TOSSIM will manipulate different set of shared variables, thus guarantee that the signals to P-Sim side are raised in the correct time. Based on this mechanism, TOSSIM will be able to notify P-Sim when some actions need to be taken in an accurate time, for example, when the node is receiving a packet.

On the other hand, P-Sim can also insert some Co-sim events. Thus TOSSIM will also be signaled when some actions need to be taken, for example, when the node is going to send out a packet. This two-way signal mechanism is shown in Figure 3.3.

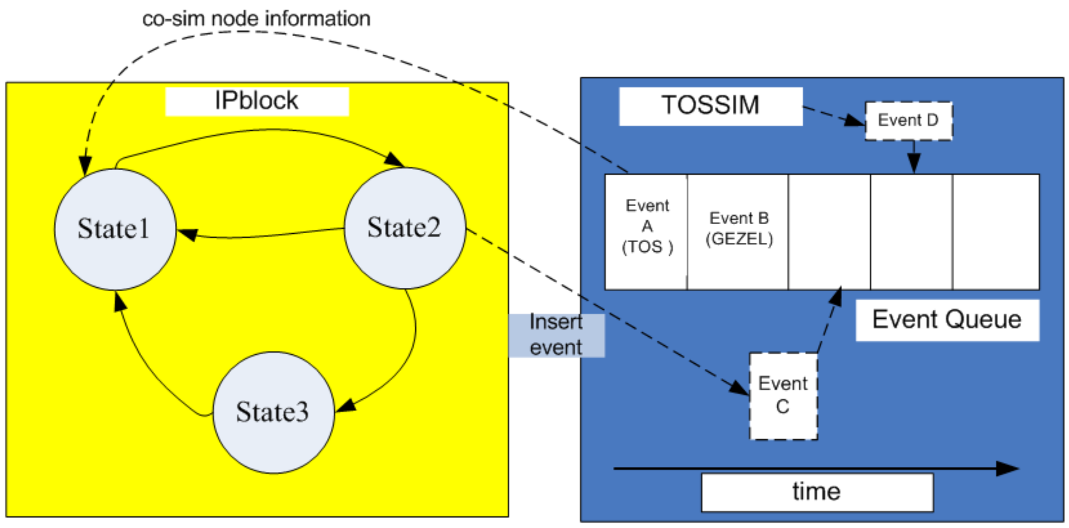

Figure 3.3: Co-sim events

We will discuss more details regarding the determination of TX and RX signals in 3.1.2 and 3.1.3.

3.1.2 TX

The detailed timing points during TX are shown in Figure 3.4. We need to generate the "after preamble" and "TX completed" signal.

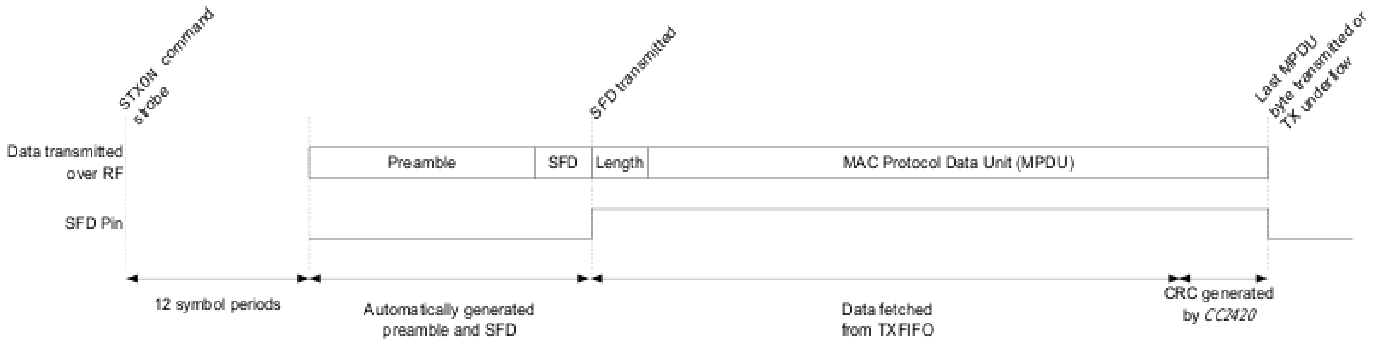

Figure 3.4: TX Timing

Thus in the code we follow the logic and create co-sim events accordingly. The logic is shown in Figure 3.5.

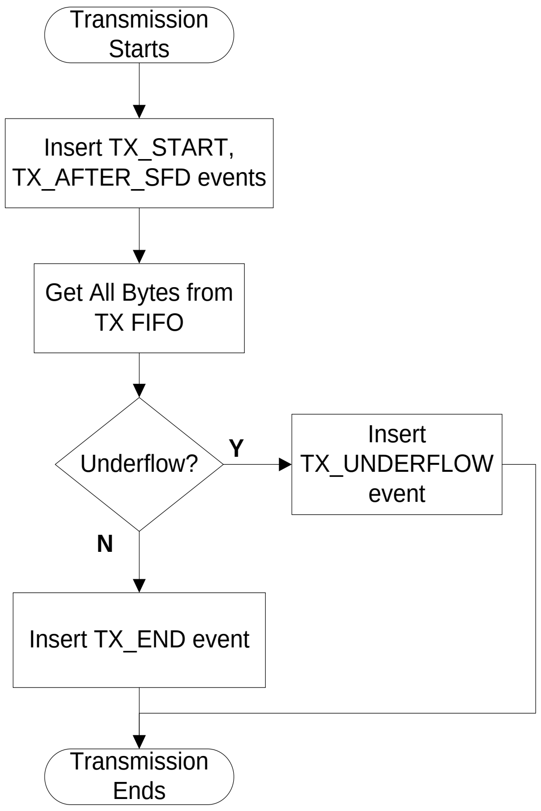
Transmission Starts
Insert TX_START, TX_AFTER_SFD events
Get All Bytes from TX FIFO
Underflow?
Y
Insert TX_UNDERFLOW event
N
Insert TX_END event
Transmission Ends

Figure 3.5: TX time conversion flowchart

During TX, in addition to inserting co-sim events, we are also getting data in TX FIFO. This is the data packet that needs to be sent out through the TOSSIM radio model, simulating the process of sending over a packet through the air. Packet conversion is involved in the process and will be discussed in section 3.2.

3.1.3 RX

The detailed timing points during RX are shown in Figure 3.6. We need to generate the "after preamble", "after length", "threshold", and "TX completed" signal. Also if the address recognition fails the "address recognition failed" signal also needs to be raised.

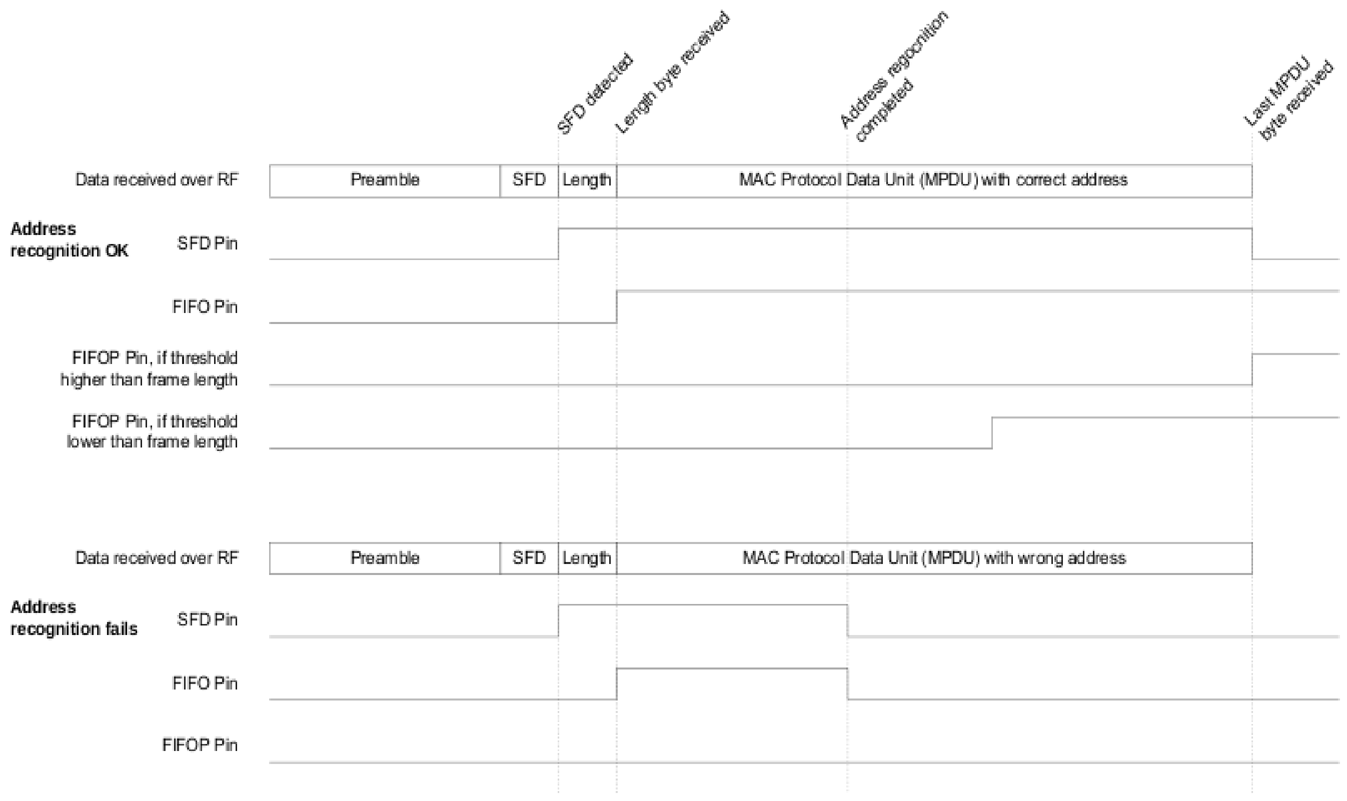
SFD detected
Length byte received
Address regocnition completed
Last MPDU byte received
Data received over RF
Preamble
SFD
Length
MAC Protocol Data Unit (MPDU) with correct address
Address recognition OK
SFD Pin
FIFO Pin
FIFOP Pin, if threshold higher than frame length
FIFOP Pin, if threshold lower than frame length
Data received over RF
Preamble
SFD
Length
MAC Protocol Data Unit (MPDU) with wrong address
Address recognition fails
SFD Pin
FIFO Pin
FIFOP Pin

Figure 3.6: RX Timing

Thus in the code we follow the logic and create signal events accordingly. The logic is shown in the flowchart in Figure 3.7.

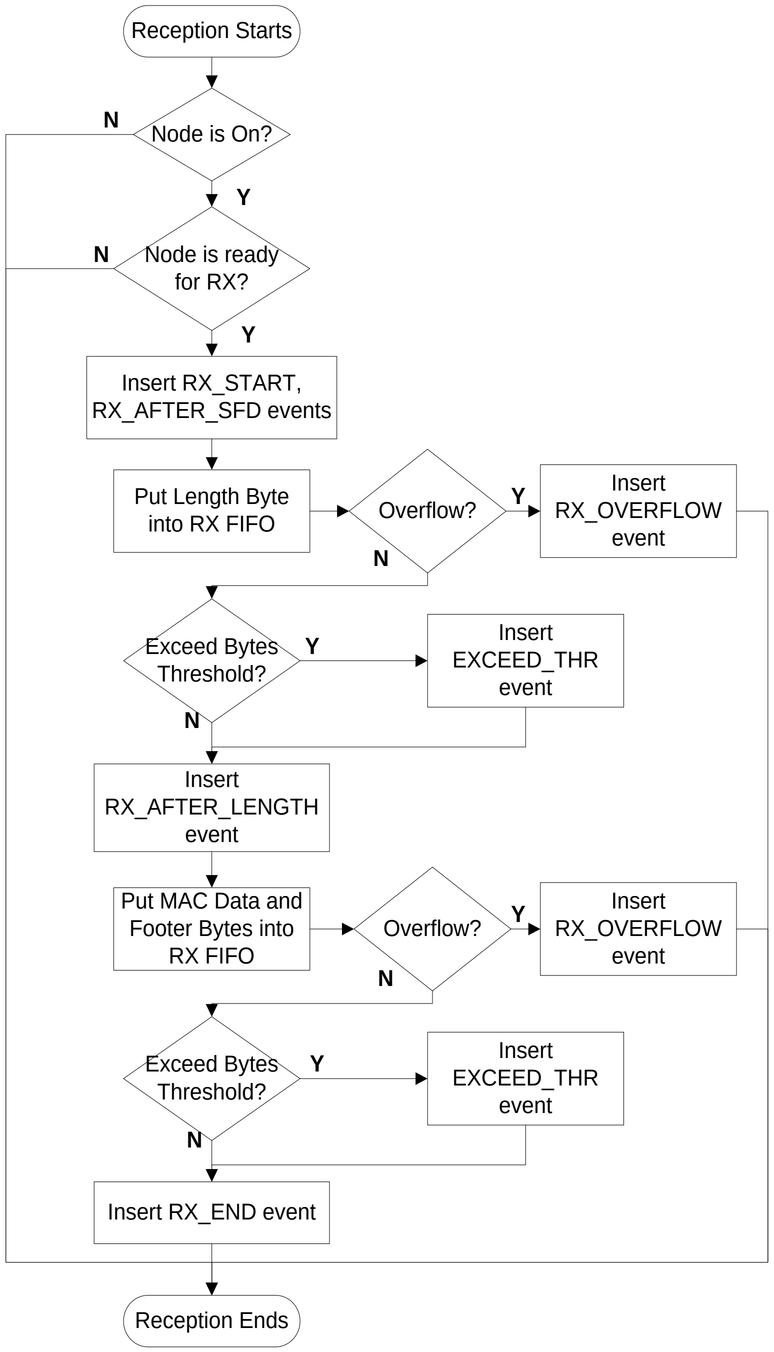

Figure 3.7: RX time conversion flowchart

During RX, in addition to inserting co-sim events, we are also putting data in RX FIFO. This is the data packet that a co-sim node received from TOSSIM radio model, simulating the process of receiving a packet from the air. Packet conversion is also involved here and will be discussed in section 3.2.

3.2 Packet Converter

Packet converter translates the TOSSIM packet abstraction to bit-level packet in RX FIFO that can be fetched by P-Sim side, also it translates TX FIFO bit-level packet that loaded by P-Sim to TOSSIM packet abstraction.

3.2.1 CC2420 Packet Format

For co-sim node, SUNSHINE captures every single bit transfer during TX and RX, in the way a real sensor node works. Thus we need to feed data to P-sim Side in the same way as what happens in real sensor node. Figure 3.8 shows the CC2420 packet format.

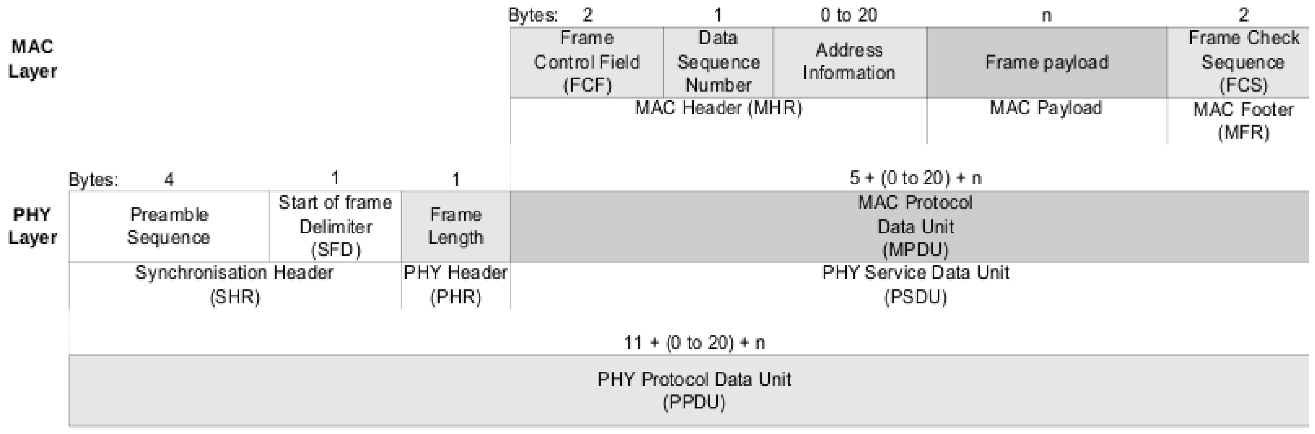

Figure 3.8: CC2420 packet format

3.2.2 TOSSIM Packet Format

TOSSIM abstracts packet in a different way. It simplifies the packet as the data structure shown as the following code:

```
typedef nx_struct tossim_header {
 nx_am_addr_t dest;
 nx_am_addr_t src;
 nx_uint8_t length;
 nx_am_group_t group;
 nx_am_id_t type;
} tossim_header_t;
```

```
typedef nx_struct tossim_footer {
  nxle_uint16_t crc;
} tossim_footer_t;

typedef nx_struct tossim_metadata {
  nx_int8_t strength;
  nx_uint8_t ack;
  nx_uint16_t time;
} tossim_metadata_t;
```

From the code above we see that the packet data used in TOSSIM can be retrieved from the real packet data, and vice versa. For example, *dest* in *tossim_header* maps to the destination node address in "Address Information" in Figure 3.8. We convert the packet data based on this mapping relationship discussed in the next section.

3.2.3 Packet Data Conversion

Packet data conversion happens during RX and TX process. More specifically, we need to convert the data when fetching data from TX FIFO to TOSSIM or from TOSSIM to RX FIFO. We provide some example code here to illustrate the conversion.

First we show the conversion from TX FIFO to TOSSIM, using the destination node address as an example. The destination address is 2 Bytes long, and since the storage unit is 1 Byte, it is important to interpret the data in the correct order. From the code we can see the low byte comes first, then the high byte. Also we check underflow and signal it if happens when getting data from TX FIFO, following the real sensor node's behavior.

```
  // 2 Bytes Dst Addr
  if (tx_fifo_get(id,&temp)==-1) {
    //printf("Underflow!\n");
    *underflow = 1;
    return counter;
  }
  counter++;
  len--;
  header->dest = temp;
  printf("Byte %d: %x Dst Addr Low >>>> TOSSIM Dest: %x\n", counter, temp, header->dest);

  if (tx_fifo_get(id,&temp)==-1) {
    //printf("Underflow!\n");
    *underflow = 1;
    return counter;
  }
  counter++;
  len--;
  header->dest ^= temp<<8;
  printf("Byte %d: %x Dst Addr High >>>> TOSSIM Dest Final: %x\n", counter, temp, header->dest);
```

Similarly, we follow the same rule in conversion from TOSSIM to RX FIFO. Below still uses destination node address as an example. Notice that when converting the bytes, we also check if RX FIFO overflows or total number of bytes in RX FIFO exceeds a certain threshold, following the real sensor node's behavior.

```
temp = header->dest;
if (rx_fifo_put(id, temp)==-1) {
  cosimEvent = allocate_cosim_event(rx_after_sfd + BYTE_TO_TICK(byte_count), id, RX_OVERFLOW);
  sim_queue_insert(cosimEvent);
  return;
}  // Byte 7 - Dest Low
byte_count++;
if (rx_fifo_count(id) > threshold_get(id)) {
  cosimEvent = allocate_cosim_event(rx_after_sfd + BYTE_TO_TICK(byte_count), id, EXCEED_THR);
  sim_queue_insert(cosimEvent);
}
```

```
if (rx_fifo_put(id, temp>>8)==-1) {
  cosimEvent = allocate_cosim_event(rx_after_sfd + BYTE_TO_TICK(byte_count), id, RX_OVERFLOW);
  sim_queue_insert(cosimEvent);
  return;
}  // Byte 8 - Dest High
byte_count++;
if (rx_fifo_count(id) > threshold_get(id)) {
  cosimEvent = allocate_cosim_event(rx_after_sfd + BYTE_TO_TICK(byte_count), id, EXCEED_THR);
  sim_queue_insert(cosimEvent);
}
```

We convert other data fields in the same way as shown in the above two examples. The conversion guarantees that both TOSSIM and P-Sim receive correct data from the other side thus to give correct simulation results.

3.3 Time Synchronization

TOSSIM and P-Sim have different time granularity, thus simulations on these two modules run on different paces. It is important to maintain the time synchronized on both sides.

3.3.1 Time Synchronization Problem

Figure 3.9 illustrates mismatches in simulation time between event-driven simulation and cycle-level simulation. As shown in Figure 3.9, P-Sim runs 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.

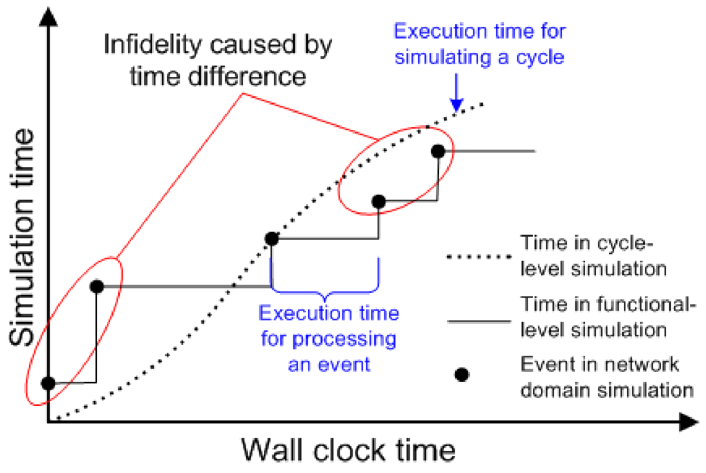

Figure 3.9: Simulation time in different domains

3.3.2 Solution

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 3.10.

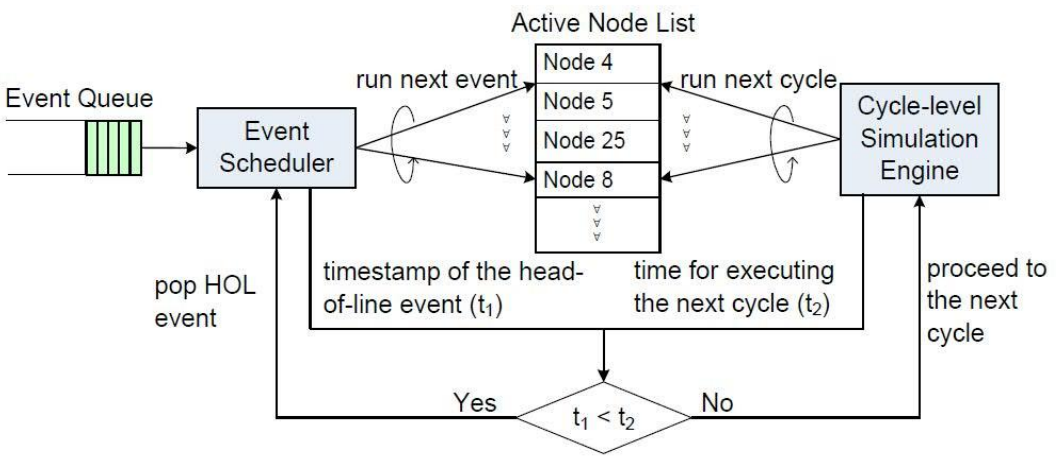
Active Node List
Node 4
Node 5
Node 25
Node 8
Event Queue
run next event
run next cycle
Event Scheduler
Cycle-level Simulation Engine
pop HOL event
timestamp of the head-of-line event (t1)
time for executing the next cycle (t2)
proceed to the next cycle
Yes
t1 < t2
No

Figure 3.10: Synchronization scheme

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 3.10 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. This way, a node's sleep or wakeup state does not need to be checked every clock cycle. Given the fact that in sensornets, a sensor node spends most of its time in sleep mode, this design will greatly accelerate SUNSHINE's simulations. Below shows the core code based on the algorithm.

```
 for (cyclecount=0; (cyclecount<max_cnum); cyclecount++){
   gez_time = (double) glbRTCycle*1.25e-7;
   tos_time = next_tos_time();
   if (gez_time <= tos_time  || tos_time == -1){
     if (RTsimgen)
           RTsimgen->step();
           if ((ignoresleep || (glbRTSimMode == RTSimMode_sleep)) && (glbRunningISS == 0)) {
           break;
           }
   }
   else {
      t->runNextEvent();
   }
}
```

Based on the synchronization algorithm, the desired behavior of a synchronized simulation can be achieved. As is shown in Figure 3.11, 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 cycl-level simulation, and the SUNSHINE system simulator correctly interleaves cycle-level processing with event-driven processing.

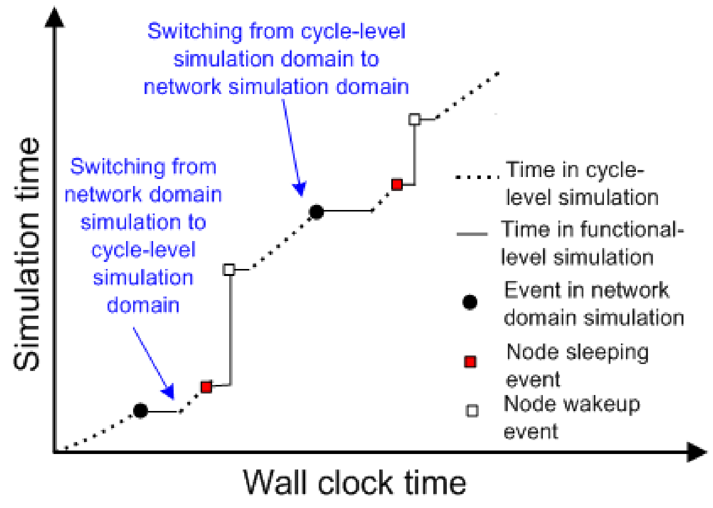
Switching from cycle-level simulation domain to network simulation domain
Switching from network domain simulation to cycle-level simulation domain
Simulation time
Wall clock time
Time in cycle-level simulation
Time in functional-level simulation
Event in network domain simulation
Node sleeping event
Node wakeup event

Figure 3.11: Synchronized simulation time in SUNSHINE

Chapter 4 Evaluation

We 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 2.1.1, SimulAVR and GEZEL version 2.5. We used the latest version of the simulators available at the time of performing the experiments. The hardware platform configured in these simulations is MICAz.

4.1 Scalability

In the following, we simulated several applications to analyze SUNSHINE's scalability. In the first application, we varied the number of nodes that are randomly distributed from 2 to 128. Nodes are paired to communicate with each other. We wrote an application to let the paired nodes send packets between each other. The simulation ends when all of the nodes receive one message from its neighbor. We considered four cases: the first case is pure co-sim nodes network, the second one is pure TOSSIM nodes network, the third is the combination of 50% co-sim nodes with 50% TOSSIM nodes network, and the fourth is 25% co-sim nodes with 75% TOSSIM nodes network.

Figure 4.1 shows SUNSHINE's wall clock time which represents the time required by SUNSHINE to complete the simulation. As expected, pure TOSSIM simulation outperforms SUNSHINE in terms of simulation speed by abstracting away the detailed behaviors of sensor nodes, such as hardware clock cycles and microprocessor's instructions. On the other hand, SUNSHINE's low execution speed comes from its fine-grained simulation accuracy. Moreover, Figure 9 shows that SUNSHINE has the ability of simulating hybrid network consist of co-sim nodes and TOSSIM nodes. When simulating the mixed network, SUNSHINE's execution speed is accelerated and hence can be suitable for even large networks.

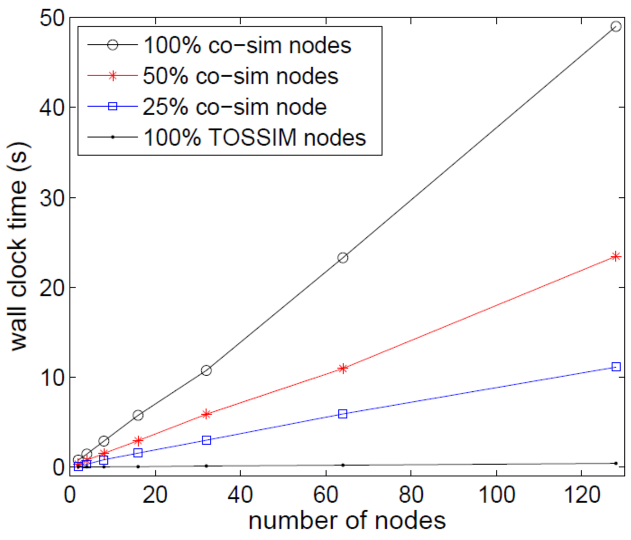

Figure 4.1: Scalability

Figure 4.2 shows the memory utilization of the simulation. The simulation with 100% co-sim nodes utilizes large CPU memory because cycle-level simulation needs to cache a lot of co-sim nodes' data and states from GEZEL, SimulAVR and TOSSIM. These data and states can take a large amount of memory space when simulating a large network. To reduce the memory consumption, SUNSHINE can combine TOSSIM nodes with co-sim nodes to decrease the memory utilization.

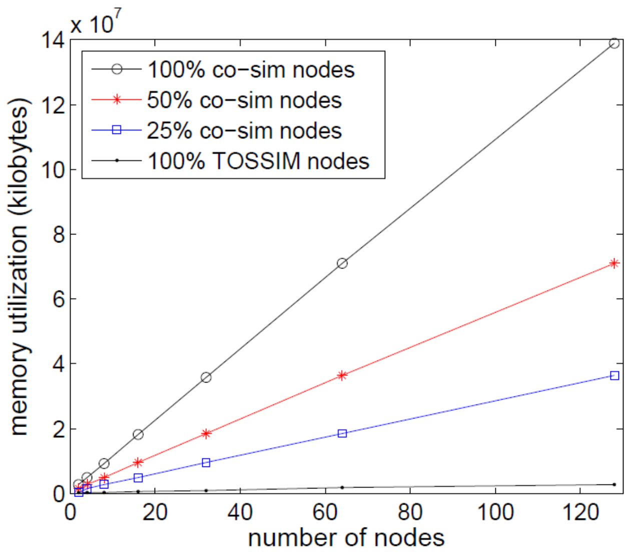

Figure 4.2: Memory utilization

Given above information, combining co-sim nodes with TOSSIM nodes becomes an advantage of both speeding up the simulator's run time and decreasing memory usage. Also, this combination is acceptable since in most network scenarios, only important nodes need to be simulated at cycle-level fine granularity (i.e. simulated as co-sim nodes) to evaluate their hardware and software performance. Other nodes, whose detailed behaviors are not important, can be simulated in TOSSIM.

4.2 Simulation Fidelity

In this section, we show SUNSHINE's simulation fidelity by simulating a computational-intensive sensor node application. On the testbed as shown in Figure 4.3, we ran the TinyOS TxThroughput application. In this application, the first sensor node keeps sending packets to the radio channel using the largest message payload size. The other node runs the Reception application, listening to the channel and receiving packets from the channel.

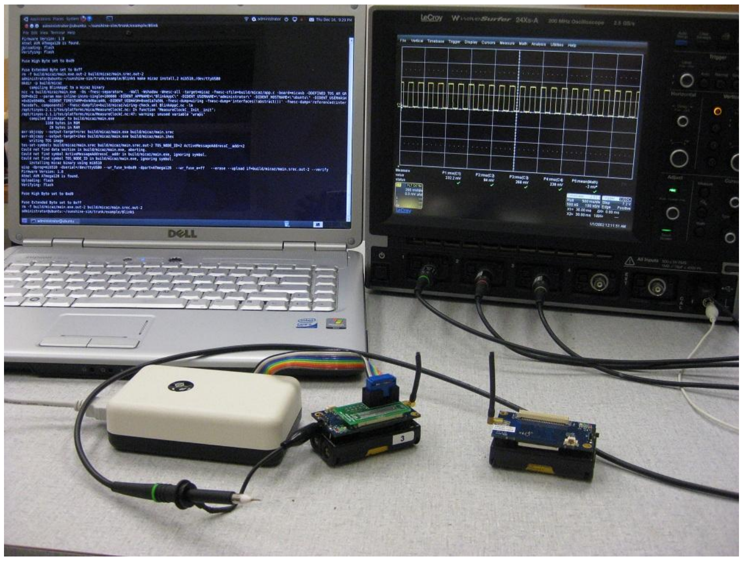

Figure 4.3: Testbed

The first sensor node executes a dummy computational task of multiple empty loops before sending packets to the other node, and we varied the number of empty loops to represent various levels of computation intensity. We compared SUNSHINE, TOSSIM, and the real mote in terms of the task execution time in simulation/experiment, and the results are shown in Figure 4.4.

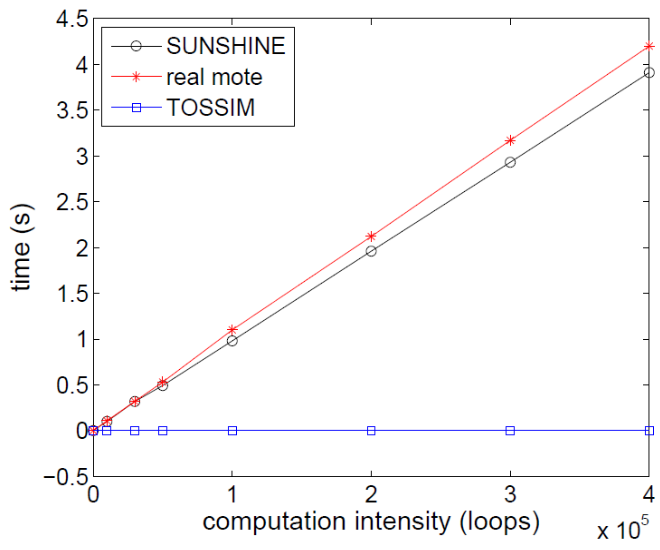

Figure 4.4: Validation result

From the results, we observe that (1) TOSSIM runs fastest as expected, and its predicted task execution time is much less than the real task execution time; and (2) SUNSHINE is able to provide a simulated task execution time that coincides with that of the real mote experiment. TOSSIM's fast simulation speed is attributed to its inability of capturing the task execution time on the microcontroller, which will apparently limit its applicability for time-sensitive applications/protocols. Many security protocols, such as the distance bounding protocol [16], require precise time-out behavior to thwart physical man-in-the-middle attacks. When testing and verifying these protocols, SUNSHINE will out-compete TOSSIM since SUNSHINE is able to correctly capture the impact of computation intensity on sensornet performance. This advantage is gained by the integration of simulators from different domains, as discussed in previous sections.

Chapter 5 Conclusion and Future Work

In this thesis, we presented one of SUNSHINE's important integration: the integration of TOSSIM and P-Sim (SimulAVR and GEZEL). We discussed solutions to several challenges around the integration: time conversion, data conversion, and time synchronization. The strength gained from the integration of simulators from different domains is also shown in the experiment results.

For future steps, we can consider a few points to improve. One is how to extend SUNSHINE to other radio platforms. Currently SUNSHINE uses CC2420 radio chip as the target radio platform. How to make SUNSHINE flexible enough to be able to easily simulate different radio chips can be a big enhancement. The other one is how to make SUNSHINE more user-friendly. Providing some additional UI features can be another improvement.

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, “Avrora: 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] P. Levis and N. Lee, TOSSIM: A simulator for TinyOS Networks. http://www.cs.berkeley.edu/pal/pubs/nido.pdf.

[8] 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.

[9] A. Hodjat, L. Batina, D. Hwang, and I. Verbauwhede, “A hyperelliptic curve cryto coprocessor for an 8051 microcontroller,” in In IEEE Workshop on Signal Processing Systems (SIPS), Nov. 2005.

[10] 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.

[11] M. Knezzevic, 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.

[12] S. Ahuja, D. A. Mathaikutty, and S. Shukla, “Model-based power estimation using least squares regression on fsmd models,” tech. rep., 2008.

[13] 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.

[14] 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.

[15] 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B). http://focus.ti.com/docs/prod/folders/print/cc2420.html.

[16] S. Capkun and J. P. Hubaux, “Secure positioning in wireless networks,” IEEE Journal of Selected Areas in Communications, vol. 24, Feb. 2006.