ABSTRACT

Title of dissertation: ENERGY-DRIVEN OPTIMIZATION OF HARDWARE AND SOFTWARE FOR DISTRIBUTED EMBEDDED SYSTEMS

Chung-Ching Shen, Doctor of Philosophy, 2008

Dissertation directed by: Professor Shuvra S. Bhattacharyya
Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies

Distributed embedded computing systems are special-purpose computer systems designed for particular applications and set up in a networked or distributed manner. A practical application domain for such a distributed system setup is the domain of wireless sensor network (WSN) applications. In this thesis, studies of architectures, applications, and methodologies for distributed embedded systems will be covered by addressing a number of important energy and performance optimization problems for translating high-level representations of distributed embedded applications into system implementations. This thesis is also concerned about systematic design methodologies and optimization problems for both software and hardware implementations.

With advances in integrated circuit technology, distributed embedded platforms such as wireless sensor nodes can be equipped with increasing amounts of computational resources, such as digital signal processing (DSP) subsystems that can handle intensive

computational tasks. By incorporating such dedicated DSP processing, a distributed embedded platform can enhance its functional capabilities for processing data before transmitting the data to other parts of the network or to an associated base station (central node). Therefore, this thesis presents a design methodology for distributing DSP applications across wireless sensor networks and optimizing associate trade-offs between computation and communication.

A low-power, application-specific sensor node platform for distributed WSN systems is designed and demonstrated in this thesis. This platform provides mixed-signal integration of streamlined digital circuits for protocol control and data processing, along with required analog subsystems, such as transceiver circuitry. Building on this optimized platform, this thesis demonstrates a complete system design of an application-specific WSN system with compact size and low power features. This system design is the result of an integrated effort across design space exploration, algorithm development, cross-layer protocol design, and most importantly, the completion of various hardware prototype implementations for validating and demonstrating proposed design techniques.

This thesis also presents a system-level synthesis methodology for finding the most suitable resource configurations for distributed, embedded systems. System-level synthesis is attractive because the carefully designed system-level models can be analyzed and evaluated rapidly, and the complex, inter-related design decisions can be explored and evaluated at a high level before mapping into low level implementations. We demonstrate the accuracy and efficiency of our system-level synthesis approach, and its ability to capture an important range of high level interactions that are relevant to the design of distributed, embedded systems.

ENERGY-DRIVEN OPTIMIZATION OF HARDWARE AND SOFTWARE FOR DISTRIBUTED EMBEDDED SYSTEMS

by

Chung-Ching Shen

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2008

Advisory Committee

Professor Shuvra S. Bhattacharyya, Chair/Advisor
Professor Neil Goldsman
Professor Gang Qu
Professor Marty Peckerar
Professor Amitabh Varshney, Dean's Representative

DEDICATION

to my parents, brother, and to Pei-Tzu Tsai

ACKNOWLEDGMENTS[1]

It is a pleasure to thank my advisor and the members of the committee, Dr. Shuvra Bhattacharyya, Dr. Neil Goldsman, Dr. Marty Peckerar, Dr. Gang Qu, and Dr. Amitabh Varshney for guiding me to complete this thesis through the process with helpful discussions, constructive comments, and providing resources. I would like to express my deep gratitude to my advisor, Dr. Bhattacharyya, who has been encouraging, inspiring and very patient all along. His enthusiasm and professional insights into research have always motivated me during the study of this work, and without his support it is difficult to make this thesis possible.

Special thanks to Dr. Goldsman and Dr. Peckerar for giving me the chance to use their lab and providing me their feedbacks to work on the collaborative Maryland smart dust project. My gratitude also goes to their students, including Bo, Yiming, Zeynep, Ben, Amrit, Thomas, Datta, and Sanaz, who spent valuable time discussing with me and oriented me to the equipment.

I want to thank Dr. Roni and Dr. Plishker for working with me and reviewing some of my papers. Our collaborations and discussions have strengthened this thesis and enriched my experience here.

1. The research reported on in this thesis was supported in part by the Laboratory for Physical Sciences and the Advanced Sensors Collaborative Technology Alliance.

It has been a pleasure to work and interact with many students in the DSPCAD research group, including Mingyung, Vida, Dong-Ik, Mainak, Sankalita, Chia-Jui, Perttu, Sebastian, Celine, Ivan, Ruirui, Hojin, Nimish, Mary, Hsing-Huang, George, and Soujanya. I feel grateful to all my friends here for their generous friendship, which supported me through the process of completing this thesis.

I would like to thank my friends, Aydin, Ankush, Eric, Charles, and all others that I fail to mention, who shared their research experience and guided me to use the design tools and equipment corresponding to the experiments in this thesis.

I would like to give my deepest gratitude to my parents for their love, support, and patience during these years while I focused on studies and my brother for encouragement and belief.

Last, but not the least, thanks to my beloved wife, Pei-tzu, who accompanies me from the beginning to the end of this thesis with full support.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1. Introduction

Embedded computing systems are special-purpose computer systems designed for particular applications to perform dedicated tasks. As technology advances, such specialized systems are networked or distributed and can be applied to broad application domains. In a distributed setup, computational tasks are carried out cooperatively by a collection of embedded processors that is distributed over some region in space and interconnected as a network by communication components in a wired or wireless fashion. A practical application domain for such a distributed system setup is the domain of *wireless sensor network* (*WSN*) applications [3]. A wireless sensor network (WSN) is a wireless networked system consisting of spatially distributed sensor platforms (also called *sensor nodes*). Typically, a sensor platform is equipped with various devices to capture, process, transmit, and receive data. These devices include sensors for capturing data; a radio transceiver for executing communication tasks; a microcontroller (microprocessor) for executing control and processing tasks; and an energy source, such as a battery.

The study of embedded system design involves three key aspects — architectures, applications, and methodologies [58]. Figure 1, which is adapted from [58], illustrates different characteristics associated with these aspects. Compared to designers of general-pur-

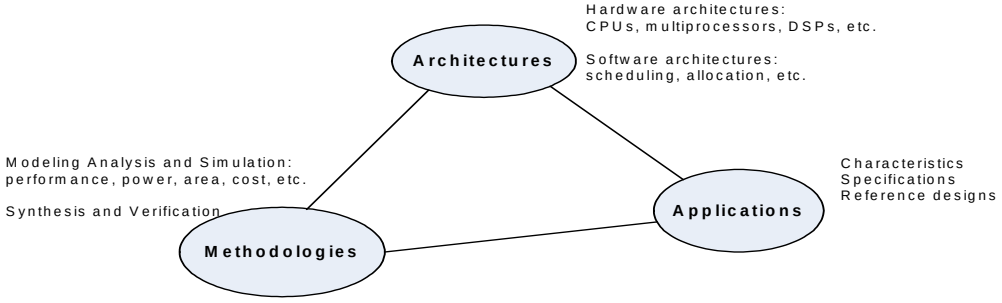


Figure 1. Aspects of embedded system design.

pose computers, embedded system designers rely much more heavily on both *methodologies* and *application expertise*. Methodology development for distributed embedded systems must be carefully considered to meet quantifiable goals, such as real-time performance, power/energy consumption, size, and cost. In many circumstances, such as in WSN systems, portable devices are densely deployed in areas that are dangerous or otherwise inaccessible to humans. Considerations of energy and power consumption are particularly important for such systems since the energy sources for the associated devices are mainly in the form of batteries. Moreover, since these systems are dedicated to specific tasks, designers can have more flexibility to optimize their design, including reduction of power/energy consumption, size, and cost, and enhancement of performance, reliability, and reconfigurability.

The design space of a WSN system is vast [60]. Varying design considerations for a WSN system among protocols, software, and hardware can directly affect energy consumption of the targeted sensor platforms in major ways. For example, the power consumption of a transceiver usually dominates the overall power consumption on a sensor node. Therefore, reducing the turn-on time of a transceiver while executing application tasks intuitively improves energy consumption. Figure 1.1 compares power consumption

Vendor name product name	**MCU**	**TX (0 dBm)**	**RX**
Texas Instruments/Chipcon CC1110	15 mW	63 mW	61.2 mW
Texas Instruments/Chipcon CC2510	20.2 mW	74.1 mW	81 mW
Ember Em250	25.5 mW	72 mW	84 mW
Atmel ATmega64RZAV	14.4 mW	49.5 mW	46.5 mW
Crossbow MICAz	24 mW	52.2 mW	59.1 mW

Figure 1.1 Power consumption comparison among various off-the-shelf platforms.

among microcontrollers and radio components in various off-the-shelf platforms that are relevant for implementation of distributed embedded systems. In Figure 1.1, *MCU* denotes the current consumed by the microcontroller component when it is running at full speed, and *TX* (*RX*) denotes the current consumed by the radio component in transmit (receive) mode.

As discussed before, a distributed embedded system such as a WSN is typically designed for a specific purpose; has features of low power and small size; and has a vast underlying design space leading to a wide range of implementation trade-offs. An important emerging trend in the development of distributed embedded systems is the trend toward *system-on-chip* (*SoC*) devices for use in embedded platforms. Such devices involve integration of digital and analog processing parts into single-chip platforms, leading to significant reductions in cost, power consumption, and size. Designing wireless digital communication systems for WSN applications is a challenging task, especially for implementing digital signal processing algorithms on application-specific integrated circuits (ASICs). ASICs are required to achieve the highest levels of application-specific streamlining, and may be desirable, for example, in applications where performance and power consumption constraints greatly dominate constraints on cost, or in scenarios where very high volumes of components are produced (e.q. [57]).

Many embedded systems in the domains of digital signal processing and digital communication have cross-layer and complex system configurations, and therefore rapid development strategies are strongly desired. High-level modeling and synthesis from such high-level models is attractive because the carefully designed system-level models can be analyzed and evaluated rapidly, and complicated design decisions can be made at high lev-

els of abstraction before synthesizing to low level implementations through an optimized, automated process. Embedded platforms targeted by such an approach can include general-purpose microprocessors or microcontrollers (e.g., see [29] [40]) or fully-customized designs, such as application-specific integrated circuits (ASICs).

1.1. Contributions of this thesis

This thesis work is concerned with systematic design methodologies and optimization problems for software and hardware implementation on distributed embedded systems. We focus especially on energy-conscious design of distributed embedded systems such as wireless sensor networks. First, we examine how to use the transceivers on sensor nodes effectively so that the overall system lifetime (based on how long the sensor nodes have sufficient power to remain operational) is managed well and/or maximized. Second, the design of low power, application-specific embedded processors is explored under constraints involving throughput and circuit area. Therefore, the work developed in this thesis centers around energy-driven hardware/software codesign, and the domain of distributed embedded systems for wireless sensor networks is the primary targeted class of applications. The problems addressed in this thesis and our contributions are summarized in the following subsections.

1.1.1. Energy-driven design methodology

In a typical WSN configuration, a distributed embedded platform such as a sensor node is resource-limited (e.g., has limited memory storage, a single low performance microcontroller/microprocessor, and relatively small energy resources) due to requirements on having small area, low cost, and low power consumption. Resource-limited sen-

sor nodes are unable to handle computationally-intensive tasks due to lack of hardware and software support. Thus, sensor nodes are often equipped to carry out light-weight computations and transmit only the required processed data to a central node for more computationally-intensive tasks. In such a scenario, the set of central nodes is typically made up of one or more higher performance nodes of the WSN that have much more abundant resources for computation, energy, and communication.

With advances in integrated circuit technology, sensor nodes are gradually being equipped with increasing amounts of computational resources, such as digital signal processing (DSP) subsystems. Such extended sensor node platforms are still highly power constrained, but provide for new power/performance trade-offs that can be exploited by the application. With such advances, the microcontroller on a sensor node can be applied to perform protocol functions and control tasks, while the DSP subsystem performs or helps with more intensive computational tasks. DSP applications are often relevant to processing sensor data, and usually require intensive computation. Here, DSP refers to the digital analysis and manipulation of data streams, such as those arising from audio signals, images, video streams and digital communication waveforms. By incorporating such dedicated DSP processing, a sensor node can enhance its functional capabilities for processing data before transmitting the data to the associated base station (central node). As a consequence, there is a significantly broadened design space for different trade-offs between computation and communication functionality in the sensor node. In particular, different kinds of preprocessing at the sensor node lead to different volumes and values of data that need to be communicated from the node.

To explore this significantly enhanced design space, we present a design methodology for modeling and implementing DSP applications for WSN systems to optimize associate trade-offs between computation and communication. This methodology explores efficient modeling techniques for DSP applications, including acoustic sensing and data processing; derives formulations of *energy-driven partitioning* for distributing such applications across wireless sensor networks; and develops efficient heuristic algorithms for finding partitioning results that maximize the network lifetime.

1.1.2. System design and implementation for a distributed sensor network: line-crossing recognition application

We design and demonstrate a low-power application-specific embedded processor for distributed WSN systems. This ASIC enables mixed-signal integration of digital circuits for protocol control and digital signal processing (DSP), along with required analog subsystems, such as transceiver circuitry.

As an important step towards the design of such a mixed-signal, systems-on-chip for WSN applications, our efforts have resulted in a complete system design of an application-specific WSN system with compact size and low power features. Here, our efforts have included application exploration, algorithm development, cross-layer protocol design, and most importantly, the completion of various hardware prototype implementations for validating and demonstrating our developed design techniques.

1.1.3. System-level synthesis of configurations of application-specific embedded systems

Distributed embedded systems require complex, inter-related high-level configuration settings such as transmission power, data transmission rate, parallelism depth for data processing. Finding the an efficient set of configurations and synthesizing them onto application-specific embedded platforms is a challenging task since the associated design space is vast. In this thesis, we study two application-specific system configuration problems for distributed embedded systems. Based on our formulations of these problems, we develop evolutionary algorithms to derive optimized solutions, and demonstrate through extensive experiments the effectiveness of these methods.

The *globally asynchronous, locally synchronous* (GALS) [14] design style is an intermediate digital design style, between the fully-synchronous and fully-asynchronous approaches (e.q., see [28], [48], [49], [73]). In this thesis, a preliminary exploration of GALS-based design is discussed for low-power, application-specific processors in distributed WSN systems. We propose a novel synthesis framework as our future work for high-level design and optimization of embedded SoCs based on GALS in which the framework incorporates dataflow-based modeling, design analysis, and optimization techniques for such an automatic, DSP-oriented synthesis process. More specifically, our approach in developing a GALS-based design methodology involves formulating and analyzing functional dataflow graph specifications for DSP applications, understanding problems that may affect hardware that is derived from these dataflow graphs, and developing efficient design methods and algorithms to overcome the problems. The overall goal of this effort is

to help improve the efficiency and degree of automation with which GALS-based design can be applied to application-specific embedded system implementation.

1.2. Outline of the thesis

An outline of this thesis is as follows. Chapter 2 presents background on the data-flow model of computation for DSP applications, and introduces associated opportunities and problems associated with energy-driven software and hardware optimization. Chapter 3 introduces an energy-driven design and synthesis methodology for systematically (re)distributing DSP applications across a distributed network environment. Chapter 4 demonstrates a case study for applying this energy-driven methodology to a distributed embedded application for speech recognition. Chapter 5 presents a complete system design and analysis — including algorithm streamlining, communication protocol configuration, hardware/software implementation, and lifetime modeling — for a compact and low power, distributed, sensor network system. Chapter 6 demonstrates a case study for implementing a distributed line-crossing recognition system on different platforms. Chapter 7 introduces two system-level synthesis methods for generating efficient configurations for application-specific embedded systems. Conclusions and summaries of the research topics studied in this thesis are drawn in Chapter 8.

Chapter 2. Background

2.1. Dataflow Models of Computation

In embedded computing, models of computation help us understand how to correctly and efficiently design complex systems from a high-level point of view. This section considers a number of models of computation that are useful for embedded signal processing. Basic concepts pertaining to these models are discussed, as well as relationships among the different models. The study of models of computation has significantly influenced the way real embedded systems are designed (e.g., see [23]). In our work, we apply dataflow models of computation, which are well-suited for the design and implementation of DSP systems.

Dataflow modeling techniques underlie many popular graphical tools for DSP system design (e.g., see [10]). When dataflow is used to model applications, the applications are modeled as directed graphs called *dataflow graphs*. A dataflow graph is a directed graph in which each node (*actor*) represents a computational module for executing (*firing*) a given task, and each directed edge represents a first-in-first-out (FIFO) buffer for holding data values (*tokens*) and imposing data dependencies. Actors can be executed only if sufficient numbers of tokens are available on their input edges. Whenever an actor fires, it consumes certain numbers of tokens from its input edges, executes its associated computational tasks, and produces certain numbers of tokens on its output edges. Also, a non-negative-integer *delay* is associated with each dataflow edge. Each unit of delay corresponds to an initial data value that is stored in the associated buffer.

2.1.1. Synchronous Dataflow

Synchronous dataflow (SDF) [41] is a restricted form of dataflow that is useful for an important class of applications in which certain behavioral properties are know a priori. SDF is employed in a variety of commercial design tools (e.g., see [4], [23], [56]) due to the powerful optimizations and performance guarantees that SDF's restricted dataflow model allows. In SDF, the number of data values (tokens) produced and consumed by each graph vertex (actor) is fixed and known at compile time. As a result of this restriction, graphs can be scheduled statically based on the so-called *repetition vector* q, which is a vector that is indexed by the actors in the graph, and gives the number of times that each actor needs to be invoked in a static (periodic) schedule for the graph. The repetitions vector can be obtained through the *topology matrix* of G — shown as Γ in Eq. 2.1 — as well as the balance equations for G — shown as the right-side equality in Eq. 2.1 — in matrix-vector form [41]:

$$\Gamma(e, v) = \begin{cases} prd(e), \text{ if } v = src(e) \\ -cns(e), \text{ if } v = snk(s) \\ \quad 0, \text{ otherwise} \end{cases} \text{ and } \Gamma \bullet q = 0. \qquad (2.1)$$

In Eq. 2.1, $prd(e)$ is the number of tokens produced onto edge e by each execution of $src(e)$, which denotes the source actor of e. Similarly, $cns(e)$ is the number of tokens consumed from e by each execution of $snk(e)$, which is the sink actor of e.

When modeling signal processing applications using dataflow, it is often important to analyze the memory requirements associated with the FIFO buffers for the dataflow edges. The static schedule of a properly-constructed SDF graph (called a *consistent* SDF graph) can be repeated indefinitely with bounded memory requirements to process the

kinds of indefinite-length data streams that are characteristic in the signal processing domain.

2.1.2. Parameterized Dataflow

DSP applications that involve purely static dataflow behavior are modeled using SDF. However, modern WSN applications do not always conform to the restricted semantics of SDF. Instead, they may require more general semantics to model application-specific sensing activities. For example, in some applications, the analog data input must be controlled so that the following digital processing sections are executed only when needed. Therefore, in this thesis, we also use the more general parameterized synchronous dataflow (PSDF) model of computation. PSDF results from the integration of SDF with the meta-modeling framework of parameterized dataflow [8]. PSDF expresses a wide range of dynamic dataflow behaviors, while preserving much of the useful analysis and synthesis capability of SDF [4]. Furthermore, PSDF provides a systematic framework for integrating various forms of SDF analysis techniques into a more general, dynamic dataflow setting.

Each hierarchical subsystem in a PSDF specification consists of three distinct graphs: the init graph (Φ_i), subinit graph (Φ_s), and body graph (Φ_b). In our use of PSDF, we employ the body graph to model the main functional behavior of an application and the init and subinit graphs to describe any input-dependent changes to the body graph functionality. According to PSDF semantics, the init graph is invoked prior to each invocation of the associated (hierarchical) parent subsystem, while the subinit graph is invoked prior to each invocation of the associated body subsystem. Such interaction allows for two distinct levels of reconfiguration control.

In this thesis, we choose PSDF for modeling WSN-oriented DSP applications because the PSDF body graph concept is well-suited to modeling the main functional behavior of such applications, while the init and subinit graphs are relevant to describing application input behavior along with the interactions between input streams and the control of core processing functionality. For example, for WSN applications, sensor nodes are often required to sense analog signals through various kinds of sensors, such as acoustic, temperature, or image sensors. By applying these sensed signals to analog-to-digital converters (ADCs), digitized data is generated for further processing. In some applications, the continuous data input must be monitored carefully so that the subsequent digital processing sections are executed only when necessary.

2.1.3. Dataflow Interchange Format

The Dataflow Interchange Format (DIF) [32] is a standard language for specifying mixed-grain dataflow models for digital signal processing (DSP) systems. It is used as a unified textual language for expressing different kinds of dataflow semantics. The associated DIF package is a software package that provides representations and utilities for analyzing dataflow graphs that are captured using DIF. The overall DIF-based design flow is shown in Figure 2.1.

In this thesis, DIF is also used as an input interfacing tool for experimenting with the developed methodology and framework for analyzing dataflow-based models of applications. A simple example described using DIF is shown in Figure 2.2, where a 2-channel CD (compact disc) to DAT (digital audio tape) sample rate conversion application is presented. In DIF specifications, each dataflow actor is allowed to have a set of attributes. This feature provides a convenient way to assign platform-specific properties of an actor

when a specific hardware platform is assumed to be used for implementing such an actor. For example, an FIR actor can be implemented on different platforms, such as a programmable DSP or 8051 microprocessor, and its associated power consumption and execution cycle estimates can be set as corresponding properties of the actor at design time when a specific platform is being targeted. DIF graphs and actor attributes will be discussed and utilized further in the parts of this thesis that pertain to energy-driven design methodology, and system-level synthesis (Chapter 3 and Chapter 7, respectively).

2.2. Design Space Exploration using Evolutionary Algorithms

Evolutionary algorithms (EA) form a family of search methods belonging to the field of evolutionary computing [5], which is inspired by biological evolution concepts such as inheritance, crossover, mutation, and selection. Evolutionary algorithms are often

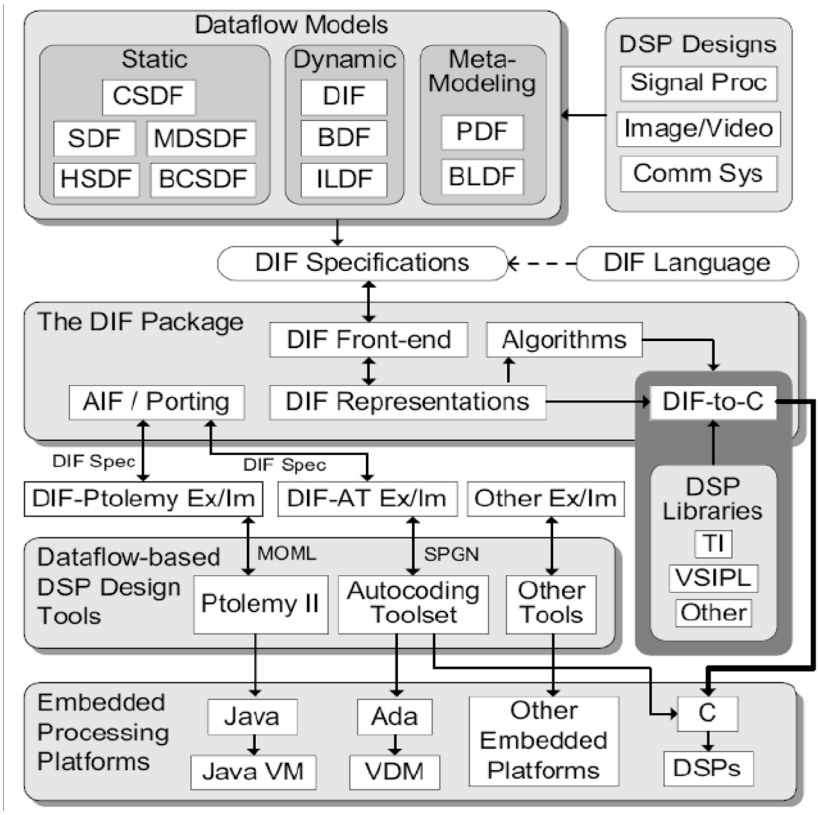
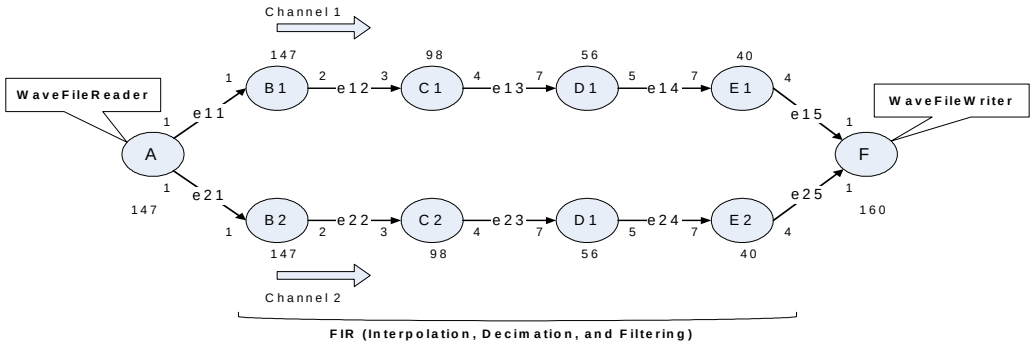


Figure 2.1 DIF-based design flow [31].




```
sdf cd2dat{
        topology {
                nodes = A, B1, C1, D1, E1, B2, C2, D2, E2, F;
                edges = e11 (A, B1), e12 (B1, C1), e13 (C1, D1),
                        e14 (D1, E1), e15 (E1, F),  e21 (A, B2),
                        e22 (B2, C2), e23 (C2, D2), e24 (D2, E2),
                        e25 (E2, F);
        }

        production {
                e11 = 1;
                e12 = 2;
                e13 = 4;
                e14 = 5;
                e15 = 4;
                e21 = 1;
                e22 = 2;
                e23 = 4;
                e24 = 5;
                e25 = 4;
        }

        consumption {
                e11 = 1;
                e12 = 3;
                e13 = 7;
                e14 = 7;
                e15 = 1;
                e21 = 1;
                e22 = 3;
                e23 = 7;
                e24 = 7;
                e25 = 1;
        }

        attribute datatype {
                e11 = "float";
                e12 = "float";
                e13 = "float";
                e14 = "float";
                e15 = "float";
                e21 = "float";
                e22 = "float";
                e23 = "float";
                e24 = "float";
                e25 = "float";
        }

        actor A {
                computation = "WaveFileReader";
                signalOut1 = e11;
                signalOut2 = e21;
        }

        actor B1 {
                computation = "CDtoDATFIR";
                signalIn = e11;
                signalOut = e12;
                interpolation = 2;
                decimation = 1;
                decimationPhase = 0;
                coefSel = 1;
        }

        actor B2 {
                computation = "CDtoDATFIR";
                signalIn = e21;
                signalOut = e22;
                interpolation = 2;
                decimation = 1;
                decimationPhase = 0;
                coefSel = 1;
        }

        actor C1 {
                computation = "CDtoDATFIR";
                signalIn = e12;
                signalOut = e13;
                interpolation = 4;
                decimation = 3;
                decimationPhase = 0;
                coefSel = 2;
        }

        actor C2 {
                computation = "CDtoDATFIR";
                signalIn = e22;
                signalOut = e23;
                interpolation = 4;
                decimation = 3;
                decimationPhase = 0;
                coefSel = 2;
        }

        actor D1 {
                computation = "CDtoDATFIR";
                signalIn = e13;
                signalOut = e14;
                interpolation = 5;
                decimation = 7;
                decimationPhase = 0;
                coefSel = 3;
        }

        actor D2 {
                computation = "CDtoDATFIR";
                signalIn = e23;
                signalOut = e24;
                interpolation = 5;
                decimation = 7;
                decimationPhase = 0;
                coefSel = 3;
        }

        actor E1 {
                computation = "CDtoDATFIR";
                signalIn = e14;
                signalOut = e15;
                interpolation = 4;
                decimation = 7;
                decimationPhase = 0;
                coefSel = 2;
        }

        actor E2 {
                computation = "CDtoDATFIR";
                signalIn = e24;
                signalOut = e25;
                interpolation = 4;
                decimation = 7;
                decimationPhase = 0;
                coefSel = 2;
        }

        actor F {
                computation = "WaveFileWriter";
                signalIn1 = e15;
                signalIn2 = e25;
        }
}
```

Figure 2.2 An application example of 2-channel CD to DAT described in DIF format.

used to solve optimization problems in which the design spaces quickly become intractable and difficult to understand. In an EA, candidate solutions are represented by abstract *individuals* (candidate solutions), which consist of *genes*. Each gene of such a genotype affects some feature of the solution. After creating the initial population, new generations are created by crossing over [5] the selected "parents," (candidate solutions from which new candidate are derived) and the process is iterated, usually until a pre-specified maximum number of generations is reached. Each iteration of an EA involves a competitive selection process that is carried out by evaluating and comparing "fitness values" for the individuals. Then solutions with "good" fitness value are selected and randomly mutated to generate new candidate solutions for the next iteration.

In this thesis, different synthesis problems and related application-specific platform optimization problems will be formalized and discussed in Chapter 7. Both single-objective (e.g. [34]) and multi-objective (e.g. [82][83]) evolutionary algorithms will be used to explore the associated design spaces. Our methods here for synthesis and optimization are to geared towards finding appropriate abstract, application-specific configurations on targeted embedded platforms before further refinement to lower-level evaluation or implementation.

Chapter 3. Energy-driven Distribution of Signal Processing Applications across Wireless Sensor Networks

Wireless sensor network (WSN) applications have been studied extensively in recent years. Such applications involve resource-limited embedded sensor nodes that have small size and low power requirements. Based on the need for extended network lifetimes in WSNs in terms of energy use, the energy efficiency of computation and communication operations in the sensor nodes becomes critical.

Digital signal processing (DSP) applications typically require intensive data processing operations and as a result are difficult to implement directly in resource-limited WSNs. The operational complexity of each sensor node and the amount of data to be transmitted across sensor nodes strongly influence the energy consumption of the nodes, which ultimately determines the network lifetime. In this thesis, we present a novel design methodology for modeling and implementing high-level DSP applications applied to wireless sensor networks. This methodology explores efficient modeling techniques for DSP applications, including data sensing and processing; derives formulations of energy-driven partitioning (EDP) for distributing such applications across wireless sensor networks; and develops efficient heuristic algorithms for finding partitioning results that maximize the network lifetime. To address such an energy-driven partitioning problem, this thesis provides a new way of reducing data traffic across nodes. By considering low

data token delivery points and the distribution of computation in the application, our approach finds energy-efficient trade-offs between data communication and computation.

3.1. Introduction

A wireless sensor network (WSN) system is composed of a collection of sensor nodes, where each node contains components for sensing, data processing, and communication. Traditionally, sensor nodes are limited in size, power, and memory, and perform relatively light-weight computational tasks. Also, sensor nodes are often deployed very densely, or in remote, inaccessible, or dangerous areas, which makes replacement of their batteries costly or infeasible. Therefore, WSN systems are typically designed with low power consumption, and energy-constrained lifetime maximization as primary objectives. Energy-constrained WSN applications include habitat monitoring, environmental observation, and battlefield surveillance (e.g., see [3], [40]).

In a sensor network, as we increase the number of nodes, requirements on network lifetime, and volume of data traffic across the network, it is often efficient to move towards hierarchical network architectures (e.g., see [39]). In a hierarchical WSN, sensor nodes are clustered into groups, and the nodes within each clustered group are divided into the “master” (e.g., the cluster head) and “slave” nodes for more efficient structuring of network traffic. The master node is typically fully-featured — that is, the platform is equipped with a relatively high-performance processor and transceiver, larger size memory storage, and sizable energy source. Conversely, slave nodes are lean in terms of features — the platforms are equipped with simple processors (e.g., small microcontrollers), simple transceivers, sensors, limited memory storage, and relatively small energy

resources. Thus, slave nodes are typically equipped to carry out simple computations and transmit only the required processed data to the associated master nodes for more computationally-intensive tasks.

In this thesis, we target this kind of hierarchical WSN organization. We assume that after a self-organizing, cluster/network startup period, every slave node is able to communicate directly with its associated master node, and the size of each clustered group is in the range of 2 to 10 nodes. Here, we define the network size as the number of active nodes in a system. This type of network structure is similar to so-called infrastructure-based networks [60].

We assume that a fixed amount of processing must be performed within each cluster so that minimal information needs to be communicated globally (across clusters) within the overall network, and our objective is to maximize the energy efficiency of this predefined amount of cluster-level processing. Such static configurations of network structure have simple routing characteristics, and protocol demands, and provide useful opportunities to optimize the network lifetime in terms of energy consumption, as well as network scalability through hierarchical organization.

Digital signal processing (DSP) algorithms are widely used in the processing of sensor data, and often require intensive computation. The behavior of many DSP applications can be characterized by regular patterns of computation and modeled efficiently through dataflow graphs. By analyzing a well-designed dataflow model of an application, operational efficiency can be estimated and optimized accordingly, such as with the use of dataflow-based algorithms for scheduling, memory management, consistency analysis among different sample rates, and hardware/software partitioning (e.g., see [10], [33], and [41]).

In a typical WSN configuration, resource-limited slave nodes are unable to handle computationally intensive tasks due to lack of hardware and software support. In such a case, microcontrollers in slave nodes can perform relatively light-weight computations. Conventionally, to minimize their computational requirements, slave nodes transmit all sensed data (i.e., without any pre-processing) to the master node for further processing. With advances in integrated circuit technology, slave nodes can be equipped with increasing amounts of computational resources, such as digital signal processor subsystems. For example, Calhoun et al. [13] propose a sensor node architecture that contains a DSP processor for executing signal processing programs. In such platforms, microcontrollers can perform protocol and control tasks, while the DSP processor performs more intensive computational tasks. For many applications, doing some processing on the slave nodes reduces the amount of data that must be communicated across the network.

The energy consumption of the nodes in a wireless sensor network, including the energy consumption for computation- and communication-related tasks, must be carefully optimized to increase network lifetime. In general, the communication tasks carried out by the transceiver dominate the overall power consumption on a sensor node (e.g., see [25], [72]).

To reduce the energy consumed by the transceivers on the sensor nodes, the amount of data to be communicated across the wireless channel should be minimized. When distributing DSP-oriented computations across a WSN, this consideration is significant since DSP applications usually process large amounts of data in an iterative fashion. We carefully consider this problem of workload distribution, including the costs associated with

computation and communication, so that we can maximize energy efficiency for WSN applications. We define this problem as the *energy-driven partitioning* (EDP) problem.

In this thesis, we develop a precise formulation of the EDP problem for DSP applications that are modeled as dataflow graphs, and we present an effective algorithm to address the problem. Specifically, our algorithm finds an efficient trade-off between the workload distribution of computation and communication tasks. The technique that we develop in this thesis is novel in that it analyzes the pattern of internal data exchange rates within an application to minimize the overall energy consumption of a sensor network system, while also taking into account latency and/or real-time constraints based on application requirements. The approach is especially well-suited for multirate signal processing applications, which exhibit complex and nonuniform patterns of data exchange across functional modules and subsystems.

3.2. Related Work

Many useful approaches have been suggested previously to reduce energy consumption in sensor nodes. Shih et al. [68] have distributed the fast Fourier transform (FFT) function over a master node and slave nodes to reduce energy consumption by moving the function from a cluster head node to slave nodes. In [39], energy and latency trade-offs are considered for different computational capabilities between master and slave nodes.

Many researchers have suggested a hierarchical, physical-layer driven sensor network design to reduce data traffic and energy consumption of a sensor node in connection with the physical-layer network functions (e.g., see [42], [69]). To manage network lifetime, design of distributed medium access control protocols based on integrating physical

layer parameters has also been discussed in [16]. In these approaches, node optimization needs to be performed carefully in conjunction with the underlying protocol characteristics.

Park proposes different heuristics in [53] and [54] to maximize network lifetime based on a well-designed topology and routing schemes. Wang [76] develops an approach that partitions applications between master and slave nodes, and also applies dynamic voltage scaling to further reduce power consumption. In contrast to the above approaches, our proposed energy-driven analysis and partitioning for an application graph are targeted at the application level. Moreover, the partitioning method presented in this thesis applies coarse-grain analysis of dataflow graphs, as well as integration within a dataflow-based DSP design tool. This tool, called the DIF (dataflow interchange format) package, is introduced in [32]. In the proposed methodology, we use DIF to specify the behavior of DSP applications. Moreover, we assign power and timing estimation results to the components of each dataflow graph. These estimation results are based on measurements and simulations carried out for those components on the targeted hardware platforms.

In [72], Tang and Xu develop adaptive data collection strategies to maximize the accuracy of data collected by a base station from sensor nodes under constraints on network lifetime. In contrast to Tang's approach, we consider options for pre-processing data on sensor nodes before information is transmitted to the "base station" (the master node, in our context). Therefore, our objective is to maximize network lifetime by finding the most appropriate resource distribution for data processing, while considering the computation and communication requirements and their underlying trade-offs for a given application.

Figure 3.1 presents the overall design flow associated with our proposed methodology for energy-efficient master/slave signal processing in sensor networks.

3.3. Modeling Workload Distributions

When a DSP applications is mapped into a clustered sensor network, each slave node generally captures data from its set of one or more sensors, and this captured data can then be sent to the associated master node immediately, or the data can be pre-processed to some degree within the slave node before it is sent to the master node.

When analyzing data processing functionality within a dataflow graph in terms of energy-efficient workload distribution, it is useful to consider carefully the rates at which actors produce and consume data from their incident output and input edges, respectively. For this purpose, we use a specialized form of dataflow called synchronous dataflow

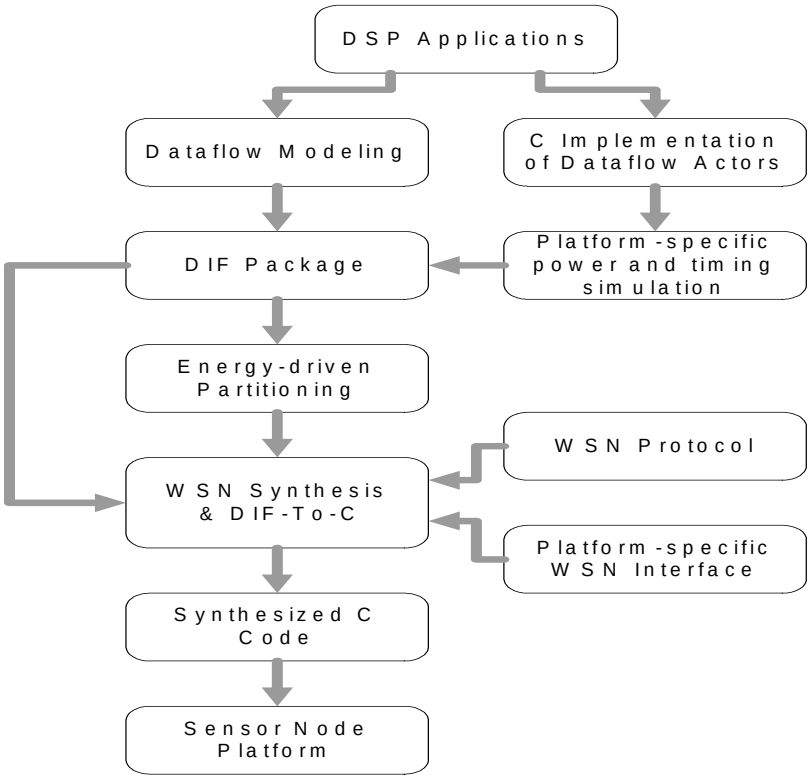


Figure 3.1 Overall design flow.

(SDF) [41], which is widely used in the design and implementation of DSP applications. Background of SDF has been introduced in Chapter 2.

3.3.1. Partitions of SDF Graphs

Given a targeted WSN system, we start by modeling its signal processing functionality as an acyclic SDF graph. A broad class of useful signal processing applications can be modeled in this way (e.g., see [10, 41]). Extensions of our techniques to arbitrary SDF topologies (i.e., those containing cycles) are discussed in Section 3.5.3.

We model workload distributions between a master node and a slave node as *feed-forward partitions* (*FFPs*) of the SDF graph that represents the overall signal processing computation to be performed. Here, by an FFP of a directed graph, we mean a decomposition of the given graph into two disjoint subgraphs G_1 and G_2 such that all edges in G that cross the partition are directed from vertices in G_1 to vertices in G_2 (i.e., the "crossing edges" are oriented in the same direction with respect to the two subgraphs). Formally, a subgraph of a directed graph $G = (V, E)$ is a directed graph $G' = (V', E')$, where $V' \subseteq V$, and E' consists of all edges in G whose source and sink vertices are both contained in V' — in other words,

$$E' = \{e \in E | (src(e) \in V') \text{ and } (snk(e) \in V')\}.$$

Thus, an FFP of G is an ordered pair (G_1, G_2) of subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that $V_1 \cap V_2 = \varnothing$, $V_1 \cup V_2 = V$, and for every edge $e \in E$ that is not contained in E_1 nor E_2, we have that $src(e) \in V_1$ and $snk(e) \in V_2$.

3.3.2. Parameterized Dataflow and Adaptive Distributions

In this thesis, DSP applications that involve purely static dataflow behavior are modeled using SDF. However, modern WSN applications do not always conform to the restricted semantics of SDF. Instead, they may require more general semantics to model application-specific sensing activities. For example, in some applications, the sensed data input must be controlled so that the following digital processing sections are executed only when needed. Thus, in this thesis we also use the more general parameterized synchronous dataflow (PSDF) model of computation. PSDF results from the integration of SDF with the meta-modeling framework of parameterized dataflow [8]. PSDF expresses a wide range of dynamic dataflow behaviors while preserving much of the useful analysis and synthesis capability of SDF [8]. Furthermore, PSDF provides a systematic framework for integrating various forms of SDF analysis techniques, such as the ones we develop in this thesis, into a more general, dynamic dataflow setting. Background of PSDF has been introduced in Chapter 2.

Another motivation that we have for using PSDF in this work — in addition to modeling dynamic behavior in signal processing functionality — is to help develop an advanced workload redistribution scheme for dynamic topology management. Based on our redistribution approach, whenever the network size changes, a the overall workload distribution result is efficiently adapted in terms of the new network structure. This is useful for scenarios in which sensor nodes can enter or exit the system at run time. Such dynamics may arise, for example, due to incremental node deployments and exhausted batteries, respectively.

3.4. Energy-driven Partitioning

In our proposed concept of EDP, an application dataflow graph (e.g., an SDF graph) $G = (V, E)$ is taken as input, and an FFP (G_s, G_m) is computed for G, where $G_m = (V_m, E_m)$ and $G_s = (V_s, E_s)$ respectively represent the distribution of the partitioned application graph onto master and slave nodes. The *partition cut* E_c associated with such an FFP is defined to be the set of edges that "cross the partition":

$$E_c = \{e \in E | (src(e) \in V_s) \text{ and } (snk(e) \in V_m)\}$$

The network size is initialized at design time, and can be changed at run time. The master node performs the function of collecting data from all other slave nodes; therefore, the communication is one-way. In order to maintain predictability and fairness across the slave nodes, we define a *scheduled iteration* (S) of a partitioned application graph as one functional iteration of the application such that all master and slave nodes complete a predefined amount of computation and transmit/receive processed data as needed based on the computation. When the application graph is an SDF graph, such an iteration in our methodology corresponds to a minimal periodic schedule for the graph.

In the particular solution demonstrated in this thesis, we define such a scheduled iteration based on the periodic behavior of the TDMA-based communication pattern in the network. Our energy consumption analysis for partitioning an application graph is targeted at the application level, and we assume that the underlying protocol design can handle problems of packet loss and collision without changing the higher level application behavior. Therefore, we formulate the system energy consumption model at a high level, based on the schedule iteration concept, and independent of implementation details for the

underlying communication protocols. We have validated the correctness and utility of this multi-level design model in our experiments.

Each actor v of the application graph is characterized by two pairs of attributes $(P_m(v),\ t_m(v))$ and $(P_s(v),\ t_s(v))$, where $P_m(v)$ $(P_s(v))$ denotes the power consumed and $t_m(v)$ $(t_s(v))$ denotes the execution time when actor v executes on the master node (on a slave node). Similarly, each edge e of the application graph is characterized by pairs of attributes $(P_m(e),\ t_m(e))$ and $(P_s(e),\ t_s(e))$ that represent the power consumption values and latencies for local (intra-node) communication on the master and slave nodes, respectively. These values are associated with the transfer of one token of data across the edge.

For modeling the energy consumption of communication, we distinguish between edges that cross the partition cut (edges in E_c), and edges that do not (edges in $(E - E_c)$). Note that, based in our application partitioning model, edges in E_c correspond to points in the dataflow graph where tokens are transmitted and received through the wireless communication channel.

3.4.1. Energy formulation on data processing and communicating

The total number of data values transferred across an SDF edge e in a minimal scheduled iteration of an SDF graph G can be expressed [41] as

$$\tau(e) = q(src(e)) \cdot prd(e) = q(snk(e)) \cdot cns(e) \tag{3.1}$$

Thus, given an FFP for G, the number of tokens X_c that crosses an FFP partition within each scheduled iteration of an SDF application graph is expressed as

$$X_c = \sum_{l \in E_c} \tau(l). \tag{3.2}$$

The computational energy consumed on each scheduled iteration by the slave and master nodes, respectively, can then be formulated as

$$E(s) = \sum_{\forall v \in V_s} q(v) \cdot P_s(v) \cdot t_s(v) + \sum_{\forall e \in E_s} 2\tau(e) \cdot P_s(e) \cdot t_s(e), \tag{3.3}$$

and

$$E(m) = \sum_{\forall v \in V_m} q(v) \cdot P_m(v) \cdot t_m(v) + \sum_{\forall e \in E_m} 2\tau(e) \cdot P_m(e) \cdot t_m(e). \tag{3.4}$$

Furthermore, the energy consumption of communication for transmitting and receiving data tokens that cross the partition cut can be expressed as

$$E(t) = \sum_{\forall l \in E_c} \tau(l) \cdot P_t \cdot t_c = X_c \cdot P_t \cdot t_c, \tag{3.5}$$

and

$$E(r) = \sum_{\forall l \in E_c} \tau(l) \cdot P_r \cdot t_c = X_c \cdot P_r \cdot t_c, \tag{3.6}$$

where P_r represents receive-mode transceiver power consumption; P_t represents transmit-mode transceiver power consumption; and t_c models the time required for inter-node communication, including transceiver turn-on time and time for executing modulation/ demodulation tasks. The value of t_c is dependent on transceiver configuration settings, such as data rate, that are associated with communication tasks.

3.4.2. Energy cost per scheduled iteration for EDP

The total energy consumption on a single node for one scheduled iteration is denoted by E_{iter}. In a network cluster that consists of a single master node and N_s slave

nodes, the master node iterates N_s times to process data frames from all of its slave nodes. Therefore, the energy consumption on each slave node for one scheduled iteration is

$$E_{iter}(s) = E(s) + E(t) + E_{idle}(s), \tag{3.7}$$

where, $E_{idle}(s)$ denotes the energy consumption when a slave node stays in the power-saving "idle" mode after processing and communicating data tokens. The total energy consumption on the master node for one scheduled iteration is formulated as

$$E_{iter}(m) = N_s \cdot (E(m) + E(r)) + E_{idle}(m). \tag{3.8}$$

Note that $E_{idle}(m)$ could be small because the workload on the master node is significantly heavy compared to the slave nodes in each scheduled iteration. Therefore, *the system energy cost per scheduled iteration*, which we refer to more concisely as *the system energy cost* is defined as

$$E_{sys} = E_{iter}(s) + E_{iter}(m). \tag{3.9}$$

To compare the network lifetime according to the selection of different FFPs, we define the worse case network lifetime as the total time that elapses before the first node in the network runs out of energy. Thus, our network lifetime estimation is formulated as

$$min\{T_{lifetime}(master), T_{lifetime}(slaves)\}, \tag{3.10}$$

where $E_{source} = T_{lifetime} \cdot E_{iter} \cdot R_{avg}$. R_{avg} is the average rate at which scheduled iterations are executed; E_{source} is the total energy stored in a given energy source (e.g., battery). Based on a given partitioning result, the network lifetime is determined in terms of the minimum lifetime between the resulting master and slave node configurations.

3.4.3. Latency and data dependency constraints

To apply our energy-aware optimization techniques in conjunction with a specific sensor network communication protocol, we derive protocol-dependent local and global latency constraints in our targeted partitioning problem. The protocol that we use to demonstrate this approach is a TDMA-based protocol, and is described in more detail in Section 3.4.4. Here, the *local* latency indicates the execution time period from the input of the master (slave) subgraph to the output of the master (slave) subgraph. This is expressed as

$$L_l = \sum_{\forall v \in V_s or V_m} q(v) \cdot t(v) + \sum_{\forall e \in E_s or E_m} 2\tau(e) \cdot t(e) + \sum_{\forall l \in E_c} \tau(l) \cdot t(l). \qquad (3.11)$$

The *global* latency, L_g, is defined as the latency of one scheduled iteration. This represents the overall computation and communication time period from the input to the output of a partitioned application graph. That is,

$$L_g = L_l(slaves) + N_s \cdot L_l(master) + N_s \cdot X_c \cdot D, \qquad (3.12)$$

where, D is the propagation delay for transmitting each data token. The term, $N_s \cdot X_c \cdot D$, is added to incorporate transmission delays in the network. In general, D is very small compared to the local latency, and therefore the whole term $N_s \cdot X_c \cdot D$ is negligible in our latency model. From Eq. 3.12, we notice that the master node should process the data from all of the slave nodes, and therefore the term $N_s \cdot L_l(master)$ is induced for presenting the total local latency that the master node requires in each scheduled iteration. However, the slave nodes can operate in parallel within each scheduled iteration, and thus, the latency required for slave node processing is independent of N_s.

We incorporate the global latency across the network into the formulation of protocol latency. For this purpose, we first define L_{sync} as the synchronization latency added

during each synchronization period. Therefore, $N_s \cdot L_{sync}$ denotes the total synchronization latency within a scheduled iteration during each synchronization period. In our pre-defined topology, the routing delay is omitted because of the one-hop distance between master and slaves nodes. In our experiments, the MAC protocol design is based on TDMA with N_p time slots of uniform time period t_p. We define $L_p = N_p \cdot t_p$ (i.e., the length of a complete TDMA time frame) as the protocol-dependent communication latency. Therefore, the latency constraint across the network for one scheduled iteration can be bounded as

$$N_s \cdot L_{sync} + L_g \leq L_p . \tag{3.13}$$

Here, both N_p and t_p are parameters determined by the TDMA configuration that is being used. Recall that the scheduled iteration is defined to maintain predictability and fairness for processing and communicating data tokens across the network. From Eq. 3.13, we observe that this property is satisfied when the TDMA-based protocol is applied. In TDMA, the nodes in a network communicates with one another at pre-defined time slots to preventing collisions when accessing the wireless channel. Therefore, there is a consistent frame-by-frame communication pattern for TDMA-based protocols. When the TDMA frame length is fixed by setting N_p and t_p, Eq. 3.13 captures the property that the latency of scheduled iteration is also constrained. That is, the operations within one scheduled iteration across the network are predictable in terms of latency when the TDMA-based protocol is applied and satisfies Eq. 3.13.

Note that if $L_{sync} \ll L_p$ (i.e., L_{sync} is small enough compared to L_p), $D \approx 0$, and there exists a partitioning cut such that $L_l(slaves) = N_s \cdot L_l(master)$, then a fully-balanced workload is achieved between the master node and the slave nodes. In such a case,

from Eq. 3.12 and Eq. 3.13, we can find the minimum requirement of t_p as $2 \cdot L_l(slaves)/N_s$ if $N_s = N_p$. Generally, $N_p \geq N_s$, and it is difficult to find a fully balanced partition in terms of given actor and edge attributes for an application graph.

To prevent cyclic dependencies (to avoid bi-directional communication complexity), a data dependency constraint is developed. That is, given an application graph, $G = (V, E)$, a partition cut E_c is *valid* if

$$\forall l \in E_c, src(l) \in G_s \Rightarrow preds(src(l)) \in G_s \wedge snk(l) \in G_m \Rightarrow succs(snk(l)) \in G_m \quad (3.14)$$

where $preds(v)$ ($succs(v)$) represents the set of immediate graph predecessors (successors) of actor v.

In summary, solving the *energy-driven partitioning problem* means to find a partition cut E_c of an application graph G so that a master-slave WSN topology having maximum lifetime results from the partitioned subgraphs G_m and G_s. That is, we wish to find a solution that satisfies

$$max\{ min[T_{lifetime}(master), T_{lifetime}(slaves)] \} \quad (3.15)$$

subject to 1) $G_m \neq \varnothing$; 2) $G_s \neq \varnothing$; 3) $N_s \cdot L_{sync} + L_g \leq L_p$; and 4) E_c is a valid partition cut, based on Eq. 3.14.

3.4.4. Customized TDMA protocol

A simple but robust TDMA-based protocol is constructed to manage network traffic. Based on this protocol, all slave nodes can receive information about the current network status from the master node. Figure 3.2 illustrates the communication pattern in the protocol between the master node and slave nodes. In this protocol design, we assume that each node in the system has a unique identifier (ID). The master node uses some pre-

defined number of time slots to establish a time frame. These time slots will be requested by slave nodes as the slave nodes join the network sequentially. We create several special packets, as shown in Figure 3.2, which are used to progressively synchronize the master node with the slave nodes that are joining the network. After synchronization, each slave node occupies a single time slot in the periodic TDMA schedule for the network cluster, and each slave node sends data to the master node at its reserved time slot.

For example, as shown in Figure 3.2, the master node broadcasts a packet initially with control information in TDMA slot 0, while all other slots are in an idle state. When the first slave node, S1, is turned on, it stays in the receiving mode until it receives a broadcasted packet from the master node. Then S1 sends a packet to the master node requesting that TDMA time slot 0 be reserved for node S1. The master node subsequently sends an acknowledgement packet to S1 confirming this reservation for slot 0. Now, time slot 0 of the master node is changed to the active state and is reserved exclusively for S1. Henceforth, node S1 will transmit data packets to the master node at time slot 0 in each TDMA time frame.

Next, the master node broadcasts a control packet at slot 1 in the next time frame. When the second node S2 joins the network, it follows the same procedure to synchronize

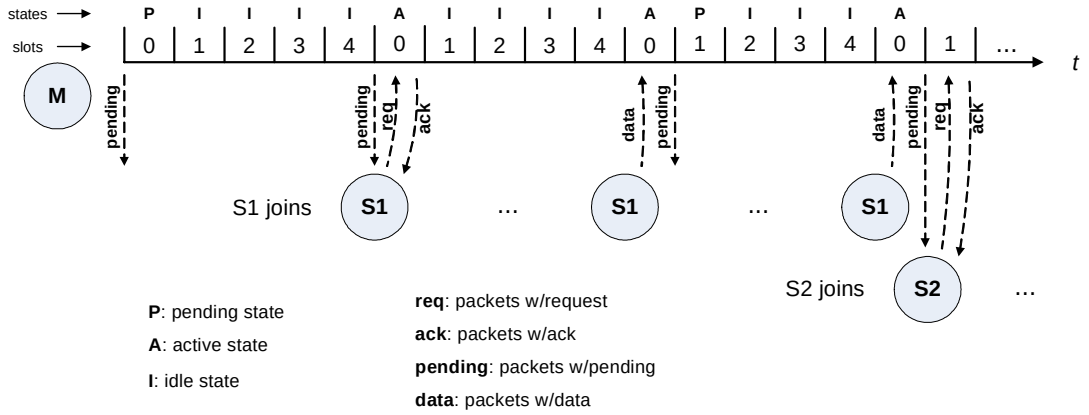


Figure 3.2 Communication pattern for TDMA-based protocol.

with the master node as described above for S1. This process of adding nodes one-by-one to the network continues until all slave nodes within communication range have been added to the network. After all nodes have been added, the network enters a data processing state in which TDMA time frames operate periodically, and in each such frame, each slave node transmits its newly collected data during its corresponding time slot. Based on this protocol design, the master node uses the utilization of time slots to track current network size.

3.4.5. Dynamic topology management

By applying the PSDF model in conjunction with the EDP approach, we provide an advanced form of workload redistribution that can be used for dynamically-changing network sizes — i.e., for scenarios in which sensor nodes can enter or exit the system at run time. Such dynamics may arise, for example, due to incremental node deployments and exhausted batteries, respectively. The number of network nodes is characterized by a dynamic parameter that represents the number of slave nodes existing in the system. This parameter is maintained and broadcasted by the master node. From the customized protocol described previously, the master node can determine the number of active nodes existing in the system based on the requests it receives from the slave nodes and the usage of time slots for its TDMA schedule. Thus, soon after any change in network size, all active slave nodes are informed about the change by the master node, and furthermore, the associated EDP configuration will generally be adjusted so that the workload distribution is efficiently adapted to the new network structure.

Based on our PSDF modeling approach, at design time, a given application graph is analyzed and the corresponding EDP configurations are determined in terms of different

network size settings. These EDP results correspond to different partition cuts on an application graph as the network size varies. The results are stored on the sensor nodes for driving workload redistribution at run time. In other words, the number of slave nodes is a parameter that is maintained on the master node, and broadcast periodically to the slave nodes. On the slave nodes, this parameter drives a *quasi-static schedule* (a dynamic schedule with a relatively large portion of the structure fixed statically, at design time) that allows the schedule to adapt efficiently to changes in network size, and associated changes in energy-driven partitions of network functionality. We refer to this integration of quasi-static scheduling with the EDP approach as *quasi-static energy-driven partitioning* (QS EDP).

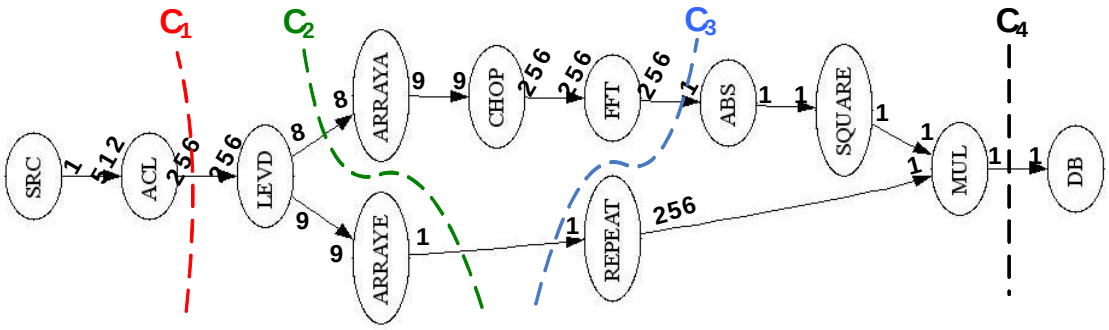
For a reasonable number of candidate EDP results (supported network sizes), many practical platforms can easily accommodate the program memory requirements for the scheduling information associated with QS EDP. For example, on the Texas Instruments CC2430F128 platform, a total 128K bytes of on-chip program memory is available, and based on our implementation, the program memory cost of QS EDP is approximately 1K bytes per unit of supported network size.

Figure 3.3 gives an example to illustrate the idea of this work redistribution scheme. Figure 3.3(a) shows an application graph with four partition cut candidates (C_1—C_4) that are considered individually. We refer these EDP results to “static EDP” results. Then, we compare these static EDP results in terms of system energy costs (i.e. Eq. 3.9) as a function of the number of slave nodes existing in the network. The workload redistribution scheme with respect to QS EDP results is illustrated by the blue solid curve in Figure 3.3(b). From Figure 3.3(b), we observe that by adding sensor nodes to a system that allows

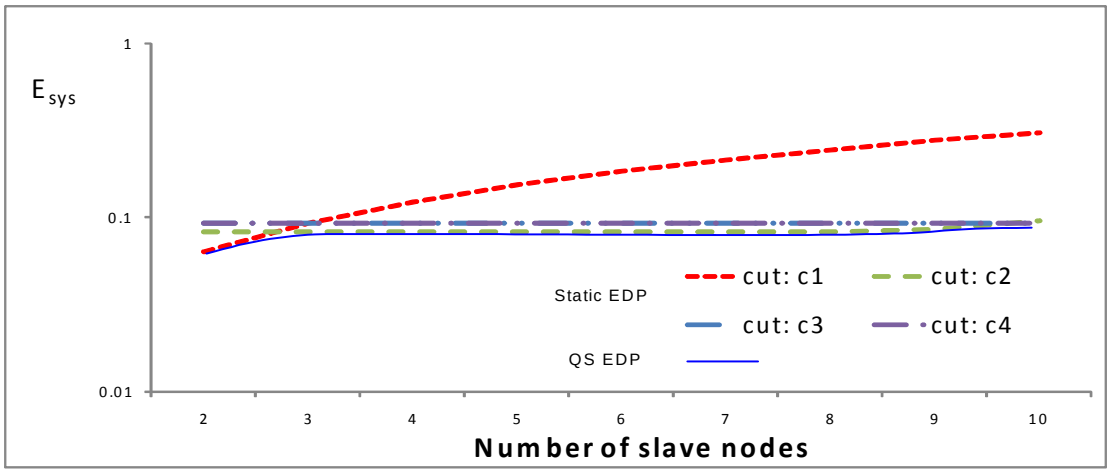
for workload redistribution, a quasic-static EDP always adapts to the minimum energy cost from its available static EDP results. This energy efficient adaptive approach results in overall system lifetime improvement in our experiments, which are be demonstrated in Sections 3.6.3 and 3.6.4.

3.5. Analysis and Solutions for the EDP Problem

The EDP problem is NP-complete, where the NP-hardness of EDP can be established with a reduction from the well-known *partition problem* [26]. Here, we consider a decision version of the EDP problem, where we are given latency and energy constraints, and the objective is to determine whether an EDP solution exists that satisfies the con-



(a)



(b)

Figure 3.3 Example of workload redistribution scheme.

straints. The energy constraint here refers to the maximum energy consumption among master and slave nodes, which directly influences the lifetime of the network.

Claim 1: The EDP problem is NP-complete.

Proof: We show in this section that the energy-driven partitioning (EDP) problem is NP-complete in the size of the input graph. We first show that the EDP problem is in NP (i.e., $EDP \in NP$). The certificate for the EDP problem consists of a slave graph G_s, a master graph G_m, and a partition cut E_c. Given pairs of power consumption and time attributes as the weights of actors and edges, we can compute $E(s) + E(t)$ and $E(m) + E(r)$ with respect to various partition cuts based on Eq. 3.3, Eq. 3.4, Eq. 3.5, and Eq. 3.6. Each such computation process can be performed in polynomial time by stepping through each node and edge; summing up the node and edge weights; and summing the cut-edge weights.

Next, we show that the partition problem is polynomially reducible to the EDP problem. The partition problem is a well-known NP-complete problem (e.g., see [17][26]). Intuitively, in the partition problem, we are given a multiset M of positive integers, and we must determine whether or not there is a way to partition M into two subsets M_1 and M_2 such that the sum of the elements in M_1 equals the sum of the elements in M_2.

To show that the partition problem is polynomially reducible to EDP, we first suppose that we are given an instance I of the partition problem. From I, we can derive a corresponding instance I' of EDP by constructing a *homogeneous* SDF (HSDF) graph with a single source actor v_{src}, and a single sink actor v_{snk}, and in which every item in I is instantiated as a corresponding actor in I'. An HSDF graph is an SDF graph in which $prd(e) = src(e) = 1$ for every edge e, and hence, $q(A) = 1$ for all every actor A [41].

Here, both v_{src} and v_{snk} have 0-valued weights (for both power consumption and time). The construction continues by having each "corresponding actor" in I' (i.e., each actor in I' that corresponds to an element of I) connected with an edge *from* the source actor and with another edge *to* the sink actor. The weight of each actors in I' is assigned as

$$(power, time) = (x, 1), \tag{3.16}$$

where x is the positive integer value of the corresponding element in I. The weight of each edge in I' is assigned as

$$(power, time) = (0, 0). \tag{3.17}$$

The latency constraint associated with the derived EDP instance I' is taken to be infinite (equivalently, the latency constraint can be taken to be so large compared to the actor execution time weights that it will always be satisfied), and the energy constraint is taken to be

$$E_{constraint} = \frac{S}{2}, \tag{3.18}$$

where S is the sum of all elements of I. By our construction, S can also be expressed as

$$S = \sum_{v \in V} p(v), \tag{3.19}$$

where V is the set of all actors in I', and $p(v)$ represents the power consumption weight associated with actor v in I'.

Figure 3.4 illustrates this graph construction process with an instance I of the partition problem and the corresponding instance I' of the EDP problem. Now for a graph $G = (V, E)$ with zero-valued edge weights, no constraint on latency, and energy constraint $E_{constraint}$ as defined in Eq. 3.18, our decision version of EDP involves finding a

FFP $G_s = (V_s, E_s)$ and $G_m = (V_m, E_m)$ such that the maximum master/slave energy consumption associated with the partition is less than or equal to $E_{constraint}$. By construction (since each actor execution requires unit time, and edges have zero cost), the energy consumption associated with an FFP $((V_s, E_s), (V_m, E_m))$ for I' can be expressed as

$$max\left\{\left(\sum_{v \in V_s} p(v)\right), \left(\sum_{v \in V_m} p(v)\right)\right\}. \tag{3.20}$$

From Eq. 3.18, Eq. 3.19, and Eq. 3.20, it follows that an FFP with the given energy constraint (i.e., a solution that demonstrates feasibility of I') exists if and only if the vertices can be partitioned into two subsets V_1 and V_2 such that

$$\sum_{v \in V_1} p(v) = \sum_{v \in V_2} p(v) = E_{constraint} = \frac{S}{2}. \tag{3.21}$$

This would in turn mean that

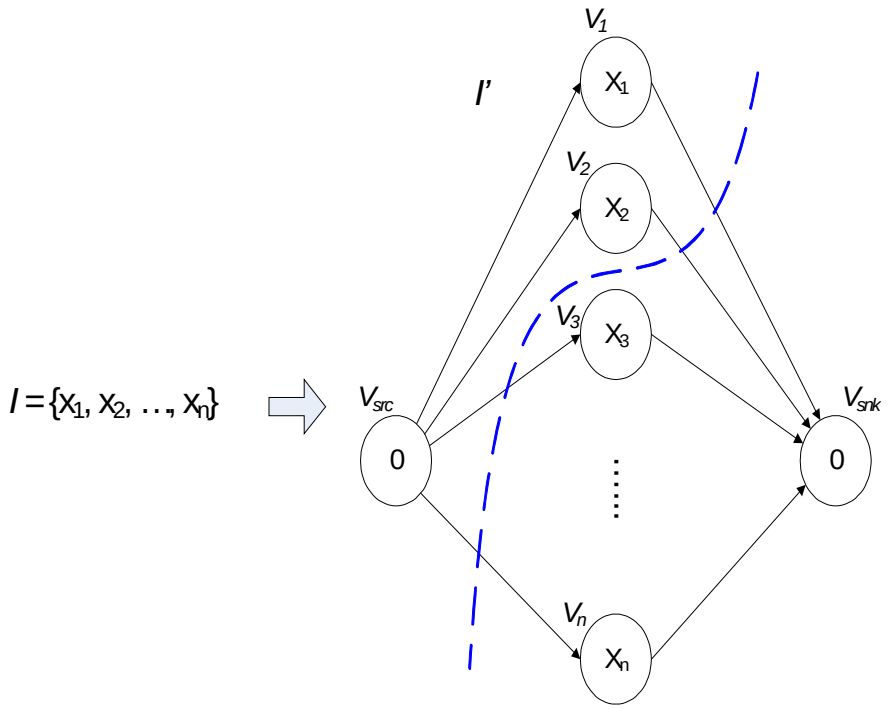


Figure 3.4 .An illustration of an EDP instance that is derived from an instance of the partition problem.

$$\sum_{v \in V_1} I(v) = \sum_{v \in V_2} I(v) = \frac{S}{2} \tag{3.22}$$

which demonstrates the feasibility of I.

In summary, we have shown that given an instance I of the partition problem, we can derive a corresponding instance I' of the EDP problem such that I is feasible (has a solution) if and only if I' is feasible.

In the example of Figure 3.4,

$$V_s = \{v_{src}, v_1, v_2\},\ V_m = \{v_3, \ldots, v_n, v_{snk}\},$$

$$E_s = \{e_{s1}, e_{s2}\},\ E_m = \{e_{3s}, \ldots, e_{ns}\},$$

$$\text{and } E_c = E - E_s - E_m = \{e_{1s}, e_{2s}, e_{s3}, \ldots, e_{sn}\}. \tag{3.23}$$

Furthermore, $\sum_{i=1}^{2} x_i = \sum_{j=3}^{n} x_j = E_{constraint}$, and the weight of the cut E_c is 0.

It is easily verified that all steps involved in the transformation between I and I' can be performed in polynomial time. Therefore, from the known NP-hardness of the partition problem, we can conclude that EDP is NP-hard, and since we have already shown that $EDP \in NP$, it follows that EDP is NP-complete.

Our proof in this section actually demonstrates that a significantly restricted version of the EDP problem (infinite-valued latency constraint, zero-valued edge weights, and HSDF-based dataflow) is NP-complete. It should be noted that the partition problem is readily solvable by approximation schemes, but the additional constraints and dimensions of EDP appear to make a more complex heuristic approach more appropriate.

3.5.1. The heuristic approach for EDP

A variety of heuristic algorithms for graph partitioning have been developed. The Kernighan-Lin (K-L) [35] and Fiduccia-Mattheyses (F-M) [24] algorithms are both popu-

lar heuristic algorithms for graph partitioning. These algorithms involve incrementally exchanging vertices across the partition cut if the exchange improves the targeted figure of merit.

To formulate our heuristic approach for EDP, we adopt useful ideas from the K-L and F-M algorithms. To help describe our algorithm, we define the cost function of a partitioning result as

$$CT(G_m, G_s, E_c) = max\{E(s) + E(t), E(m) + E(r)\} . \quad (3.24)$$

We seek to minimize *CT* so that the corresponding maximized network lifetime can be obtained. Based on this, we formulate the *Gain* function δ for moving actor v from one side of a given cut to the other. Intuitively, $\delta(v)$ gives the potential energy reduction or increase for both G_m and G_s whenever an actor v is switched from one subgraph to the other. δ values can be computed and updated efficiently based on the formulations developed earlier in the previous section. To discuss the algorithm formulation in more detail, it is useful to define the "cost" of a given partitioning to be the maximum energy consumption for transmitting ($E(t)$) and receiving ($E(r)$) data tokens across the partition cut, E_c, plus the energy consumption of computation ($E(s)$ and $E(m)$) for the two subgraphs, G_s and G_m. That is, we seek to minimize *CT* for a given SDF graph $G = (V, E)$ so that the corresponding maximum system lifetime can be obtained.

To help derive $\delta(v)$, we define a *gain pair function* $d(v)$ for switching vertex v from one subgraph into the other subgraph based on possible energy variations on master and slave nodes. The value $d(v)$ is expressed as

$$d(v) = \{\eta(G_s), \eta(G_m)\} =$$

$$\begin{cases} \{E(v) + \eta(cut, t), N_s \cdot (-E(v) + \eta(cut, r))\} & \text{if } v \in G_s \\ \{-E(v) + \eta(cut, t), N_s \cdot (E(v) + \eta(cut, r))\} & \text{if } v \in G_m \end{cases}$$

where

$$\eta(cut, t) = X_c(v) \cdot P_t \cdot t_c - \overline{X_c}(v) \cdot P_t \cdot t_c,$$

$$\eta(cut, r) = X_c(v) \cdot P_r \cdot t_c - \overline{X_c}(v) \cdot P_r \cdot t_c,$$

$$X_c(v) = \sum_{e \in cutedges(v)} prd(e) \cdot q(src(e)),$$

$$\text{and } \overline{X_c}(v) = \sum_{e \in noncutedges(v)} prd(e) \cdot q(src(e)).$$

Here, $cutedges(v)$ is the set of edges of vertex v that cross the partition cut (thus, $cutedges(v) \subseteq E_c$), and $noncutedges(v)$ is the set of edges of vertex v that do not cross the cut. Therefore, $X_c(v)$ denotes the number of data tokens to be transmitted and received by v due to the existing partition cut, and $\overline{X_c}(v)$ denotes corresponding number of data tokens to be transmitted and received due to the partition cut that would result from moving v across the existing cut. Moreover, for a given vertex v, $\eta(cut, t)$ ($\eta(cut, r)$) denotes the improvement in communication energy for transmitting (receiving) data tokens respectively across the partition cut if vertex v is moved across the existing cut. Here, a negative "gain" means that such a move would cause a net increase in communication energy.

Based on the gain pair function for each candidate vertex v, we derive $\delta(v)$ by:

$$\delta(v) = CT - max\{\{E(s) + E(t), N_s \cdot E(m) + E(r)\} - d(v)\} = \begin{matrix} \\ CT - max\{E(s) + E(t) - \eta(G_s), N_s \cdot (E(m) + E(r)) - \eta(G_m)\} \end{matrix} . \quad (3.25)$$

Note that $\delta(v)$ can be positive- or negative-valued. A positive value of $\delta(v)$ means that there is an improvement in the cost function *CT* if node v is moved across the existing

cut. On the other hand, a negative value for $\delta(v)$ represents a deterioration of *CT* if v is moved.

3.5.2. Performance comparison and analysis

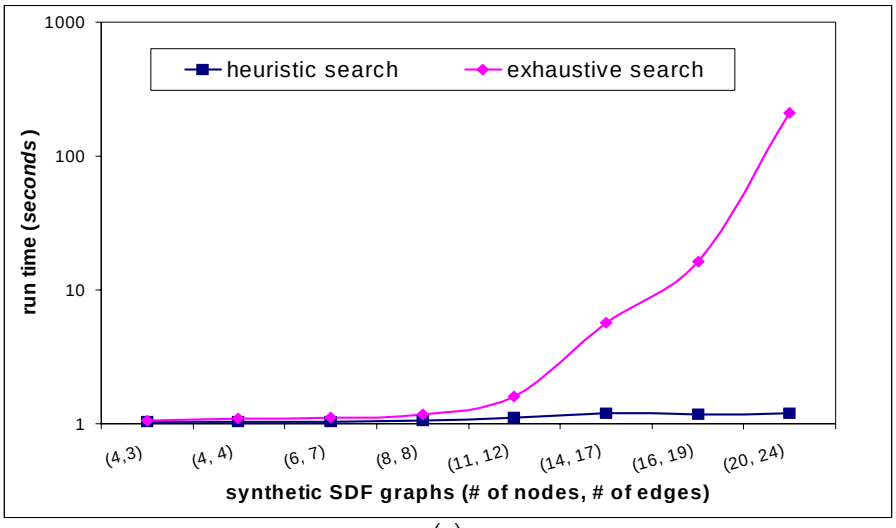
Based on the *Gain* formulation mentioned above, Figure 3.5 shows a pseudocode specification of our heuristic for solving the EDP problem, and Figure 3.6 provides performance comparisons between our heuristic approach and the exhaustive search for the EDP. In Figure 3.6, we examine the performance of our approach on several randomly-generated, synthetic SDF graphs. Figure 3.6(a) shows that while the exhaustive search

```
input: G = (V, E)
output: G_m = (V_m, E_m), G_s = (V_s, E_s), and E_c.
begin
1   Create an initial partition.
2   Compute CT(G_m, G_s, E_c) and check constraints for the
      initial partition.
3   do
4     for (each v ∈ V) do
5       if v violates constraints in Eq. 3.15
6         Mark v locked.
7       else
8         Mark v unlocked, and Compute δ(v) for v.
9     while (unlocked nodes exist) do
10      Find one unlocked node v with maximum δ(v).
11      Add (v, δ(v)) to the ordered list, order_L.
12      Lock v, and update δ for all unlocked nodes.
13    Select the first k nodes that maximizes Σ_v δ(v)
      from order_L
14    if Σ_v δ(v) > 0

15      for (each v ∈ selected k nodes) do
16        Update G_m and G_s based on switching v.
17        Update
            CT(G_m, G_s, E_c) = CT(G_m, G_s, E_c) − δ(v).

18  while Σ_v δ(v) > 0

end
```

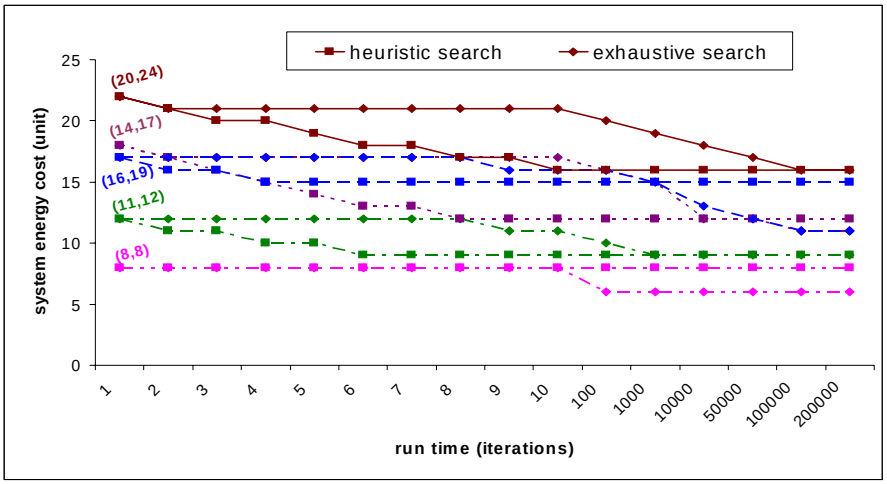
Figure 3.5 Algorithm pseudocode for our heuristic approach to solving the EDP problem.

time quickly becomes infeasible for moderate size examples, our heuristic produces comparable results in a fraction of the time. In our implementation of exhaustive search, the constraints in Eq. 3.15 are verified for each candidate partitioning to filter out invalid results.

Figure 3.6(b) shows a comparison of E_{sys} versus run time for successive algorithm iterations ("run time testing iterations") on several synthetic graphs. Here, the total number of run time testing iterations represent the maximum number of allowable node



(a)



(b)

Figure 3.6 Performance comparison for EDP schemes: (a) Run time comparison based on the complexity of synthetic SDF graphs. (b) Cost versus run time comparison for selected synthetic SDF graphs.

switches. Moreover, to examine the impact of graph complexity in terms of the numbers of actors and edges, we normalize certain graph attributes when constructing the synthetic graphs. Specifically, we normalize the assignments of production and consumption rate to 1 for all the edges, and we also normalize all energy-related attributes (i.e. the power and time estimates) to 1 for all the actors and edges. We observe from Figure 3.6(b) that our heuristic algorithm converges significantly faster than exhaustive search for each synthetic graph. Note here that if both search algorithms converge on the same point, then both algorithm have targeted results that have identical (optimal) quality; in other cases, the exhaustive search algorithm finds an optimal solution, whereas the solution returned by the heuristic algorithm is suboptimal. Our heuristic algorithm for solving the EDP problem has $O(|V|^2|E|)$ time complexity.

3.5.3. Extension to Application Graphs that Contain Cycles

Until now, we have assumed that the overall signal processing application is represented as an acyclic SDF graph. Our implementation of EDP assumes an acyclic input graph, and our experimental results are carried out on graphs that are acyclic. Indeed, many practical signal processing applications take the form of acyclic SDF or parameterized SDF graphs.

However, our partitioning techniques can be extended in a straightforward way to handle more general topologies — in particular topologies that contain cycles. For this purpose, it is useful to first pre-process the application dataflow graph by computing its *strongly connected components* (SCCs). An SCC of a directed graph is a maximal subgraph C such that for any distinct vertices x and y in C, there is a directed path in C

from x to y, and a directed path in C from y to x. SCCs can be computed efficiently from arbitrary directed graphs (e.g., see [17]).

If the original SDF graph G is not acyclic, our extended EDP algorithm first computes the SCCs of G, and then applies the *SDF clustering transformation* to each SCC [10]. This transformation allows each SCC to be abstracted as a single actor (a "hierarchical actor") so that the EDP process can operate on an acyclic graph. Dataflow properties (production and consumption rates) of the hierarchical actors are computed as part of the clustering transformation. Once the partition is constructed, each hierarchical actor is replaced by its corresponding subgraph as the partitions are mapped to their respective target processors.

Implementation of this extended EDP algorithm and experimentation with the algorithm on practical, cyclically-structured signal processing algorithms are useful directions for further study.

3.6. Experiments and Results

3.6.1. Experimental DSP computations

In this chapter, we first choose several DSP computations modeled by SDF graphs to illustrate the operation of EDP, and show the corresponding EDP results. Then, we demonstrate the development of a practical application for distributed speech recognition as a case study in Chapter 4. In the case study, we use PSDF to the real-time sensing activities and digital data processing associated with the speech recognition functionality at each network node. We also provide a detailed analysis of real-time latency and buffer management on this holistically-developed system implementation, integrating careful

design considerations at the levels of the algorithm, application partitioning, network, communication protocol, embedded software, and platform power analysis.

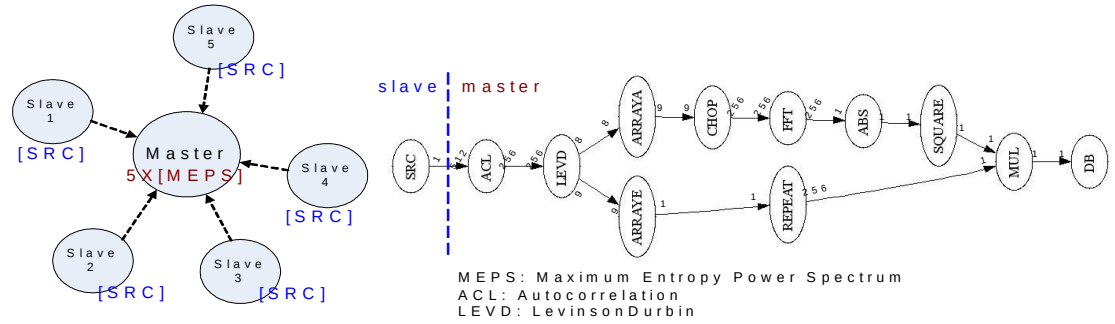
The first DSP computation, which involves maximum entropy power spectrum (MEPS) computation, is adapted from the Ptolemy II design environment [23]. We use this example to illustrate step-by-step the operations involved in EDP based on SDF modeling. Here, we assume that there is 1 master node and there are 5 slave nodes in the targeted network. Figure 3.7 shows three different cases of partition cuts for the MEPS computation, where (a) and (b) show two extreme cases of workload distribution and (c) shows one that has a more balanced distribution. MEPS processing can be divided into two subgraphs, which are allocated to the master and slave nodes as illustrated in the figure. The dotted lines on graph edges represent partition cut candidates. The order (a parameter relating to the complexity and accuracy of the operation) of the MEPS computation in this presented example is 8. Moreover, the repetitions vector of the associated SDF graph model is shown in Figure 3.7(d).

In Figure 3.7(a), the slave nodes send raw data directly to the master node without any processing, where all of the MEPS processing is performed. This configuration involves a partition cut that assigns the source actor of the graph to G_s and the remaining actors to G_m. Therefore, the total data transmission (i.e., X_c) for each scheduled iteration from the 5 slave nodes is $5 \times 512 = 2560$ tokens.

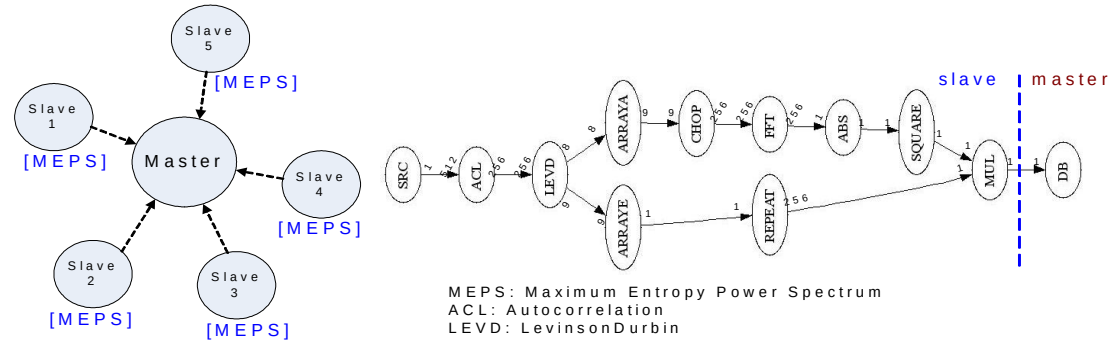
In Figure 3.7(b), each slave node executes a complete MEPS computation, thereby fully processing a captured data frame before communicating to the master node. This is a fully distributed scenario, which minimizes the workload of the master node. In this sce-

nario, each slave node sends 256 tokens to the master node. Thus, the X_c from the 5 slave nodes is $5 \times 256 = 1280$.

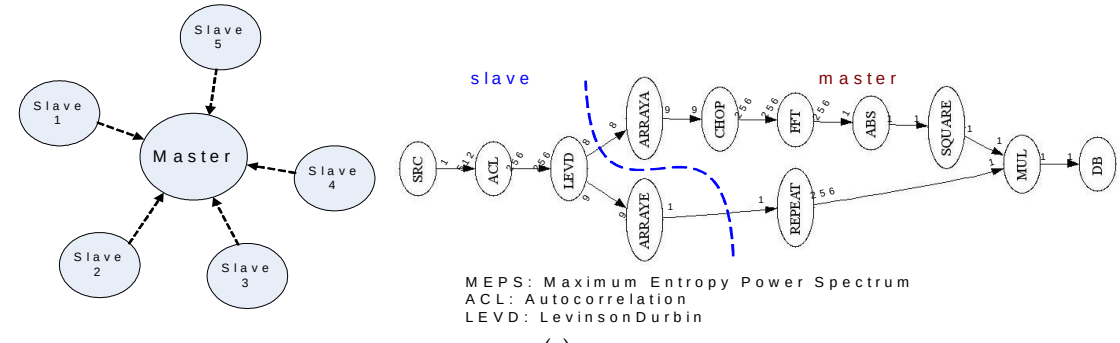
In Figure 3.7(c), on the other hand, the application graph is divided more evenly into two subgraphs. The carefully-constructed partition cut between G_s and G_m reduces X_c to 9, which results in total slave-to-master X_c of $5 \times 9 = 45$ tokens per schedule iteration.



(a)



(b)



(c)

$q(\text{SRC, ACL, LEVD, ARRAYA, ARRAYE, REPEAT, CHOP, FFT, ABS, SQUARE, MUL, DB}) = [512, 1, 1, 1, 1, 1, 1, 256, 256, 256, 256]$

(d)

Figure 3.7 (a)-(c) show various partitioning cases for a WSN that performs maximum entropy power spectrum computation. (d) represents the repetition vector of the modeled SDF graph.

The example of Figure 3.7 illustrates, in terms of the amount of data tokens to be processed and communicated, trade-offs involved in workload balancing among sensor nodes in a network. As discussed in the previous section, our heuristic algorithm can be used to explore such trade-offs effectively in terms of our energy consumption formulation, and the underlying dataflow graph modeling approach. Our experimental results for this application will be demonstrated in the next section.

We have also examined two other DSP applications in our EDP experiments. The first is spectrum computation [23], which can be used in converting signals from time domain to frequency domain representations. The second is a seven-level, tree-structured filter bank [75], which is commonly used in sub-band coding with perfect reconstruction for audio coding applications.

3.6.2. Experimental setup

Our experiments are constructed using TDMA-based *homogeneous* and *heterogeneous* wireless sensor networks that have master-slave topologies. Here, by "heterogeneous," we mean that the master and slave nodes, respectively, are equipped with different kinds of data processing components (having, in general, different speeds, supply voltages, etc.) in addition to the microcontrollers that are used for protocol control. Conversely, by "homogeneous," we mean that the master and slave nodes are equipped with identical data processing components.

Each experiment includes one master node and varying numbers of slave nodes. In order to have each experiment satisfy latency constraints appropriately, we set parameters such that $N_p = N_s$ and

$$t_p = max\{L_l(slaves), L_m(master)\}$$

for the TDMA-based protocol setup. Thus, all computation and communication operations for each scheduled iteration can be performed within a given TDMA time frame.

For our homogeneous WSN target systems, we use the Texas Instruments/Chipcon CC2430 [88] system-on-chip (SoC) device on all master and slave node platforms for executing processing and communication tasks. This device provides a single-chip, integrated transceiver and embedded microprocessor.

For the heterogeneous systems, we again use the CC2430 device on all slave nodes. However, for the master node, we incorporate, in addition to the CC2430, a Texas Instruments TMS320C5509A [93] as a dedicated DSP processor. On the master node, we use the transceiver subsystem in the CC2430 for executing communication tasks, and we use the microcontroller in the CC2430 only for protocol control. We use simulators from the IAR Embedded Workbench and Texas Instruments Code Composer Studio to derive task-level timing estimates. Figure 3.8 shows hardware specifications for both of the processors that we use in the experiments.

Texas Instruments/Chipcon CC2430

Name	Value
Clock frequency	32MHz
Radio frequency	2.4GHz
Supply voltage	3V
Process power	36.9mW
Transmit power	80.7mW
Receive power	80.1mW
Radio bit rate	250 kbps
Code memory	128KB
Data memory	8KB

Texas Instruments TMS320C5509A DSP

Name	Value
Clock frequency	200MHz
Core voltage	1.6V
I/O voltage	3.6V
Core supply current (CPU + internal memory access)	120mA

Figure 3.8 Hardware specifications for the Texas Instruments/Chipcon CC2430 microprocessor and TMS320C5509A DSP processor.

We measure voltage variations and calculate the associated current, power, and energy consumption for the CC2430-based platforms. The voltage variations are measured through the Tektronix 4GHz digital phosphor oscilloscope (TDS 7404). For measuring the power consumption on the DSP core, we use the Texas Instruments C55X Power Optimization DSK for the TMS320C5509A.

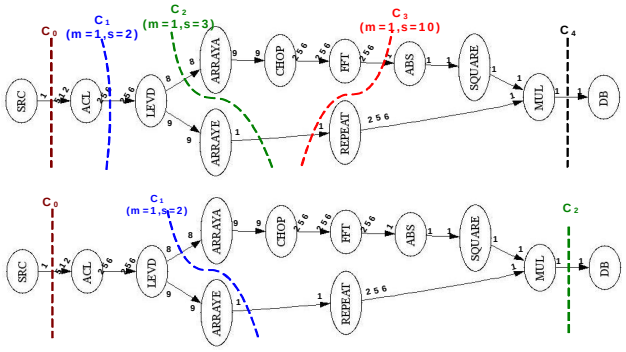
3.6.3. Simulation results

Figure 3.9 shows experimental results for the targeted DSP applications when distributed across homogeneous and heterogeneous WSNs. The EDP results are simulated to derive partition cuts along with the changes of network size by using the proposed heuristic algorithm on each application graph. In Figure 3.9, C_0 denotes the initial partition cut that assigns the source actor to G_s and remaining actors to G_m, and C_i $(i > 0)$ denotes the EDP cut using the proposed heuristic approach in terms of different network sizes (increases in the index i correspond to increases in network size). Note that the initial partitioning corresponds to the conventional configuration of having maximal data processing performed on the master node.

For the tables shown in Figure 3.9, the profiled task-level timing estimates are reported for showing the execution time of individual actors in each application. Note that in a homogeneous WSN configuration, computation tasks on all nodes are executed by the same kind of processor; therefore, the timing information in columns 2 and 3 of the tables are valid for either master and slave nodes. However, in a heterogeneous WSN configuration, since computation tasks on the master node and the slave nodes are executed by a different processor, the timing information in columns 4 and 5 of the tables shows each

actors' execution on the master node, and the corresponding timing information on the slave nodes are provided from columns 2 and 3 of the tables.

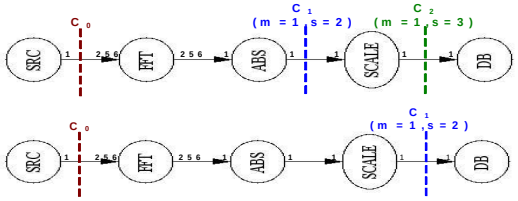
Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Average execution cycle on TMS320C5509A	Average execution time (sec.) on TMS320C5509A with 200MHz CLK
SRC	2534.21	7.92E-5	928.75	4.64E-6
ACL	52830479	1.65	24779839	1.24E-1
LEVD	17977524	0.56	1181	5.91E-6
ARRAYA	2186	6.83E-5	323	1.62E-6
ARRAYE	325	1.01E-5	14	7.0E-8
REPEAT	28093	8.77E-4	5899	2.95E-5
CHOP	39672	1.24E-3	9377	4.69E-5
FFT	7479003	0.23	962008	4.81E-5
ABS	1428.24	4.46E-5	5245	2.6E-5
SQUARE	478.85	1.5E-5	181	9.05E-7
MUL	618.29	1.93E-5	580	2.9E-6
DB	722.25	2.26E-5	5310	2.66E-5



q(SRC, ACL, LEVD, ARRAYA, ARRAYE, REPEAT, CHOP, FFT, ABS, SQUARE, MUL, DB) = [512, 1, 1, 1, 1, 1, 1, 256, 256, 256, 256]

(a) EDP results for the MEPS computation

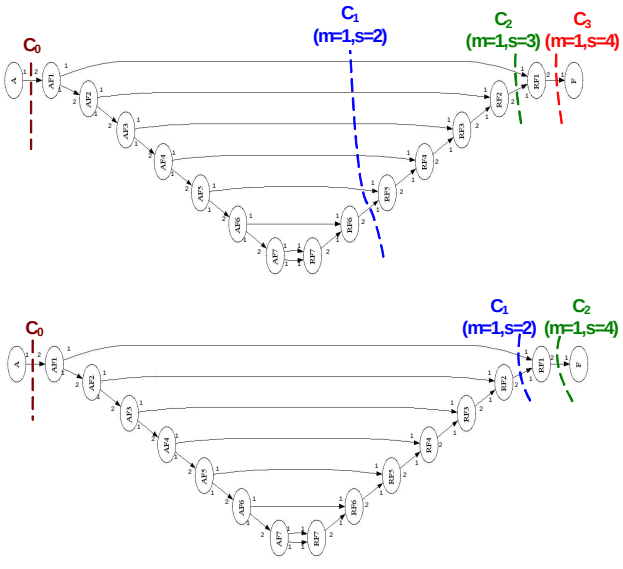
Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Average execution cycle on TMS320C5509A	Average execution time (sec.) on TMS320C5509A with 200MHz CLK
SRC	2532.99	7.92E-5	926	9.74E-6
FFT	8163758	0.26	962008	4.81E-3
ABS	1409	4.4E-5	5325	2.66E-5
SCALE	1606.3	5.02E-5	5421	2.7E-5
DB	707	2.21E-5	5252	2.63E-5



q(SRC, FFT, ABS, SCALE, DB) = [256, 1, 256, 256, 256]

(b) EDP results for the spectrum computation

Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Average execution cycle on TMS320C5509A	Average execution time (sec.) on TMS320C5509A with 200MHz CLK
SRC	1398	4.37E-5	804	4.02E-6
FilterAnalyzer (AF)	40868	1.28E-3	8625	4.31E-5
FilterReconstructor (RF)	56317	1.76E-3	11243	5.62E-5
SINK	274	8.56E-6	116	5.8E-7



q(A, AF1, AF2, AF3, AF4, AF5, AF6, AF7, RF7, RF6, RF5, RF4, RF3, RF2, RF1, F) = [128, 64, 32, 16, 8, 4, 2, 1, 1, 2, 4, 8, 16, 32, 64, 128]

(c) EDP results for the seven-level tree-structured filter bank

Figure 3.9 EDP results for experimental DSP applications across homogeneous and a heterogeneous WSNs with various network size settings (m = # of master nodes, s = # of slave nodes).

The EDP simulation results are shown graphically in Figure 3.9 based on derived partition cuts. Here, the top and bottom figures represent the associated EDP results when a target application is applied to a homogeneous and a heterogeneous WSN system, respectively. The EDP results (C_i) are compared with the initial partitioning (C_0) in terms of various network size settings for each WSN configuration. As shown in the figure, partition cuts are shifted gradually from the G_s side to the G_m side of each application graph as the network size is increased. This is because a more balanced workload distribution is found between the master and slave nodes on the chosen applications. As more slave nodes are added, more of the data processing burden should generally be assigned to the slave nodes since the master node, as the central recipient of communication from all slave nodes, needs to take care of more computational requests.

3.6.4. Energy cost comparison with the workload redistribution scheme

We demonstrate the associated system energy cost, E_{sys} (i.e. Eq. 3.9), in Figure 3.10 for the chosen applications in Figure 3.10 according to the simulated EDP results obtained from Figure 3.9. For more detailed analysis, E_{sys} can be applied to Eq. 3.10 along with the appropriate battery capacity values so that E_{sys} can be converted to an estimate of network lifetime.

We compare the system energy cost with and without our the workload redistribution scheme for all of the experimental DSP applications. We initialize the targeted WSN systems with 1 master node and 2 slaves. Then additional slave nodes are added into the systems one at a time until a total of 10 slave nodes is reached for each system.

We demonstrate a comparison in terms of system energy cost (E_{sys}) in Figure 3.10 for each experimental application. This comparison is based on a homogeneous WSN con-

figuration. Here, "QS EDP" stands for the results from quasi-static EDP, and each partition cut (C_i, where $i \geq 1$) is derived by using our proposed heuristic algorithm with an appropriate network size setting as shown in Figure 3.9. In Figure 3.10, we observe that the system energy cost is reduced consistently with the derived partitioning results for balancing the workload distribution between the master node and the slave nodes. Our

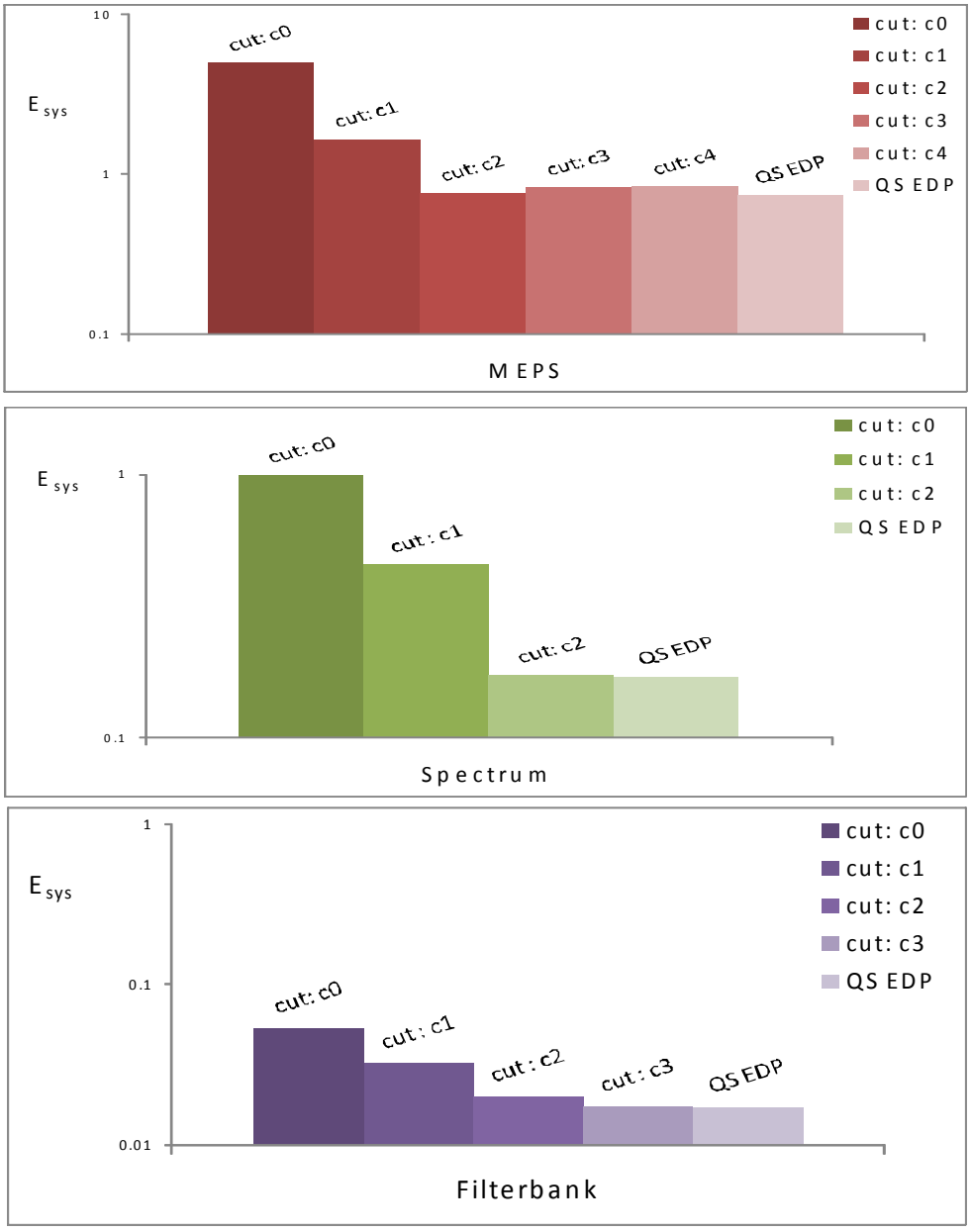


Figure 3.10 Simulation results involving energy cost for the experimental applications.

approach obtains at least a 50% improvement in energy cost compared to the conventional approach of having maximal data processing performed on the master node. Moreover, as the network size changes, the EDP result is adapted automatically to solutions that are better matched to the new scenarios.

Note that as discussed in Section 3.5.2, the exhaustive search algorithm always finds an optimal solution, whereas a solution returned by the heuristic algorithm may be suboptimal. We have used the exhaustive search approach to find EDP solutions for all of the applications that we experimented with. In these experiments, we found that the result of exhaustive search was 5% better on average; however, as we have shown in Figure 3.10, our heuristic approach still improves energy consumption significantly compared to the conventional master/slave processing approach, and also achieves results that are close in quality to or equivalent to the optimal results obtained from exhaustive search.

Chapter 4. Case Study: Distributed Automatic Speech Recognition

In this chapter, we present a case study involving the design and implementation of a distributed sensor network application for embedded, isolated-word, real-time speech recognition. In our system design, we adopt a parameterized-dataflow-based modeling approach to model the functionalities associated with sensing and processing of acoustic data, and we implement the associated embedded software on an off-the-shelf sensor node platform that is equipped with an acoustic sensor. The topology of the sensor network deployed in this work involves a clustered, hierarchical organization. A customized time division multiple access protocol is developed to manage the wireless channel. We analyze the distribution of the overall computation workload across the network to improve energy efficiency. In our experiments, we demonstrate the recognition accuracy for our speech recognition system to verify its functionality and utility. We also evaluate improvements in network lifetime to demonstrate the effectiveness of our energy-aware optimization techniques. Summaries of the work presented in this chapter have been published in [66] and [67].

4.1. Introduction

Speech recognition involves converting acoustic signals, captured by a microphone or an acoustic sensor, to a set of words. Then, these words are compared with some predefined words and some sort of indication is given if there is a match. The recognized

words can then be analyzed and used for back-end applications such as command and control, commercial information retrieval, and linguistic processing for speech understanding. Figure 4.1 presents a design flow for basic speech recognition systems.

From an embedded system design aspect, a major design challenge for speech recognition is to assure the processing of large amounts of data in real-time. Various prior efforts on embedded speech recognition, e.g. [2], [52], and [59], focused on implementing various speech recognition algorithms on embedded platforms, such as programmable digital signal processors (PDSPs) and comparing their performance. These efforts typically have not explored further optimization for real-time operations and energy usage beyond what is already available through a standard PDSP-based design flow. These existing design approaches therefore are not fully suited for heavily resource-limited, distributed systems, such as wireless sensor networks.

WSN systems have a variety of potential applications [40], such as environmental monitoring and intrusion detection. Sensor nodes are often deployed in inaccessible or, in the case of certain military and security-related applications, dangerous areas and communicate with each other through self-organizing protocols. To maximize the useful life of these systems, power consumption must carefully be considered during sensor node design.

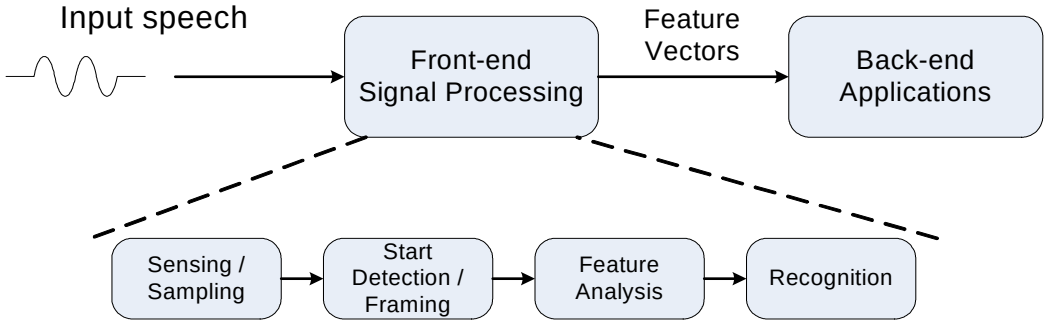


Figure 4.1 Basic design flow for automatic speech recognition systems.

Integrating speech recognition into a WSN system enables a new class of applications for speech recognition that have distributed configurations. We refer to such speech-recognition-equipped WSN systems as distributed automatic speech recognition (DASR) systems.

The DASR system developed in this thesis is an isolated-word and speaker-dependent speech processing system, where templates of extracted coefficients of words have to be created and stored at a central node. The system functionality is to have all sensor nodes collect speech data within their sensing ranges, and transmit this data periodically — in the form of recognized words (or simple indicators for the absence of any words) — to the central node. Any application-specific analysis and usage of the recognized words is handled as back-end processing on the central node.

Based on different requirements on recognition accuracy, we describe two practical application scenarios in which our developed DASR system can be applied. The first scenario involves using a DASR system as a speech-based command and control system in a battlefield environment. Since the DASR system is speaker-dependent, its word templates need to be trained by the person who will be using the system and, as we will show in our experiments, it is capable of achieving a high accuracy for word recognition in this context. When we apply the system in a battlefield for recognizing command words, the speaker-dependent property provides a benefit by rejecting command words that are spoken by enemies and author unauthorized people.

Another application scenario is to use a DASR system as a surveillance system for collecting large amounts of speech data with similar patterns from arbitrary speakers. Since sensor nodes are usually designed to be small, they can be hidden in a battlefield

environment. Therefore, acoustic signals from the enemy can be secretly sensed, collected, and translated into useful data on the sensor nodes. Through a well-designed communication protocol, this data can then be transmitted to a central node for further processing and back-end analysis. The recognized words, for example, can be used to distinguish among a diversity of languages or to survey specific words in a crowd for special monitoring and detection purposes.

We observe, however, that it is difficult to apply speech recognition techniques for secretive speech monitoring (e.g., for security- or defense-related applications) if the recognition is constrained to be performed on a single, stand-alone embedded platform. This is because the sensing range of individual sensor nodes is limited, and sensors are often deployed in difficult-to-reach areas, where their placement cannot be fine-tuned. However, a distributed WSN configuration can help make such applications feasible by distributing the computation across a multitude of embedded platforms (i.e., sensor nodes), and then collecting and consolidating large amounts of monitored information in a systematic way.

4.2. Related Work

Pauwels [50] gives an overview of recent WSN developments for ambient intelligence applications, including speech recognition. In these applications, information provided by sensors is used to drive systems that automatically analyze human behavior.

As shown in Figure 1, the main component in the front-end of speech processing algorithms is feature extraction. Several popular feature extraction techniques, such as Mel-Frequency Cepstral Coefficients (MFCC), Weighted OverLap Add (WOLA), and

Noise-Robust Auditory Features (NRAF) have been used extensively in speech recognition technology. Ahadi [2] has demonstrated that both MFCC and WOLA approaches can achieve high (reasonable) recognition accuracy for clean (noisy) speech. Ravindran [59] presents a comparison between their proposed NRAF approach and the MFCC approach and claims that their proposed NRAF approach can be implemented for low-power sensor networks.

A few research efforts have integrated speech recognition front-ends into WSN systems and demonstrated overall system feasibility. Phadke [52] presents a hardware/software co-design solution to implementing an embedded speech recognition system with the use of a modified MFCC approach for feature extraction, and a dynamic time warping (DTW) technique for template alignment and matching. In this chapter, we build on Phadke's design flow for the embedded software development of our front-end speech recognition processing.

Delaney [20] investigates both computation and communication energy consumption of distributed speech recognition on a targeted embedded system and proposes optimization techniques for energy reduction in the application and network layers. However, Delaney's design and optimization approach are developed for general wireless mobile devices and sophisticated wireless networks such as 802.11b and Bluetooth.

In contrast, our objective is to customize DASR systems for heavily resource-limited sensor nodes, and to employ an application-specific, point-to-point protocol configuration for this purpose. Compared to the efforts of Phadke and Delaney, we target a very different region of the DASR system design space involving potentially higher cost, due to the use of more specialized and streamlined sensing devices, but also involving greater

potential for miniaturization and longer-lifetime operation. The latter objectives are useful for our targeted class of defense- and security-related speech recognition applications, where it is important to have sensor nodes that are small enough so that they are not easily detected, and that can be deployed for long periods of time without maintenance in remote, difficult-to-access geographical areas.

4.3. Distributed Embedded System for Speech Recognition

4.3.1. Master-slave network topology

As we have introduced in the previous chapter, a hierarchical network organization is efficient for lifetime management, and a master-slave network topology is a clustered-based network. Within each cluster of the hierarchy, we refer to the designated central node as the master node, and we refer to all other nodes in the same cluster as slave nodes.

In the experiments, which are based on the off-the-shelf wireless transceiver described in [88] as the communication device for each sensor node, the distance between the master node and all slave nodes should be within approximately 30 meters to ensure reliable communication. We define a parameter to indicate the number of nodes (i.e., the initial network size) being used to set up the overall system. After the system has been initially established (i.e., all nodes have joined the network) based on this number of nodes, the network size can be dynamically changed as nodes are added to or removed from the system. We use the updated value of the network size as an input to our workload redistribution scheme.

In summary, the network developed for the DASR system is a clustered, hierarchical network, where each cluster forms a master-slave network topology. The analysis and

experimental results throughout the remainder of the chapter are based on such a network organization.

4.3.2. Real-time acoustic sensing

In our developed system, we employ a sampling frequency of 8KHz for human speech; therefore, a 125ms timer is set up for the microcontroller [88] to enable an 8-bit analog-to-digital Converter (ADC) for sampling and converting sensed signals from an acoustic sensor. Since sampling and conversion by the selected ADC [88] takes around 20μs, all sensed samples by the targeted 8 KHz sampling frequency can be captured accurately. Moreover, in order to use limited memory size efficiently, we select words of duration 0.25s or less in our experiments so that the number of samples at 8 KHz is bounded by 2000, where each sample is stored as an 8 bit integer value.

When the DASR system is initialized, slave sensor nodes capture samples from background noise within a certain threshold, and calculate the average. The average noise value is subsequently used to compare signal values that are being monitored and captured. In this process of monitoring, threshold-exceeding and zero-crossing schemes are used to detect the start of an utterance in the presence of background noise. Typically, ambient noise generates very few zero crossings compared to normal speech, so that increasing the rate of detected zero crossings can be used as an indicator for the beginning of a speech utterance [52]. Similarly, since the energy level of noise is typically low compared to that for detected speech, a threshold-based scheme can be used to detect signal levels that are likely to correspond to speech.

Once a possible utterance is detected using one or both of the threshold- and zero-crossing-based schemes described above, 2000 consecutive signal samples are captured

and stored in the memory of the associated sensor node for further processing. The processing steps for speech recognition are applied to such blocks of 2000 collected samples.

4.3.3. Parameterized dataflow modeling

We model and implement the targeted real-time sensing and speech recognition processing behavior on slave nodes using the parameterized synchronous dataflow (PSDF) model of computation [8]. According to PSDF semantics, the sensing inputs and start detection schemes are modeled using init and subinit graphs, and the speech recognition processing algorithm is modeled using a body graph. Based on this modeling approach, and the scheduling features provided by PSDF, low-overhead, "quasi-static" schedules can be generated for hand-coded implementation or software synthesis [8]. Here by a quasi-static schedule, we mean an ordering of execution for the functional modules (dataflow actors) whose structure is largely fixed at compile time, with a small number of decision points or symbolic adjustments made at run-time based on the values of relevant input data.

Figure 4.2 illustrates our PSDF application model and an associated quasi-static schedule. Here, *ASR.init* sets the length of a circular window for the past $L - 1$ samples at each point of time, and the frame size M for each data frame. *ASR.subinit* reads sample inputs and maintains a circular window of $L - 1$ samples for real-time processing. In our experiments, we set $L = 2000$, $M = 200$, and use synchronous dataflow (SDF) [41] to model the core speech recognition processing functionality, which is integrated into the enclosing PSDF body graph (i.e., *ASR.body* in Figure 4.2) of our overall PSDF-based modeling framework.

If a valid speech token is detected from the acoustic input stream, a dynamic parameter N is configured to enable speech recognition processing in the body graph ASR.body. This parameter is configured based on the number of speech tokens that should be processed in the newly-initiated recognition step. Otherwise, the value of the parameter N is set to zero, which effectively disables speech recognition processing for the current application iteration.

4.3.4. Speech data processing and recognition

As described in Figure 4.2, speech recognition processing is modeled as an SDF graph — i.e., the ASR.body graph — that is integrated into an enclosing PSDF framework. Once ASR.body is triggered for execution on a block of 2000 collected samples, the block of samples is first partitioned into 10 overlapping frames, where each frame consists of 256 samples with padding zeros. On each frame, a 256-point FFT is applied. The result of each frame-wise FFT is processed by a 16-tap Mel-scaled filter bank to implement the

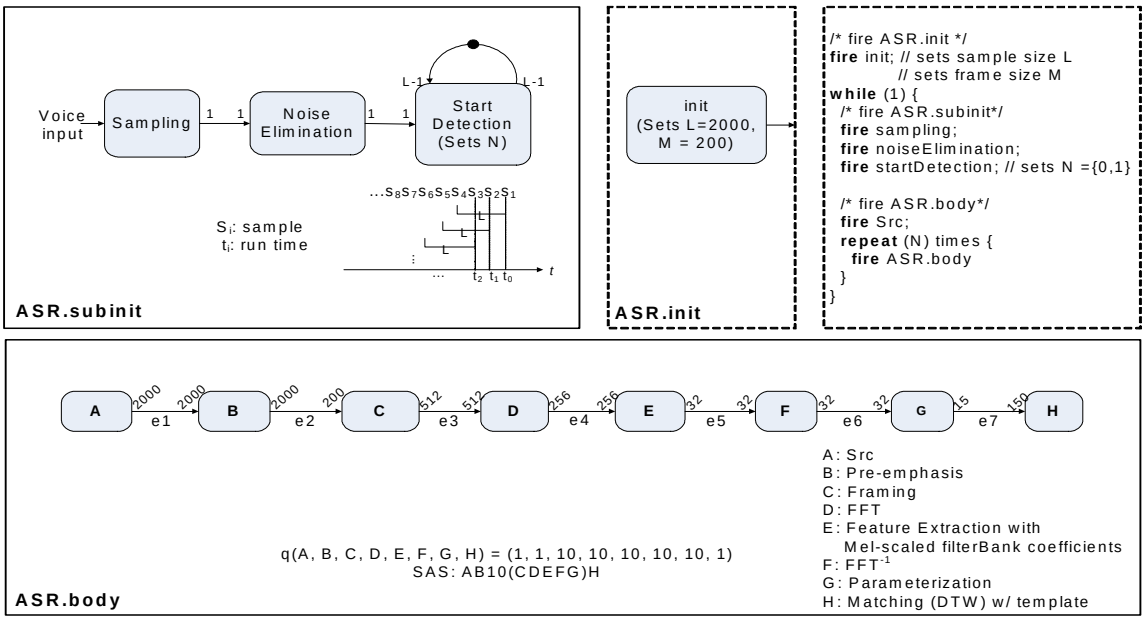


Figure 4.2 PSDF modeling and associated quasi-static schedule for the DASR system.

feature extraction function for the frame. The feature extraction function is used to identify the speaker's vocal tract in the speech. From this feature extraction step, 15 coefficients are obtained from an inverse discrete cosine transform to represent the parameters a given frame. Then, a dynamic timing warping (DTW) technique (e.g., see [12], [52]) is used in the master node to search for a match between the spoken word and one of the template words.

Regardless of which partitioning result of workload distribution is applied, spoken word recognition is always executed at the master node under our WSN configuration. Thus, we unconditionally store a set of parameters for template words in the master node; these parameters are trained and stored a priori through a stand-alone speech recognition process. The set of template words is chosen based on the back-end application.

In the master node, the DTW technique is used to find the best match (i.e., the smallest DTW distance) between the parameters of the spoken word and each of the template words. Here, the smallest DTW distance represents the best match for the spoken word from the inventory of template words.

4.3.5. Memory management and real-time constraints

For embedded system design, memory management is important for real-time operation, especially if resource-limited platforms are used. We analyze buffer (i.e. memory required during the computation process) and latency requirements based on the discussed dataflow structure for modeling and implementing the DASR system.

By applying a looped, single appearance schedule (a compact form of dataflow graph schedule in which looping constructs are organized carefully so that each dataflow actor appears only once in the schedule) to execute the ASR.body graph for the DASR

system, the maximum data memory requirement for slave nodes ($buf(G_s)$) can be bounded as $2000 \le buf(G_s) \le 2982$, in terms of bytes (each sample size is 8 bits). For example, as shown in Figure 4.2, if the slave nodes only capture input signals and transmit them directly without any pre-processing, 2000 bytes of memory are needed to buffer the signal samples. In the other extreme, if the master node is designed to only execute the recognition operation, and the slave nodes perform the entire signal processing chain before transmitting their data, then the slave nodes require $2000 + 512 + 256 + 32 + 32 + 150 = 2982$ bytes of memory to buffer data values throughout the computation process. However, in this case, the slave nodes need to transmit only 150 bytes of data to the master node, since the volume of data is reduced significantly through the signal processing that is performed on the slave nodes.

The memory requirement in the master node is determined in terms of the amount of data required for storing the received data from the slave nodes, the required buffer size for signal processing operations, and the size of the template word inventory. By analyzing the dataflow structure in the system design of Figure 4.2, the memory requirement in the master node ($buf(G_m)$) can be bounded as $150 + buf_T \le buf(G_m) \le 2982 + buf_T$ bytes, where $buf_T = (\#\ of\ template\ words) \cdot (\#\ of\ parameters\ per\ word)$.

In order to achieve the goal of real time sensing and processing for this application, we define a minimum duration T_d between consecutive spoken words that the application must be able to handle. In a command-and-control context, for example, such a value would impose a constraint on how fast successive commands could be applied at a given node. The parameter T_d can be translated into a latency constraint on slave node processing and communication. That is, T_d must be larger than the time needed to execute the

ASR.subinit and *ASR.body* graphs plus the time needed to transmit the required data to the master node.

For example, when the application is implemented on our target platform [88] with a 32 MHz processing speed, and 250 kbps transceiver data rate, and if all sensing and processing tasks except for word matching are handled by the slave nodes, then the minimum allowable "word interval" T_d is approximately 13.675s (0.27s sensing and detection time plus 13.4s processing time and 4.8ms transmission time). On the other hand, if the slave nodes only execute sensing tasks, T_d is constrained below by approximately 0.334s (0.27s sensing and detection time plus 64ms transmission time). Therefore, when an EDP result is applied to the target platform, T_d is bounded by $0.334s \leq T_d \leq 13.675s$. This kind of analysis, which is based on the dataflow-based application model together with relevant details of the target platform, can be used to constrain real-time specifications for the implemented system. The corresponding latency measurements for the execution of individual actors in the system are examined in the next section. Note that T_d is the real-time constraint applied to the slave nodes of DASR system for satisfying requirement of sensing and processing speech signals.

4.4. Experiments

4.4.1. Experimental setup

For demonstrating the DASR system and analyzing its performance in a complete implementation, we use the Texas Instruments/Chipcon CC2430 [88] system-on-chip (SoC) device as the main processor and transceiver on all sensor nodes. This device is therefore used on each node for executing all processing and communication tasks. In

addition, each slave node platform is equipped with an acoustic sensor. We use the profiling tool in the IAR Embedded Workbench to derive a task-level timing estimate for each actor's execution in the DASR system.

For the simulated heterogeneous systems, we use the CC2430 device on all slave nodes. However, for the master node, we incorporate, in addition to the CC2430, a Texas Instruments TMS320C5509A [93] as a dedicated DSP processor. On the master node, we use the transceiver subsystem in the CC2430 for executing communication tasks, and we use the microcontroller in the CC2430 only for protocol control. We use simulators from the IAR Embedded Workbench and Texas Instruments Code Composer Studio to derive task-level timing estimates. The profiling results of task-level timing estimation for the DASR system are shown in Figure 4.3.

We implement the DASR system, measure voltage variations and calculate the associated current, power and energy consumption across multiple CC2430 platforms. The voltage variations are measured through the Tektronix 4GHz digital phosphor oscilloscope (TDS 7404). Figure 4.4 shows this experimental setup for energy consumption measure-

Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Average execution cycle on TMS320C5509A	Average execution time (sec.) on TMS320C5509A with 200MHz CLK
startDetection	364	1.14E-5	33	1.65E-7
SRC	2496	7.8E-5	1025	5.13E-6
Pre-emphasis	2372682	0.074	100150	5.0E-5
Framing	17285507	0.54	105120	5.26E-4
FFT	87028273	2.72	45053953	0.225
featureExtraction	218199980	6.82	10896790	0.54E-1
invFFT	87028273	2.72	45053953	0.225
Parameterization	20096	6.28E-4	3540	1.77E-5
Matching	126874305	3.96	57580815	0.288

Figure 4.3 Task-level timing estimation for the DASR system implementation.

ment and the associated equations to estimate overall energy consumption of a sensor node platform within a period of time t_s, where N_{t_s} represents the total number of points within t_s that are sampled by the oscilloscope.

We choose some common words as experimental examples and compare the recognition accuracies for the different words after the master node receives its required information from the slave nodes and finishes its recognition task. Figure 4.5(a) shows the

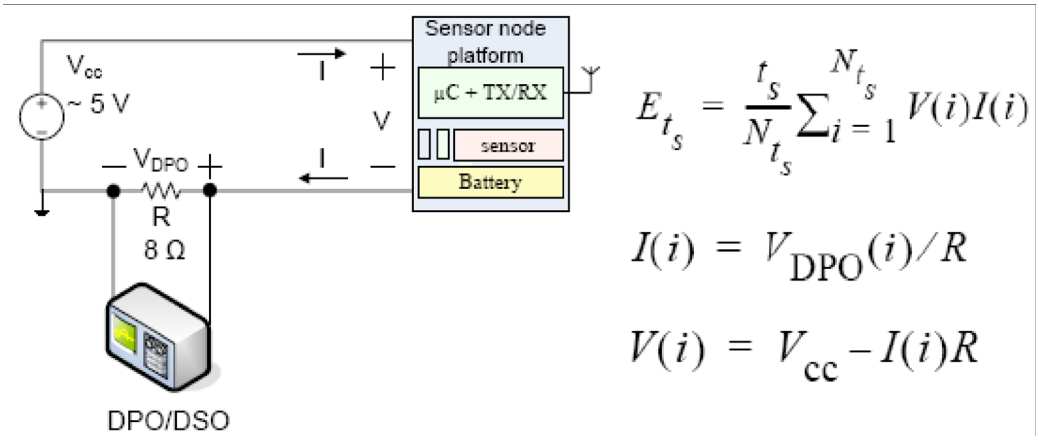
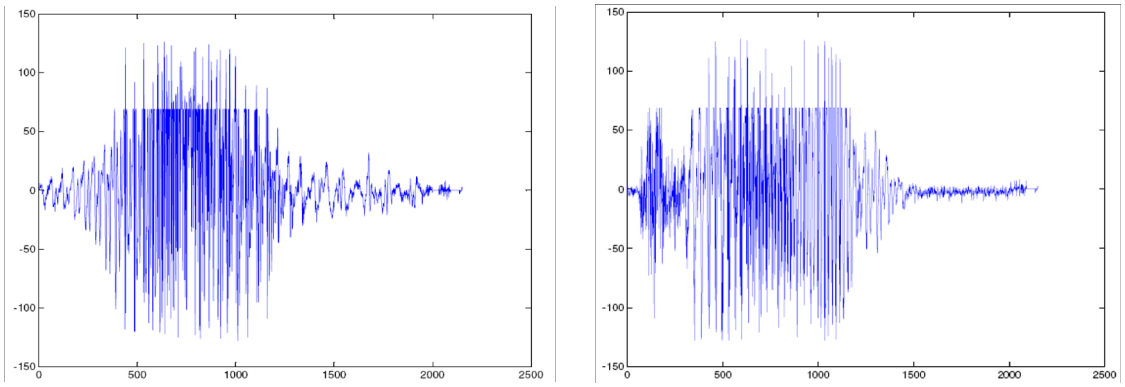


Figure 4.4 Experimental setup for energy consumption measurement.



(a)

Words	'one'	'two'	'eight'	'hello'	'bomb'	'shoot'
Accuracy %	95.3	90.1	96.4	86.2	93.5	79.6

(b)

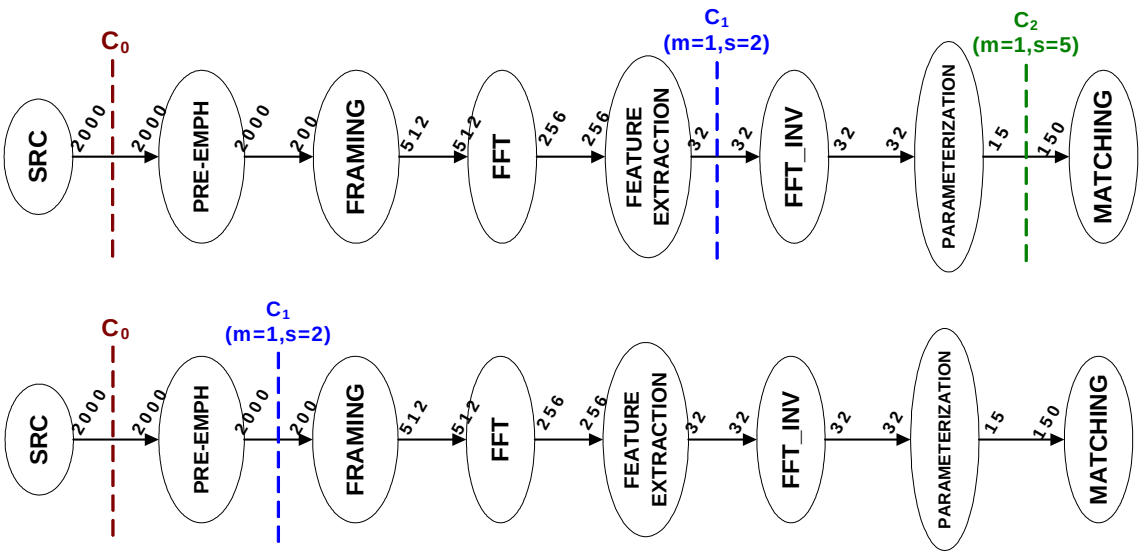
Figure 4.5 (a)Captured signal samples for example words: "one" and "two". (b) Recognition accuracy.

captured signal samples for some spoken word examples. Figure 4.5(b) shows a comparison of recognition accuracy for the example words. Note that our experiments are established in a relatively clean (non-noisy) environment. This helps us to achieve the relatively high recognition accuracies shown in Figure 4.5(b).

Ahadi [2] has demonstrated that by applying different filter bank techniques, recognition accuracy can be further improved even in noisy environments. Elaboration on this method is beyond the scope of this thesis.

4.4.2. Simulation results

In similar to the experiments shown in Chapter 3, the simulated EDP results for the developed DASR system across a homogeneous and a heterogeneous WSN have demonstrated in Figure 4.6. The EDP results are simulated to derive partition cuts along with the changes of network size by using the proposed heuristic algorithm on each application graph. Both top and bottom figures represent the EDP results when the DASR application



q(SRC, PRE-EMPH, FRAMING, FFT, FEATURE-EXT, FFT_INV, PARAM, MATCHING) = [1, 1, 10, 10, 10, 10, 10, 1]

Figure 4.6 EDP results for the experimental DASR system across a homogeneous and a heterogeneous WSN with various network size settings (m = # of master nodes, s = # of slave nodes).

is simulated in a homogeneous and a heterogeneous WSN system, respectively. In Figure 4.6, C_0 denotes the initial partition cut that assigns the source actor to the G_s and remaining actors to the G_m. C_i $(i > 0)$ denotes the EDP cut using the proposed heuristic approach in terms of different network size set to the system.

We also demonstrate the associated system energy cost, E_{sys} (i.e. Eq. 3.9), for the developed DASR application in Figure 4.7 according to the simulated EDP results obtained from Figure 4.6. Here, in similar to the scenario described in the previous chapter, we compare the system energy cost with and without using the workload redistribution scheme for the developed DASR system.

4.4.3. Measurement results and lifetime comparison

According to the TDMA-based communication protocol discussed in Chapter 3, a sensor node in the DASR system transmits and receives data at its reserved time slot after joining the network. Based on this regular communication pattern, we estimate the lifetime for sensor nodes in terms of the workload distribution for computation and communication activities. To this end, we first measure the energy consumption when the master

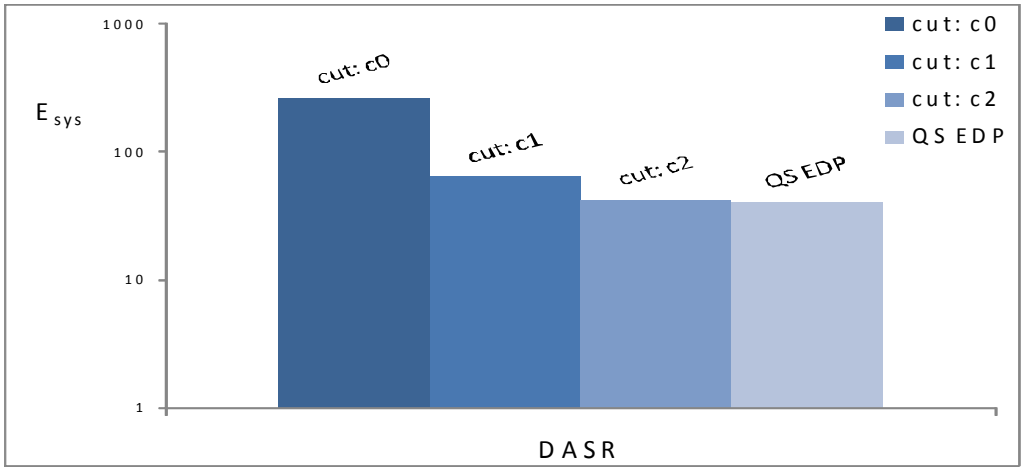


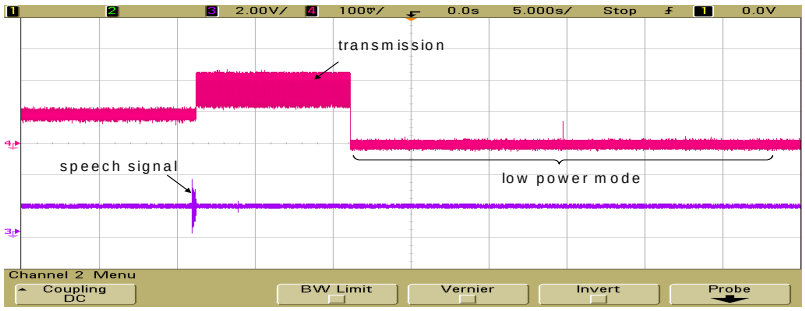
Figure 4.7 Simulation results of energy cost comparison with the workload redistribution scheme for the experimental applications.

node and slave nodes are at their reserved time slots for executing their assigned tasks. Then, this measurement is translated into a lifetime analysis estimate by considering the battery usage on each sensor node. Figure 4.8 shows the results from our measurements and analysis.

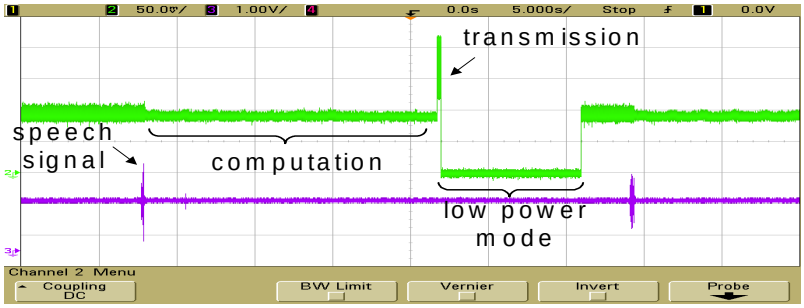
In Figure 4.8, we show voltage variation measurements when two different partitioning schemes are applied to distribute computations across the master and slave node platforms. That is, Figure 4.8(a-d) shows the voltage variation on a TDMA active slot for (a) slave nodes that transmit raw data (i.e., 2000 samples); (b) slave nodes that execute the full signal processing chain and only transmit necessary parameters; (c) the master node configuration that receives all raw data and executes a full computation including the recognition task; and (d) the master node configuration that only receives necessary parameters and executes the recognition task.

By applying the equations in Figure 4.4 to the measured voltage variation presented above, an estimate of the corresponding energy consumption of an active TDMA time slot is obtained.

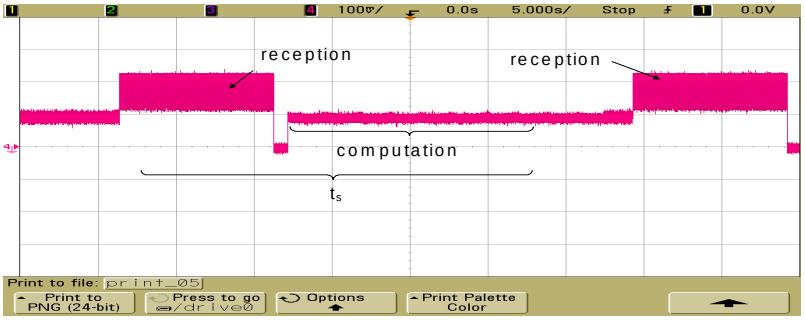
In this experimental setup for the DASR system, the overall system consists of one master node and three slave nodes; thus, we set $N_s = 4$ and $t_s = 25$ seconds so that a TDMA time frame consists of four time slots and the whole computation on a sensor node can be completed within an active time slot. For example, as we see in Figure 4.8(a), a slave node takes around 10 seconds to transmit its raw data (i.e., 2000 samples) to the master node and stays in the idle state (i.e., switches to its low power mode) for the remaining 15 seconds of an active slot. Note that a sensor node may also stay in an idle state whenever its designated slot is not active.



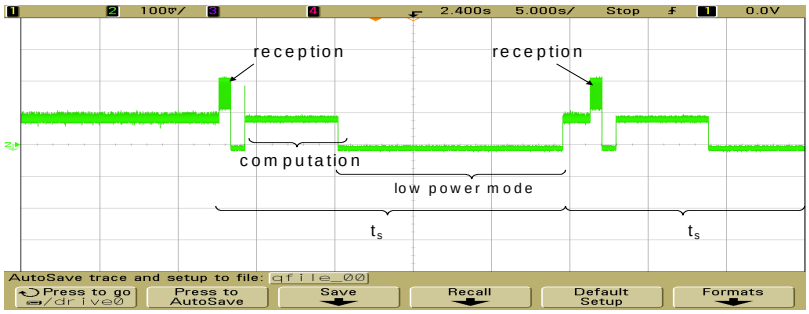
(a)



(b)



(c)



(d)

Node type / Partition type	slave	master
partition 0	157.84 hours	32.55 hours
partition 1	161.52 hours	147 hours

(e)

Figure 4.8 Energy consumption measurement and lifetime comparison.

Furthermore, by considering the use of a $3V \cdot 650mAh$ Lithium battery on each sensor node, lifetime analysis results for sensor nodes in the DASR system are shown in Figure 4.8(e). Here, we compare sensor node lifetime (i.e., sensor nodes continuously execute their tasks frame-by-frame under the given TDMA protocol until they run out of energy) in terms of hours between two different workload distributions. In Figure 4.8(e), the label "partition 0" represents the conventional configuration of having maximal data processing performed on the master node — i.e., the partition is "cut" on $e1$ of ASR.body in Figure 4.2. On the other hand, the label "partition 1" represents a balanced workload distribution after EDP analysis — i.e., the partition is cut on $e7$ of ASR.body in Figure 4.2.

We observe from Figure 4.8(e) that the partitioning of computation and communication affects the lifetime of sensor nodes significantly. It can also be observed that if the system lifetime is defined as the time that the first node in the network runs out of energy, then a DASR system whose workload distribution is based on the EDP result has a system lifetime improvement of approximately a factor of four compared to a system that uses a conventional workload configuration (i.e., the configuration in which slave nodes transmit raw data only).

Chapter 5. Design and Analysis for Distributed Sensor Networks

In the past, primary focus has been given to novel sensor elements for deployment against urban terrorists and in limited force engagements. The issue explored in this chapter is the adequacy of electronic system support for these new sensing elements. For example, ad hoc distributed networks must lie dormant for long periods of time and “come alive” when threats are nearby. This presents a unique challenge in the storage, generation, and management of power. In this chapter, we demonstrate designs of processor algorithms and telecommunication protocols that alleviate current power-system shortcomings for these stationary networks. These advances include 1) low power protocols for data fusion and fault tolerance; and 2) system-level energy modeling and analysis. As a concrete example, we define a distributed sensor support system for line crossing recognition. We demonstrate that threat detection is a system-level problem. Single elements of the system chain individually have small impact on overall performance. Through the development of a pre-amplifier/amplifier chain for optimum signal-to-noise (S/N) ratio, we show the degree to which system-level architecture can improve reliable detection. Specifically, the use of sensor redundancy to improve performance is analyzed from a statistical viewpoint. The work presented in this chapter has been published in [63], [64], [65], and [79].

5.1. Introduction and related work

Sensor support systems, such as wireless sensor networks (WSN), address a great diversity of asymmetric defense and security applications. They include chem/bio threat detection, explosive detection, intrusive detection, and battlefield surveillance [40]. In many circumstances, sensor nodes are densely deployed in areas that are dangerous or otherwise inaccessible to humans. Thus, nodes must communicate with one another wirelessly through self-organizing protocols [29], [38]. Often, when designing such a distributed sensor system, the size of individual sensor nodes should be small enough so that they can easily be hidden in the environment. Issues of energy and power consumption are especially important due to the requirement of extended system lifetime [60]. A long autonomous system lifetime is an important evaluation metric for sensor support systems since any system is required to stay alive as long as possible. Also, fault tolerance features of the system are desired so that the system functionality can be reliably maintained.

Design of each element in a sensor support system is, in itself, a research topic. For example, Figure 5.1 shows a typical block diagram for a system platform that consists of distinct single elements. In order to extend the system lifetime, low-power design of single elements for a sensor node platform has been extensively studied in recent years (e.g. see

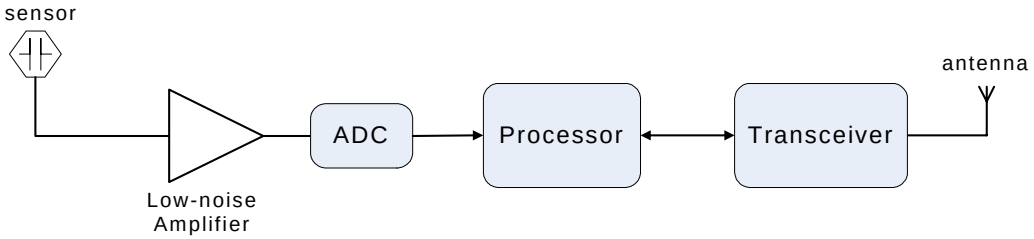


Figure 5.1 Block diagram of a sensor support system platform

[7] and [21]). However, few of these design techniques for lifetime improvement have been discussed from a system-level point of view.

In this chapter, we consider the design of sensor support systems for threat detection as a system-level problem. Here, the system-level problem is defined to be considered at the application level, protocol level, physical design level, and performance modeling level. For example, it includes algorithm streamlining, communication protocol configuration, and hardware/software implementation. We target system lifetime as the key system-level optimization objective and present our designs and experiments all aimed at lifetime improvement. This includes integration of a sensor support system for threat detection with single-element designs; system-level energy modeling and analysis; and simulation-based experimental results.

We start with the introduction of a distributed sensor network for line-crossing recognition as a threat detection application. This system is sensor supported with an acoustic sensor installed on each sensor node. Threats are detected via acoustic signals. The purpose of this system is to periodically reach consensus in deciding whether or not an object ("intruder") has crossed a specific boundary ("line") in a noisy environment that is contiguously monitored. Furthermore, upon detecting an intrusion, the system determines where the line was crossed (i.e., between which nodes in the line). For example, in Figure 5.2, the sensor nodes are placed in a circle inside a room. In this practical configuration, the system recognizes when and where a subject has crossed the circle through an integrated application, protocol, and system architecture development.

When we consider lifetime improvement at the system level, the energy consumption of each sensor node in the system must be carefully optimized to increase system life-

time. Experimental results presented in [25] and [62] show that the power consumption for communication devices such as transceivers dominates overall power consumption on a sensor node. Therefore, for application-level algorithm development, we develop a light-weight distributed algorithm for line-crossing recognition so that the transceiver use time — in terms of the data size (decoded payload bits) to be communicated — can be minimized. Furthermore, for protocol-level consideration, all sensor nodes in such a threat detection application communicate with each other through an efficient, wireless time division multiple access (TDMA) protocol so that each node can transmit and receive at designated time slots, and can "sleep" during other times for energy savings. The packet routing path for such an application is based on a ring topology.

In [30], Hirschberg and Sinclair proved an upper bound of $O(logN)$ on the number of bits that are sent by every node during a consensus task of N nodes arranged in a bidirectional ring topology. Every node that executes its algorithm has an initial input, and has no additional inputs during its execution. In [22], Dinitz, Moran, and Rajsbaum proved an upper bound of $O(N)$ on the number of bits that is sent by all nodes during a consensus

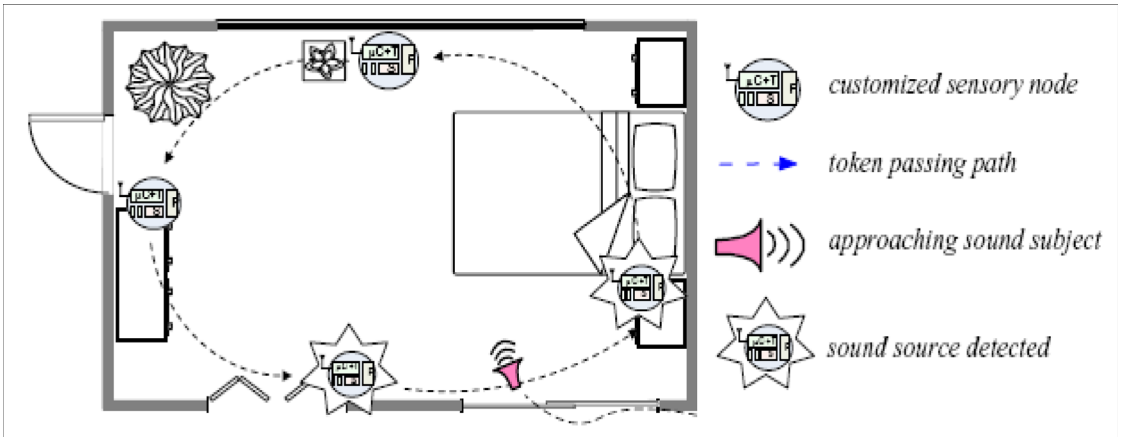


Figure 5.2 An indoor environment scenario with the use of the threat detection system for line-crossing recognition

task of N nodes arranged in a tree topology (a chain topology is a special case). This proof is based on the collection of information with feedback (CIF) algorithm. Every node that runs the CIF algorithm may have an initial input without additional inputs during its execution. After one round that includes two phases — the 'collect" phase and "feedback" phase — all nodes reach consensus.

In our proposed distributed algorithm — in contrast to the approaches described above — each node can obtain many inputs (i.e., either from the received data or from the sensed data) during its execution, and based on these inputs, all of the nodes decide whether or not a subject is approaching and crossing the given line. Also, our algorithm has the property that either $O(logC)$ or $O(logN)$ data bits are needed depending on the protocol stage (i.e., synchronization stage or communication stage), instead of $O(N)$ bits. Furthermore, during most its lifetime, our system communicates with only $O(logC)$ data bits. Here, C — a design parameter — is the minimum number of nodes that must sense the subject in order to reach consensus that an intruder is approaching and crossing the line. Higher values of C provide higher system accuracy at the expense of higher communication requirements and higher recognition latency. Since energy consumption during transmission and reception is high, our approach reduces energy consumption significantly by reducing the number bits that need to be communicated for overall system operation.

Any sensor support system currently envisioned monitors threats in a noisy environment. False detection is inevitable at some level. This, in turn, creates unnecessary energy consumption especially if the monitored environment is severely noisy. In this chapter, we analyze that using a low-power pre-amplifier/amplifier chain (or called amplifier con-

cisely) with ultra high signal-to-noise (S/N) ratio is effectively preventing such a false detection problem and reducing redundant energy consumption. The amplifier that we choose is referred to [1] and designed to use low noise read-out circuitry, including a pre-amplifier and shaping amplifier, to increase the signal-to-noise ratio. The shaping amplifier is used to achieve two conflicting goals. The first goal is to increase the signal-to-noise ratio by restricting the bandwidth. A large bandwidth will increase the noise without increasing the signal. The pulse shaper takes a narrow pulse and turns it into a broader, gradually rounded peak. The second goal is to limit the pulse width in order to measure consecutive signal pulses without pileup or overlap. A trade-off exists because reducing the signal pulse width will increase the signal rate but at the expense of higher noise. Optimum shaping depends on the desired application. In this case, the goal is to increase the signal-to-noise ratio in detector sensors. Therefore, the main focus will be on limiting the bandwidth to achieve a higher signal-to-noise ratio.

5.2. Distributed Sensor System for Line Crossing Recognition

5.2.1. Lightweight Distributed Algorithm for Line-Crossing Recognition

Our proposed distributed system uses a TDMA-based communication protocol that consists of two stages: synchronization and communication. The maximum number of nodes (upper bound) is determined according to the actual environment where the system needs to be deployed. Therefore, the number of nodes is fixed at design time. All node-to-node communications are based on a ring topology. Based on our experiments, we assume that at most two nodes may fail in the system. Define $(N + i - 1)\ mod\ N$ to be the neighbor of node $(N + i)\ mod\ N$. Assuming that there is no node failure, the only requirement

is that every node "hears" (receives communication from) its neighbor. If there are at most two node failures (e.g., node $(N + i - 1)\ mod\ N$ and then $(N + i - 2)\ mod\ N$), the requirement is that node $(N + i)\ mod\ N$ hears nodes $(N + i - 1)\ mod\ N$, $(N + i - 2)\ mod\ N$, and then $(N + i - 3)\ mod\ N$, respectively. Therefore, under our assumption, it is not necessary that all nodes hear all other nodes. All N nodes within the system run the distributed algorithm, and reach a consensus based on local decisions of C nodes while a subject is being detected ($C \leq N$). Here, C is a pre-determined parameter that allows the designer to control a trade-off between recognition accuracy and communication requirements.

The full operation of the line-crossing recognition application, which runs on each node after the whole system is synchronized (i.e., at the communication stage), involves two phases of operation (phase 0 and phase 1). In phase 0, the nodes reach a consensus and decide whether the subject has crossed the line, and in phase 1, the nodes find the place where the subject crossed the line.

Figure 5.3 shows a pseudo-code representation of the proposed distributed algorithm, as well as an example to illustrate the message structure of the data packets used in the communication protocol. According to the use of different phases of operation, only the number of least significant bits is used in each phase. This is explained in our discussion of Figure 3 later in this section. We implement such an algorithm at a node level in terms of mode operations (i.e. *Transmission* and *Reception*) in a TDMA-based protocol design. Note that in the algorithm operation, hardware-controlled receiving and transmitting operations are dependent on the targeted transceiver module. Since this thesis is concerned primarily with system-level design, we do not address the details of these operations, which encapsulate lower level hardware configurations.

In phase 0 of the proposed algorithm (lines 4-22), the nodes use a counter C_{glo} to decide whether the subject is approaching based on local decisions of C nodes ($C \leq N$) that sense the subject. In its turn, each node receives C_{glo} from its previous neighbor (i.e., the node with ID i receives from node $(N + i - 1)\ mod\ N$, which is its "left" neighbor in

```
0 function receiveMode() {
1   (F_RX, CM, ID, C_global or M_sense or MAX_sense) = receiveOp(period T_RX);
2   if (F_RX is true) then
3      switch CM then
4         "phase 0, round 0":
5            update C_glo from the least 3 bits of the received message.
6            if (F_sen is true) then
7               if (F_app is false and C_sen is 0) then
8                  increment C_glo;
9                  set F_inc to true;
10                 increment C_sen;
11
12              if (F_app is false) then
13                 if (C_glo is larger than C_G) then
14                    set F_app to true;
15                 else
16                    if (F_sen is false and F_inc is true) then
17                       decrement C_glo;
18                       set F_inc to false;
19
20                    if (C_glo equals to 0) then
21                       update CM to "phase 1 and round 0";
22                       set M_sen to C_sen;
23
24         "phase 1, round 0":
25            if (phase equals to 0) then
26               update CM to "phase 1 and round 0";
27               if (the least 6 bits of the received message is less than
28                     C_sen) then
29                  set M_sen to C_sen;
30               else
31                  set M_sen to the least 6 bits of the received message;
32         else
33            update CM to "phase 1 and round 1";
34            set MAX_sen to the least 6 bits of the received message;
35
36         "phase 1, round 1":
37         set MAX_sen to the least 6 bits of the received message;
38         if (round equals to 0) then
39            update CM to "phase 1 and round 1";
40
41         if (MAX_sen equals to C_sen) then
42            "the subject crossed the line near node I";
43            reinitialize the system;
44 }
```

```
function transmitMode() {
   switch CM then
      "phase 0, round 0":
         update CM to "phase 0, round 0"
         transmitOp(CM, ID, C_glo);

      "phase 1, round 1":
         update CM to "phase 0, round 0"
         transmitOp(CM, ID, M_sen);

      "phase 1, round 1":
         update CM to "phase 0, round 0"
         transmitOp(CM, ID, MAX_sen);
}
```

Notation for distributed line-crossing algorithm

N: number of nodes
$\mathbf{F_{RX}}$: local received flag
$\mathbf{F_{app}}$: global approaching flag
$\mathbf{F_{inc}}$: local incremented flag for $\mathbf{C_{global}}$
$\mathbf{F_{sen}}$: local sensed flag
phase: phase indicator
round: round indicator
CM: control mode message
$\mathbf{C_{glo}}$: global sensed counter
$\mathbf{C_{sen}}$: local sensed counter
$\mathbf{C_G}$: global sensed threshold
$\mathbf{M_{sen}}$: local maximum sensed count
$\mathbf{MAX_{sen}}$: global maximum sensed count
$\mathbf{T_{RX}}$: maximum waiting time for receiving message

receiveOp: hardware-controlled receiving operations
transmitOp: hardware-controlled transmitting operations

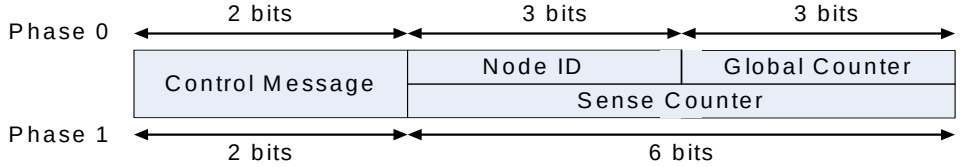


Figure 5.3 Pseudocode specification for the proposed distributed algorithm of line-crossing recognition and an illustration of data packet structure.

the circular, virtual linkage of nodes based on their identifiers). When a node senses that the subject is approaching, it increments C_{glo} by one (line 8). Every node increments C_{glo} at most once. Therefore, at any moment during the first stage of phase 0, the value of C_{glo} indicates the number of nodes that sense the subject. When C_{glo} reaches C, all the nodes reach a consensus and decide that the subject is approaching. The node that increments C_{glo} to C is the first node to set its approaching flag (F_{app}) to 1 (line 14). In their respective turns, all of the other nodes set their F_{app} values to 1. Note that in this phase of operation, the execution of the algorithm relies on the count of sensing samples. The reliability of the sensing activities determines the system performance, which further affects the energy consumption for the overall system. Thus, a high signal-to-noise ratio device is preferred to help sensing elements improve performance. This will be discussed in more detail in the following sections.

Then, the second stage of phase 0 starts (line 15). During the second stage of phase 0, the subject is stepping away from the line. Every node has increased C_{glo} and stops sensing the subject, decrements C_{glo} by one (line 17). Since every node increments C_{glo} at most once, eventually all the nodes will stop sensing the subject. Then, C_{glo} will be decremented to zero. The first node, s, that decrements C_{glo} to zero, starts phase 1 (line 24).

In the message structure example shown in Figure 5.3, the system is assumed to consist of at most 8 nodes. Thus, in phase 0, the least significant 6 bits are used to represent a set of 8 nodes using 3 bits, and to represent C_{glo} by using the other 3 bits. In phase 1, where it is not necessary to identify the nodes, the least significant 6 bits are used to represent C_{sen}.

In phase 1 of the proposed algorithm (lines 24-43), the nodes find the place where the subject crossed the line. This is done by finding which node sensed the subject the maximum number of times, which in turn is determined using a *sensing counter* C_{sen} in every node. Phase 1 consists of 2 rounds. In the first round (lines 24-34), the maximum sensed number is found. Node s transmits $C_{sen}(s)$ to its next neighbor. In its turn, each node i transmits the maximum of the number it received from node $(N + i - 1)\ mod\ N$ and $C_{sen}(i)$. At the end of the first round, node s receives the global maximum sensed count, $MAX_{sen} = max\{C_{sen}(i)\}$. Afterward, node s starts the second round (line 36), where it transmits MAX_{sen} to its next neighbor. Every node j—in its turn—compares MAX_{sen} to $C_{sen}(j)$ (line 41). If there is a match, node j claims that it obtained the maximum number of senses, which means that the subject crossed the line near node j. Otherwise, node j transmits MAX_{sen} to its next neighbor. At the beginning of the second round, node s holds MAX_{sen}, which is equal to $C_{sen}(k)\ (0 \le k \le N - 1)$. Therefore, the second round ends at node k. That is, the subject crossed the line near node k, and the system can be reinitialized. Suppose several nodes sense the approaching subject the same number of times which is equal to MAX_{sen}. In phase 1 round 0 the MAX_{sen} is found, and in phase 1 round 1 the first node out of these nodes that receives MAX_{sen} decides the approaching subject has crossed nearby.

From our algorithm, it can be observed that only $logC$ are needed to represent C_{glo} and a maximum of $logN$ bits are needed to represent C_{sen}. However, the time that the system operates in phase 1 is much less than the time of operation in phase 0. This is because the subject is stepping across the line continuously, and when phase 1 starts, the nodes can

find the place where the subject crossed the line relatively quickly. Therefore, the critical number of data bits for transmitting and receiving is $O(logC)$ instead of $O(logN)$.

We determine the consensus threshold C based on noise in the actual environment. For example, if the environment is clear and there is minimal noise, we set $C = 1$. If the environment is noisy, we experiment with higher values of C so that multiple nodes must filter out noise, and agree about an approaching subject.

Based on a worse case analysis of the distributed algorithm, we give our fault tolerance protocol a chance of r rounds for every node before another neighbor is chosen. Suppose one round takes T time units and m nodes fail simultaneously. In this worse case, each node requires $T \cdot m \cdot r$ time units for a new neighbor to be found and assigned.

5.2.2. TDMA-based Low Power Protocol for Communication

TDMA-based communication protocols are often applied in small scale wireless communication systems [19], [80] due to their simplicity and low power communication patterns. Furthermore, with TDMA, collision avoidance can be guaranteed throughout the system. Therefore, we have employed TDMA-based communication in the WSN system presented in this chapter. The specific protocol that we have implemented consists of two stages: synchronization and communication. During the synchronization stage, the nodes are synchronized with each other. Whenever a node is powered on, it starts in the synchronization stage with a periodic communication pattern of transmitting and receiving one packet during each TDMA time frame, which consists of several pre-defined time lots. In each time frame, each powered node transmits one packet and receives at most one packet. Node i stays in the synchronization stage until it receives a packet from its previous node

$(N + i - 1)\ mod N$. Afterward, it enters the communication stage for regularly communicating with other nodes in the network.

During the communication stage, every node transmits, receives, and idles periodically based on a pre-defined TDMA time schedule. In such a case, each node can power down its main computation and communication resources when the node is idle so that energy consumption is reduced to a minimal level. Figure 5.4 shows a schedule for a four-node TDMA-based communication protocol with the ring topology routing scheme.

5.2.3. The Fault Tolerance Approach

We enhance the TDMA-based protocol mentioned above so that the fault tolerance feature is supported. This is preventing any node failures from translating into failures in the overall system. Here, a node failure means that a node stops processing and communicating, for example, due to lack of energy. Without considering fault tolerance, any node failure, in general, causes abnormal termination of all other nodes in the system.

The approach that we use in the synchronization stage is to have all of the nodes separately determine whether or not the system is synchronized, in addition to the initial

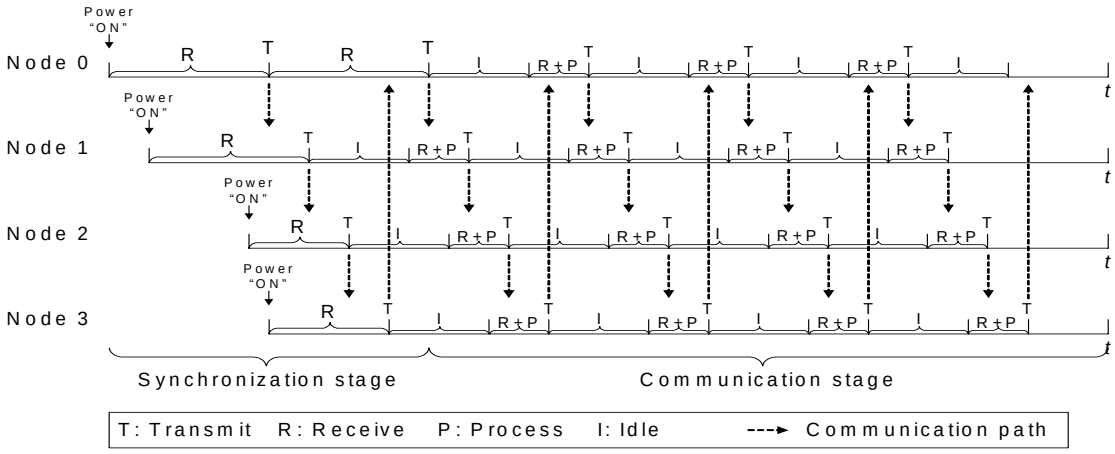


Figure 5.4 TDMA-based communication pattern with a ring topology routing scheme for a four node example.

node clock synchronization process discussed previously. The system is synchronized only if all of the functioning nodes are synchronized and agree on this situation. Once the system is synchronized, the algorithm of system failure prevention at the communication stage can be activated.

During the synchronization stage, there are only functioning (powered-on) nodes and powered-off nodes. Moreover, powered-on nodes will be synchronized and will enter the communication stage in a relatively short period of time. Our algorithm for system failure prevention is not incorporated in the synchronization stage. This is because a node might consider a powered-off neighbor as a failed node. Whenever a node fails in the communication stage, the distributed system will be reorganized automatically within the time period of a single TDMA frame.

The fault tolerance algorithm that we employ in the synchronization stage operates in the following way. Suppose that there are N nodes in the system, and every node has a unique identifier (ID) i such that $0 \le i \le N-1$. Initially, once node i is turned on, it periodically transmits a packet to node $(i+1) \bmod N$, every s seconds. As mentioned previously, such a packet includes a control message field, an id field, and a global counter field (C_{glo}) with an initial value of 1. In the synchronization stage, C_{glo} counts the number of nodes that are on and are synchronized. In addition, node i keeps an internal synchronization flag (F_{sync}), which is set to 0 when the node is powered-on. F_{sync} indicates that node i knows whether all the nodes in the system are synchronized. Note that during the synchronization stage, some nodes might be on while others might be off. Therefore, the overall distributed system is not necessarily synchronized at this time.

While node i transmits a packet every s seconds, where s denotes the duration of a TDMA time frame, it continuously tries to receive a packet from node $(N + i - 1)\ mod\ N$ (i.e., from its "left" neighbor in the circular, virtual linkage of nodes based on their identifiers). Whenever node i receives a packet from node $(N + i - 1)\ mod\ N$ with $C_{glo} \leq N - 1$, it reads the associated value C_{glo}, increments this value by 1, and transmits it within a data packet to node $(i + 1)\ mod\ N$.

Whenever a node j $(0 \leq j \leq N - 1)$ is the first to receive a packet from its predecessor (i.e., from node $(N + i - 1)\ mod\ N$) with $C_{glo} = N - 1$, node j sets its internal F_{sync} to 1. Then node j transmits a packet to node $(j + 1)\ mod\ N$ with a unique control message α, and C_{glo} is set to 0. That is, at this point, node j knows that all the nodes are on and that they are synchronized with their neighbors. Therefore, node j starts the process of informing all other nodes in the system that all the nodes are on and are synchronized. It does this by transmitting the control message α to its neighbor.

Now suppose that a node k $(0 \leq k \leq N - 1$ and $k \neq j)$ whose F_{sync} value is 0 receives a packet from node $(N + k - 1)\ mod\ N$ with the control message α. Then node k sets its F_{sync} value to 1, and transmits a packet to node $(k + 1)\ mod\ N$ with the control message α, and with a C_{glo} value of 0. When node j (i.e., the first node that set its F_{sync} value to 1) receives a packet from node $(N + j - 1)\ mod\ N$ with the control message α, node j knows that all the nodes in the system have received α. Then, node j starts the communication stage by transmitting a packet to node $(j + 1)\ mod\ N$ with the control message β and with a C_{glo} value of zero. At this point, all the nodes are synchronized in the system.

Note that α is transmitted as long as there are still nodes that are not synchronized, and β is transmitted once all the nodes are on and are synchronized. Moreover, the time period over which the whole system remains in the synchronization stage must be larger than the time difference between when the first and the last nodes join the system plus an additional s seconds.

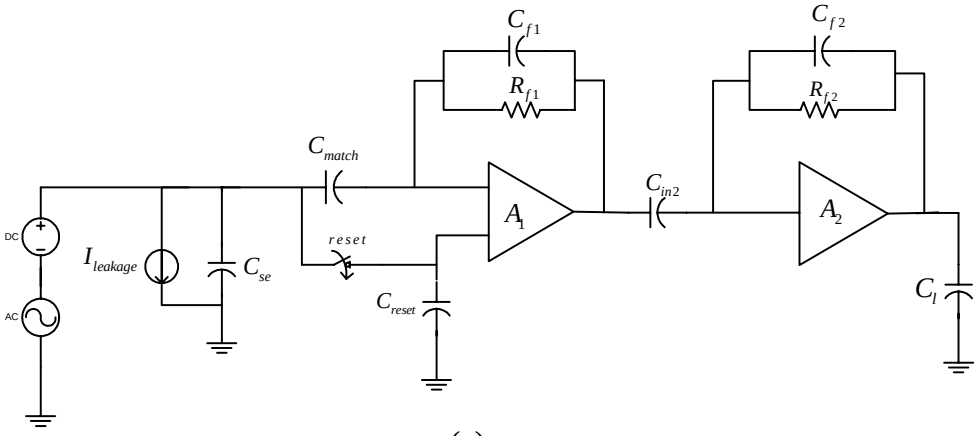
If our TDMA-based distributed system consists of N nodes, the TDMA time frame of s seconds is divided into N time slots, and each such slot lasts for a period of s/N seconds. During slot i $(0 \le i \le N-1)$, node i transmits a packet, and node $(i+1) \bmod N$ receives that packet in its receiving window. Here, the receiving window is defined as the longest time period allowed for receiving packets within a given TDMA time slot.

When node i does not receive packets from node $(N+i-1) \bmod N$ (i.e., if node $(N+i-1) \bmod N$ has failed), then node i starts a process to find a new neighbor. Based on the pre-defined TDMA schedule, node i shifts its receiving window from slot $(N+i-1) \bmod N$ to slot $(N+i-2) \bmod N$ and tries to receive a packet from node $(N+i-2) \bmod N$ during the next TDMA time frame. If node i succeeds in receiving a packet from some node $(N+i-\delta) \bmod N$ $(0 \le \delta \le N-2)$, then δ becomes the new neighbor of i in the new, fault-adapted, virtual linkage structure of the remaining $N-\delta$ functional nodes. All the nodes between nodes i and $(N+i-\delta) \bmod N$ (non-inclusive) are considered from this point onward as being non-functioning nodes. That is, the corresponding TDMA time schedule on each node is pre-defined initially, and is adapted as execution evolves based on changes in system status.

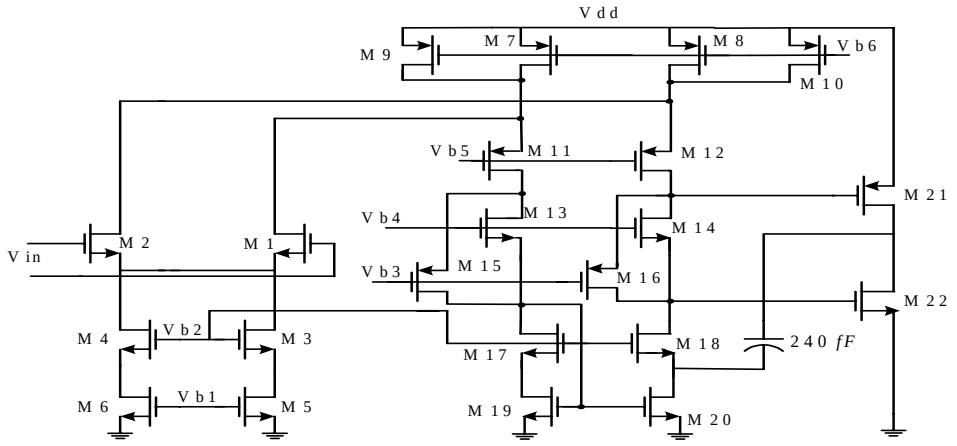
In our proposed system, acoustic sensors will require some amplifier and pre-processing. Optimizing the analog pre-processing saves power, as shown in the next section.

5.3. Low-Noise Pre-Amplifier/Amplifier Chain for High Capacitance Sensors

A low noise amplifier/pre-amplifier chain [1] is used for high capacitance sensors as an example in this chapter. The large sensor capacitance increases the charge uncertainty on the sensor node after reset. Due to the fact that a large capacitance enhances the reset noise component of the system's signal-to-noise-ratio, innovative techniques need to be incorporated in readout arrays to increase the signal-to-noise ratio. Figure 5.5(a) portrays the block diagram of the used readout array from [1] for high capacitance sensors. The



(a)



(b)

Figure 5.5 (a)Preamplifier/amplifier chain block diagram. (b)Folded cascode amplifiers schematic for each stage. [1]

readout circuitry consists of a pre-amplifier and amplifier. Each stage is a folded cascode with a class AB push-pull output stage, as shown in Figure 5.5(b). A large capacitor along with a leakage current source is utilized for electronic simulation of the detector. The detector capacitance is simulated by a 100nF capacitor. The capacitor used for simulating the detector has a capacitance that is 5 orders of magnitude more than that of capacitors used in previous techniques. The sensed signal is the voltage stored on the sensor capacitance.

Capacitive matching is also used at the amplifier input to reduce the reset noise at the sensor node. This method calls for an additional capacitor to be placed at the input of the pre-amplifier/amplifier chain. The capacitance of this capacitor must be equivalent to the capacitance seen at the pre-amplifier input. The capacitance seen at the input of the pre-amplifier is the feedback capacitor, C_{f1}, multiplied by the open loop gain of the pre-amplifier, A_1. The capacitor referred to as C_{match} in Figure 5.5(a) is used at the amplifier input for the purpose of matching, in order to increase the signal-to-noise ratio at the input of the pre-amplifier/amplifier chain. The matching capacitor is chosen to be equal to the product of C_{f1} and A_1. By using capacitive matching, the principle of impedance matching is followed, which maximizes the signal power transmission into the amplifier system.

5.4. Energy Modeling and Lifetime Analysis

Based on the design schematic of a sensor support system platform shown in Figure 5.1 and the designed scheme of TDMA operations in the network, we integrate various fine-grained energy models to evaluate energy consumption for the whole system. Moreover, the system-level energy model is used to evaluate the system lifetime in the experi-

ments and to demonstrate how single elements of the system chain impact overall performance.

5.4.1. System-level Energy Modeling

The proposed energy model estimates energy consumption from a system-level point of view, i.e. considering the energy consumption from the use of all system devices (including software and hardware components) within the framework of our TDMA-based protocol control scheme. Therefore, the system energy model for which the system is running at various protocol modes can be generalized to

$$E_m = E_{sens} + E_{amp} + E_{ADC} + E_{mcu} + E_{tran}, \tag{5.1}$$

where E_{sens}, E_{amp}, E_{ADC}, E_{mcu}, and E_{tran} respectively denote the energy consumed by the sensor, pre-amplifier/amplifier chain, analog-to-digital converter, microcontroller, and transceiver when the system runs in different protocol modes — i.e., transmission, reception, and idle modes. Figure 5.6 lists all notations with their descriptions for system-level energy modeling as well as experimental values that are being used in our tests, where $t_{ADC\text{-}on} = N_{pt} \cdot t_{ADC\text{-}c}$, $t_p = N_{clk}/f_{clk}$, and $t_{tran\text{-}on} = t_{tran\text{-}st} + M/R$.

Symbols	Description	Values	Symbols	Description	Values
V_{cc}	system supply voltage	$3.3V$	t_s	TDMA slot time	$50ms$
R	transceiver data rate	$500Kbps$	t_p	CPU processing time	$4\mu s$(RX) $12.3\mu s$(TX)
f_{clk}	processor clock frequency	$26MHz$	$t_{tran\text{-}st}$	transceiver startup time	$200\mu s$
M	data length	$8bits$	$t_{tran\text{-}on}$	transceiver turn-on time	0.22ms
N_{ck}	# of cycles for processing	320(RX) 105(TX)	$t_{ADC\text{-}c}$	ADC conversion time/sample	$58.4\mu s$
N_{pt}	# of sensed samples	1 ~ 5000	$t_{ADC\text{-}on}$	ADC turn-on time	$58.4N_{pt}\mu s$
t_{sens}	sensing time per sample	$1\mu s$	$t_{tmr\text{-}on}$	timer turn-on time	$50ms$

Figure 5.6 Table of notations and experimental values for system-level energy modeling

Specifically, when we consider the time to use various devices on a sensor node platform, the energy consumption for which the system is in either transmission, reception, or idle mode is modeled as

$$E_m = N_{pt} \cdot [(P_{sens} + P_{amp})t_{sens} + P_{ADC}t_{ADC\text{-}on}] + P_{tmr}t_{tmr\text{-}on} + P_{proc}t_p + P_{tran}t_{tran\text{-}on} + P_{lp}t_{lp} \quad . \tag{5.2}$$

Here, $N_{pt} \cdot [(P_{sens} + P_{amp})t_{sens} + P_{ADC}t_{ADC\text{-}on}]$ is considered as the energy consumption for all sensing elements. $P_{lp}t_{lp}$ denotes the energy consumption for the processor when the processor stays in its low-power state. For example, in transmission (reception) mode, the platform will turn to a low-power state after finishing its processing and communicating tasks, and therefore $t_{lp} = t_s - (t_{tran\text{-}on} + t_p)$. However, in idle mode, each sensor node platform stays in a low-power state during the whole time slot without executing processing and communicating tasks. That is, $t_p = t_{tran\text{-}on} = 0$ and $t_{lp} = t_s$ when $E_m = E_{idle}$. We define the sensing time, t_{sens}, as the time that a sensor and a sensor support amplifier complete a signal processing operation for one sample. For example, our designed amplifier takes approximately 1μs to complete such a signal processing operation for one sample in terms of its RC time constant. Note that the timer device is turned on all the time due to the execution of run-time TDMA protocol control and synchronization, and therefore $t_{tmr\text{-}on} = t_s$.

5.4.2. System Lifetime Analysis

We analyze the average system lifetime according to the system-level energy models described above. The system lifetime is represented in Eq. , where the lifetime analysis is derived based on the pre-scheduled TDMA-based communication operations along with the characteristics of hardware and software components on each sensor node. According

to the communication protocol discussed in Section 5.2 along with fault tolerance, all operations within a TDMA frame (T_{fr}) are executed periodically, frame-by-frame, until the system fails (i.e., all nodes run out of energy). Thus, in our lifetime model, we first estimate the average energy consumption for each sensor node within a single communication time frame, and then based on this estimation, the average lifetime of the overall system (i.e., T_{sys}) can be estimated in terms of total energy stored in the available batteries for each sensor node

$$T_{sys} = \frac{1}{N} \cdot \sum_{n=1}^{N} \frac{E_{bat}(n)}{P_{fr}(n)},$$

$$\text{and } P_{fr} = \frac{1}{t_s}\left[N_{tx} \sum_{m \in tx} E_m + N_{rx} \sum_{m \in rx} E_m + N_{idle} \sum_{m \in idle} E_m \right] \tag{5.3}$$

where N denotes the number of sensor nodes in the system; $E_{bat}(n)$ denotes the total energy stored in a given battery and P_{fr} denotes the power consumption for each TDMA time frame for node n, respectively; N_{tx}, N_{rx}, and N_{idle} denote the number of transmission (tx), reception (rx), and idle (idle) mode occurrences in a T_{fr}, respectively.

The energy model shown in this section can be used to estimate energy consumption in both the synchronization stage and the normal communication stage. However, the time of the synchronization stage is relatively short (e.g., $t_{sens} = t_{ADC\text{-}on} = 0$, and $t_{tmr\text{-}on}$, t_p, $t_{tran\text{-}on}$ are active at most once per T_{fr}) compared to the time spent for the normal communication, so the energy consumption during this stage can be omitted with minimal loss in accuracy. Our experiments corresponding to the use of system-level energy modeling outlined in this section will be demonstrated in Section 5.5.

5.4.3. Fidelity Analysis

We calculate the fidelity of the system-level energy model based on the results of simulated versus measured energy consumption. Our reason for using the metric of fidelity here is that the design of sensor network systems depends on various cross-layer configurations, and therefore, it is difficult to obtain perfect accuracy of simulated energy for the purposes of fast energy consumption evaluation and associated design space exploration considerations. For this estimation, we use the fidelity metric to determine the trend of energy estimation based on our energy model as an alternative to physical measurement. In this way, the proposed energy model can be shown, in a quantitative way, to have high accuracy when compared to actual measured results. Thus, the model can be applied with high confidence to evaluate our sensor support system — and in particular, to compare alternative system configurations — in terms of lifetime for arbitrary network size. Here, by network size, we mean the number of sensor nodes in the network.

To measure the accuracy of our approach to system-level energy modeling, we use the estimation fidelity metric defined by

$$fidelity = \frac{2}{n(n-1)}\left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} f_{ij}\right), \tag{5.4}$$

where $f_{ij} = 1$ if $sign(M_i - M_j) = sign(S_i - S_j)$, and $f_{ij} = 0$ otherwise. Here, M_i (M_j) denotes the measured results and S_i (S_j) denotes the simulated results, respectively.

For experimenting with fidelity calculation for the proposed system-level energy modeling approach, we generated 20 testing points for configuring the sensor node platforms with different combinations of supply voltages. The supply voltages across the different platforms in our experiments could be homogeneous or heterogeneous. For each

configuration of supply voltages, we observed the energy consumption variation (simulated results from the energy model versus measured results) on the nodes corresponding to the voltage changes. Our fidelity experiments are summarized in Figure 5.7, where the fidelity value is determined to be 0.91. Note that a fidelity value of 1 corresponds to perfect fidelity.

5.5. Experiments

5.5.1. Experimental Setup

Demonstration of the discussed sensor support system for line-crossing recognition will be presented in the next chapter. Here, based on the use of pre-amplifier/amplifier chain in [1], we construct a simulation-based experiment for estimating the system lifetime. An arbitrary number of sensor nodes in the system can be chosen for this simulation-based experimental environment. According to the fidelity analysis in the previous section, the analyzed results from this simulation-based experiment are similar to the results that we observed when we implemented the system in actual hardware with the same configurations.

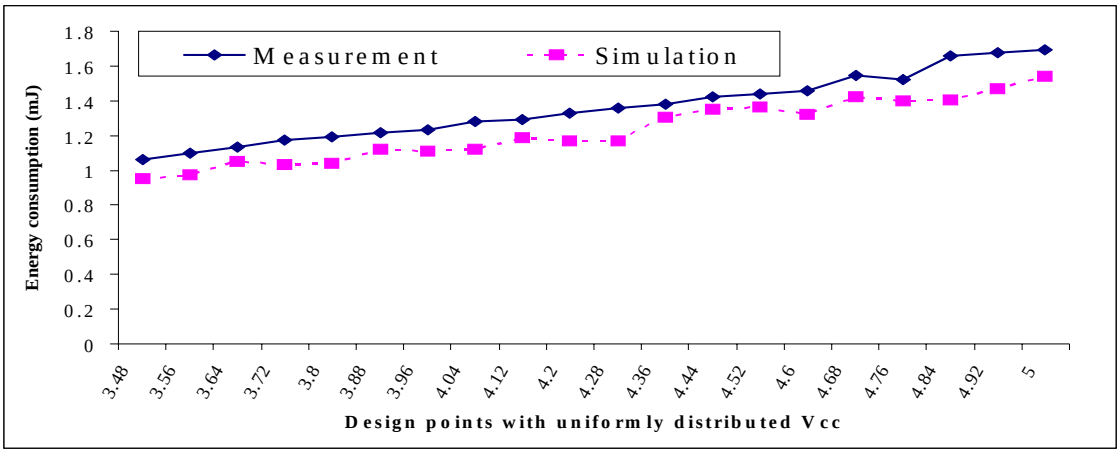


Figure 5.7 Fidelity analysis for energy modeling

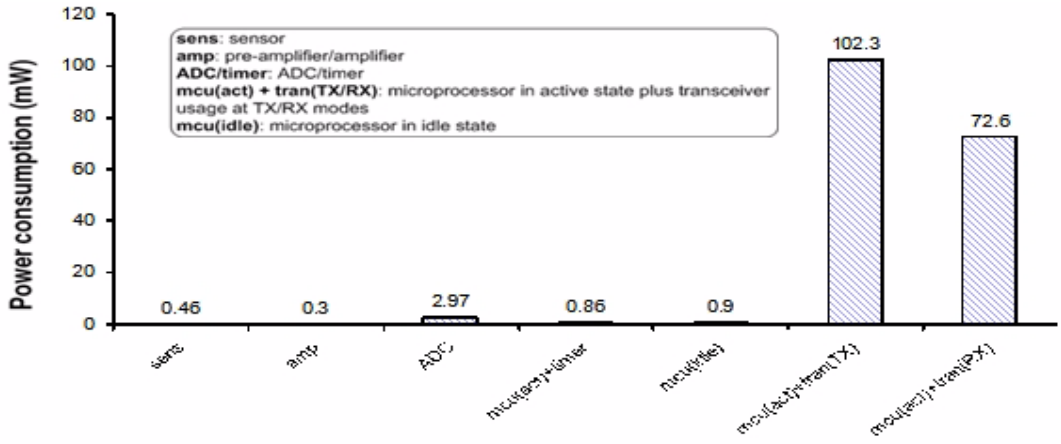
For the experimental configurations in the thesis, we set up 10 sensor nodes for the integrated sensor support system such that each sensor node takes 0.5 s to pass around information across the whole system. That is, in this configuration, we set $T_{fr} = 0.5$ s, $N = 10$, $t_{slot} = 50$ ms, $N_{tx} = N_{rx} = 1$ and $N_{idle} = 8$. Based on these system-level configurations, we conduct simulations for estimating the system lifetime of the integrated sensor support system in the following context.

The chosen pre-amplifier/amplifier chain referred from [1] and designed in 0.13μm CMOS8RF IBM technology using the Cadence design package (Assura). The circuit has a gain of 63 dB.

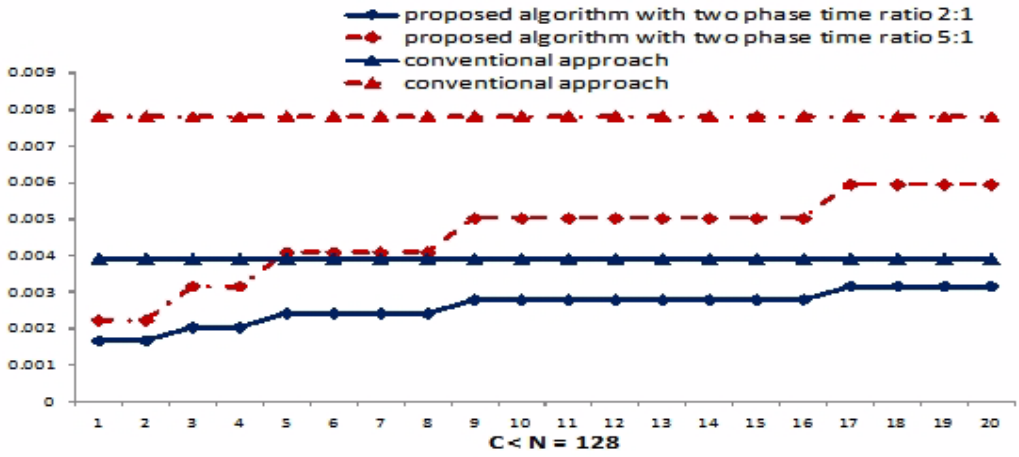
The input to the pre-amplifier/amplifier chain from the capacitive sensor is simulated by a pulse with a pulse width of 0.5μs and amplitude of 1mV. The pulse contains a DC offset of 1.8V to provide the necessary input DC bias voltage for the signal processing chain. The amplifier has a maximum equivalent output noise of 280 $\mu V/\sqrt{Hz}$. The signal-to-noise ratio for the pre-amplifier/amplifier chain is about 5300 V/V. This value is 80 times larger than the signal-to-noise ratio of previously-reported signal processing chains [1].

5.5.2. Simulation Results for System Lifetime Analysis

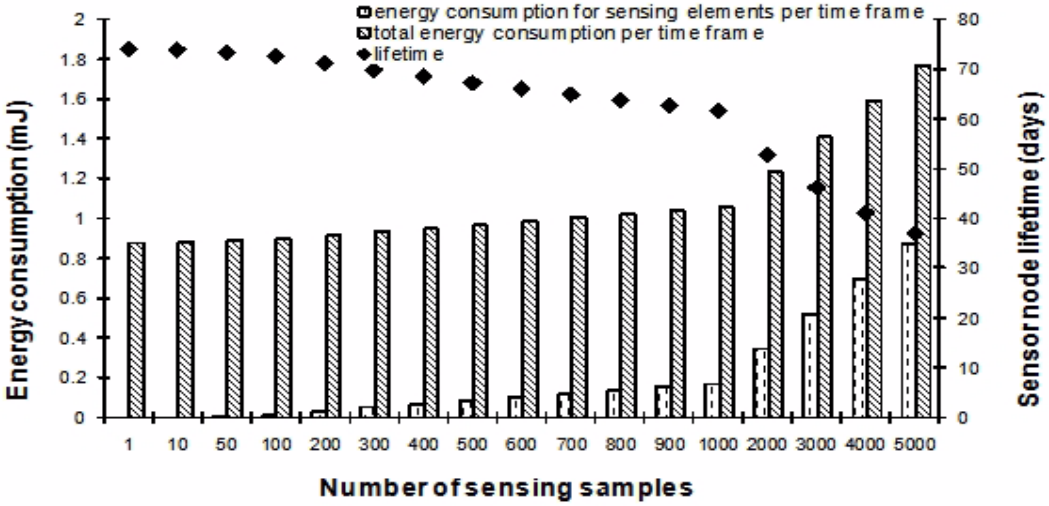
In our experiments for lifetime analysis, we consider the case where all sensor nodes in the system execute tasks for line-crossing recognition frame-by-frame periodically without additional power-off mechanisms. As a practical example, Figure 5.8(a) first presents power consumption comparison between each single element used in the sensor node platform, where the electrical specifications except for the designed amplifier are obtained from [12] and [91]. Note that in [91], we only consider the power consumption of the sen-



(a)



(b)



(c)

Figure 5.8 (a) Power consumption comparison among all devices on a sensor node platform. (b) Energy consumption comparison depending on *C*. (c) Energy consumption and lifetime comparison, where simulated results are calculated in terms of the specifications in [1], [12], [91].

sor part for our experiments according to its provided gain information. In Figure 5.8(a), it shows explicitly that the communication element (i.e., the transceiver) dominates overall power consumption on a sensor node platform. Therefore, we need to consider the lifetime problem at the system level so that designs of single elements become appropriately tailored for integration into the overall system. Figure 5.8(b) shows how energy depends on the number C of nodes needed to reach consensus, where N is assumed to be 128. In Figure 5.8(b), we use two experiments to compare energy consumption for data communication between our proposed distributed algorithm with two phase operation and a conventional approach with single-phase operation, where the time ratios of the proposed algorithm spent in phase 0 and phase 1 are assumed to be 2:1 and 5:1, respectively. We observe from Figure 5.8(b) that the longer the duration of system operation, the more the potential for energy savings by the proposed algorithm (compared to the single-phase approach) for a smaller value of C. Note that the data to be communicated in a conventional approach only depends on N and is independent of C.

To consider the lifetime problem at the system level, in Figure 5.8(c), we first demonstrate a comparison between the total energy consumption and the energy consumption for sensing elements on a node platform. This comparison is done based on energy consumption values during a single TDMA time frame. With this system-level consideration, we reduce the transceiver use time by minimizing the data that is required to be communicated across sensor nodes. This minimization is based on our proposed algorithm for the targeted threat detection application. Moreover, a TDMA-based communication protocol is used to schedule the tasks among processing, communicating, and idling so that the pro-

cessing and communicating elements can remain in idle states when their tasks are complete.

As the statistical results show in Figure 5.8(c), once the number of required sensing samples increases, the energy consumption for sensing elements increases and significantly impacts overall energy consumption for the system. When using a traditional signal processing chain with lower signal-to-noise ratio for processing sensing signals, multiple sensing activities are required under different circumstances with different noise levels, because noise interference is not predictable during the system run time. Many false sensing detections may occur when producing samples from the received signals. As a result, it is difficult to formulate a general, exact solution for the number of required sensing samples.

Using the amplifier with high signal-to-noise ratio solves this problem when it is integrated into the system. Regardless of interference at any noise level, the noise will be canceled through the correlated double sampling technique in our design. Therefore, only one sensing task is required for producing each sample from the received signal.

We also demonstrate the system lifetime comparison in Figure 5.8(c) in terms of the number of sensing samples required in our system. In this lifetime experiment, a $3.3V \cdot 950mAh$ capacity lithium battery is provided in each sensor node platform so that a practical result in terms of days of useful operation is shown in the figure. From the lifetime results, we observe that by using the designed amplifier with high signal-to-noise ratio, the energy consumption for sensing elements is not critical anymore because there is no redundant sensing activity executed for maintaining the system functionality, and one sensing activity only consumes 0.02% of the overall energy consumption within a TDMA

time frame. In this case, our system can last 75 days since all sensor nodes are powered-on.

However, without using the optimized amplifier, the system lifetime drops since we require too many undesirable sensing activities. For example, in the worse case statistical results, the system lifetime will drop to approximately 37 days when 5000 sensing samples within a TDMA time frame are required. In this case, the energy consumption for sensing elements may have equivalent influence to computation and communication elements on overall energy consumption - for example, 49.2% of overall energy consumption is consumed by the sensing elements in our system when a conventional amplifier is used. Note that in many practical applications where tamper-resistant deployment is important, the size of each sensor node has to be limited so that the nodes cannot easily be found or manipulated. In such cases, batteries with very limited capacity must be used, and the system lifetime may drop significantly (e.g., well below a single day) if redundant sensing activities cannot be avoided.

Chapter 6. Case Study: Software and Hardware Implementations for the DLCR System

In this chapter, we present a complete system design flow for implementing the distributed application for line-crossing recognition (DLCR) on the platforms that we have designed. Our developed platforms have been designed carefully to provide features of low power usage and small size. In the previous chapter, we have presented the theoretical development of the DLCR algorithm with the motivation for reducing computation and communication energy consumption. In addition, we have also shown the developments of a low power protocol for data fusion and fault tolerance, and system-level energy modeling and analysis. Since DLCR is a sensor support system, we have analyzed from a statistical basis that single elements of the system chain individually have small impact on overall performance, and we have examined how to consider the design of a single element as a system-level problem when it is integrated into the overall system chain. We also present an asynchronous handshaking approach for providing synchronization between the transceiver and digital processing subsystem in a sensor node. This provides a general method for achieving such synchronization with reduced hardware requirements and reduced energy consumption compared to conventional approaches, which rely on generic interface protocols.

In this chapter, we will demonstrate three designed platforms and corresponding implementations of our DLCR system based on these platforms. These three platforms include a microcontroller (MCU)-based design, field programmable gate array (FPGA)-

based design, and application-specific integrated circuit (ASIC)-based design. Figure 6.1 shows the complete system design flow that encompasses all three designed platforms. Summaries of these chapters have been published in [63], [64], and [79].

6.1. MCU-based Design

There are two types of platforms developed for our MCU-based designs. These are based on the Texas Instruments CC1110 [87] and CC2430 [88] devices, and each platform supports a customized miniature antenna [79]. This antenna operates at 916MHz center frequency for the CC1110-based platform, and at 2.4GHz center frequency for the CC2430-based platform. For example, for the antenna with a center frequency of 916MHz, with a return loss of 20dB and bandwidth of 13MHz, the reflection coefficient at

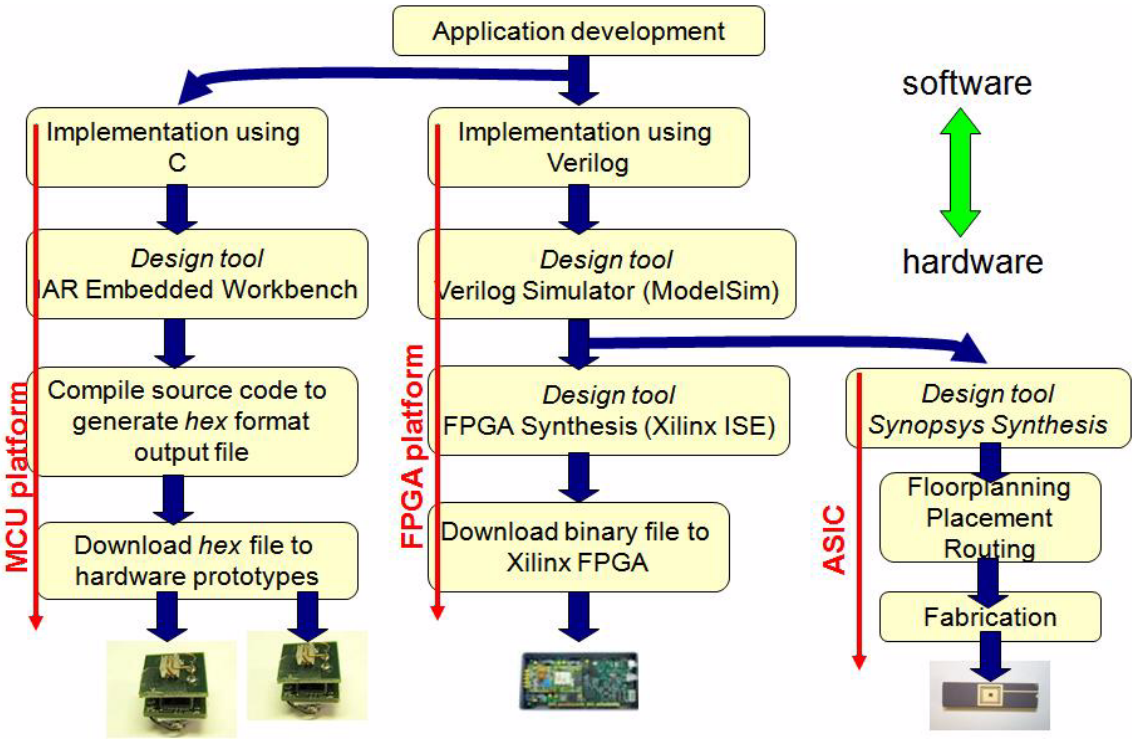


Figure 6.1 System design flows.

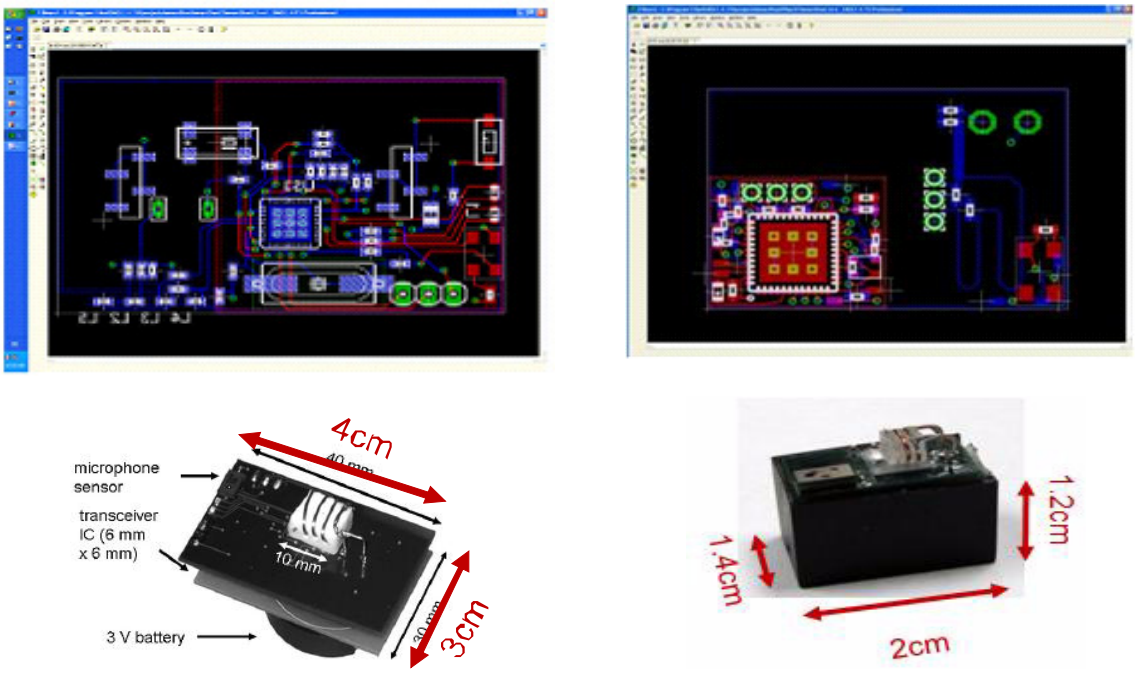
the feeding point of the antenna is measured through the Agilent Network Analyzer (PNA Series 8364B).

To implement the DLCR algorithm — based on the algorithms and protocols discussed in the previous chapter and shown in Figure 5.3 — we employed an emerging family of system-on-chip (SoC) devices (e.g., see [87] [88]) as an integrated, single-chip solution for performing computation and communication tasks as well as an acoustic sensor [91] for sensing tasks. In addition to the main algorithm developed for DLCR, a simple program is also developed for monitoring digitized sensed values using an acoustic sensor [91] in the experiment. Figure 6.2 shows the implemented functional module for sampling on the CC2430 embedded platform. Here, in order to confirm that the sensed data is coming from the real-time sensed signal instead of noise, consecutive sensed samples, which are already converted to the corresponding digital values, are stored and compared with a user-defined threshold value (e.g., plus_ADC_OFFSET and minus_ADC_OFFSET in Figure. 6.2). Such a threshold value is adjustable at design time based on the present environment condition. A *sense* flag, which represents whether a real-time signal is sensed, will be set to *true* if the distance between two consecutive sensed samples is greater than the threshold value.

```
void sampling() {
        oldSample = sample;
        sample = halAdcSampleSingle(ADC_REF_1_25_V, ADC_8_BIT, ADC_AIN1);
        if (!sense) {
                if ((sample - oldSample >= plus_ADC_OFFSET) || (sample - oldSample < minus_ADC_OFFSET))
                        sense = TRUE;
        } else {
                if ((sample - oldSample < plus_ADC_OFFSET) || (sample - oldSample > minus_ADC_OFFSET))
                        sense = FALSE;
        }
}
```

Figure 6.2 A program for sampling sensed data through digital I/O on the experimental CC2430-based platform.

The designs of MCU-based sensor node prototypes are shown in Figure 6.3. The designs include 4-layer Printed Circuit Boards (PCB) with integrations of microcontrollers, acoustic sensors, customized miniature antennas, and batteries. For example, Figure 6.3(a) demonstrates the designed platform with the CC1110 microcontroller and 916MHz customized antenna. Figure 6.3(b) demonstrates the designed platform with the CC2430 microcontroller and 2.4MHz customized antenna. Compared to other designs of sensor node platforms, e.g. [29], our designed platforms have the minimal size as shown in the figure. This is gained from the customized miniature antenna since the antenna size is one of the major limitations in miniaturizing wireless communication equipment [79].



(a) (b)

Figure 6.3 MCU-based sensor node prototypes

6.2. FPGA-based Design

As an intermediate step towards the development of and application-specific digital integrated circuit that is fully specialized for our targeted sensor nodes, a field programmable gate array (FPGA)-based platform is designed and used to prototype the structure of the targeted ASIC component. We use the FPGA to implement the digital processing functionality that controls the sensor node. In addition, in this prototype, we use a commercial transceiver for wireless communication, and an off-the-shelf acoustic sensor for sensing tasks. This prototype system provides the complete functionality for a sensor node in a distributed sensor system application.

When building a sensor node by combining various subsystem platforms, as described above, synchronization must be handled across the different platforms, and such synchronization requires special care when the platforms employ separate clocks. The conventional approach to this synchronization problem is that both platforms negotiate with each other via generic synchronization protocols, such as the universal asynchronous receiver-transmitter (UART) or serial peripheral interface (SPI) protocols. To run these protocols, the platforms that are being interfaced require additional hardware requirements, which may increase their size and energy usage. However, without generic synchronization protocols, such as UART and SPI, synchronous interfacing of separately-clocked platforms in a sensor node prototype system requires a master clock that is potentially much faster than other clocks in the system. Such a synchronous interfacing approach may cause major problems due to clock skew.

Thus, in this thesis we introduce an asynchronous approach based on a light-weight handshaking scheme for interfacing two platforms for sensor node prototyping when the

platforms have separate clocks. Unlike generic interfacing methods, such as UART and SPI, this approach is specialized to the specific needs of sensor node integration, and therefore involves less complexity, and therefore less hardware and energy cost. We demonstrate this approach by interfacing an FPGA-based digital processing platform to an off-the-shelf transceiver platform. We show in our experiments that through our interfacing approach, the FPGA and transceiver platforms interact asynchronously in a robust manner.

The proposed asynchronous interfacing approach can be generalized to any platform pair that conforms to a master-slave structure. In our prototype implementation, the master platform is the field programmable gate array (FPGA) platform, and the slave platform is the transceiver platform. In our design, we use separate channels for the data, request, and acknowledgement signals. The FPGA platform runs as the main processor to deal with computation and control tasks, and the transceiver platform is controlled by the FPGA to execute communication tasks (i.e., to transmit and receive signals through the wireless channel). Therefore, in accordance with the TDMA-based protocol described previously, the FPGA determines and sets control signals (e.g., requests and acknowledgments) to the transceiver for executing tasks in transmit (TX) and receive (RX) modes, or for changing its status to the idle mode.

Without loss of generality, the scenario of the proposed asynchronous approach between two separate clock platforms in TX mode is as follows. Whenever the FPGA is running in the TX mode and is ready to transmit, it sends a TX signal (W-TX) to the transceiver platform. The transceiver platform receives the W-TX (R-TX) signal, enters TX mode, acknowledges back (W-Ack) to the FPGA platform, and then monitors the request

channel from the FPGA. Once the FPGA receives W-Ack (i.e., notification that the transceiver platform is ready in TX mode), it places a data bit on the data channel and changes its request signal (C-Req). Whenever the transceiver platform detects the C-Req signal, it reads a data bit from the data channel (R-Data), and then sends an acknowledgement back to the FPGA (W-Ack). Every time the FPGA receives a W-Ack in this way, it can place another data bit on the data channel and repeat the handshaking process described above until it successfully passes all the data bits that it needs to send at a given point in time. Once the FPGA passes all the data bits, the transceiver platform may enclose the data bits in appropriate packets and transmit the resulting packets through the wireless channel. Figure 6.4 illustrates the state transition graph for the handshaking schemes associated with control signals in TX and RX mode.

In this prototype development, we use the Xilinx Virtex-4 FPGA [94] as the control core for a sensor node. That is, the embedded software targeted to the microcontroller in the first prototype is replaced by the FPGA. Therefore, in the second prototype, we only use the transceiver part of the CC1110 device [87]. This prototype implementation demonstrates a complete sensor node in our distributed sensor application with custom logic used

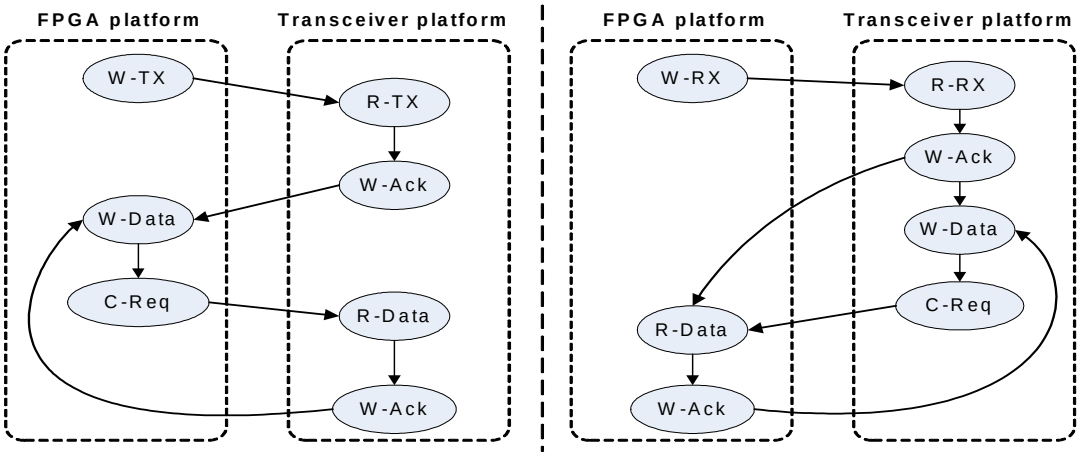


Figure 6.4 State transition diagrams for handshaking in (a) TX mode and (b) RX mode.

to implement all control functionality. We have verified its correct operation (including both communication and computation) in conjunction with other nodes in the system.

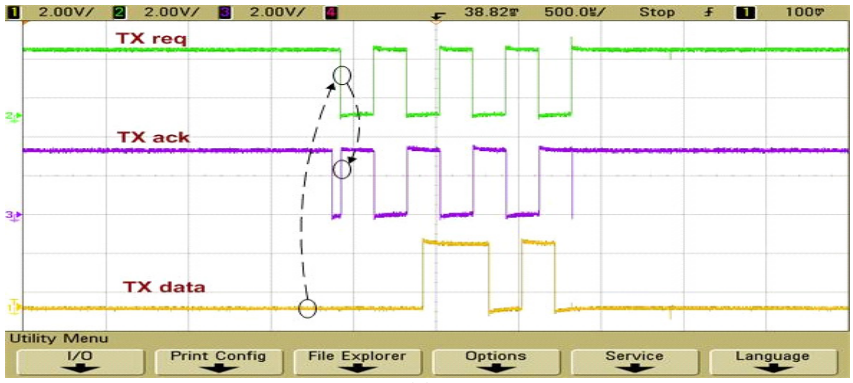
All control and associated data processing for the application has been modeled and implemented in Verilog, and our Verilog implementation has been synthesized onto the targeted FPGA device to demonstrate the sensor node behavior. For example, if the sensor node is ready to transmit data, the FPGA will first set up the data and then send a request signal to the transceiver. Conversely, each time the transceiver receives a request along with the associated data from the FPGA, it will send an acknowledgement back to FPGA to complete each transaction.

To interact between the FPGA and the transceiver for a sensor node using our proposed asynchronous approach, the handshaking protocol mentioned previously is modeled and implemented in Verilog for the FPGA platform, and in C for the CC1110 microcontroller subsystem. Figure 6.5 shows snapshots captured from an Agilent DSO 6041A oscilloscope that depict handshaking interactions between the FPGA device and the CC1110 transceiver subsystem in either transmit (TX) and receive (RX) mode.

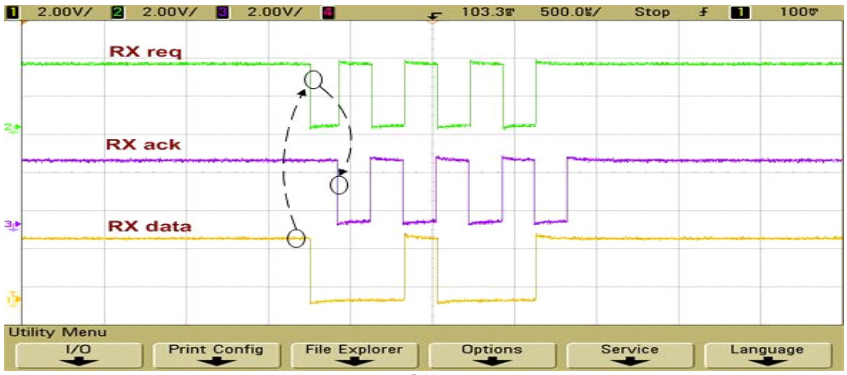
6.3. Design of Hardware Modules for DLCR

The DLCR algorithm and the associated TDMA protocol is implemented in the FPGA using Verilog-HDL so that the implemented digital circuits can be tested and verified before the final synthesis and implementation onto ASICs. There are totally 8 major modules and 12 sub-modules to be designed and implemented for the DLCR algorithm and the associated TDMA protocol. Figure 6.6 shows the corresponding schematic design, and the functionality of all of the modules are illustrated in the following context:

ReceivePacket: the functionality of this module is to receive the data of a transmitted packet from another node in a bit-serial manner and reconstruct the packet with the *PacketFilter* module. In our experiments, a 20KHz off-chip clock source is used, and the



(a)



(b)

Figure 6.5 Handshaking interactions between the FPGA and CC1110 platforms in (a) TX and (b) RX mode.

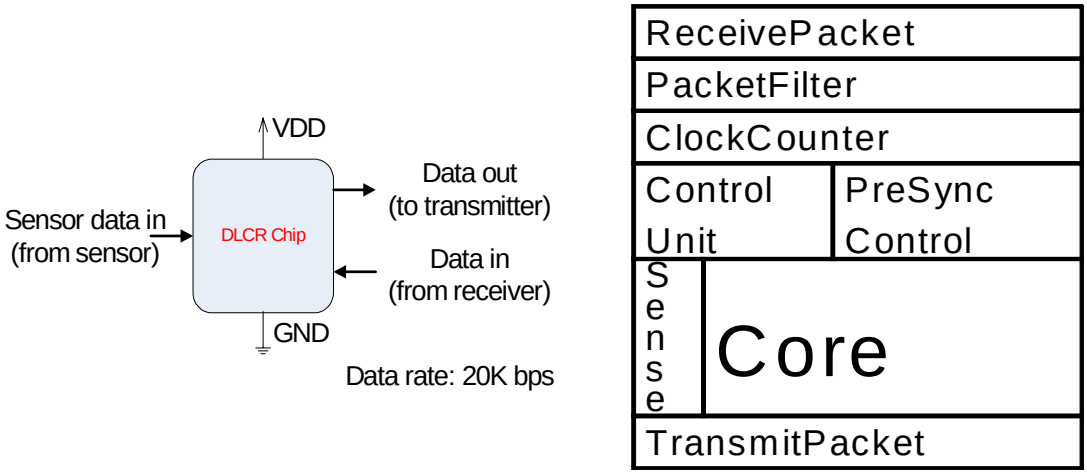


Figure 6.6 Schematic design for the DLCR algorithm and the associated TDMA protocol.

associated data rate is 20K bps. Thus, for receiving a data packet with M bits, the receiving time is about $M/(20K)$ seconds.

PacketFilter: the functionality of this module is to validate the received data packets. For data validation, it first confirms whether the packet was received from the node's neighbor by checking the identifier field (i.e., ID) of a received packet. If the packet was not received from its neighbor, then the packet is discarded. Also, this module confirms whether the received packets have the desired format, where the format is defined experimentally (illustrated in Figure 6.6). If the desired format is not matched, then the packet is discarded. Only a packet with a correct ID and format will be processed further.

PreSyncControl: the functionality of this module is to control, schedule and launch the different operations that take place before all of the nodes are synchronized. When a node is turned on, it will be assigned to a pre-defined initial TDMA schedule. By following this initial schedule, a node will stay mainly in a receiving mode and broadcast an initial packet to all other nodes at a desired transmitting time slot. The node stays in such a *pre-synchronization stage* until it receives an initial packet from its neighbor nodes. Then, the node will change its status to the synchronization stage and reschedule its transmitting and receiving time slot accordingly.

ControlUnit: as described in the previous chapter, a node will enter a communication stage when all of the nodes in the system are synchronized with one another. The Control Unit module is used to control, schedule and launch the different operations that are taking place when all of the nodes in the system are synchronized. At this communication stage, a node will follow its schedule to transmit (receive) packets to (from) its neighbor node at a desired a TDMA time slot. The fault tolerance scheme is also implemented

in this module. That is, a node will readjust its schedule accordingly based on the scheme, developed in section 5.2.3, whenever its neighbor fails.

Core: This module implements the main DLCR algorithm, which has been introduced in section 5.2. The DLCR algorithm is implemented in this module according to its operation mode (transmitting or receiving) in a TDMA time frame. This module controls and sets internal registers, and internal flags according to the input information from the associated sensor, received packets, and the fields of packets that are to be transmitted. It has interfaces with other modules and also indicates when a subject has crossed the boundary region that is monitored by nearby nodes. This module can also be replaced with modules that implement different algorithms.

Sense: the functionality of this module is to determine and set internal flags according to whether incoming signals are due to an approaching subject. Assume that an off-chip sensor is used in the experiment and connected to the sensor input pin of the chip. The sensor should continuously sense and send signals via an analog-to-digital (ADC) device setup. The digitized signal is first compared to a given signal threshold and then given to the chip via the sensor input pin.

TransmitPacket: the functionality of this module involves obtaining packets to be transmitted from the *Core* module and transmitting the bits of the packets in a bit-serial manner. Similar to the *ReceivePacket* module, a 20KHz off-chip clock source is used in our experiments, and the associated data rate is 20K bps. Thus, for transmitting a data packet with M bits, the transmission time is also about $M/(20K)$ seconds.

ClockCounter: the functionality of this module is to count clock ticks to drive a TDMA schedule that is used by the *ControlUnit* and *PreSyncControl* modules. An exter-

nal clock source, which generates the reference clock ticks, is connected to the input port of this module. For example, for a 1 second TDMA time frame, an internal register will be increased 20,000 times in this module if a 20KHz clock source is used.

Figure 6.7 demonstrates a 4-node DLCR system integration that consists of our MCU-based and FPGA-based platforms. In the FPGA-based platform shown here, a Xilinx Virtex-4 FPGA [94] is used for executing the DLCR algorithm and the associated TDMA protocol. A pair of LINX on-off keying (OOK)-based transmitter and receiver devices [43] is used to handle real-time packet stream processing.

6.4. ASIC Design for DLCR Digital Subsystem

Follow the ASIC design flow shown in Figure 6.1, the designed DLCR system using Verilog-HDL is first to be synthesized with the use of the Synopsys Design Compiler [92] and the desired technology library for standard cells (e.g., see [84], [89]). The synthesized Verilog code for the digital DLCR implementation is verified via a well-developed benchmark suite for post synthesis simulation. Then, we use Cadence SOC Encounter [85] to

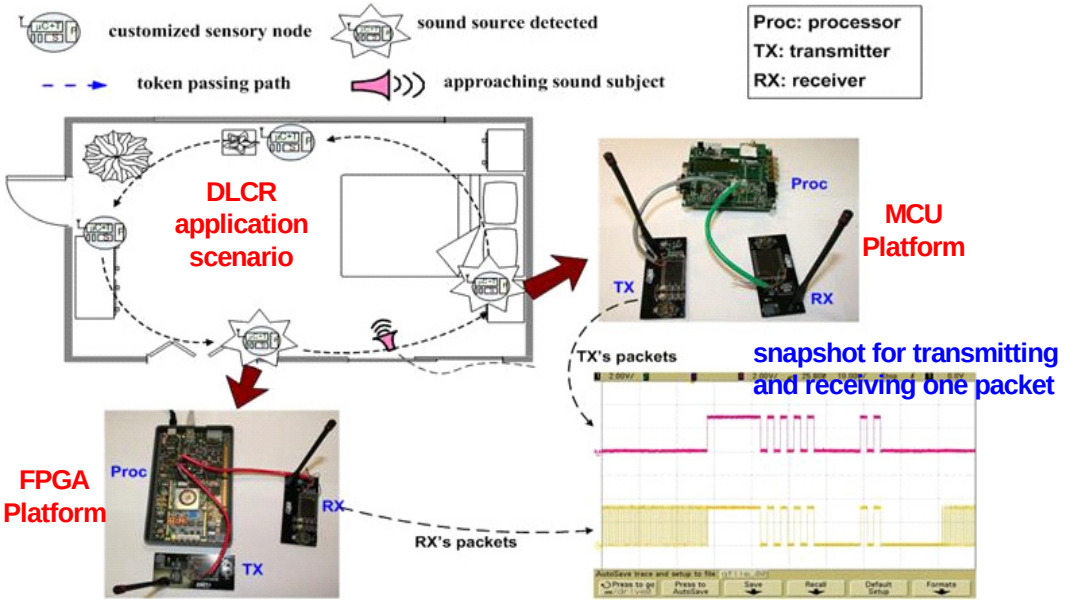


Figure 6.7 DLCR system integration with MCU-based and FPGA-based platforms.

layout the chip following a standard ASIC design flow as shown in Figure 6.8. The post layout process, including design rule checking and verification, is carried out using Cadence Custom IC Design Tools: Virtuoso [86].

Figure 6.9 shows the design and layout snapshot for the designed digital ASIC for DLCR using the MOSIS AMI 0.5 micron process. Figure 6.10 shows the corresponding snapshot for another version that we developed using the IBM 0.13 micron process. Figure 6.11 shows the fabricated chip for DLCR using the MOSIS AMI 0.5 micron process. This MOSIS chip has a size of 2.4 mm^2, uses about 30,000 transistors, and consumes 1.2 mW of power with a 5V supply voltage. The IBM chip shown in Figure 6.10 has a size of

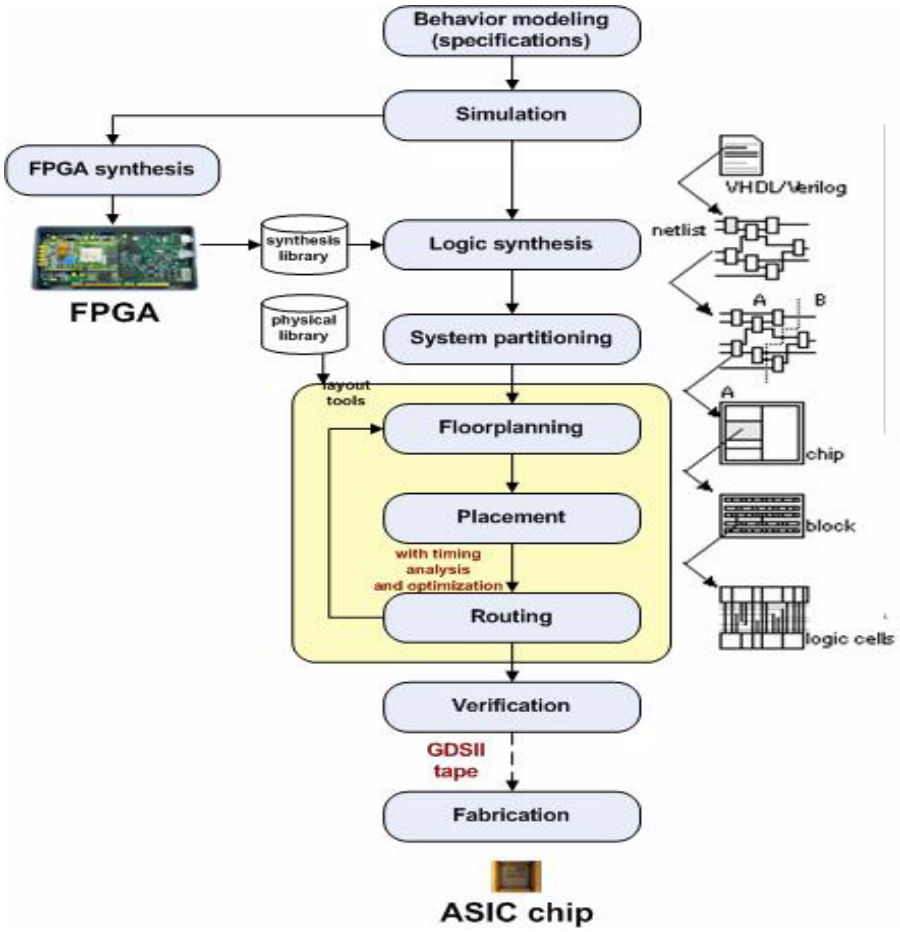


Figure 6.8 ASIC design flow.

1.0 mm^2, also uses about 15,000 transistors, and consumes 10.7 µW of power with a 1.32V supply voltage.

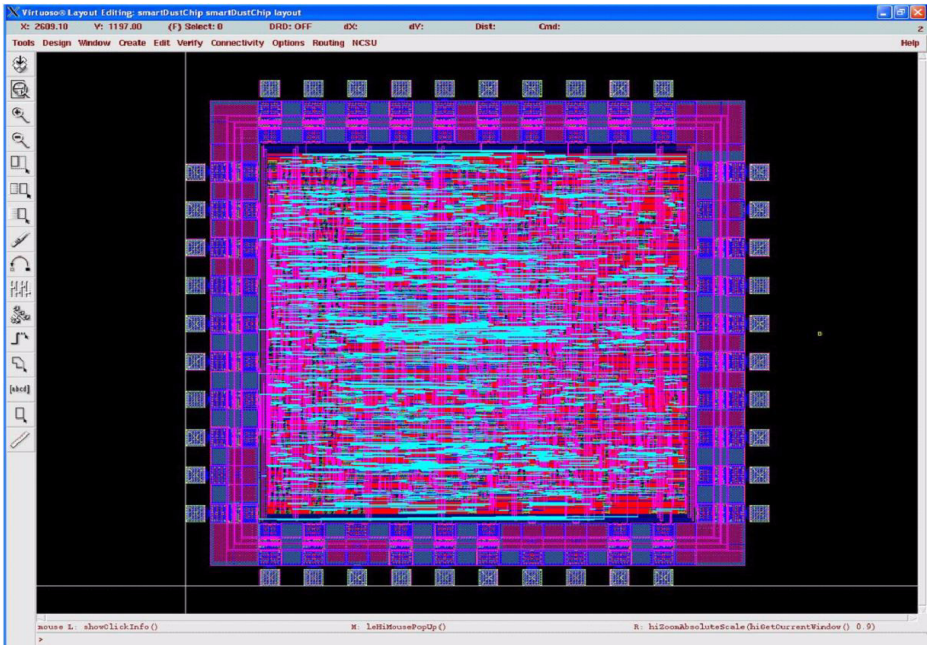


Figure 6.9 The design and layout for the DLCR digital ASIC using the MOSIS AMI 0.5µm process.

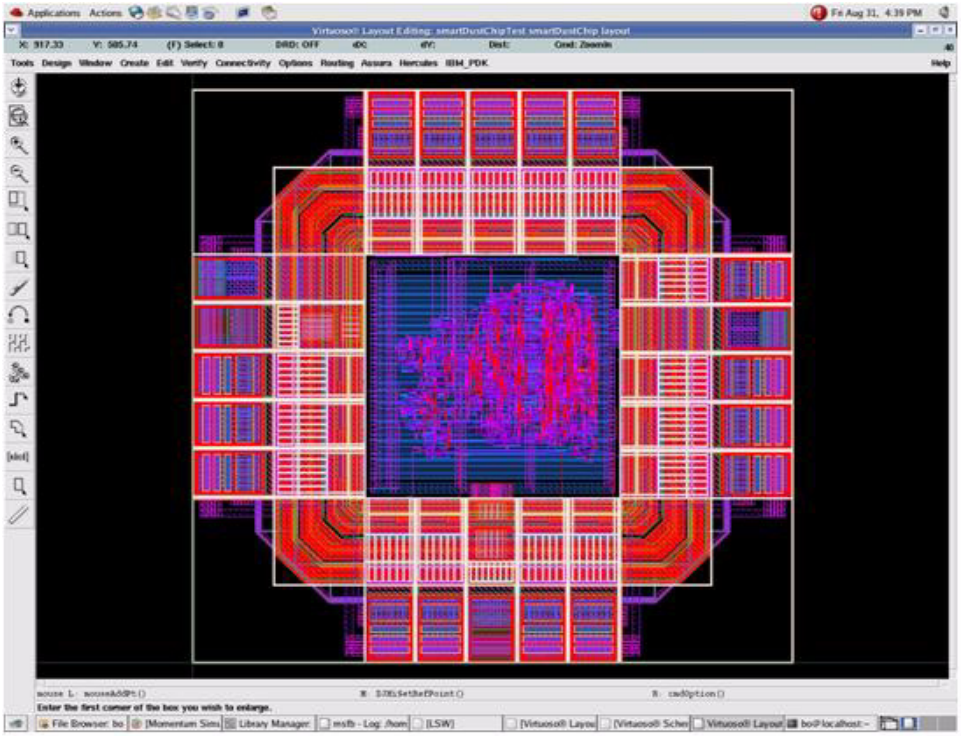


Figure 6.10 The design and layout for the DLCR digital ASIC using the IBM 0.13µm process.

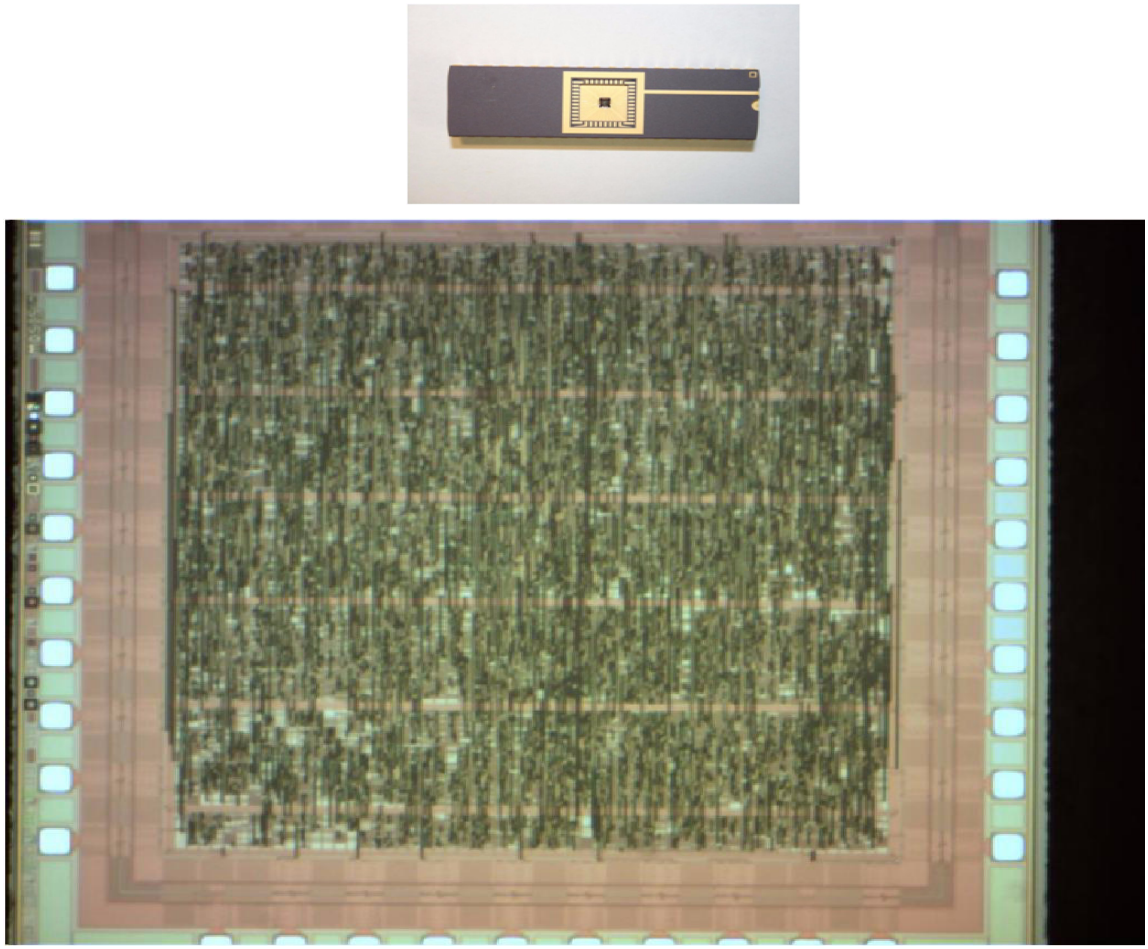


Figure 6.11 Fabricated chip using the MOSIS AMI 0.5µm process with a DIP package.

Chapter 7. System Synthesis for Configurations of Application-specific Embedded Systems

Many embedded systems in the domains of digital signal processing and communication have cross-layer and complex design considerations, and effective design space exploration strategies are desirable to improve the efficiency with which such systems can be designed. Resource limited embedded platforms need to be configured efficiently so that low power consumption, performance constraints, and other application-specific requirements can be met. As the work that we present in this chapter shows, design alternatives for this class of problems can be evaluated and explored rapidly at high levels of abstraction with careful development and calibration of system-level models.

The problem of finding efficient configurations for an application-specific embedded platform can be very difficult even for applications of moderate complexity due to the large number of different possible resource combinations. These problems are typically formalized as optimization problems, which may be resolved using evolutionary algorithms that are relevant for design space exploration. In this chapter, we present two configuration synthesis methods for finding efficient resource configurations for application-specific embedded systems. We also propose a new synthesis framework as our on-going and future work to design application-specific system-on-chips based on globally-asynchronous locally-synchronous configurations. Summaries of this chapter are published in [61] and [62].

We are very grateful to Prof. Jarmo Takala, Perttu Salmela, and Celine Badr for their time and effort in collaborating with us in this work.

7.1. A Rapid Prototyping Methodology for Application-Specific Sensor Networks

Sensor network embedded systems depend on many inter-related system parameters. The associated design space is vast, and effective optimization in this space is challenging. In this thesis, we introduce a system-level design methodology to find efficient configurations for an application-specific sensor network system where optimization of energy consumption is a primary implementation criterion. This methodology incorporates fine-grained, system-level energy models; analyzes critical parameters of candidate off-the-shelf components; integrates the associated parameters into a comprehensive optimization framework; and applies optimized configurations to the actual hardware implementation of the targeted sensor network system. The results demonstrate the accuracy and applicability of our methodology and supporting tools for optimized configuration of application-specific sensor networks.

7.1.1. System-level energy modeling

For finding effective application-specific sensor network configurations based on energy consumption considerations, a system-level energy model is required for the design methodology. System-level energy consumption is determined by processors and communication interfaces, hardware configurations, and the dependencies imposed by the application. This integrated energy model is used in an optimization framework [6] so that alternative system configurations can be evaluated by running simulations for estimating-

system-level energy consumption. The resulting approach to system-level modeling is one important contribution of this thesis, and this contribution helps us to more comprehensively explore the design space of a sensor network application.

We classify our system-level cross-layer energy models into the two levels of network-level and node-level modeling. For the network-level modeling, a network topology is defined based on a graph-based representation of the application, and then the critical parameters are identified that can affect energy consumption throughout the system. Next, constraints are formulated associated with maintaining the minimum acceptable functionality from the overall application. Based on methods presented in [55], a power model is derived to represent the minimum received power strength at a given node for correct communication of data across nodes. Here, we assume that nodes can receive the correct data pattern as long as the received power strength is above a particular threshold.

Considering the application example of N-nodes distributed line crossing recognition discussed in Chapter 5, the topology is specified by $G = (V, E)$, where $V = \{v_1, \ldots, v_N\}$, and $E = \{(v_i, v_{i+1}), 0 \leq i \leq N-1\}$. The WSN-related parameters associated with this application are given by $P_T(e)$, $P_R(e)$, $A(v)$, $d(e)$, and f_c, where $P_{T/R}(e)$ are the transmitted/received power values associated with the edge e in the network topology; $A(v)$ is the effective area of the antenna for the node v; $d(e)$ is the distance between the transmitter and receiver that is associated with the edge e; and f_c is the carrier frequency. Based on these parameters, the power model described above can be formulated as $P_R(e) = (\lambda/(4\pi d))^2 \cdot P_T(e) \cdot G_T \cdot G_R$, where $G_R = 4\pi A(v)/\lambda^2$, and $\lambda = c/f_c$ gives the receiver antenna gain and wavelength, respectively; c represents the speed of light; and G_T gives the transmitter antenna gain. The goal in network-level mod-

eling and optimization is to find the minimum $P_T(e)$ for each node so that the energy consumption of the whole system-level application can be minimized while maintaining the functionality of the application.

The node-level energy modeling is considered similar to the one discussed in Chapter 5.4.1. With the use of a MSP430 [90] as a targeted microcontroller platform, for energy modeling of the transmission mode, we model the active time for each device that is used in this mode. For example, the active time for the CPU core, A/D converter, UART, timer, and transceiver can be modeled, respectively, with the following equations:

$$f_{clk} = (2.14V_{cc} + 0.296)MHz,$$

$$t_{cpu\text{-}active}\big|_{tx} = N_{cpu\text{-}active}/f_{clk}\big|_{tx},$$

$$t_{ADC\text{-}active}\big|_{tx} = t_{sample} + (13ADCCLK)/5 \times 10^6,$$

$$t_{UART\text{-}active}\big|_{tx} = t_{UART\text{-}startup} + M/R,$$

$$t_{radio\text{-}active}\big|_{tx} = t_{radio\text{-}startup} + M/R.$$

Using these models of active time, the energy consumption in transmission mode can be modeled according to

$$E_{t_s}\big|_{tx} = (E_{mcu} + E_{radio} + E_{sensor})\big|_{tx} = E_{cpu\text{-}sleep} + E_{cpu\text{-}active} + E_{ADC\text{-}active} + E_{UART\text{-}active} + E_{timer\text{-}active} + E_{sensor\text{-}active} + E_{radio\text{-}sleep} + E_{radio\text{-}active}, \quad (7.1)$$

where,

$$E_{cpu\text{-}sleep} = I_{cpu\text{-}sleep} \cdot V_{cc} \cdot (t_s - t_{cpu\text{-}active}),$$

$$E_{cpu\text{-}active} = CV_{cc}^2 f_{clk} t_{cpu\text{-}active} + (V_{cc} I_0 e^{V_{cc}/(nV_T)}) t_{cpu\text{-}active},$$

$$E_{radio\text{-}sleep} = I_{radio\text{-}sleep} \cdot V_{cc} \cdot (t_s - t_{radio\text{-}active}),$$

$$E_{radio\text{-}active} = (I_{radio}\big|_{tx} V_{cc} + P_{out}) t_{radio\text{-}active},$$

$$E_{sensor\text{-}active} = I_{sensor\text{-}active} \cdot V_{cc} \cdot t_{ADC\text{-}active}.$$

The equations above are formulations that we have derived to provide energy models for data acquisition, computation, and transmission for sensor, microcontroller, and transceiver devices. We also need models for peripheral control in the microcontroller. Here, for the internal devices in the microcontroller, we just use the average current consumption values for calculations because it is difficult to observe the actual current variations of each internal device on a chip. Thus, we employ models of the following forms:

$$E_{ADC\text{-}active} = I_{ADC} \cdot V_{cc} \cdot t_{ADC\text{-}active},$$

$$E_{UART\text{-}active} = I_{UART} \cdot V_{cc} \cdot t_{UART\text{-}active},$$

$$E_{timr\text{-}active} = I_{timer} \cdot V_{cc} \cdot t_s.$$

For energy modeling in reception mode, where the sensor stays in a powered-down state, for example, the energy consumption can be modeled by using a similar approach as in transmission mode. The resulting models can be formulated as

$$E_{t_s}\big|_{rx} = (E_{mcu} + E_{radio})\big|_{rx} = E_{cpu\text{-}sleep} + E_{cpu\text{-}active} + E_{UART\text{-}active} + E_{timer\text{-}active} + E_{radio\text{-}active}, \quad (7.2)$$

where,

$$t_{cpu\text{-}active}\big|_{rx} = N_{cpu\text{-}active} / f_{clk}\big|_{rx},$$

$$t_{UART\text{-}active}\big|_{rx} = t_{UART\text{-}startup} + M/R,$$

$$E_{cpu\text{-}sleep} = I_{cpu\text{-}sleep} \cdot V_{cc} \cdot (t_s - t_{cpu\text{-}active}),$$

$$E_{cpu\text{-}active} = CV_{cc}^2 f_{clk} t_{cpu\text{-}active} + (V_{cc} I_0 e^{V_{cc}/(nV_T)}) t_{cpu\text{-}active},$$

$$E_{radio\text{-}active} = I_{radio}\big|_{rx} \cdot V_{cc} \cdot t_s.$$

Also, the corresponding energy model for using the internal devices (UART and timer devices) in the microcontroller in reception mode are represented in the same way as in transmission mode.

For energy modeling of the idle mode, we need to consider that a typical sensor node platform can be turned off so that there is no execution of operations for computation nor communication. After such turning off, the microcontroller and transceiver remain in power saving states until they are activated again. For this scenario, energy models can be derived as follows:

$$E_{t_s}\big|_{idle} = (E_{mcu} + E_{radio})\big|_{rx} = E_{cpu\text{-}sleep} + E_{timer} + E_{radio\text{-}sleep} = (I_{cpu\text{-}sleep} + I_{timer} + I_{radio\text{-}sleep}) \cdot V_{cc} \cdot t_s \quad . \qquad (7.3)$$

Note that a timer is required in our assumed implementation target for coordinating TDMA operations. Thus, the energy consumption for that timer device is considered in the energy model for each mode. Figure 7.1 summarizes the symbols that we use for the energy models that are developed in this section.

7.1.2. Design space exploration using particle swarm optimization

Particle swarm optimization (PSO) [34] has been chosen as the optimization strategy to find the most suitable configuration for a sensor network embedded system based on the system-level energy modeling discussed in the previous section. The experiments

Symbols	Description	Symbols	Description
$N_{cpu\text{-}active}$	number of clock cycles executed by CPU	f_{clk}	processor clock frequency
$t_{radio\text{-}startup}$	startup time from power-on to valid transmit/receive	$ADCCLK$	sampling cycles for ADC device
$t_{tx/rx}$	transmit/receive on time	P_{out}	transmission output power
t_s	slot time	C	total switching capacitance
$t_{device\text{-}active}$	active time for devices: ADC, UART, or timer	M	transmission message length
t_{sample}	ADC sampling time	R	data rate
$t_{UART\text{-}startup}$	UART startup time	I_0	processor leakage current
I_{sleep}	average current consumption in sleep mode with respect to corresponding devices	V_T	processor threshold voltage
I_{device}	average current consumption for device: ADC, UART, or timer		

Figure 7.1 Table of notation for energy modeling.

have been carried out with a prototype WSN platform, illustrated in Figure 7.2,. The platform is equipped with a Texas Instruments MSP430 microcontroller and an 916MHz transceiver. For evaluating WSN-related optimized configurations, we implemented the line-crossing application described in Chapter 5. We conducted experiments with mutable parameters chosen from V_{cc}, $ADCCLK$, and P_{out} to compare energy consumption results associated with simulation from the PSO-based optimization framework, and measurement from the constructed prototype platform. In addition, in the experiments we employed the settings in Figure 7.3 as immutable parameter values. The measured current consumption result from one node on the prototype platform is shown in Figure 7.4(left).

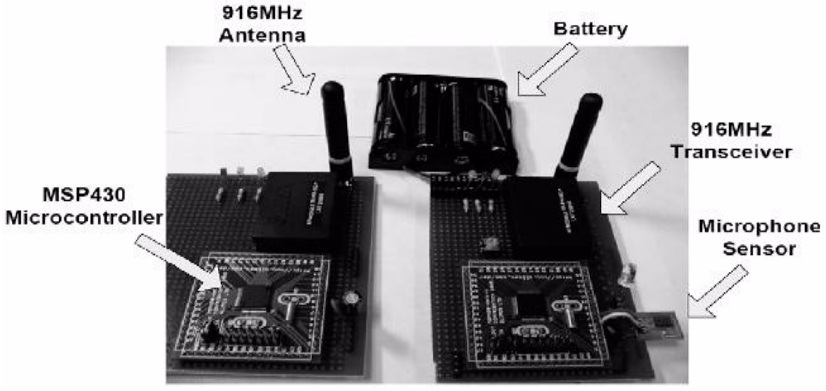


Figure 7.2 WSN prototype platform.

immutable parameters	values	immutable parameters	values	immutable parameters	values
$N_{cpu\text{-}active\|tx}$	579	$I_{cpu\text{-}sleep}$	$200uA$	R	$9600bps$
$N_{cpu\text{-}active\|rx}$	309	$I_{sensor\text{-}sleep}$	$200uA$	f_{clk}	$8MHz$
$t_{cpu\text{-}active\|tx}$	$0.072ms$	I_{ADC}	$300uA$	C	$100pF$
$t_{cpu\text{-}active\|rx}$	$0.039ms$	I_{timer}	$300uA$	I_0	$2mA$
$t_{radio\text{-}startup\|tx}$	$3mA$	$I_{radio\text{-}sleep}$	$200uA$	t_s	$125ms$
$t_{ADC\text{-}active}$	$53.6us$	I_{UART}	$300uA$	$t_{radio\text{-}active}$	$6.33ms$
$t_{radio\text{-}startup\|rx}$	$6ms$	N/A	N/A	N/A	N/A

Figure 7.3 Table of settings for immutable parameters and values.

We used a 4GHz digital phosphor oscilloscope to measure the corresponding voltage variations on each platform for 10 time frames of execution, where each time frame takes 8 TDMA time slots with 125ms for each time slot. The actual energy consumption on the prototype platform can be calculated according to: $E = \int_t P(t)dt$. The experimental results of the optimized configurations for the whole 5-node line-crossing application through our optimization framework are shown in Figure 7.4(right). We compared the results from the simulation of the optimization framework and measurement from the corresponding implementations on our WSN testbed. For these comparisons, we chose 20

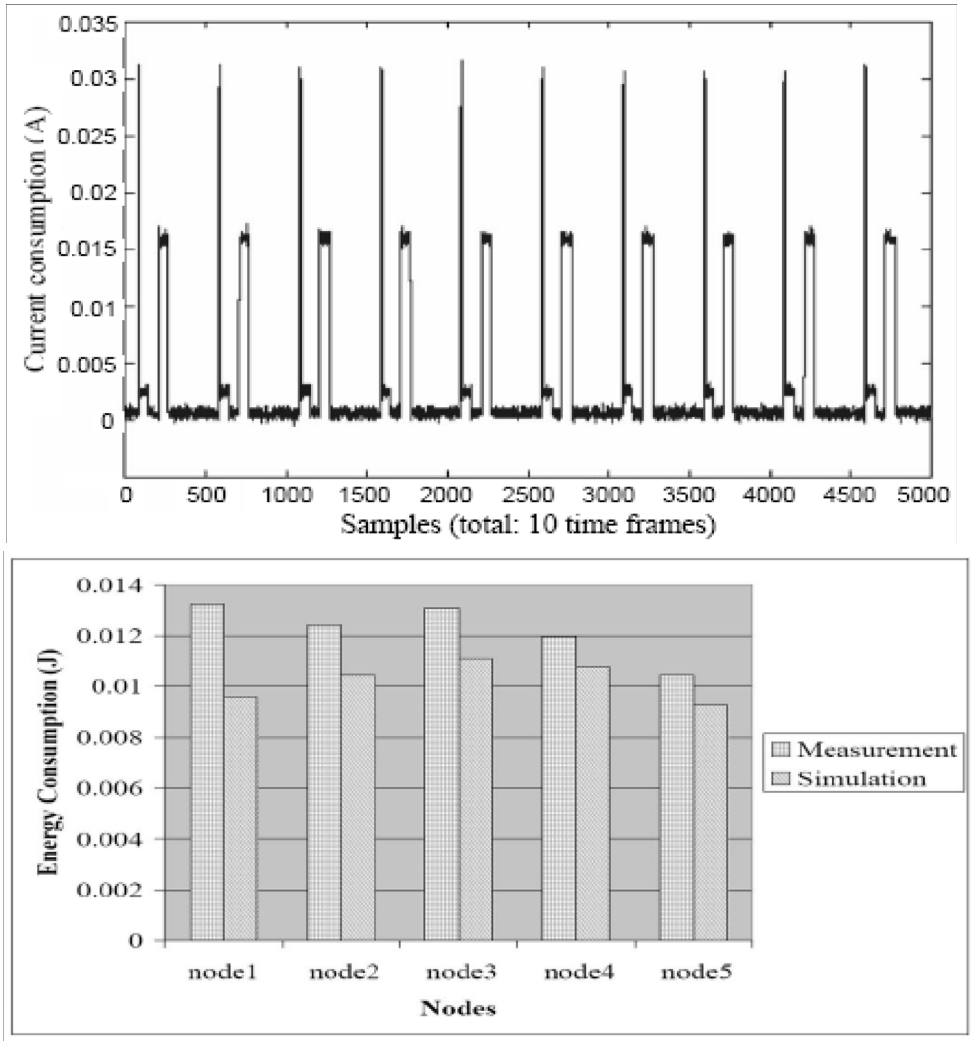


Figure 7.4 Plot of current consumption measured from the prototype platform.

particles with $c_1 = c_2 = 2.0$ and $\omega = 0.95$ when running the PSO optimization algorithm for the experiment [6].

In the example of the 5-node line-crossing application, we chose the target optimum value as 0.05(J) for total energy consumption during one simulation time frame. One set of candidate results generated from our optimization framework for configuring the whole application with chosen mutable parameters and a binding constraint of 0.0013 are listed in Figure 7.5. From the results, we can verify that required transmission power increases with distance, and we can quantify this fundamental dependence in terms of the technology that we are using in the targeted platform.

7.2. Synthesis of DSP Architectures using Libraries of Coarse-Grain Configurations

Another method for exploring the design space of parallel elementary computing resources is introduced. The method can be used to find a suitable set of computing resources for processors applying instruction level parallelism (ILP) or pure hardware designs. The computing resources are explored at the system level so that they can be evaluated rapidly, and the extensive size of the design space handles such coarse level modeling and evaluation. The method presents the targeted system as a union of multisets

Node I.D.	V_{cc}(V)	ADCCLK(CC)	P_{out}(dBm)	d
1	5.25	1024	-10	15.79
2	4.59	1024	-3	28.13
3	3.81	1024	-7	27.85
4	3.94	512	-11	10.62
5	3.70	256	-8	11.79

Figure 7.5 Table of candidate result for configuring the 5-node line-crossing application with binding constraint *±0.0013* .

of computing resources. This formulation provides a general framework for efficient, multi-objective optimization in terms of relevant cost metrics, including processing latency, area, and power consumption. We demonstrate this framework by developing a multiobjective evolutionary algorithm based on it, and applying this algorithm to a rake receiver application.

7.2.1. System-level modeling for resource configurations

The main objective here is to find suitable computing resource configurations, i.e., multisets of function units (FU), for a given application specification, where a targeted platform is assumed to be used. The highest-level specification of a system is modeled as a dataflow graph, $G = (V, E)$, that consists of communicating edges in E between actors (i.e. computation modules) in V that have fixed consumption and production rates of tokens. The computation platform to which G is mapped is an abstract multiset of parallel computing resources.

The resource configurations are defined with the aid of multisets. A multiset (X, m) consists of a set X and a multiplicity function $m: X \rightarrow N$. The multiplicity given by $m(x)$ represents how many instances of the element x are included in the multiset. The FUs of an experimental architectures typically consists of adders, multipliers, shifters, etc. In addition, there can be special FUs targeted to a limited application domain. If the set of all the available FUs is denoted by S_{FUs}, the resource configuration CFG can be defined as a multiset $CFG = (X, m)$, where $X \subseteq S_{FUs}$ and m gives the multiplicity of each FU. An architecture example that consists of multi-purpose FUs is transport triggered architecture processors [18].

We formalize our objectives for system synthesis and evaluation for clock cycles, area, and power in the following context. The number of clock cycles of the actor processed with the resource configuration CFG is denoted with $D(v, CFG)$ where $v \in V$ in G. In general, an actor can be invoked several times to match differing production and consumption rates with its adjacent actors or due to the feedback loops. Therefore, $D(v, CFG)$ must contain all the invocations of the actor. The number of invocations can be obtained from a repetitions vector, which can be computed easily if the system is designed with the aid of synchronous dataflow graphs [41].

In the targeted platform, all the actors are assigned to the same hardware resources and the actors are executed sequentially. Thus, the clock cycles of data transfers between actors, by passing the data via memory or registers, can be included in the cycle count $D(v, CFG)$ of the actor who is the sender of the data. Since real-time applications are targeted, it is assumed that $D(v, CFG)$ is independent of the input data. If such a dependency exists, the maximum number of clock cycles, i.e., the worst case, must be represented by $D(v, CFG)$. Since all the actors are executed sequentially, the total number of clock cycles taken by the system is the sum of the number of clock cycles used by all the actors, i.e.,

$$D_{total}(CFG) = \sum_{v \in V} D(v, CFG). \tag{7.4}$$

The area cost of a FU u is defined as $A(u)$ and the total area costs, A_{total}, contributed by FUs are given by,

$$A_{total}(CFG) = \sum_{u \in CFG} A(u). \tag{7.5}$$

It is assumed that the costs are not affected heavily by clock frequency. Such an assumption is justified, if the operating conditions are far enough from the limits of the applied technology. In other words, there should be a sufficiently long slack on the critical path, which is typically true for modest clock frequencies. In Eq. 7.5, there can be several FU instances of the same type and all of them are included in the sum. In other words, each FU is counted according to the multiplicity of FU in multiset CFG.

The dynamic power consumption is considered here and defined as $P = \alpha C_L f_{clk} V_{cc}^2$, where α is the activity ratio, C_L is the capacitive load, f_{clk} is the clock frequency, and V_{cc} denotes the supply voltage. The dynamic power consumption used in FUs utilized by actor v is denoted with $P(v, CFG)$. Thus, the average dynamic power consumption of all the actors is defined as

$$P_{total}(CFG) = \sum_{v \in V} \left(\frac{D(v, CFG)}{D_{total}(v, CFG)} (P(v, CFG)) \right), \tag{7.6}$$

which is obtained with the aid of utilizations of each actor, i.e., the percentage of execution time of an actor of the total execution time of the system. Since real-time systems are targeted, there is a given time frame, t_f, in which the system must be executed. The required system clock frequency is the number of clock cycles per time frame, i.e., $f_{clk} = D_{total} / t_f$.

For each actor v in G, the associated actor specific configurations, $CFG_{v,i}$, $i = 1, \ldots, N_v^{CFG}$, are created, where N_v^{CFG} denotes total number of configurations allowed for v. Assume that a library is used at design time that contains the associated $D(v, CFG_{v,i})$ and $P(v, CFG_{v,i})$ for each configuration of actor v. The resource configu-

ration, CFG_S, for the system is defined as a multiset union of actor specific configurations, where one configuration is selected for each actor. That is,

$$CFG_S = \bigcup_{v \in V} CFG_{v,i}, \text{ where } i \in \{1, \ldots, N_v^{CFG}\}. \quad (7.7)$$

For considering the design space exploration, each distinct resource configuration, CFG_S, of the whole system is one point in the design space, and it is possible that some configurations are identical and they are located at the same point.

7.2.2. Design space exploration using multiobjective evolutionary algorithm

When an actor is executed, it is assumed that the utilization of available FUs optimizes for the speed but not for other objectives such as power consumption. With this assumption the number of clock cycles taken by an actor, v, with resource configuration, CFG_S, is estimated with the fastest compatible resource configuration. Especially, the configuration, CFG_S, is compatible in respect to the actor v, if $CFG_{v,i} \subseteq CFG_S$. Therefore, the number of clock cycles can be estimated as

$$D(v, CFG_S) = \min_{CFG_{v,i} \subseteq CFG_S} D(v, CFG_{v,i}). \quad (7.8)$$

The power consumption is estimated according to the same $CFG_{v,i}$ as in Eq. 7.8 and based on the FUs that are utilized by the actor v. That is,

$$P(v, CFG_S) = P(v, CFG_{v,i_0}), \quad (7.9)$$

where

$$CFG_{v,i_0} = \operatorname{argmin}_{CFG_{v,i} \subseteq CFG_S} D(v, CFG_{v,i}) \quad (7.10)$$

represents one of the configurations of the actor v which pre-estimated power consumption is available in the library. The area cost for a resource configuration is estimated directly based on Eq. 7.5.

The general quality of the system is determined by the three objectives namely the power consumption, area costs, and number of clock cycles. Thus, the goal of the search is to find resource configurations, CFG_S, which minimize the multi-objective vector, i.e.,

$$\min \begin{pmatrix} P_{total}(CFG_S) \\ A_{total}(CFG_S) \\ D_{total}(CFG_S) \end{pmatrix}. \tag{7.11}$$

In the design space, a solution $CFG_S^{'}$ dominates solution $CFG_S^{''}$ if

$$P_{total}(CFG_S^{'}) \le P_{total}(CFG_S^{''}) \wedge A_{total}(CFG_S^{'}) \le A_{total}(CFG_S^{''}) \wedge D_{total}(CFG_S^{'}) \le D_{total}(CFG_S^{''}) \quad . \tag{7.12}$$

The size of design space is strongly related to the number of configurations for a given system. That is, there are

$$N_{total}^{CFG} = \prod_{v \in V} N_v^{CFG} \tag{7.13}$$

alternative multiset union configurations. Therefore, evolutionary algorithms are preferred to be applied to search for solutions in such a vast design space. In this study, the encoding of the problem for evolutionary algorithm relies on the previously defined actor specific configurations $CFG_{v,i}$ and multiset union configurations, i.e. Eq. 7.10. For each actor, v, there is a gene in the individual. The value of the gene represents the selected actor specific configuration. The size of the domain of the gene is limited by the number of respective actor specific configurations, N_v^{CFG}, of the actor v. Crossing over two parents

reproduces two new offsprings. For each gene of the reproduced offspring, a random function selects the parent from which the gene is inherited.

Pareto optimal sets can be searched with multiobjective evolutionary algorithms where all the objective functions are taken into account, instead of selection with a scalar valued fitness function [83]. In our implementation, the platform and programming language independent interface for search algorithms (PISA) [11] is used. The selection process is carried out with the strength Pareto evolutionary algorithm (SPEA) [83].

We choose a high-level computational load model of the rake receiver, which is shown in Figure 7.6. In load modeling, the required FUs, number of clock cycles with different configurations, and FU utilizations to obtain power estimates are analyzed for each actor, instead of targeting a functional simulation model. The rake receiver is applied in 3G telecommunications systems and on the edge between processor, accelerated processor, or application specific hardware based implementation alternatives. In this example, the despreading actor includes four rake finger operation which multiply the input samples

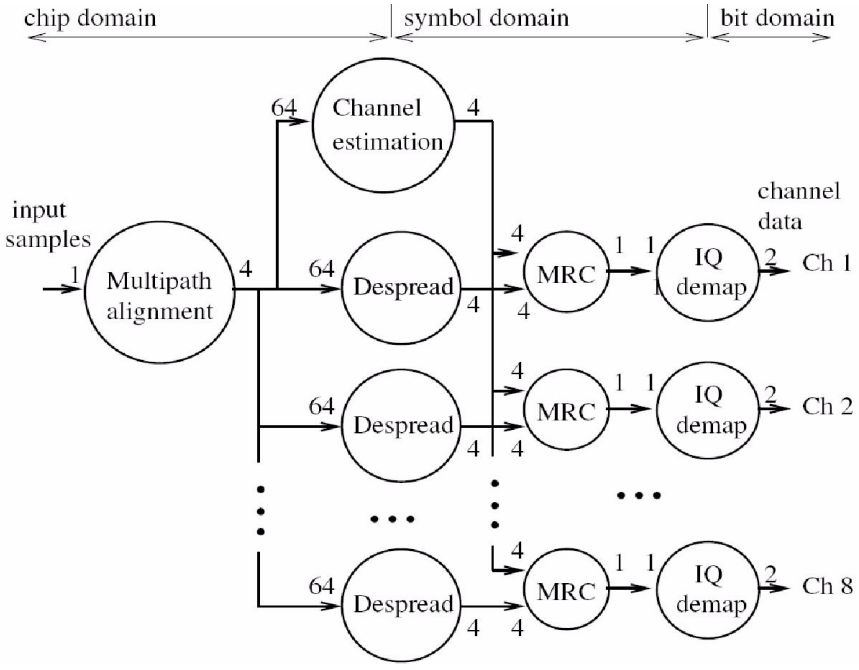


Figure 7.6 Experimental rake receiver model.

with a despreading code and integrate the result for the length given by a spreading factor (SF) to obtain one symbol. The SF determines ratio between chip rate and symbol rate. All the fingers access the same input samples, but with different offsets. The offsets are determined by a multipath search, which is not included in the model. The channel estimation is based on pilot data, which is despread and filtered with a moving average filter, whose output is fed to the maximum ratio combining (MRC). The MRC computes the weighted sum of the results of rake fingers. The alignment of the multipaths accesses the memory banks with different offsets.

The computational load of actors can be modeled with different multisets of FUs. Especially, many of the actors can utilize parallelism, i.e., multiplying the number of kernel FUs lowers the number of clock cycles. The range of clock cycles per designed actor specific configurations are shown in Figure 7.7. In this study, in addition to arithmetic or logic functionality, the FUs include minor control logic and internal registers. Naturally, with very simple functions the control logic dominates costs. The area and power estimates of FUs are shown in Figure 7.8. Here, the finger resource can be applied to compute the inner product of code and input samples. The spreading code generation resource pro-

actor type	# of cfgs.	range of # of clock cycles
despreading	8	1–4 / chip
multipath alignment	3	1–4 / chip
MRC	5	1–16 / symbol
IQ-demapping	2	1–2 / symbol
channel estimation	8	1–4 / chip

Figure 7.7 Performance ranges of the applied actor specific configuration with SF=16 and 8 data channels.

duces spreading code and is implemented as a simple shift register with feedback connections.

The results of the multiobjective design exploration are presented in the form of Pareto optimal points in Figure 7.9 In total 26 different kind of actor specific configurations contribute to the union configuration of the receiver. Since there are three objectives, the Pareto points are located in three dimensional space. The plots show some points which cannot be interpreted as Pareto points if only two objectives, i.e., the axis of the figure are considered. However, they are Pareto points in reality, since all the three objectives are considered. The Pareto points show, as it is expected, that the area costs increase when the number of clock cycles decreases, i.e., the more parallelism is applied to decrease the delay, the more resources are required. The negative relation of the area costs and dynamic power consumption originates from the lower voltage with lower clock frequency. The clock frequency is lower since higher parallelism allows execution with decreased number of clock cycles.

resource	area (gates)	power (μW)
multiply	3184.50	657.67
finger	1433.00	340.44
add / sub	1095.50	344.96
shift	1021.50	287.18
and / or	794.00	303.77
load / store	643.50	240.68
code gen.	530.00	185.37

Figure 7.8 Area costs and dynamic power consumption of the elementary computing resources.

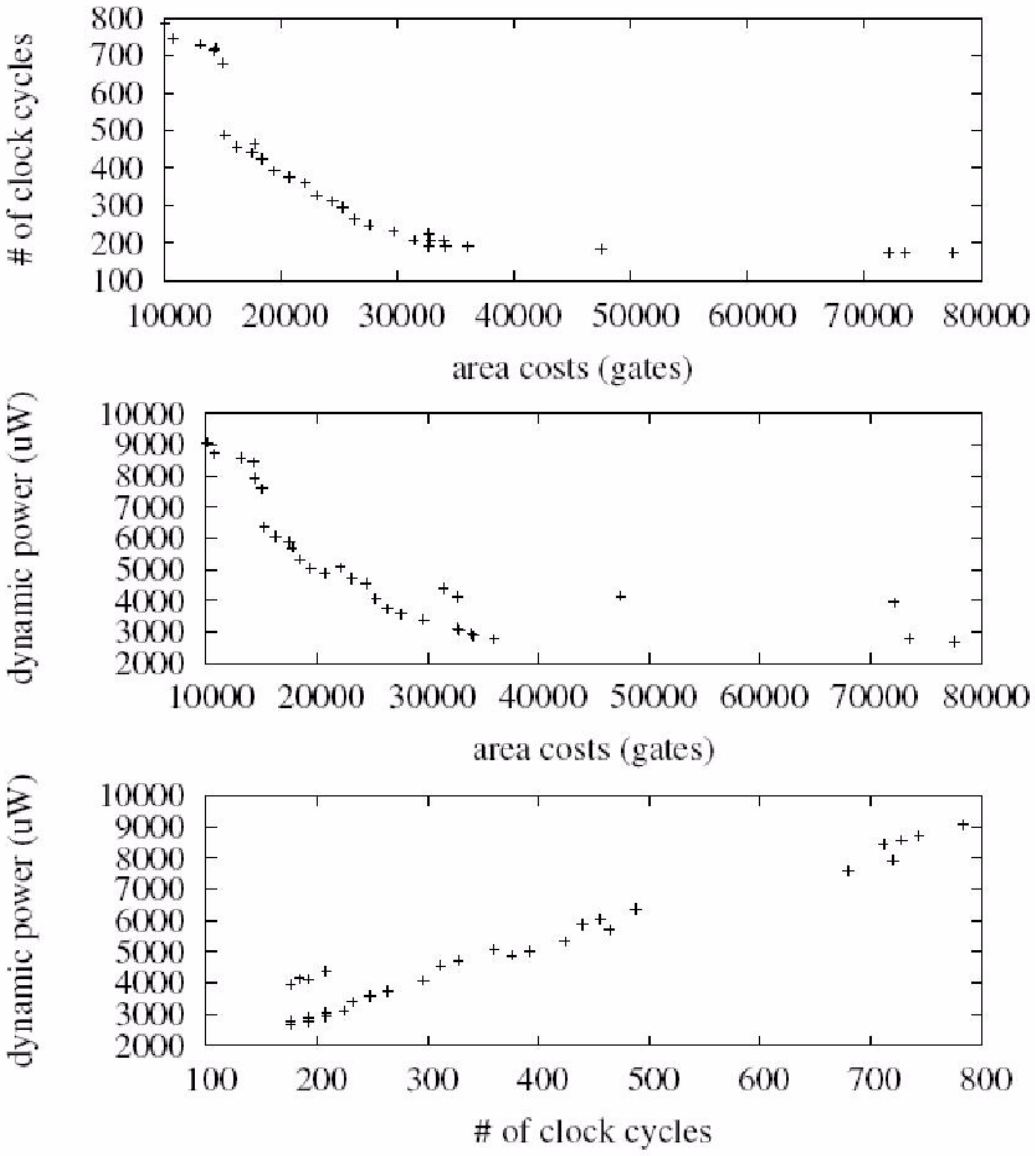


Figure 7.9 Pareto optimal points are projected to 2-D presentations. Each point corresponds to a resource configuration.

Chapter 8. Conclusions and Future Work

8.1. Conclusion

In this thesis, we have presented a novel algorithm, and associated design methodology for distributing DSP applications across master-slave WSN systems in an energy-efficient fashion. This methodology integrates high-level application modeling, and task- and network-level energy and latency modeling to comprehensively optimize system performance. We have discussed efficient dataflow-based modeling techniques for different DSP applications, integrated topology and protocol requirements of WSNs into these modeling techniques, formulated an important energy-driven partitioning problem, integrated this problem with a quasi-static scheduling approach, and developed an efficient heuristic algorithm for finding partitioning results that maximize the network lifetime. Our developed workload redistribution scheme is adapts automatically to changes in network size. Results on synthetic benchmarks and on practical applications demonstrate the utility of our proposed methods. From the experimental results, our approach can be seen to run efficiently and improve conventional partitioning solutions significantly, by least 50% of the energy cost. As applications become more complicated, the proposed methodology becomes even more useful.

We have demonstrated a distributed automatic speech recognition (DASR) system implementation as a case study by integrating embedded processing for speech recognition with a wireless sensor network system. We have discussed several promising applications for which the proposed system can be further customized. In our system implementation, we have adopted a parameterized-dataflow-based modeling approach for

structuring a well-known algorithm for embedded speech recognition. This model-based design facilitates latency and memory analysis, and helps to structure the embedded software for efficient implementation.

Through a detailed case study, we have demonstrated in detail our key design steps, including the definition of the network topology, protocol design, implementation of the targeted speech recognition algorithm, and distribution of computation and communication with careful consideration of energy usage. Measurement results on recognition accuracy and energy consumption demonstrate the functionality and efficiency of our DASR system implementation.

We have also demonstrated in this thesis that threat detection, an important application area for WSN technology, should be considered as a system-level problem. Single elements of the system chain individually have small impact on overall performance, and understanding the interactions among different elements is critical to meeting constraints on power consumption, performance, and reliability. We have demonstrated a complete system design methodology for a practical application of a distributed line-crossing recognition (DLCR). This methodology includes algorithm streamlining, communication protocol configuration, hardware/software implementation, and lifetime modeling. Our proposed distributed algorithm is useful in reducing the amount of data that must be communicated across nodes in the network. Furthermore, the communication protocol that we employ carefully manages the duty cycle to achieve further improvements in energy efficiency. Our designed protocol is fault tolerant so that node failures are prevented from translating into failures in the overall system; the capability to add nodes dynamically into our distributed system is being considered in our ongoing and future work. Through simu-

lation-based experiments, we have employed various design techniques to improve the system lifetime. From our experimental results, we have demonstrated the importance of considering the network lifetime problem at the system level. We have also observed that full system integration for sensor support systems plays an important role in influencing the design for individual components (“elements”) that are to be used on sensor node platforms.

This thesis has also demonstrated different sensor node platforms, including microcontroller- FPGA-, and ASIC-based platforms, for the targeted DLCR system. Customized printed circuit board (PCB) designs of various system prototypes are demonstrated to compactly support the developed sensor node embedded platform. The platform in turn employs emerging, off-the-shelf, system-on-chip technology, and a custom-designed miniaturized antenna. The digital ASIC that we have designed and implemented in this work demonstrates the functionality of the DLCR system, and significantly reduces the size and power consumption compared to other commercial embedded platforms.

In this thesis, we have also explored system-level design methodologies to derive optimized configurations for different applications. We have derived a number of fine-grained, system-level energy models as efficient evaluation metrics for our application-specific WSN system analysis and optimization. To demonstrate the efficacy of the models and optimization methods, we used configurations that were derived from our design framework to map a practical WSN application into complete hardware/software implementations. From these implementations, we analyzed various parameters from the models that we employed and calculated the fidelity of the high level estimation methods used in the optimization framework. Our results showed that, relative to their high level of

abstraction and efficiency in exploring the design space, the integrated models and estimation techniques result in a high accuracy of relating candidate solutions during optimization to their equivalent realizations as actual WSN system implementations.

As a complementary form of platform optimization, we demonstrated a method for system-level exploration of computing resources on targeted DSP architectures. The method was based on presenting actor specific resource configurations as multisets of functional units (FUs). A coarse level modeling was derived to help evaluate and explore the associated design space rapidly. The derived formulations for this coarse-level modeling lead naturally to efficient evaluation and encoding in the context of multi-objective evolutionary algorithms. Experiments on a dataflow-based rake receiver application and design showed the applicability of our proposed methods to identifying strategic design alternatives in the space of architectural configurations.

8.2. Future Work

With advances in integrated circuit technology, distributed embedded systems such as sensor nodes will be equipped with increasing amounts of computational resources, such as digital signal processing (DSP) subsystems. We believe that the work presented in this thesis will be of increasing utility and impact as this trend develops further. The energy-driven design methodology that we have developed introduces a systematic framework that can be extended for automatic synthesis and code generation for implementing embedded software on distributed embedded platforms.

The work that we have presented for design and implementation of streamlined digital ASICs is relevant for incorporation in complete distributed embedded systems such as

wireless sensor networks. Our ongoing and future work aims for further miniaturization and energy efficiency for the targeted line crossing recognition application by developing a fully customized design, including a streamlined ASIC implementation of the mixed-signal SoC.

Based on the experience gained from our work on system-level synthesis and design space exploration, a novel design and optimization framework is proposed to find the most suitable GALS configuration for realizing a DSP application. A central optimization goal here is to suitable clustering (i.e., grouping into synchronous regions) results from a given application graph, where the system power consumption, area cost, and processing performance are to be efficiently traded off (e.q. [36]). We will continue our work on developing our framework for GALS-based DSP system design and optimization. This framework can be integrated with exiting tools or packages such as DIF that are based on well-developed dataflow modeling and graph theory (e.q., see [10], [71]). Furthermore, integration of hardware synthesis and code generation steps are useful for incorporation to complete the overall design flow, and make it more automated.

BIBLIOGRAPHY

[1] S. Adl and M.Peckerar. A low-Noise Pre-amplifier/Amplifier chain for high capacitance sensors. In *IEEE Sensors Applications Symposium (SAS) Proceedings*, San Diego, February 2007.

[2] S. M. Ahadi, H. Sheikhzadeh, R. L. Brennan, and G. H. Freeman. An efficient front-end for automatic speech recognition. In *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems*, Sharjah, United Arab Emirates, December 2003.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. In *Computer Networks*, pages 393-422, March 2002.

[4] H. Andrade and S. Kovner. Software synthesis from dataflow models for embedded software design in the G programming language and the LabVIEW development environment. In *Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers*, November 1998.

[5] T. Back, U. Hammel, and H. P. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transaction on Evolutionary Computation*, vol. 1, no. 1, pp. 3–17, 1997.

[6] C. Badr. Synthesis of embedded software for sensor nodes. Master's thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park, 2006.

[7] K. Bhagavathula, A. H. Titus, and C. S. Mullin. An extremely low-power CMOS glare sensor. *IEEE Sensors Journal*, vol. 7, issue 8, pages 1145-1151, August 2007.

[8] B. Bhattacharya and S. S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, 49(10):2408-2421, October 2001.

[9] S. S. Bhattacharyya, J. T. Buck, S. Ha, and E. A. Lee. Generating compact code from dataflow specifications of multirate signal processing algorithms. *IEEE Transactions on Circuits and Systems --- I: Fundamental Theory and Applications*, 42(3):138-150, March 1995.

[10] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. Optimized software synthesis for synchronous dataflow. In *Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors*, July 1997.

[11] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA—a platform and programming language independent interface for search algorithms. In *Evolutionary Multi-Criterion Optimization, ser.* LNCS, pp. 494 – 508, Faro, Portugal, 2003.

[12] M. Brown and L. Rabiner. Dynamic time warping for isolated word recognition based on ordered graph searching techniques. In *Proceedings of the International Conference on Acoustic, Speech, Signal Processing*, vol. 7, pp. 1255-1258, May 1982.

[13] B. H. Calhoun, D. C. Daly, N. Verma, D. F. Finchelstein, D. D. Wentzloff, A. Wang, S. Cho, and A. P. Chandrakasan. Design considerations for ultra-low energy wireless microsensor nodes. *IEEE Transactions on Computers*, vol. 54, issue 6, pages: 720-740, Jun. 2005.

[14] D. M. Chapiro, *Globally-asynchronous locally-synchronous systems*, Ph.D thesis, Stanford University, October 1984.

[15] T. Chelceq and S. M. Nowick. Low-latency asynchronous FIFO's using token rings. In *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuirs and Systems*, pages 210-220, April 2000.

[16] Y. Chen and Q. Zhao. An integrated approach to energy-aware medium access for wireless sensor networks. *IEEE Transactions on Signal Processing*, vol. 55, issue 7, pages 3429-3444, July 2007.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, The MIT Press, 2001.

[18] H. Corporaal. Design of transport triggered architectures. In *4th Great Lakes Symp. on Design Autom. of High Perf. VLSI Syst.*, pp. 130–135, Notre Dame, IN, USA, March 1994.

[19] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the 1st international conference on embedded networked sensor systems*, pp. 171–180, November 2003.

[20] B. Delaney, T. Simunic, and N. Jayant. Energy aware distributed speech recognition for wireless mobile devices. *HP Labs Technical Reports*: HPL-2004-106, June 2004.

[21] P. D. Dimitropoulos, D. P. Karampatzakis, G. D. Panagopoulos, and G. I. Stamoulis. A low-power/low-noise readout circuit for integrated capacitive sensors. *IEEE Sensors Journal*, vol. 6, issue 3, pages 755-769, June 2006.

[22] Y. Dinitz, S. Moran, and S. Rajsbaum. Exact communication costs for consensus and leader in a tree. *Journal of Discrete Algorithms* 1(2), pp. 167–183, April 2003.

[23] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. In *Proceedings of the IEEE*, January 2003.

[24] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristics for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, pages 175-181, 1982.

[25] D. Ganesan, A. Cerpa, W. Ye, Y. Yu, J. Zhao and D. Estrin, Networking issues in wireless sensor networks. *Journal of Parallel and Distributed Computing*, vol. 64, issue 7, pages 799-814, July 2004.

[26] M. R. Garey and D. S. Johnson, *Computers and Intractability,* W. H. Freeman and Co., NY, 1979.

[27] J. Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *Proceedings of the ACM International Conference on Wireless Sensor Networks and Applications*, pages 11-19, September 2003.

[28] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Oberg, P. Ellervee, and D.Lundqvist. Lowering power consumption in clock by using globally asynchronous locally synchronous design style. In *Proceedings of the 36th ACM/ IEEE conference on Design automation*, pages 873-878, 1999.

[29] J. Hill and D. Culler, Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, pp. 12–24, November 2002.

[30] D. S. Hirschberg and J. Sinclair. Decentralized extrema-finding in circular configurations of processors. *Commun. ACM* 23 (ll), pp. 627–628, 1980.

[31] C Hsu, I. Corretjer, M. Ko., W. Plishker, and S. S. Bhattacharyya. Dataflow interchange format: Language reference for DIF language version 1.0, user's guide for DIF package version 1.0. *Technical Report UMIACS-TR-2007-32, Institute for Advanced Computer Studies*, University of Maryland at College Park, 2007. Also *Computer Science Technical Report CS-TR-4871*.

[32] C. Hsu, F. Keceli, M. Ko, S. Shahparnia, and S. S. Bhattacharyya. DIF: An interchange format for dataflow-based design tools. In *Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation*, July 2004.

[33] A. Kalavade and P. A. Suhrahmanyam. Hardware / software partitioning for multifunction systems. In *Proceedings of the International Conference on Computer Aided Design*, pages 516-521, November 1997.

[34] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, November 1995.

[35] W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, pages 291-307, 1970.

[36] V. Kianzad and S. S. Bhattacharyya. Efficient techniques for clustering and scheduling onto embedded multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 17(7):667-680, July 2006.

[37] D. Ko, C. Shen, S. S. Bhattacharyya, and N. Goldsman. Energy-driven partitioning of signal processing algorithms in sensor networks. In *Proceedings of the International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 142-154. Springer, July 2006.

[38] M. Kohvakka, M. Hannikanen, and T. Hamalainen. Wireless sensor prototype platform. In *Proceedings of the IEEE International Conference for Industrial Electronics, Control and Instrumentation*, November 2003, pp. 1499–1504.

[39] R. Kumar, V. Tsiatsis, and M. B. Srivastava. Computation hierarchy for in-network processing. In *Proceedings of the ACM International Conference on Wireless Sensor Networks and Applications*, pages 68-77, 2003.

[40] M. Kuorilehto, M. Hannikainen, and T. D. Hamalainen. A survey of application distribution in wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, pages 774-788, 2005.

[41] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. In *Proceedings of the IEEE*, 75(9):1235-1245, September 1987.

[42] S. Lindsey, C. Raghavendra, and K. Sivalingam. Data gathering in sensor networks using the energy delay metric. *IEEE Transactions on Parallel and Distributed Systems*, pages 924-935, April 2002.

[43] LINX LR Series RF Transmitter and Receiver Module Data Guides, *LINX Technologies*.

[44] D. Lymberopoulos and A. Savvide, Xyz: A Motion-enabled Power Aware Sensor Node Platform for Distributed Sensor Network Applications. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, April 2005.

[45] R. Mehra and J. Rabaey. Behavioral level power estimation and exploration. In *Proceedings of the International Workshop on Low Power Design*, pages 197-202, April 1994.

[46] A. Nahapetian, P. Lombardo, A. Acquaviva, L. Benini, and M. Sarrafzadeh. Dynamic reconfiguration in sensor networks with regenerative energy sources. In *Proceedings of the Design Automation and Test Europe Conference*, April 2006.

[47] A. Nisbet and S. Dobson. A systems architecture for sensor networks based on hardware/software co-design. In *Proceedings of the 1st IFIP Workshop on Autonomic Communications*, pages 115-126, Springer Verlag, July 2005.

[48] K. Niyogi and D. Marculescu. Speed and voltage selection for GALS systems based on voltage/frequency islands. In *Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 292-297, January 2005.

[49] K. Niyogi and D. Marculescu. System level power and performance modeling of GALS point-to-point communication interfaces. In *Proceedings of the 2005 international symposium on Low power electronics and design, pages 381-386*, August 2005.

[50] E. Pauwels, A. A. Salah, and R. Tavenard. Sensor networks for ambient intelligence. In *Proceedings of the IEEE International Workshop on Multimedia Signal Processing*, Chania, Crete, Greece, October 2007.

[51] K. K. Parhi. *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, Inc., 1999.

[52] S. Phadke, R. Limaye, S. Verma, and K. Subramanian. On design and implementation of an embedded automatic speech recognition system. In *Proceedings of the 17th International Conference on VLSI Design*, pages 127-132, 2004.

[53] J. Park and S. Sahni. Maximum lifetime broadcasting in wireless networks. *IEEE Transactions on Computers*, vol. 54, issue 9, pages: 1081 - 1090, Sept. 2005.

[54] J. Park and S. Sahni. An online heuristic for maximum lifetime routing in wireless sensor networks. *IEEE Transactions on Computers*, vol. 55, issue 8, pages: 1048 - 1056, Aug. 2006.

[55] J. Paul and M. G. Linmartz. Wireless Communication, The Interactive Multimedia CD-ROM. Baltzer Science, 1996.

[56] J. L. Pino and K. Kalbasi. Cosimulating synchronous DSP applications with analog RF circuits. In Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, Nov. 1998.

[57] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall. December 2002.

[58] J. M. Rabaey and M. Pedram. *Low Power Design Methodologies*. Kluwer Academic Publishers, 1996.

[59] S. Ravindran, D. Anderson, and M. Slaney. Low-power audio classification for ubiquitous sensor networks. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pages iv-33-iv-340, May 2004.

[60] K. Romer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, vol. 11, issue 6, pages 54-61, December 2004.

[61] P. Salmela, C. Shen, S. S. Bhattacharyya, and J. Takala. Synthesis of DSP architectures using libraries of coarse-grain configurations. In *Proceedings of IEEE Workshop on Signal Proceeding Systems*, pages 475-480, October 2007.

[62] C. Shen, C. Badr, K. Kordari, S. S. Bhattacharyya, G. L. Blankenship, and N. Goldsman. A rapid prototyping methodology for application-specific sensor networks. In

Proceedings of the IEEE International Workshop on Computer Architecture for Machine Perception and Sensing, Montreal, Canada, September 2006.

[63] C. Shen, R. Kupershtok, S. S. Bhattacharyya, and N. Goldsman. Design techniques for streamlined integration and fault tolerance in a distributed sensor system for line-crossing recognition. In *Proceedings of the International Workshop on Distributed Sensor Systems*, pages 95-1-95-6, Honolulu, Hawaii, August 2007.

[64] C. Shen, R. Kupershtok, B. Yang, F. Vanin, X. Shao, D. Sheth, N. Goldsman, Q. Balzano, and S. S. Bhattacharyya. Compact, low power wireless sensor network system for line crossing. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2506-2509, New Orleans, Louisiana, May 2007.

[65] C. Shen, R. Kupershtok, S. Adl, S. S. Bhattacharyya, N. Goldsman, and M. Peckerar. Sensor support systems for asymmetric threat countermeasures. *IEEE Sensors Journal*, 8(6):682-692, June 2008.

[66] C. Shen, W. Plishker, S. S. Bhattacharyya, and N. Goldsman. An energy-driven design methodology for distributing DSP applications across wireless sensor networks. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*, pages 214-223, Tucson, Arizona, December 2007.

[67] C. Shen, W. Plishker, and S. S. Bhattacharyya. Design and optimization of a distributed, embedded speech recognition system. In *Proceedings of the 16th International Workshop on Parallel and Distributed Real-Time Systems*, pages 1-8, Miami, April 2008.

[68] E. Shih, S. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor

networks. In *Proceedings of the International Conference on Mobile Computing and Networking*, April 2001.

[69] M. Singh and V. K. Prasanna. System-level energy tradeoffs for collaborative computation in wireless networks norwell. In *IEEE International Conference on Communications*, 2002.

[70] Jens Sparsø and Steve Furber. *Principles of asynchronous circuit design: a systems perspective*. Springer, December 2001.

[71] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., 2000.

[72] X. Tang and J. Xu. Adaptive data collection strategies for lifetime-constrained wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, issue 6, pages 721 - 734, June 2008.

[73] P. Teehan, M. Greenstreet, and G. Lemieux. A survey and taxonomy of GALS design styles. *IEEE Design and Test of Computers*, vol. 24, no. 5, pp. 418-428, September-October, 2007.

[74] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel and F. Baez. Reducing power in high-performance microprocessors. In *Proceedings of the Design Automation Conference*, pages 732-737, 1998.

[75] P. P. Vaidyanathan. Multirate digital filters, filter banks, polyphase networks, and applications: a tutorial. In *Proceedings of the IEEE*, vol. 78, no. 1, pp. 56-93, January 1990.

[76] A. Wang and A. Chandrakasan. Energy-efficient DSPs for wireless sensor networks. *IEEE Signal Processing Magazine*, pages 68-78, 2001.

[77] X. Wang, T. Ahonen, and J. Nurmi. A synthesizable RTL design of asynchronous FIFO. In *Proceedings of the International Symposium on System-on-chip*, pages 123-128, November 2004.

[78] W. Wolf. *High-Performance Embedded Computing: Architectures, Applications, and Methodologies*. Morgan Kaufmann, September 2006.

[79] B. Yang, F. Vanin, C. Shen, X. Shao, Q. Balzano, N. Goldsman, and C. Davis. A low profile f-inverted compact antenna (fica) for wireless sensor networks. In *Proceedings of the IEEE AP-S International Symposium*, June 2007.

[80] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, June 2004, pp. 493–506.

[81] Tzyh-Yung Wuu and Sarma B. K. Vrudhula. Synthesis of asynchronous systems from data flow specifications. *Technical Report ISI/RR-93-366*, Information Science Institute, University of Southern California, December 1993.

[82] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm. Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland, Tech. Rep. 103, May 2001.

[83] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[84] Artisan ARM Physical IP, http://www.artisan.com.

[85] Cadence SOC Encounter User Manual, *Cadence*.

[86] Cadence Virtusoso Front to Back Design Environment, *Cadence*.

[87] CC1110 Data Sheet. SWRS033A, *Texas Instruments*.

[88] CC2430 Data Sheet. SWRS036F, *Texas Instruments*.

[89] The MOSIS Service, http://www.mosis.com.

[90] MSP430 Data Sheet. SLAU049F. *Texas Instruments*.

[91] SP0103 microphone sensor, *Knowles Acoustics*.

[92] Synopsys Design Compiler User Manual, *Synopsys*.

[93] TMS320C5509A Data Sheet. SPRS205I, *Texas Instruments*.

[94] Xilinx Virtex-4 SX FPGA, *Xilinx*.