ABSTRACT

Title of dissertation: HARDWARE AND SOFTWARE ARCHITECTURES FOR ENERGY- AND RESOURCE-EFFICIENT SIGNAL PROCESSING SYSTEMS

Inkeun Cho, Doctor of Philosophy, 2014

Dissertation directed by: Professor Shuvra S. Bhattacharyya
Department of Electrical and Computer Engineering

For a large class of digital signal processing (DSP) systems, design and implementation of hardware and software is challenging due to stringent constraints on energy and resource requirements. In this thesis, we develop methods to address this challenge by proposing new constraint-aware system design methods for DSP systems, and energy- and resource-optimized designs of key DSP subsystems that are relevant across various application areas. In addition to general methods for optimizing energy consumption and resource utilization, we present streamlined designs that are specialized to efficiently address platform-dependent constraints.

We focus on two specific aspects in our development of energy- and resource-optimized design techniques:

(1) *Application-specific systems and architectures for energy- and resource-efficient design.*

First, we address challenges in efficient implementation of *wireless sensor net-*

work building energy monitoring systems (*WSNBEMSs*). We develop new energy management schemes in order to maximize system lifetime for WSNBEMSs, and demonstrate that system lifetime can be improved significantly without affecting monitoring accuracy.

We also present resource-efficient, field programmable gate array (FPGA) architecture for implementation of orthogonal frequency division multiplexing (OFDM) systems. We have demonstrated that our design provides at least 8.8% enhancement in terms of resource efficiency compared to Xilinx FFT v7.1 when it is embedded within the same OFDM configuration.

(2) *Dataflow-based methods for structured design and implementation of energy- and resource-efficient DSP systems.*

First, we introduce a dataflow-based design approach based on integrating interrupt-based signal acquisition in context of parameterized synchronous dataflow (PSDF) modeling. We demonstrate that by applying our approach, energy- and resource-efficient embedded software can be derived systematically from high level models of dynamic, data-driven applications systems (DDDASs) functional structure.

Also, we present an in-depth development of *lightweight dataflow-Verilog* (*LWDF-V*), which is an integration of the LWDF programming model with the Verilog hardware description language(HDL), and we demonstrate the utility of LWDF-V for design and implementation of digital systems for signal processing. We emphasize efficient integration of LWDF with HDLs, and emphasize application of LWDF-V to design DSP systems with dynamic parameters on FPGA platforms.

HARDWARE AND SOFTWARE ARCHITECTURES FOR ENERGY- AND RESOURCE-EFFICIENT SIGNAL PROCESSING SYSTEMS

by

Inkeun Cho

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
2014

Advisory Committee:
Professor Shuvra S. Bhattacharyya, Chair/Advisor
Professor Martin Peckerar
Professor Neil Goldsman
Professor Manoj Franklin
Professor Patrick McCluskey

Acknowledgments

Foremost, I appreciate to god for his grace to let me complete long journey of Ph.D. I owe my gratitude toward my advisor, who is Dr. Bhattacharyya, for giving me opportunities to have research in interesting and challenging works. His devotion in research made a profound impression to me, and his sincere attitude for advising and working together with his student remind me an ideal professor. As a Ph.D student of him, I have learned how to define a research problem, build an initial idea to concrete research, and prove the uniqueness and worthy of research theme in technically and formally. It was a great honor to be his student and had experiences to research with such a great person. I would like to thank to my Ph.D committees, who are Dr. Peckerar, Dr. Goldsman, Dr. Franklin, and Dr. McCluskey, for their advice and effort to finish my thesis. I am grateful to Dr. Tachwali and Dr. Hsu, who are working in Agilent Technologies Inc., for collaboration on FPGA research. I also would like to thank my DSPCAD group colleagues for working in projects. Dr. Shen helped me in embedded software design for wireless sensor network by discussing in research idea, and guiding me to solve the technical problem that I have encountered during research. Kishan helped me to apply the DDDAS into new dataflow design methodologies. It was great experience to work and interact with Hojin, Ruirui, George, Zhou, Scott, Nimish, Wu, Stephen, Lai-Huei, Ilya, Soujanya, Shenpei, William and Allen. They helped my research by discussing idea with me and giving me comments. I would give my sincere gratitude my parents, who always give me infinite support and love, and I engraved their efforts that they had for me

in my mind. I also appreciate my brother, who is Sungkeun, for his support to take care of our family. Thanks to my parents in law for their praying for me and supporting my family. I would like to give my sincere appreciation to my beloved wife, who is YouKyung Lee, for her devotion in supporting me during my Ph.D. Her great efforts to take care of family make me focus on the research, and I owed everything to her in finishing my Ph.D. Thanks to my precious daughter, who is Ellie SeoHyeon Cho, for growing up well.

The research underlying this thesis was supported in part by Agilent Technologies, the Austrian Marshall Plan Foundation, the Laboratory for Physical Sciences, the Maryland Industrial Partnerships (MIPS) Program, the US Air Force Office of Scientific Research, and the US National Science Foundation.

Table of Contents

List of Figures

Chapter 1: Introduction

Digital signal processing (DSP) technology is used in many applications, including applications for wireless communication, image processing, and speech processing, to name a few. Design and implementation of DSP systems is often challenging due to stringent, multi-dimensional constraints that must be satisfied — e.g., real-time performance constraints when processing high volume data streams, or energy consumption constraints in low power wireless sensor networks (e.g., see [3]). Resource-constraints are also highly relevant in many signal processing application areas, such as consumer-oriented areas, where cost is often a major differentiator. For such applications, use of hardware resources may need to be carefully optimized to avoid the use (and associated costs) of more devices than necessary, more complex devices or device models than necessary, etc.

In this thesis, we address challenges in design and implementation of signal processing systems that must satisfy stringent constraints in hardware resource utilization and energy consumption. We focus especially on the application areas of wireless communication systems and wireless sensor networks, which are areas in which energy and resource constraints are often critical. We develop novel architectures and design methodologies to help optimize the energy and resource efficiency of systems for wireless communications and wireless sensor networking. Specific contributions of this thesis are outlined in remainder of this chapter.

1.1 Contributions of this Thesis

1.1.1 Design Methods for Wireless Sensor Network Building Energy Monitoring Systems

Wireless sensor network (WSN) systems are widely used in real-world applications, such as applications in military, environmental monitoring, healthcare, building management, and other areas. The specific types of protocols, architectures, and design methods employed in such systems depend on the kinds of sensing, processing and communication functionality required [4]. Building energy monitoring systems represent a promising area for widespread use of wireless sensor networks due to their potential for improving home automation and energy efficiency. In such systems, it is desirable for the employed sensor nodes to be battery-powered, and support long battery lifetimes to help reduce the frequency of required maintenance.

In this thesis, we present a new energy management method that is targeted to maximizing sensor node lifetime in wireless sensor networks for building energy monitoring. Our energy management method can be viewed as an application-specific approach for enhancing energy efficiency in wireless sensor networks. Our energy management method is based on approaches for energy consumption analysis at the sensor node, application, and network levels, and for systematically applying this multi-level analysis to improve the energy efficiency of building energy management systems. We validate our proposed energy management method through experiments on a fully functional building energy monitoring system that is based on a wireless sensor network.

1.1.2 Configurable, Resource-optimized FFT Architecture for OFDM Communication

Orthogonal frequency division multiplexing (OFDM) is widely used in modern wireless communication systems, including wireless LAN, WiMAX and 3G LTE. OFDM is known to be an efficient multiple access scheme for high data rate communication. Hardware platforms such as field programmable gate arrays (FPGAs) and application specific integration circuits (*ASICs*) are used to provide implementations for OFDM communication systems. Achieving area-efficient, hardware implementation for OFDM is a challenging and important problem in wireless communications.

FFT computation is the most computationally intensive part of an OFDM system. In Chapter 4 of this thesis, we address area-efficient FFT implementation for general OFDM architectures. Our proposed architecture can be configured across a range of different specialized OFDM requirements. We have identified inefficiencies in memory and resource utilization in conventional FFT implementations, and we have developed methods to address these inefficiencies to achieve a novel, area-efficient FFT hardware architecture. To validate our FFT architecture, we have apply relevant evaluation metrics for resource efficiency in FPGA platforms, and synthesize our proposed design on an off-the-shelf FPGA platform. We apply a widely-used commercial intellectual property (IP) core for FFT as a reference for our experimental comparisons, and through these experiments, we demonstrate the area-efficiency of our proposed design.

1.1.3 Data-driven Parameterized Synchronous Dataflow

Dataflow models of computation have been used in a wide variety of development environments to aid in the design and implementation of signal processing applications, and provide systematic approaches for design processes, including modeling, simulation, and implementation (e.g., see [3]). A limitation of existing dataflow-based design methodologies is systematic support for hardware interrupts. In Chapter 5, we develop a novel dataflow modeling approach, called *data-driven parameterized synchronous dataflow* (*DDPSDF*), to support efficient model-based development of signal processing systems in a manner that provides integrated modeling and management of hardware interrupts.

DDPSDF can be viewed as a structured integration of parameterized synchronous dataflow (PSDF) modeling of signal processing applications [5], and the DDDAS paradigm for dynamic steering of system operation and instrumentation based on run-time data characteristics [1]. In the area of WSN applications, DDPSDF allows for model-based integration of optimized, interrupt-driven processing and interrupt control in energy-constrained sensor nodes. We demonstrate the efficacy of our proposed DDPSDF techniques through a case study involving a a WSN application for integrated speech recognition and temperature sensing.

1.1.4 Lightweight Dataflow Design of Digital Systems

In Chapter 6, we present LWDF-Verilog (*LWDF-V*), which is an integration of the lightweight dataflow programming model with the Verilog hardware description language (HDL), and we demonstrate the utility of LWDF-V for design and implementation of digital systems for signal processing. *Lightweight dataflow* (*LWDF*) is a recently-introduced programming model for applying dataflow techniques to DSP system design in a manner that is relatively easy to learn and integrate into existing design processes, and that provides agility in retargeting designs across different kinds of platforms [2]. Use of LWDF-V facilitates the use of formal dataflow techniques in the design and implementation of signal processing systems. These techniques in turn are effective in exposing high level application structure, which can be exploited to optimize implementations in terms of key metrics, including energy and resource utilization efficiency [3]. In this work, we emphasize new features that have been developed in LWDF for efficient integration with HDLs, and the application of LWDF-V to design of DSP systems with dynamic parameters on FPGA platforms. We demonstrate our proposed LWDF-V programming model and methods for managing dynamic parameters through design and implementation of actors for inner product computation.

Chapter 2: Background

2.1 Dataflow-based Design of Signal Processing Systems

2.1.1 Dataflow Graphs

Model based design is used widely in many areas of embedded digital signal processing (e.g., see [3]). Dataflow is often used as a model of computation to provide the foundation for model-based design methods that are targeted to digital signal processing (DSP) systems. In dataflow modeling of DSP applications, the applications are modeled as directed graphs of actors (graph vertices) and dataflow graph edges. Actors express the computational components in an application and edges represent communication channels between actors. A data value is encapsulated in a *token* as it passes from the output of one actor to the input of another. Actors consume tokens from their input edges, process the data encapsulated by the tokens, and produce the resulting output tokens on their output edges. Edges in a dataflow graph correspond to FIFO buffers.

Execution of a dataflow graph follows a data-driven execution model [6]. An actor executes as a sequence of discrete units of computation called *firings*, and each firing depends on some well-defined amount of data on the input edges of the actor. We say that such an actor is *enabled* when this well-defined amount of data is present on the actor input edges (i.e., within the corresponding FIFO buffers). Due to the data-driven semantics of dataflow-based application models, the order in which actors fire is not part of the specification. The order is typically determined by the compiler, the hardware, or both.

2.1.2 Synchronous and Parameterized Synchronous Dataflow Graphs

Synchronous dataflow (*SDF*) [7] is a specialized form of dataflow that is popular in many design environments for DSP systems. In SDF graphs, the number of data values produced and consumed by each actor is fixed and known at compile time. This restriction provides increased predictably for SDF graphs and enables static scheduling [7].

Parameterized synchronous dataflow (PSDF) is a modeling approach that can be viewed as an integration of synchronous dataflow (SDF) with the meta-modeling framework of parameterized dataflow [5]. In contrast to SDF, PSDF describes a wide range of dynamic dataflow behaviors with flexible forms of dynamic parameter configuration allowed in a design. PSDF provides a systematic framework for integrating various SDF analysis techniques into a more general, dynamic dataflow setting.

The basic unit of modeling in PSDF is a *PSDF specification*, and PSDF specifications can be composed hierarchically to construct complex systems. A PSDF specification Φ consists of three distinct subgraphs: the *init graph* (Φ_i), *subinit graph*

(Φ_s), and *body graph* (Φ_b) [5]. Intuitively, Φ_b in PSDF is typically employed to model the core signal processing behavior of a subsystem (e.g., speaker keyword identification), while Φ_i and Φ_s are used primarily to specify how parameter updates are determined to the configure or reconfigure the body graph functionality.

When a PSDF subsystem Φ executes, the associated init graph Φ_i is invoked prior to each invocation of the associated (hierarchical) parent subsystem, while Φ_s is invoked prior to each invocation of the associated body graph Φ_b. Reliable and efficient implementations can be derived from PSDF graphs using quasi-static scheduling techniques that analyze the structure and parameters of the underlying PSDF models [5].

2.1.3 Core Functional Dataflow

enable-invoke dataflow (EIDF) is a flexible dataflow model that can express actors that involve both static and dynamically determined dataflow (production and consumption) rates. For DSP system design, the most relevant subclass of EIDF specifications is *core functional dataflow* (*CFDF*), which ensures deterministic execution [8].

Intuitively, a CFDF actor A is has an associated set $modes(A)$ of *modes*, where each mode $\mu \in modes(A)$ can be viewed intuitively as a specific "configuration" for actor execution. Each firing of a CFDF actor A has a unique mode associated with it, and in each mode, A produces and consumes tokens at constant rates on the ports of A. However, production and consumption rates can differ across different modes of the same actor, which allows for specification of dynamic dataflow rates.

A CFDF actor A also has two associated functions, called the *enable* and *invoke* functions of A, respectively. Intuitively, the enable function is a Boolean-valued function that takes as arguments a mode $\mu \in modes(A)$, a vector v_i of input buffer populations, and a vector v_o of output buffer populations. Each element of v_i corresponds to a specific input port of A, and similarly, each element of v_o corresponds to a specific output port of A, and these vector elements are all non-negative-integer valued. The enable function returns `true` for a given triple (μ, v_i, v_o) if and only if actor A is fireable (has sufficient data to fire) in mode μ when the volume of available input data and output buffer capacity is given by v_i and v_o, respectively. The invoke function can be described intuitively as a function that defines the mapping from input vectors consumed by the actor on its inputs and specific modes of operation (elements of $modes(A)$) into specific output vectors that are produced on its outputs. For further details on the CFDF invoke and enable functions, we refer the reader to [8]. Note that the original definition of the enable function in [8] does not consider finite output buffer capacities as we do in this paper, but this is a straightforward extension of the core enable function concept that was introduced in [8].

2.1.4 Lightweight Dataflow

Lightweight dataflow (LWDF) is a programming methodology for implementation of signal processing systems based on the core functional dataflow (CFDF) model of computation [2].

By avoiding dependence on specialized tools and facilitating retargetability to arbitrary host languages (languages for implementing individual actors), LWDF provides a minimally intrusive methodology for implementing applications based on the CFDF model, and can be incorporated efficiently into existing design processes [2]. In our approach to design and implementation of wireless sensor network building energy management systems (WSNBEMSs), which we develop in Chapter 3, we employ LWDF-C, which is the integration of the LWDF programming methodology with the C programming language. C is commonly used in developing embedded software for sensor nodes, and LWDF-C provides a framework for implementing such software through rigorous dataflow principles.

In LWDF, each actor has an operational context (OC), which encapsulates all parameters, local variables, and state variables of the actor, as well as references to the ports of the input and output FIFOs. Actor design in LWDF involves four interface functions called the *construct*, *execute* (also called *invoke*), *terminate*, and *enable* functions. The construct function performs one-time memory allocation and initialization associated with setting up the actor, and is typically called once at the beginning of the application. Similarly, the terminate function performs memory deallocation and other actor-level wrapup tasks, and is called when the actor is no longer needed, typically after the application is shut down or otherwise terminated.

Each call to the execute function performs a single firing of the associated actor provided that the input ports of the actor have sufficient input data. If the execute function is called when there is not sufficient data, the results are in general unpredictable. For each call to the execute function, the designer or enclosing tool can ensure sufficient data through appropriate static or quasi-static scheduling techniques or by first checking the status of the input buffers using the LWDF enable function. The enable function is a function that returns a Boolean value. This return value is `true` if and only if the actor has sufficient data on in its input edges and sufficient empty space on its output edges to carry out the next firing of the actor, including all of the required data consumption and production.

In CFDF and LWDF, the firing of actors and the associated tests that are performed by the enable function are formulated in terms of distinct *modes* of the actor. In general, an actor has one or more modes. A mode can be viewed as a "firing template" (functionality for a well defined subset of firings) in which the number of data values produced and consumed from the actor ports is constant. Thus, each mode has fixed data transfer (production and consumption behavior), but data transfer behavior can vary across different modes. Such decomposition of actor firing behavior in terms of fixed data transfer units is useful since data transfer characteristics generally have a strong influence on techniques for dataflow graph scheduling and other forms of dataflow graph analysis (e.g., see [3]).

Chapter 3: Design methods for Wireless Sensor Network Building Energy Monitoring Systems

3.1 Introduction

A *wireless sensor network*(*WSN*) is a distributed, wireless communication network system, where the nodes of the network (*sensor nodes*) contain groups of sensor devices. In addition to their associated sensors, sensor nodes typically consists of microprocessors, wireless communication interfaces, and energy sources. The small size and low cost of many modern sensor devices allow designers to deploy WSNs for a wide variety of applications.

Major research issues for WSNs include efficient hardware implementation and software stacks for energy-optimized wireless communication. Various low power processing units for sensor nodes have been developed with objectives of application-specific energy optimization. Development of new sensor technologies is also an important research area that gains motivation from the rapid evolution of WSN applications.

Research on design methods for WSN applications has received relatively little attention compared to the aforementioned WSN-related research areas. Design of WSN applications requires careful attention to operating system, data acquisition, and communication protocol aspects and their interactions. The complexity of these interactions and the need to manage this complexity under strict energy constraints contribute significantly to the difficulty of WSN application development. In this chapter, we address this difficulty by developing new design methods for WSN systems based on the formal application modeling framework of signal processing dataflow graphs. Our methods help to ease the task of implementing WSN applications, and to optimize their energy efficiency so that they can operate for longer periods before they need to be serviced or replaced.

More specifically, we develop a number of contributions to energy analysis and optimization of sensor nodes in WSN application. First, we develop a new energy analysis method for this class of sensor nodes at the application level and network level. Then we apply this energy analysis method to develop a new energy management scheme that is targeted to maximizing end node lifetime in building energy monitoring system(*BEMs*). To enhance the efficiency and reliability with which this management scheme can be implemented, we develop a new application-level design approaches of two representative application in each group that uses dataflow to model the application-level interfacing behavior between the processor and sensors on an individual sensor node. At the network level, we analyze the energy consumption in the 802.15.4 Zigbee network medium access control (MAC) layer based on the state of the sensor node radios. We validate our methods for energy analysis and optimization through experiments on a fully functional building energy monitoring system in which the sensor nodes are equipped with Texas Instruments CC2530

Zigbee network-enabled micro controllers [9]. This chapter builds on our published research on design methods for WSN BEMSs [10].

3.2 Related Work

3.2.1 Wireless Sensor Network Applications

WSN systems are widely used in real world applications. Akyildiz et al. present a survey on various application areas of WSNs, and categorize WSN applications as military, environmental, health, home, and other commercial areas depending on the kinds of sensing, processing and communication functionality provided [11]. In military applications, WSN systems can be used for applications such as equipment monitoring or detection of chemical attacks. WSNs can be applied for flood or mountain fire detection in environmental applications, and used to track physical data for patients in healthcare applications.

WSNs can be categorized as monitoring systems and signal processing systems. For example, flood detection, physical data tracking, and building energy monitoring involve periodic reporting of sensed data, which is characteristic of monitoring systems. On the other hand, intruder detection and home automation typically involve intensive image, acoustic, or speech signal processing, and are thus representative of signal processing oriented WSNs. In monitoring systems, the behavior of a sensor node exhibits periodic, alternating mode changes from active to sleep states depending on the reporting schedule. In signal processing oriented WSNs, the system continually acquires and processes sensed data for event detection, and classification.

We develop new design methods for monitoring and signal processing oriented WSN systems. Our methods apply the recently introduced lightweight dataflow programming model [2] [12], which provides a flexible and retargetable approach for developing monitoring and signal processing applications based on formal dataflow graph semantics. Using lightweight dataflow programming and dataflow graph analysis, we have developed systematic methods for measuring and optimizing energy consumption in sensor nodes. We have demonstrated and refined these methods using a representative application in the monitoring domain, and in our future work, we propose to address also the signal processing domain. Specific applications that we are focusing on include building energy monitoring (for the monitoring domain), and distributed speech recognition (for the signal processing domain).

3.2.2 Wireless Sensor Network Building Energy Monitoring System

3.2.2.1 Wireless Sensor Network Existing Researches on Network Lifetime

Zhaohua and Mingjun [13] present a survey on network lifetime research for wireless sensor networks. Zhang, Jia, and Yuan [14] and Padmavathy [15] have proposed network protocols and assocaited algorithms for improving energy efficiency. Wang and Kulkarni [16] consider scenarios in which sensor nodes are over-deployed in certain regions, and explore trade-offs between network lifetime and coverage through application of partial coverage transmission. Madan et al. [17] investigate cross layer design for the physical, MAC, and routing layers to maximize lifetime in wireless sensor networks.

These are largely general-purpose approaches, which are formulated without taking into account detailed characteristics of the targeted wireless sensor network or monitoring application.

Various bodies of research have targeted WSNBEMS technology (e.g., see [18] [19] [20]). Jang, Healy and Skibniewski [19] develop a building monitoring system with associated hardware, a web-based user interface, and monitoring software; evaluate this system on a wireless sensor network; and demonstrate that such a network can be employed to provide low cost monitoring. Chintalapudi et al. [20] show how wireless sensor networks can be used in structural health monitoring, and describe actual monitoring systems that are deployed to monitor real structures. These efforts have emphasized the development and demonstration of practical wireless sensor network based monitoring systems, and have not placed significant emphasis on optimizing energy consumption or maximizing lifetime for the network nodes.

Our research differs from these approaches in that we develop methods to exploit specific characteristics of our targeted class of WSNBEMSs, and streamline the design to maximize network lifetime based on these characteristics. Also, whereas an important body of related work focuses on network level considerations, the approach developed focuses on trade-off analysis and optimizations that take into account application level quality of service (reporting accuracy).

We therefore provide an additional layer of application-driven optimization that is geared towards the important domain of WSNBEMS and goes beyond what can be provided by more generic methods. As part of this application-driven approach, we model the target application in terms of core functional dataflow [8], which allows us to formally capture the signal processing functionality in a form that can be efficiently analyzed and mapped into hardware and software. Additionally, we apply new techniques for MAC layer energy analysis and we apply these techniques to optimize transceiver energy consumption in sensor nodes.

3.3 IEEE 802.15.4 MAC Energy Modeling

3.3.1 IEEE 802.15.4 MAC Layer Overview

The 802.15.4 Zigbee network MAC layer protocol is a hierarchical protocol in which at any given time, a single *coordinator* node controls a set of one or more *end nodes* in the network. This protocol has two operation modes, which are referred to as the *beacon-enabled mode* and *non-beacon mode*. The non-beacon mode works with carrier sense multiple access with collision avoidance (CSMACA) communication.

In the non-beacon mode, data communication from end nodes is allowed at arbitrary times. To support such flexibility, the coordinator continually monitors the communication channel for messages from the associated end nodes. Such monitoring is expensive in terms of energy consumption, and furthermore the flexibility it provides is not needed in our targeted WSNBEMS applications, where data communication from end nodes can be restricted to occur at pre-specified, periodic times. Thus, we are able to perform all of the intra-node communication in our wireless sensor network architecture using only the beacon-enabled mode, which is more energy efficient.

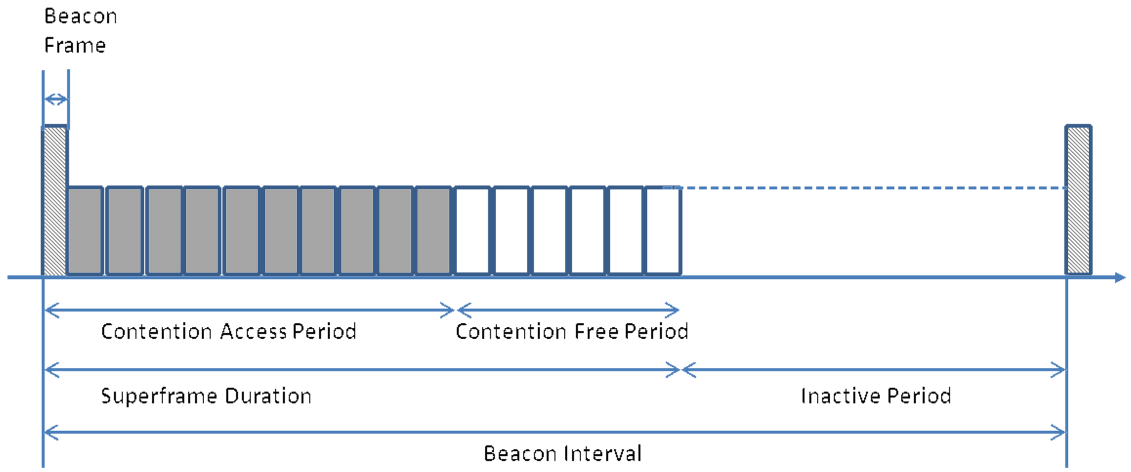


Figure 3.1: IEEE 802.15.4 Zigbee Network Superframe Structure

In the beacon-enabled mode, communication is governed by a *superframe* structure, which is illustrated in Figure 3.1. The superframe structure is composed of four parts, which are called the beacon frame, contention access period (CAP), contention free period (CFP), and inactive period. The superframe structure (indicated by the interval in Figure 3.1 labeled *superframe duration*) is composed of 16 time slots, where each slot corresponds to communication from at most one end node to the coordinator. The beacon frame is used to synchronize all of the nodes in the network. In the CAP, the end nodes that are ready to transmit data (ready nodes) check their back-off timers (the values of these timers are randomly assigned), and wait until their respective timers expire. After appropriate back-off, a ready node sends data if the channel is clear; otherwise, its backoff timer is set to another random value, and its backoff process is repeated. Through the CFP, a coordinator node can selectively assign guaranteed time slots to specific nodes, and nodes that obtain such permissions can send data during the CFP without channel sensing.

3.3.2 Energy Modeling of Beacon-enabled Mode

End node lifetime can be estimated by dividing the battery capacity (in Joules) with the average power consumption of an end node (in Watts). In our target application, the average power consumption of an end node can be divided into application layer and MAC layer power consumption. The application layer power consumption includes the power expended for acquiring sensed values from the attached sensors, such as temperature, light, and humidity sensors. The MAC layer power consumption includes the power expended for handling MAC events, the transceiver power consumption for transmission of sensed data, and sensor node sleep mode power consumption during the inactive periods of network beacon intervals. For simplicity, we take into account only the MAC layer data transmission, and ignore the effects of communication that are due to other factors. Although this is a significant simplification, we show in our experiments that it does not have a major affect on the energy consumption *trend* in our experiments; that is, we can compare alternative design configurations with reasonable accuracy under this assumption.

3.4 Energy Monitoring

In this section, we describe our approach to energy analysis for BEMS applications.

3.4.1 WSNBEMS System Model

In the model of WSNBEMSs that we apply, end nodes, which are equipped with heterogeneous sets of measurement sensors, are deployed in fixed positions, and these end nodes periodically send sensor measurements to a central network node, which we refer to as the *coordinator node*. The coordinator node collects data from all of the end nodes, and processes the data to determine properties of overall building energy consumption.

In the WSNBEMS testbed that we are experimenting with, the wireless communication range for end nodes is approximately 30 meters, and router nodes are used for multi-hop networking and clustering.

Each end node in our testbed is composed of a microcontroller for data processing and peripheral control, and a wireless transceiver subsystem for communication with other nodes. The microcontroller on an end node communicates with its associated sensors and acquires environmental measurement data using peripheral communication protocols.

We encapsulate the functionality for sensor interfaces in LWDF actor implementations so that sensor interfaces are integrated into the overall design using the same general module design approach as all other functional components. Thus, regardless of whether the sensors support I2C, SPI or some other protocol, the sensor interfaces communicate with other actors in the same way, which makes it easy to integrate sensors into the end nodes using a standard, protocol-independent approach. Moreover, the standardized interfacing and encapsulation as dataflow components facilitates more comprehensive model-based analysis, including energy estimation, for the overall application. This feature, for example, allows designers to derive useful estimates of energy consumption for alternative designs before deploying and testing the design in real environments.

3.4.1.1 Application Energy Modeling

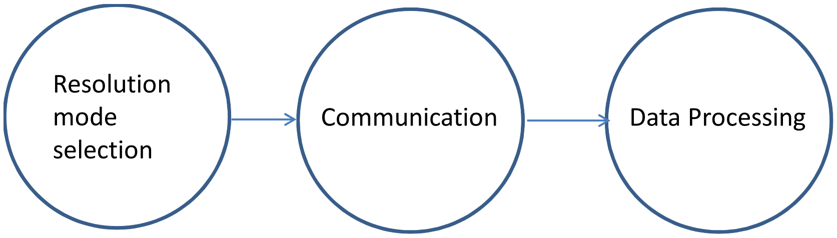


Figure 3.2: LWDF application model for the targeted WSNBEMS application

Commonly, environmental sensors support multiple sensing modes having different resolutions. Such sensing mode alternatives provide for trade-offs among the resolution of the data arriving from the sensor interfaces, number of bits required to communicate the sensed data across the interface and network, and time required to obtain the sensor readings. Resolution modes for sensors are generally configured by sending appropriate instructions to the sensors.

The mode based decomposition of LWDF firings provides a natural method for specifying functionality associated with different resolution modes within LWDF actors. In our WSNBEMS end node design, we provide for dynamic selection of sensor resolution modes through a *resolution mode selection* actor. Such an actor can, for example, be connected to a bank of switches or process configuration data sent by the coordinator node to dynamically adapt the resolution mode based on user input or coordinator node decisions. On each firing, the resolution mode selection actor reads the resolution settings from the appropriate configuration interface, packages the settings as a sequence of dataflow tokens (data packets), and passes these tokens to one or more subsequent actors for controlling how sensor data is read and transmitted.

```
boolean resolution_mode_selection_enable() {
boolean result = FALSE;
switch(context->mode) {
        case HIGH_resolution:
       result = TRUE;
           break;
        case LOW_resolution:
       result = TRUE;
           break;
        default:
           wsnbems_exception(context, INVALID_ACTOR_MODE);
           break;
}
    return result;
}

void resolution_mode_selection_invoke() {
```

```
switch(context-> mode) {

        case HIGH_resolution:

            value=generate_inst(HIGH_resol);

            lwdfc_fifo_write(&value);

            context->mode=decide_next_mode();

            break;

        case LOW_resolution :

            value=generate_inst(LOW_resol);

            lwdfc_fifo_write(&value);

            context->mode=decide_next_mode();

            break;

        default:

            wsnbems_exception(context, INVALID_ACTOR_MODE);

            break;

    }

}
```

Figure 3.2 illustrates our LWDF-based application model for our experimental WSNBEMS application. As an example of actor programming in our design, selected code is sketched below for the resolution mode selection actor.

We analyze the energy consumption of the end node functionality in terms of the LWDF application model of Figure 3.2.

The overall application energy is estimated as the sum of energy consumption estimates that are derived for the individual actors. In the estimation process, each component of energy consumption (E_{en}, E_{fifow}, etc.) is derived by instrumenting the application to determine the amount of time spent in the associated processing state, and multiplying by an estimate of the power consumption in each state. The instrumented values are obtained through actual application execution on the targeted hardware (not by simulation).

The overall energy estimation approach is summarized by the following equations.

$$
\begin{aligned}
E_{app} &= E_{res} + E_{com} + E_{dp} \\
E_{res} &= E_{en}(res) + E_{fifow} + E_{mdch}(res) + E_{inst} \\
E_{com} &= E_{en}(com) + E_{fifor} + E_{intfmode} + E_{fifow} \\
E_{dp} &= E_{en}(dp) + E_{fifor} + E_{dps}
\end{aligned}
$$

Table 3.1 provides a legend of the symbols that are used in these energy modeling equations.

3.5 Energy Efficient Management Scheme for WSNBEM

In the previous sections, we have developed models for analyzing the energy consumption of our targeted WSNBEMS, both in terms of application- and MAC-related energy consumption. To derive energy estimates from these models, we apply appropriate hardware platform parameters. For example, Table 3.2 shows relevant platform parameter values for the Texas Instruments CC2530 microcon-

E_{app}	Application Energy
E_{res}	Resolution Selection Actor Energy
E_{com}	Communication Actor Energy
E_{dp}	Data Processing Actor Energy
E_{en}	Actor Enabling Energy
E_{fifow}	FIFO Write Energy
E_{fifor}	FIFO Read Energy
E_{mdch}	Energy for Mode Change
E_{inst}	Energy for Instruction Generation
$E_{intfmode}$	Sensor Interface Communication Energy
E_{dps}	Data Processing Energy

Table 3.1: Terminology for Energy Analysis

Parameter	Value
$\text{Current}_{\text{TX}}$	39.6 mA
$\text{Current}_{\text{Active}}$	20.5 mA
$\text{Current}_{\text{Sleep}}$	1 uA
t_{timeslot}	320 us
V_{op}	3.3 V

Table 3.2: Hardware platform parameters for the Texas Instruments CC2530.

troller, which supports 802.15.4 Zigbee networks, and is the platform used in our experiments. However, we note that through its basis in dataflow-based application representations, our energy analysis methodology is easily retargeted to other platforms, and is not specific to the hardware used in our implementation.

As an example of how this kind of energy modeling can be applied, Figure 3.3 shows how end node lifetime varies as we increase the time interval for transmission, and apply the parameters in Table 3.2 and a battery with an energy capacity of 1250mAh. As we would expect, the lifetime has a tendency to increase with increasing intervals for transmission, and our detailed energy analysis approach helps to quantify this trend so that we can identify a suitable trade-off based on application requirements.

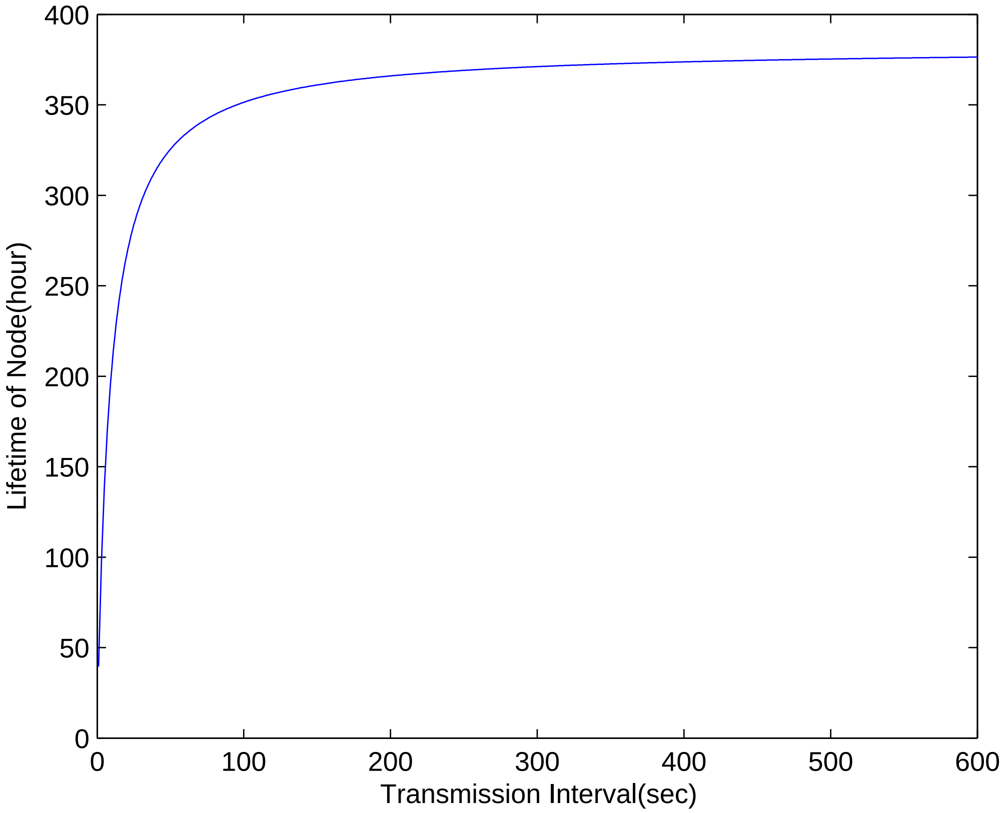


Figure 3.3: Node lifetime versus transmission interval.

3.5.1 End Node Based Energy Saving Scheme

In this section, we present the *end node based network lifetime* (*ENBESS*) scheme, which is an energy saving scheme that helps to extend end node lifetime by adjusting the transmission interval based on relevant operating conditions. This approach is motivated by the trade-off between transmission interval and node lifetime, which was discussed above.

ENBESS is motivated by the observation that environmental data of interest in our application, such as temperature and light, is relatively stable in building environments. ENBESS exploits this stability by maintaining a moving average of sensed data for each monitoring modality (temperature, humidity, light, etc.) of interest, and transmitting at increasingly long intervals when the current sensed values are sufficiently close to their associated moving average values.

When each modality is sensed, the moving average is updated. Then a weighted sum is used to determine the deviation of the current set of sensed values from the associated moving averages. This deviation is compared to a pre-specified threshold, and if the difference is within the threshold then the data transmission time is increased; on the other hand, if the deviation is moderately outside the threshold, then the transmission time is decreased. If the deviation from the threshold is outside of a pre-specified *normal range*, then a much shorter *exception interval* is used to notify the controller node. This notification alerts the controller node to an exceptional change in sensed value status, which could indicate, for example, a device failure or dangerous event in the environment.

The ENBESS method is sketched by the pseudocode shown in Algorithm 1. By strategically adjusting between shorter and longer transmission intervals, the method provides for both accurate data reporting to the coordinator node, as well as energy efficient operation on the end nodes.

In Algorithm 1, a provides a weight for updating the moving average value, which is described above. F_{int} is a weight that is applied to the difference between the newly updated moving average and stored moving average, and W_{sen} is a coefficient applied to each sensed value measurement. The value of W_{sen} should be determined based on the units in which sensor value measurements are provided and the associated magnitude ranges. For example, in our measurement scenario, temperature sensor values typically range from 20 to 35 degree Celcius, while light sensor values range from 0 to 800 degree Lux. In our experiments, we used coefficient values of 0.2 and 0.01, for temperature and light readings, respectively, to help normalize the values into similar ranges. t_{Base} provides a time offset that we used to update successive transmission intervals. In our experiments, we used 12 seconds as the value for t_{Base}.

3.6 Simulation and Experiments

3.6.1 WSNBEMS Testbed

To experiment with our methods for design and energy analysis, we have developed a fully functional, small scale WSNBEMS testbed. The network nodes in this testbed employ Texas Instruments CC2530 802.15.4 Zigbee network enabled (Z-Stack 1.4.0) microcontrollers.

Figure 3.4 shows an overview of the network in our testbed. The network includes one coordinator node, one router node, and three end nodes. Each end node is equipped with two environmental sensors — a temperature sensor and a light sensor. Both of these sensors communicate with the associated microcontroller using the I2C peripheral communication protocol [21].

The network is deployed in the A. V. Williams Building on the University of Maryland, College Park campus. End nodes periodically send sensed values for temperature and light to the coordinator node, and the router node relays informa-

Algorithm 1 End Node Based Energy Saving Scheme.

```
$Fint = 0$
**for** $i = 1$ to $Nsen$ **do**
    $Mavg[i] = a * Msen[i] + (1 - a) * Mpavg[i]$
    $Dsen[i] = Mavg[i] - Mpavg[i]$
    $Mpavg[i] = Mavg[i]$
**end for**
**for** $i = 1$ to $Nsen$ **do**
    $Fint = Fint + Dsen[i] * Wsen[i]$
**end for**
**if** $Fint > Lbase$ **then**
    $NexInterval = PreInterval + tBase * Fint$
**else if** $Fint \leq Lbase$ **then**
    $NexInterval = PreInterval - tBase * Fint$
**end if**
**for** $i = 1$ to $Nsen$ **do**
    **if** $Msen[i]$ is out of normal range **then**
        $NexInterval = ExceptionInterval$
    **end if**
**end for**
$PreInterval = NexInterval$
```

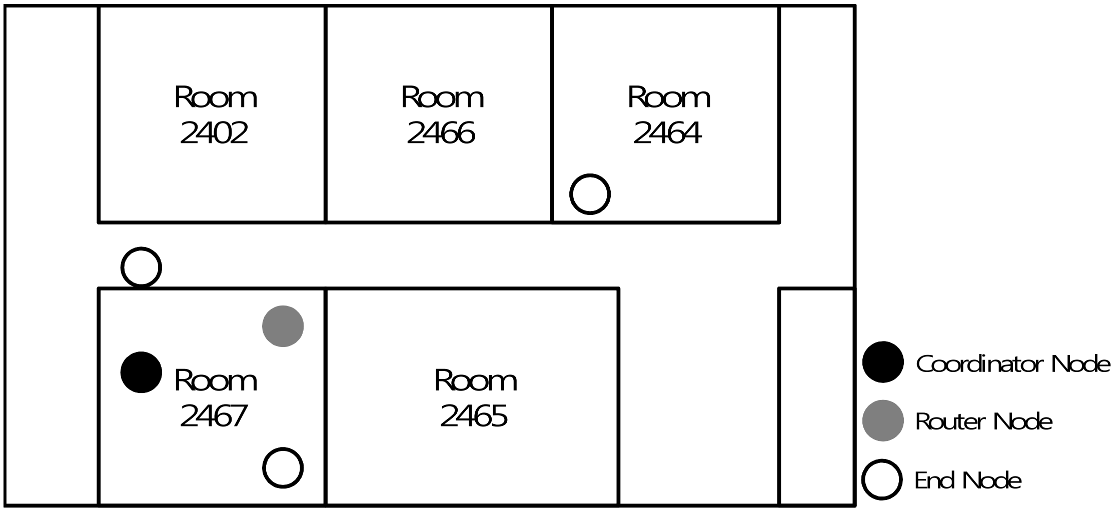


Figure 3.4: Network for WSNBEMS testbed.

tion to the coordinator node in case end nodes cannot reach the coordinator node directly.

3.6.2 Node Lifetime

In our testbed, the Texas Instruments Z-stack is used to implement the 802.15.4 Zigbee protocol, and the Operating System Abstraction Layer (OSAL) is used for node scheduling. The Z-stack is composed of 7 main layers — the MAC layer, network layer, hardware abstraction layer, application support sub-layer, monitoring task, interface for Zigbee application layer, and application layer. In our energy analysis, we consider in detail the MAC layer and application layer, so our energy analysis needs some model for differentiating results measured from our testbed with results obtained from our analysis. To help differentiate between testbed and analysis results, we define the term E_{layers} to represent the combined energy consumption of the network layer, hardware abstraction layer, application support sub-layer, and

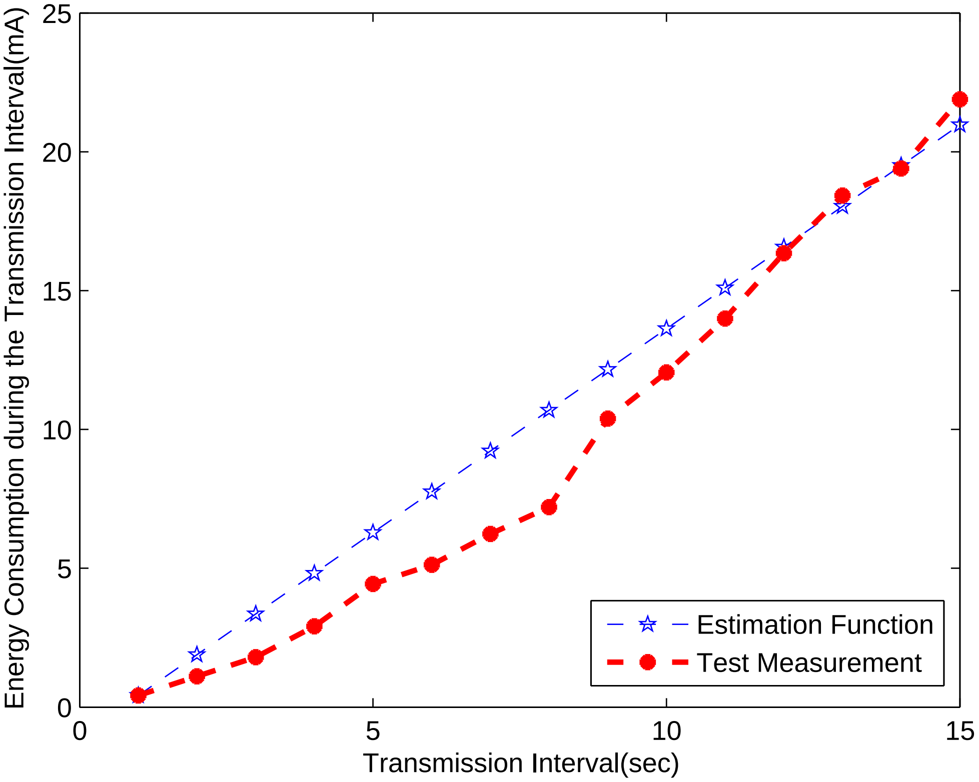


Figure 3.5: Energy consumption for all Z-stack layers except for the MAC and application layers.

interface for Zigbee application layer.

To estimate the energy consumption for these layers (i.e., all layers apart from the application and MAC layers), we measure the required numbers of clock cycles for processing the layers with increasing data transmission intervals. From these clock cycle counts, we can estimate the associated energy consumption values. The clock cycle counts are measured 15 times and the average across these 15 trials is used to derive the energy consumption estimates. Figure 3.5 shows our energy estimation results for the different layers.

Based on the results shown in Figure 3.5, we use a linear estimation function

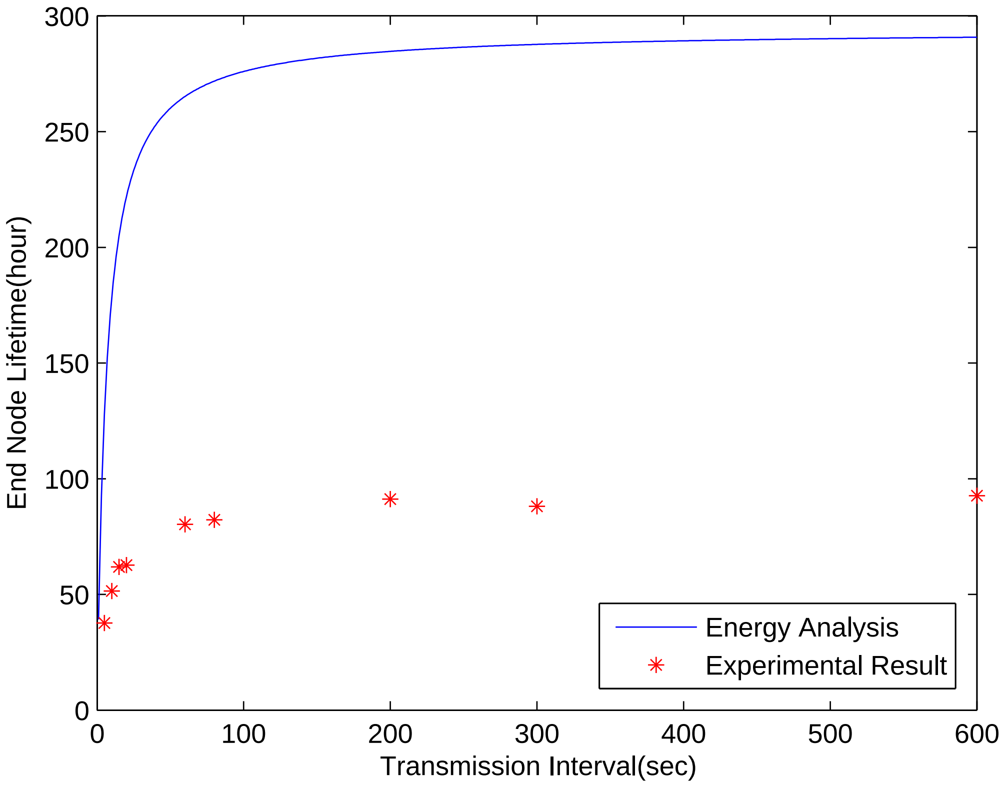


Figure 3.6: Test results for end node lifetime.

to provide the term E_{layers}.

Figure 3.6 presents experimental results based on the end nodes employed in our testbed. From these results, we can verify that end node lifetime increases with increases in the data reporting interval. However, measurements from the actual testbed give shorter end node lifetimes compared to the lifetimes estimated from our energy model analysis. Even if we revise the energy analysis by incorporating the E_{layers} term, the actual measured lifetimes are shorter than what the estimates predict. This indicates that there are other overheads that contribute to overall energy consumption but are not included or not adequately covered by our energy model.

However, the results also demonstrate that the measurement- and analysis-based energy consumption profiles follow similar trends, and thus the results from our analysis can be used to compare alternative design points with reasonable accuracy. This confirms the utility of our energy analysis approach in rapid prototyping, design exploration, and overall design methodology assessment.

3.6.3 Energy Savings Using ENBESS

To apply ENBESS, we must first ensure that the method is accurate — i.e., that the technique is effective at detecting exceptional conditions when they occur. An energy monitoring setup that has high lifetime but low accuracy will usually be of little use. In this context, we define accuracy under a given application scenario (e.g., based on a given experiment) as the ratio D/A, where D and A represent the number of detected exception cases, and the number of actual exception cases, respectively. If $A = 0$ for a given scenario, then the accuracy for that scenario is undefined.

The accuracy result can be affected by the weighted sum calculation in Algorithm 1 (i.e., the update of the `Fint` variable). Because of the difficulty involved in rigorous testing of exception conditions in an actual environment (e.g., the need to cause sudden swings in temperature, humidity, etc.), we performed accuracy assessment throught simulation. For this purpose, we modeled sensed values as having Gaussian distributions, and as a result, exception conditions were configured to occur during simulation with relatively low, but nonzero probability.

Figure 3.7 shows the accuracy and energy savings for various simulations. This is based on comparison with a static data transmission time interval of 40s, which is selected because it is the largest (most energy efficient) static interval that achieves a target accuracy of 75%.

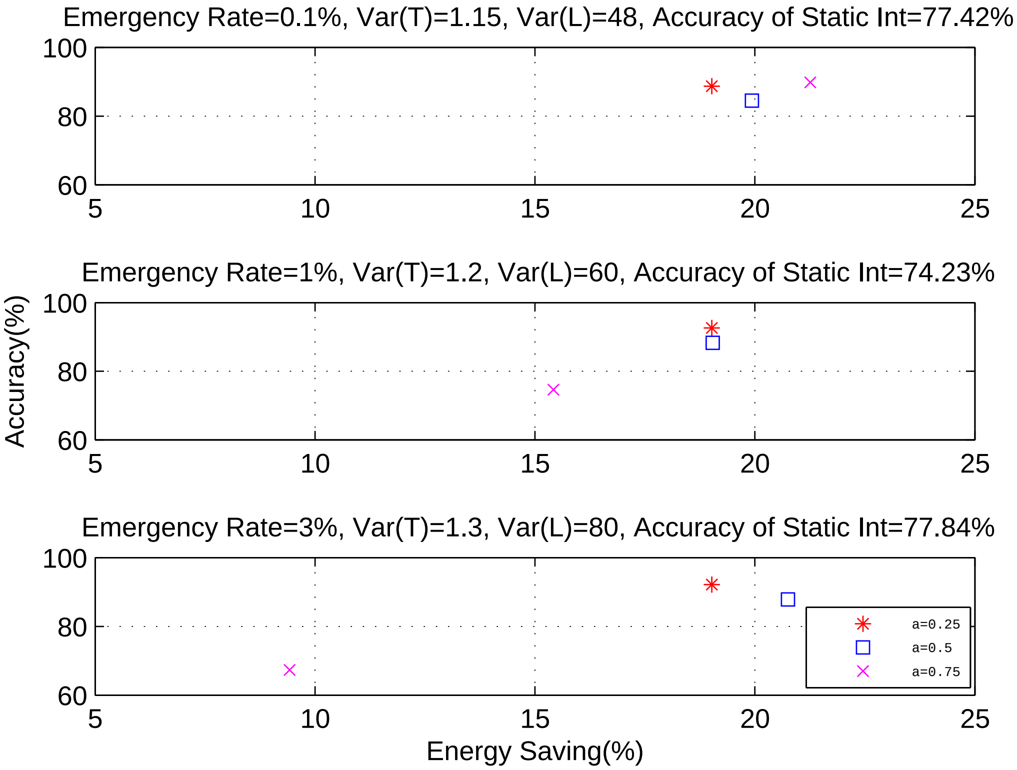


Figure 3.7: Simulation results for accuracy and energy savings.

The tests are held with different emergency rate value. As shown in Figure 3.7, ENBESS usually saves 9% or more energy and increases reporting accuracy compared to the use of static data transmission intervals. Thus, this value should be examined carefully through simulation to determine an effective setting based on

End Node Life Time	Hours	Average Interval
ENBESS with a=0.3	76h 15m	527s
ENBESS with a=0.5	77h 12m	512s
ENBESS with a=0.7	74h 23m	507s
Static Transmission Interval(40s)	60h 12m	

Table 3.3: Energy saving of ENBESS as determined from the WSNBEMS testbed.

expected operating conditions. The value of the weight values (in the update equation for `Fint`) in general will also affect the results. Analysis of this effect, as well as development of more systematic methods for determining a and the weighting factors for `Fint`, are useful directions for further investigation.

We have also experimented with the ENBESS scheme on real hardware in our WSNBEMS testbed. Table 3.3 shows the energy savings determined from these experiments. These savings are determined in comparison to the same 40s static interval case that was used in Figure 3.7. The results in Table 3.3 demonstrate significant improvement compared to the static interval case.

3.7 Conclusion

In this chapter, we have developed methods for system design and analysis of wireless sensor network building energy monitoring systems (WSNBEMs). We have developed methods for application modeling and energy analysis that help to quickly assess the energy efficiency of alternative design configurations. Motivated by results from our energy analysis, we have developed a method, called the End Node Based Energy Saving Scheme (ENBESS), for dynamically adjusting data transmission intervals in a WSNBEMS based on the degree to which sensed data deviates from corresponding moving average values. We have validated ENBESS along with our energy analysis methods in a fully functional, small scale WSNBEMS testbed deployed across multiple rooms in an actual building. Results from experiments on our testbed show that our energy analysis methods are useful in comparing alternative WSNBEMS design configurations, and that ENBESS can achieve significant improvements in energy efficiency and node lifetime through strategic adaptation of transmission intervals.

Chapter 4: Configurable Resource-optimized FFT Architecture for OFDM Communication

4.1 Introduction

As OFDM is widely used in wireless communication systems, efficient hardware implementation for OFDM is critical issue. FFT computation is the most computationally intensive part of an OFDM system. A large body of prior research exists on efficient implementation of FFTs. Two general forms of FFT architecture are the parallel input and output architecture [22] [23] [24], and the serial input and output architecture [25] [26]. The parallel input and output architecture is shown in Figure 4.2, and the serial input and output architecture is shown in Figure 4.3.

A parallel input and output architecture can offer relatively high throughput of computation, however it requires multiple computation units (*butterfly* processing units) for processing parallel inputs simultaneously, and additional hardware resources due to complex multipliers [27]. Since area efficiency is critical in mobile systems, this architecture is not well suited for OFDM implementation. On the other hand, a serial input and output architecture can be developed using single computation units along with random access memory (*RAM*), and can thereby offer resource efficient design for OFDM implementation.

Saeed et al. presented an FFT system in [26] that employs a simple radix-2 based processing unit instead of a radix-4 unit, and provides a configurable FFT size (number of points). This system also uses simple counter based control logic, which helps to reduce resource utilization. Our proposed solution adopts ideas from the work of Saeed et al., and further streamlines buffer memory and control logic implementation based on the context in the which an FFT unit operates within an overall OFDM system.

A variety of methods have been presented for implementing serial input and output FFTs for OFDM systems. Hung, Chen and Chen presented an address generator for memory and twiddle factors of variable-length FFTs for high speed and low complexity design [28]. Jung, Yoon and Kim demonstrated a method for efficient design of the radix-4 based butterfly [29] and multiplier in an FFT unit. This approach was shown to achieve enhancements in both area and throughput. Huang et al. presented a memory efficient OFDM system considering WiMAX subcarrier allocation scheduling and demonstrated results in terms of area efficiency [30]. However, these approaches are based on specialized OFDM system requirements — i.e., these methods exploit the specialized structure for specific OFDM system configurations to derive optimized designs.

In contrast, in this chapter we target efficient FFT implementation for general OFDM architectures, which can be configured across a range of different specialized OFDM requirements. Such general OFDM architectures are useful for prototyping as well as for cognitive radio applications, which may support many different communication standards within the same hardware.

For such generalized OFDM architecture design, we have identified inefficiencies in memory and resource utilization in conventional FFT implementations, and we have developed methods to address these inefficiencies to achieve a novel area-efficient FFT hardware architecture. To help support the overall context within a generalized OFDM design, our design provides significant user configurability in terms of FFT parameters, such as the number of FFT points, and fixed point representation format.

To validate our FFT architecture, we have synthesized it using Xilinx ISE version 13.4 with the Digilent Spartan 3e platform as the target. This target platform is based in turn on the Xilinx (*xc3s500efg320-4*) integrated circuit. We have selected the Xilinx FFT intellectual property (IP) core (Xilinx FFT IP core v7.1) as a reference for comparison, as it is a widely used commercial IP core.

4.2 FFT Design Optimization for OFDM

4.2.1 FFT Design with User Configurability

Orthogonal frequency division multiplexing (*OFDM*) is a multiple access scheme for high data rate communication. OFDM is based on frequency division multiplexing, where bandwidth is divided into a series of non-overlapping frequency bands, and each user is assigned a dedicated subset of the available bands. OFDM is a special case of frequency division multiplexing in which the sub-carriers are orthogonal to one another, which eliminates crosstalk between sub-carriers [31].

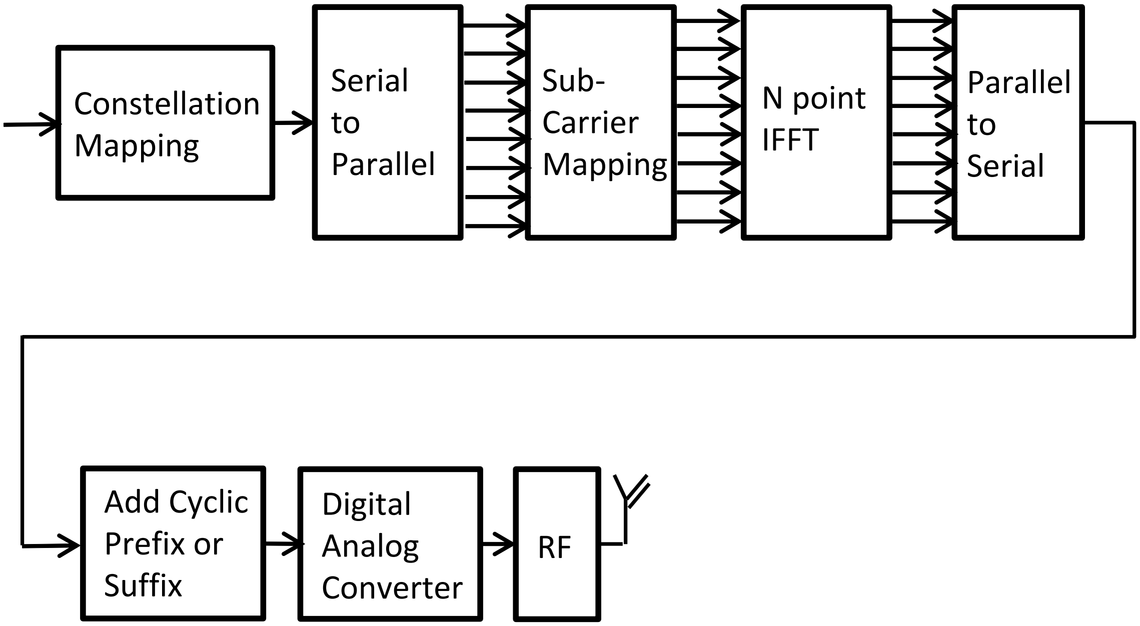


Figure 4.1: OFDM system overview.

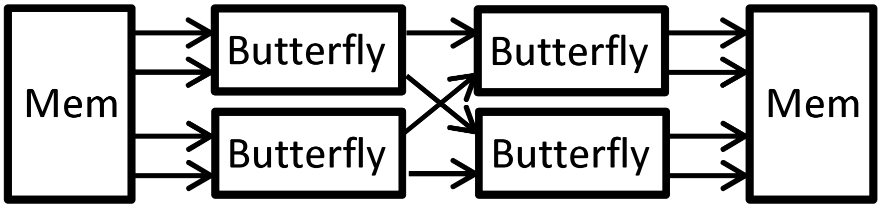


Figure 4.2: Pipelined parallel architecture for FFT.

OFDM offers a variety of advantages, including robustness against narrow band co-channel interference, inter symbol interference and fading caused by multipath propagation. Also, OFDM can readily be implemented with the fast Fourier transform (*FFT*). Due to these advantages, OFDM is widely used in modern wireless communication systems, including wireless LAN, WiMAX and 3G LTE. Figure 4.1 shows a system overview for OFDM.

OFDM modules are used in many communication systems, and OFDM configurations in general need to be varied with communication system requirements.

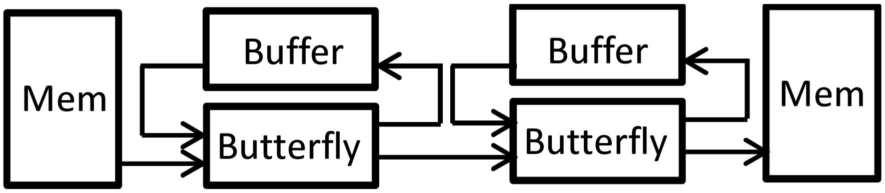


Figure 4.3: Pipelined serial architecture for FFT

To support a broad range of OFDM configurations, we consider a parameterized OFDM system design that supports flexibility across the OFDM design space. A block diagram of our targeted OFDM system design is shown in Figure 4.4. The contribution of this chapter is to present an optimized FFT architecture that is suitable for integration as a component of this system design.

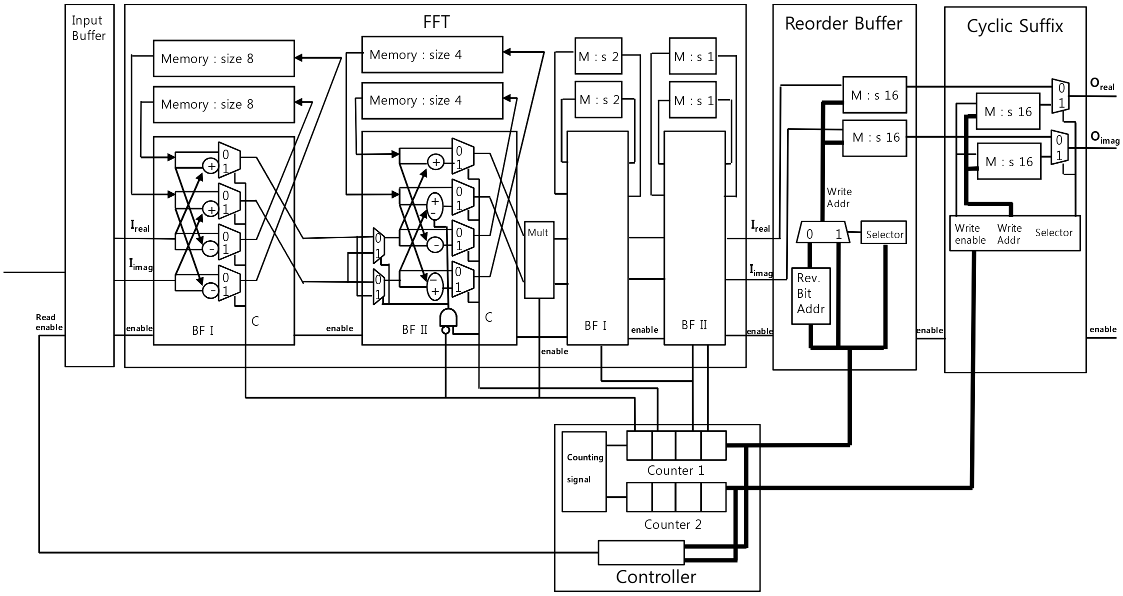


Figure 4.4: Block diagram of targeted OFDM system design.

There were plenty of FFT designs were researched and the FFT implementation is a well matured research area. In this chapter, we focused more on developing the efficient OFDM modules based on the FFT module which was presented in previous research. We chose the FPGA FFT design in [26] among plenty of system, because the system can easily expandable with the point of FFT for supporting multiple configuration and have a counter based control logic. The proposed configurable parameters for OFDM module provides user configurability in terms of parameters such as the number of FFT points, and fixed point representation format.

4.2.2 Memory Efficient Design of FFT for OFDM

In an OFDM system, a guard interval using a cyclic prefix or cyclic suffix is used between consecutive OFDM symbol durations [31]. Such prefix or suffix based guard intervals are used to combat inter symbol interference. When using a cyclic prefix, the OFDM output symbol size increases from N to $(N + C)$, where N and C represent the FFT size and cyclic prefix size, respectively. The OFDM symbol duration is the sum of the actual transmission time for an N-sized IFFT output and the guard interval time. The actual transmission time for an OFDM symbol is determined by the time taken by the physical layer electronics, and the RF transmission time. If the rate at which OFDM symbols are generated exceeds the throughput of the physical layer electronics, an additional FIFO is inserted to help compensate for the rate difference.

In OFDM system design, a buffer of size $(N + C)$ is inserted to handle cyclic prefix or suffix generation. In our design, we employ a cyclic suffix, and configure the FFT a with **stall_state**. In this state, which is employed during guard intervals, the FFT does not generate any output. After FFT data is generated, a dedicated **cyclic_suffix** module suffixes data in memory based on a suffix size parameter **C**. This approach allows for streaming generation of OFDM symbols before digital-to-analog conversion without requiring additional buffering, and provides a memory savings of

$$M_{save1} = \frac{N}{N+C}. \tag{4.1}$$

Within an OFDM system, a re-order buffer is typically used in conjunction with the FFT unit. The output streams for the IFFT/FFT computations are in bit-reversed order based on common conventions for IFFT and FFT implementation. A common buffer management scheme for re-order buffer and cyclic suffix design, which is presented in [32], uses a buffer of size $2N$ for re-ordering and continuous processing of FFT output data. However, with optimized buffer management, our design employs a smaller re-order buffer of size $(N + C)$. In particular, based on knowledge of how the FFT is employed in the surrounding OFDM context, we are able to replace the double buffering scheme of [32] with in-place buffering. This leads to a re-order buffer that is smaller by a factor of

$$M_{save2} = \frac{N-C}{2N}. \tag{4.2}$$

This represents a significant savings since typically C is much smaller than N in OFDM systems.

4.2.3 Simple Control Logic

Another opportunity that we exploit in our OFDM-oriented FFT subsystem is the ability to share control logic across key modules that interface to the FFT. In particular, we extend the controller for the core FFT unit with an additional counter, and use the resulting circuit as a unified controller for the re-order buffer and cyclic suffix modules ("interface modules"). This is in contrast to conventional approaches to OFDM system implementation that use off-the-shelf or otherwise generic FFT IP blocks and interface modules, which come with their own controllers. Our approach to sharing of control logic helps to further enhance the area efficiency of our approach, as we demonstrate quantitatively in experiments.

4.3 Experiments

4.3.1 Evaluation Metric

In this section, we demonstrate the performance and area efficiency of our OFDM-oriented FFT subsystem design. First, we define two evaluation metrics that we employ in our experiments. As a measure of resource utilization on the targeted FPGA devices, we define $R(D_i)$ to be the number of FPGA slices occupied by the synthesized result for a given design D_i. Also, given two designs D_i and D_j such that $R(D_j) > R(D_i)$, we use the metric $E_(D_i, D_j)$ to quantify the degree to which D_i is more resource-efficient compared to D_j. This metric is defined by:

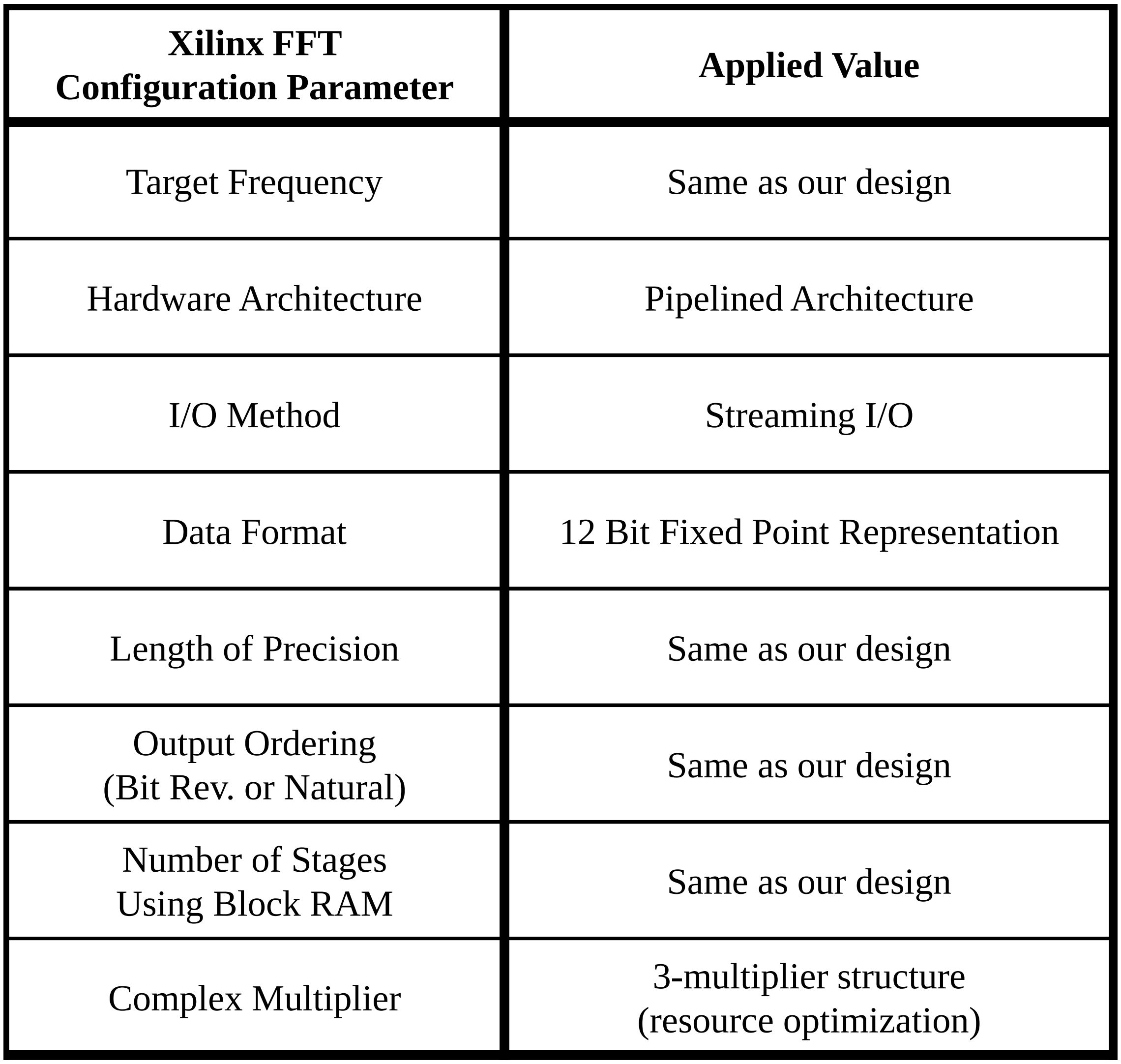
$$E_R(D_i, D_j) = (1 - \frac{R(D_i)}{R(D_j)}) \times 100(\%). \tag{4.3}$$

4.3.2 Performance of Our Proposed Design

To further test the performance of our proposed design, the design is synthesized under Xilinx ISE version 13.4 with the option of using the Digilent Spartan 3e platform, which is based in turn on the Xilinx integrated circuit. We have chosen FFT computation with cyclic suffix generation as the configuration for the evaluating system performance. The FFT module is the most computationally complex part in an OFDM system and the system resource usage is heavily related to this part. Also, the Xilinx IP generator supports FFT v7.1 instantiation along with features for reorder buffer and cyclic suffix usage. Since this core (the Xilinx FFT IP core v7.1) has these relevant features and it is a widely used commercial core, we employ it as a reference for comparison in our experiments.

Results obtained from our synthesis experiments are shown in Figure 4.6. In this Figure, R represents the amount of occupied logic resources, and M represents the amount of RAM memory blocks that are used. Each entry for R is expressed in the form x/y, followed by a percentage value z. Here, x represents the number of occupied logic resources in the synthesized design, y represents the total number of available logic resources on the targeted FPGA device, and z represents the percentage of occupied resources (i.e., the quotient (x/y)).

For fairness in our comparison, we maintained consistency where possible be-

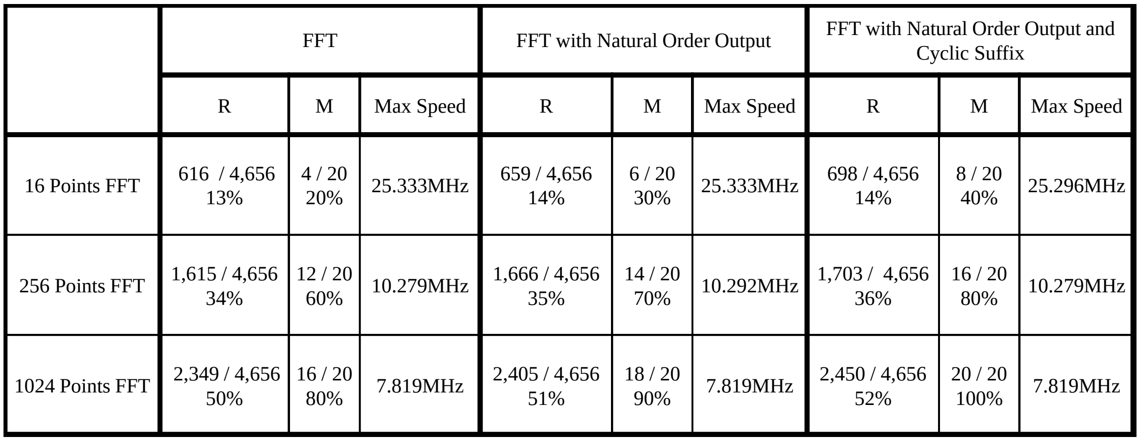


Xilinx FFT Configuration Parameter	**Applied Value**
Target Frequency	Same as our design
Hardware Architecture	Pipelined Architecture
I/O Method	Streaming I/O
Data Format	12 Bit Fixed Point Representation
Length of Precision	Same as our design
Output Ordering (Bit Rev. or Natural)	Same as our design
Number of Stages Using Block RAM	Same as our design
Complex Multiplier	3-multiplier structure (resource optimization)

Figure 4.5: Configuration for Xilinx FFT v7.1 IP Core Generation

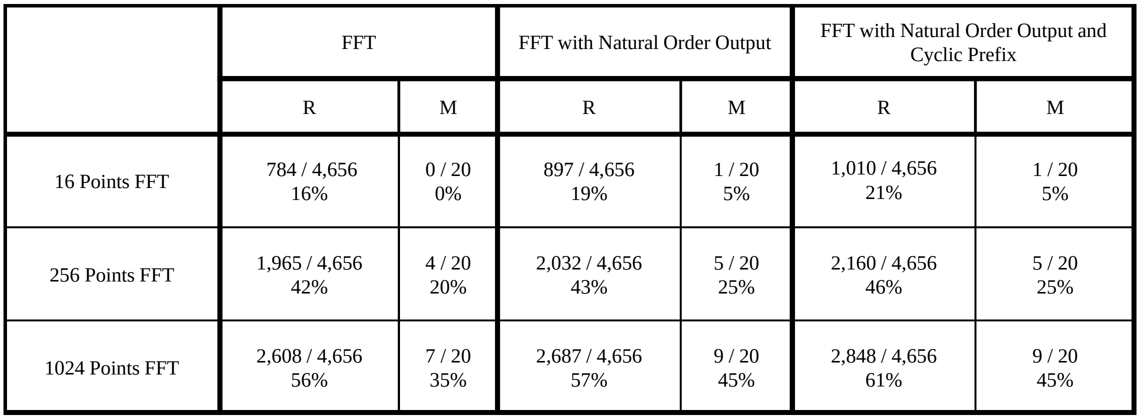
tween the configurations used for our design and the configurations used for generating the Xilinx FFT core. Figure 4.5 shows the configurations that we used to generate the Xilinx FFT core.

For the synthesis experiments using the Xilinx core, the target clock frequency is set based on the maximum frequency at which our proposed configurable FFT module can operate. Since the Xilinx core generator generally improves the resource utilization efficiency for lower speeds, this optimizes the resource utilization of the



	FFT			FFT with Natural Order Output			FFT with Natural Order Output and Cyclic Suffix		
	R	M	Max Speed	R	M	Max Speed	R	M	Max Speed
16 Points FFT	616 / 4,656 13%	4 / 20 20%	25.333MHz	659 / 4,656 14%	6 / 20 30%	25.333MHz	698 / 4,656 14%	8 / 20 40%	25.296MHz
256 Points FFT	1,615 / 4,656 34%	12 / 20 60%	10.279MHz	1,666 / 4,656 35%	14 / 20 70%	10.292MHz	1,703 / 4,656 36%	16 / 20 80%	10.279MHz
1024 Points FFT	2,349 / 4,656 50%	16 / 20 80%	7.819MHz	2,405 / 4,656 51%	18 / 20 90%	7.819MHz	2,450 / 4,656 52%	20 / 20 100%	7.819MHz

Figure 4.6: Performance of our proposed configurable FFT design.



	FFT		FFT with Natural Order Output		FFT with Natural Order Output and Cyclic Prefix	
	R	M	R	M	R	M
16 Points FFT	784 / 4,656 16%	0 / 20 0%	897 / 4,656 19%	1 / 20 5%	1,010 / 4,656 21%	1 / 20 5%
256 Points FFT	1,965 / 4,656 42%	4 / 20 20%	2,032 / 4,656 43%	5 / 20 25%	2,160 / 4,656 46%	5 / 20 25%
1024 Points FFT	2,608 / 4,656 56%	7 / 20 35%	2,687 / 4,656 57%	9 / 20 45%	2,848 / 4,656 61%	9 / 20 45%

Figure 4.7: Performance of Xilinx FFT core.

Xilinx core for the same speed that is achieved by proposed design.

For the complex multiplier configuration in the Xilinx IP core, we applied the **3-multiplier structure** among the three available options — **CLB-logic**, **3-multiplier structure** and **4-multiplier structure**. We selected the **3-multiplier structure** because it is the most resource-efficient option.

Figure 4.8 shows the resource efficiency of our proposed design over the reference design, based on the experiments summarized in Figure 4.6 and Figure 4.7. We see that our proposed configurable FFT module requires less logic resources

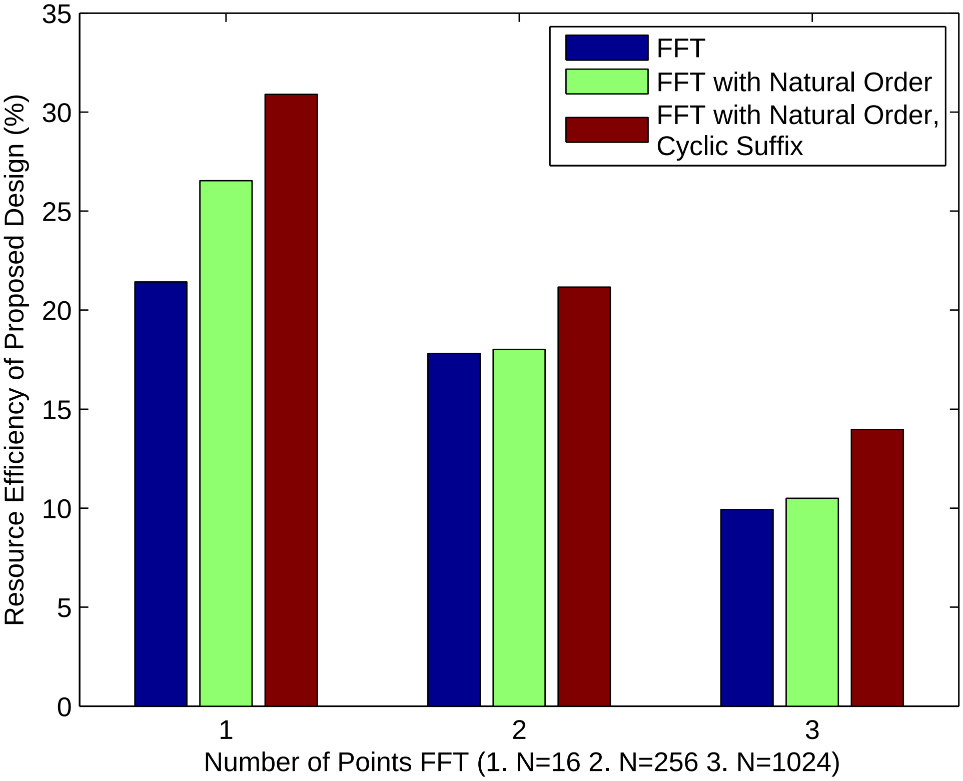


Figure 4.8: Resource efficiency of proposed design with block RAM.

than the Xilinx FFT core. Thus, our proposed module is advantageous for highly cost-constrained design scenarios with extensive requirements of computational resources, such as those that arise from trying to fit complex wireless communication subsystems into single-chip implementations.

Although, as shown in Figure 4.6 and Figure 4.7, our design is significantly more efficient in terms of logic resource requirements, it has increased requirements in terms of on-chip memory blocks.

An alternative to incurring the increased block RAM cost of our approach, is to employ distributed memory, which consumes logic resources (slices) rather than

	Num of Points	FFT	FFT with Natural Order Output	FFT with Natural Order Output and Cyclic Prefix
Proposed Design	16 Points FFT	608 / 4,656 13%	682 / 4,656 14%	737 / 4,656 15%
	256 Points FFT	1,884 / 4,656 40%	2,476 / 4,656 53%	3,016 / 4,656 64%
Xilinx FFT Core	16 Points FFT	784 / 4,656 16%	1,198 / 4,656 25%	1,247 / 4,656 26%
	256 Points FFT	2,067 / 4,656 44%	3,724 / 4,656 79%	3,847 / 4,656 82%

Figure 4.9: Performance comparison of our proposed design based on a distributed memory implementation.

block RAM. To study the impact on resource efficiency in this case, we synthesized the design using distributed instead block RAM, and and we compared the resulting resource usage with the Xilinx FFT core under the same conditions (i.e., by enforcing use of distributed RAM). Figure 4.9 shows that under these constraints, our proposed configurable FFT module requires less logic resources than the Xilinx FFT core.

Figure 4.10 shows the resource efficiency (E_R) for the distributed memory version of our proposed design compared to the Xilinx FFT core. This demonstrates an efficiency enhancement of 8.5% using our proposed OFDM-oriented FFT solution.

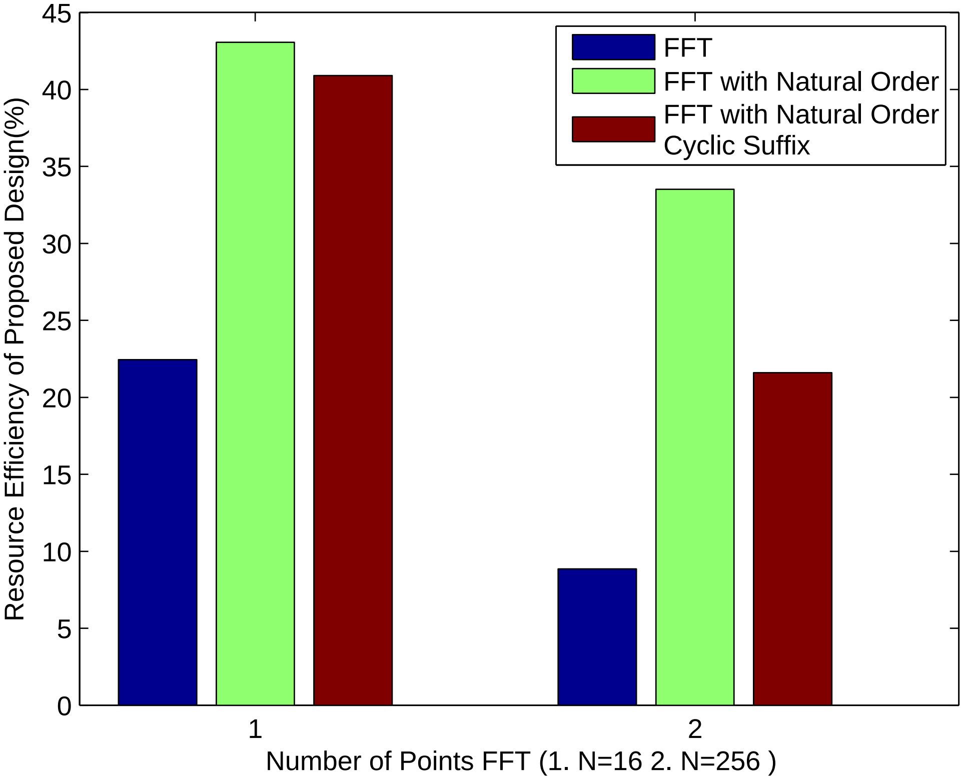


Figure 4.10: Resource efficiency of our proposed design using distributed memory.

4.4 Conclusion

In this chapter, we have presented a resource efficient design for a configurable FFT module for flexible integration into OFDM systems. Our design overcomes inefficiencies in the conventional use of FFT modules in OFDM systems, and provides extensive designer-configurability that helps to support a variety of communication system specifications. Our OFDM-targeted FFT design achieves efficient resource utilization through careful buffer memory optimization, and streamlining of control logic. To validate our proposed FFT design, we synthesized it on an FPGA target, and compared the synthesis results to an analogous configuration of a commercial, off-the-shelf FFT core. Results from synthesis show that our proposed design provides significant improvements in resource efficiency under both block RAM and distributed memory based implementations.

Chapter 5: Dynamic Resource Coordination and Energy Optimization in Sensor Network Platforms

5.1 Introduction

Wireless sensor networks (WSNs) can be applied to a variety of application domains such as environmental monitoring and surveillance. The resources of computation, communication, memory, and power on a typical sensor node platform are severely limited such that careful design and optimization are needed in order to meet mission-critical requirements. To address this problem, we have developed a system-level design approach for WSN systems, utilizing the *dynamic data driven applications systems* (*DDDAS*) paradigm [1] jointly with dataflow models of computation. Dataflow models of computation have been used in a wide variety of development environments to aid in the design and implementation of signal processing applications (e.g., see [3, 10, 33–39]).

When developing a design methodology for WSN applications, we need to carefully support optimizations for dynamic resource coordination and utilization of power-aware processing modes. Hardware interrupts are commonly used in sensor node platforms to handle events associated with data acquisition. Such interrupts help to efficiently drive signal processing subsystems and associated control functionality in a data-driven manner, where subsystem iterations, application mode changes, and other aspects of signal processing computations can be triggered based on the arrival of specific kinds of data from specific sensors. In this chapter, we have developed a novel dataflow modeling approach, called *data-driven parameterized synchronous dataflow* (*DDPSDF*) to support efficient model-based design and implementation of WSN systems based on such sensor-driven triggering of control and signal processing computations. DDPSDF can be viewed as a structured integration of (1) parameterized synchronous dataflow (PSDF) modeling of signal processing applications, and (2) the DDDAS paradigm for dynamic steering of system operation and instrumentation based on run-time data characteristics.

Dynamic configuration of resource usage and power consumption in WSN systems is critical to efficient WSN system design. For optimized performance, configuration changes need to be driven carefully by the current data characteristics and system status. This calls for rigorous integration of DDDAS design principles into design processes for WSN systems. Use of DDDAS design techniques involves tightly integrated feedback from instrumentation and application of dynamic parameters that are adapted based on such feedback, and that also control how subsequent rounds of instrumentation are performed. The DDPSDF design approach developed and demonstrated in this chapter is a first step towards systematic integration of dynamically parameterized dataflow modeling and scheduling techniques with DDDAS principles for robust and context-optimized operation of application systems under complex constraints.

To demonstrate our proposed DDPSDF methods, we present in this chapter a case study involving an embedded speech recognition system. We demonstrate that by applying the DDPSDF approach, and its associated interrupt-integrated style of PSDF modeling, energy- and resource-efficient embedded software can be derived systematically from high level models of DDDAS application structure.

5.2 Data-driven PSDF Modeling

The DDPSDF approach presented in this chapter integrates the interactions of platform-level *interrupt service routine* (*ISR*) operation with high-level PSDF modeling of signal processing applications. By integrating ISR operation within a parameterized dataflow framework, sleep and awakening operations for micro-controllers and sensor subsystems can be incorporated naturally with quasi-static scheduling strategies for efficient and structured execution of signal processing applications across WSNs. Integration of ISRs in this way also allows designers to efficiently adapt run-time operation of the system based on data acquired from various sensors under specific kinds of conditions (e.g., exceeded thresholds in signal magnitude or frequency).

DDPSDF modeling is a structured application of PSDF graphs to incorporate interrupt driven processing throughout the execution of a parameterized dataflow graph specification. DDPSDF exploits the flexibility in the parameterized dataflow framework for defining the underlying notion of a *graph iteration*, which defines the basic window of processing in PSDF during which parameter values are kept

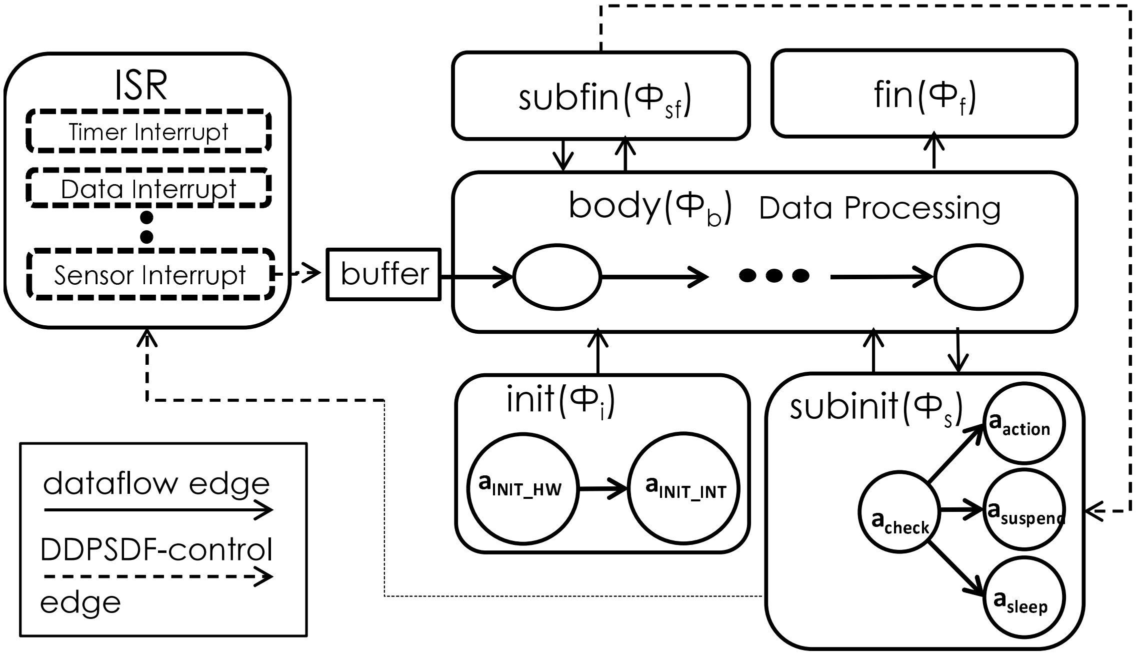


Figure 5.1: Illustration of DDPSDF modeling.

constant [5]. The DDPSDF modeling approach is illustrated in Figure 5.1.

In a sensor network system, many kinds of interrupts can be used, including timer-driven and data-driven interrupts. In DDPSDF, we model the system with a core data processing part that interacts with one or more types of hardware interrupts. Sensor data is acquired using sensor interrupts and stored into buffers that are modeled as input edges of associated body graph (Φ_b) representations.

In DDPSDF modeling, the handling of different kinds of hardware interrupts can be controlled and configured dynamically by setting appropriate ISR and data acquisition parameters. Computations required to generate new parameter values or perform related control operations can be specified as dataflow subsystems and integrated with the associated signal processing and interrupt management functionality through PSDF modeling techniques. For example, if an interrupt has low priority, it can be disabled during core "data processing phases" of system operation, and enabled again when there is no new data to process. In general, the priority associated with each interrupt can be updated at run-time through application of DDDAS techniques that are embedded as appropriate subgraphs within init graph and subinit graph specifications.

The init graph Φ_i of a DDPSDF subsystem sets up hardware interrupts for acquiring sensor data, and initializes the relevant peripherals. The subinit graph Φ_s controls the scheduling associated with data processing and interrupts based on monitored data, including run-time instrumentation information. The body graph Φ_b can then be employed to carry out the core signal processing functionality for the sensed samples. The samples are read from the input edges of Φ_b and processed using actors and nested SDF or DDPSDF subsystems that implement the relevant signal processing algorithms. Configurations of Φ_b can be updated with parameters set from Φ_s between graph iterations depending on the current status of relevant buffers and ISRs.

The DDPSDF model also incorporates a natural extension of the core PSDF model to provide symmetry with the subinit graph feature. In particular, each DDPSDF subsystem Φ is equipped with a (optionally empty) graph called the *subfin* graph (Φ_{sf}) that encapsulates computations to be performed just *after* each body graph iteration. Here, "fin" is short for "finalize" just as "init" is short for "initialize". Similarly, DDPSDF introduces the concept of a *fin* graph to provide symmetry in relation to the *init* graph. Specifically, computations associated with Φ that are to be executed at the *end* of each *parent* graph iteration can be encapsulated in the fin graph (Φ_f) associated with Φ.

Thus, a DDPSDF subsystem in general consists of five graphs — the body, fin, init, subfin, and subinit graphs, although in general some of these graphs may be unused (empty) for any given DDPSDF subsystem.

The following are specific actors that can be applied, with suitable customizations, for various useful purposes in the context of DDPSDF models. These actors are employed in the embedded speech recognition case study that we present in Section 5.4. These actors are defined here as core functional dataflow (CFDF) [8] actors so that they can be applied beyond the framework of DDPSDF. CFDF is a specific dataflow model of computation in which actor functionality is decomposed into distinct *modes* of operation. Each CFDF mode has constant production and consumption behavior (*dataflow signatures*) on the actor ports, but different modes can have different signatures, thereby allowing for dynamic, actor-level dataflow behavior. A CFDF actor A can be embedded in a DDPSDF graph as a PSDF subsystem, where each firing of A corresponds to an iteration of the subsystem. Thus, the actors defined here can be applied readily as part of the DDPSDF design methodology developed in this chapter, while also being applicable in designs using CFDF and related models.

1. Interrupt checking actor a_{check}. The a_{check} actor contains three modes — the *Check* mode (the initial mode), *EndProcessing* mode, and *Sleep* mode. When a system needs to disable selected interrupts (e.g., for QoS management), a_{check} generates a token to allow a_{suspend} (described below) to suspend interrupt gen-

eration processes, and changes its current mode to the *EndProcessing* mode.

2. Interrupt suspending actor a_{suspend}. This actor provides a parameterized interface to interrupt control registers associated with a particular embedded processor platform. When a_{suspend} executes, it sets the encapsulated interrupt control registers to suspend interrupts depending on the desired priority levels, which can be set statically or reconfigured as dynamic parameters of the actor.

3. Interrupt sleep actor a_{sleep}. This actor enables a sleep timer and transitions the processor into its sleep mode. Prior to entering the sleep mode, priority parameters can be readjusted to ensure that sleep mode status is maintained until an event of sufficiently high priority is encountered. For example, if the system is in a state of critically low battery level, then the threshold for returning from sleep mode may be relatively high. The system will resume operation upon arrival of a sensor or timer interrupt of sufficiently high priority.

4. Interrupt action actor a_{action}. This actor resumes generation of selected interrupts by appropriately configuring the associated interrupt control registers.

5.3 Energy Estimation

We outline here an energy estimation model based on run-time analysis of the available power modes in the targeted sensor node processor (microcontroller). In the case study presented in Section 5.4, we use this model to estimate the energy savings of our DDPSDF design approach.

We denote the estimated energy consumption of the targeted microcontroller across the time interval $[0, t]$ as $E(t)$. $E(t)$ is decomposed into the processing energy $E_p(t)$ (associated with computational tasks), and energy for idling in sleep mode $E_s(t)$. That is,

$$E(t) = E_p(t) + E_s(t). \tag{5.1}$$

We denote I_A and I_S as the average current consumption when the microcontroller is in its active mode and sleep mode, respectively. Also, V_{ref} denotes the reference voltage for the system. $E_p(t)$ can then be estimated as

$$E_p(t) = I_A \times V_{\text{ref}} \times t_1, \tag{5.2}$$

where t_1 is the total amount of time within time interval $[0, t]$ that the microcontroller is in its active mode.

Similarly, the "sleep energy" can be estimated as

$$E_s(t) = I_S \times V_{\text{ref}} \times t_2, \tag{5.3}$$

where t_2 is the total amount of time within time interval $[0, t]$ that the microcontroller is in its sleep mode.

In this model, we approximate as zero the time spent in transitions between the active and sleep modes. Thus, we assume that $t = t_1 + t_2$.

To compare the energy consumption values associated with different implementations, we define the *energy gain* (improvement) $G_E = G_E(I_1, I_2)$ of implementation

I_1 relative to implementation I_2 as

$$G_E(I_1, I_2) = \frac{E_2 - E_1}{E_2}, \tag{5.4}$$

where E_1 and E_2 represent the estimated energy usage of I_1 and I_2, as determined by the energy estimation model of (5.1). Note that a positive (negative) energy gain value $G_E(I_1, I_2)$ indicates that implementation I_1 is estimated as being more (less) energy efficient compared to implementation I_2.

5.4 Experiments

5.4.1 Case Study: Sensor Network System for Speech Recognition

We have experimented with the DDPSDF modeling techniques and design methods presented in this chapter using a WSN application for distributed speech recognition. The underlying speech recognition algorithm is a lightweight algorithm, designed for distributed embedded operation, developed by Phadke et al. [40]. Our implementation of this speech recognition application involves functionality for sensing, signal processing, and interrupt handling to demonstrate the data-driven and interrupt integration features of the DDPSDF model.

Our sensor network for speech recognition is integrated with a temperature sensing application to demonstrate DDPSDF methods in a system that requires heterogeneous sensing and processing tasks. Such a lightweight (limited vocabulary) speech recognition system together with temperature sensing can be used, for example, as part of a building automation system. Depending on the system energy level and processing priorities in our experimental WSN system, the temperature sensor can be turned on and off at different points in time.

In our experimental WSN system, we have used two sensors, an external acoustic sensor for speech data acquisition, and the internal temperature sensor of the targeted microcontroller. The speech data acquisition is data-driven, through use of interrupts to acquire samples when there is input of sufficient magnitude (loudness) at the microphone sensor. The temperature sensing is achieved by using timer interrupts to acquire data at periodic intervals, where the enabling of the sensor readings and the period of data acquisition can be adapted at run-time.

Figure 5.2 illustrates our experimental WSN application modeled as a DDPSDF graph. The Texas Instruments (TI) CC2530 microcontroller was used as the target processor for our implementation.

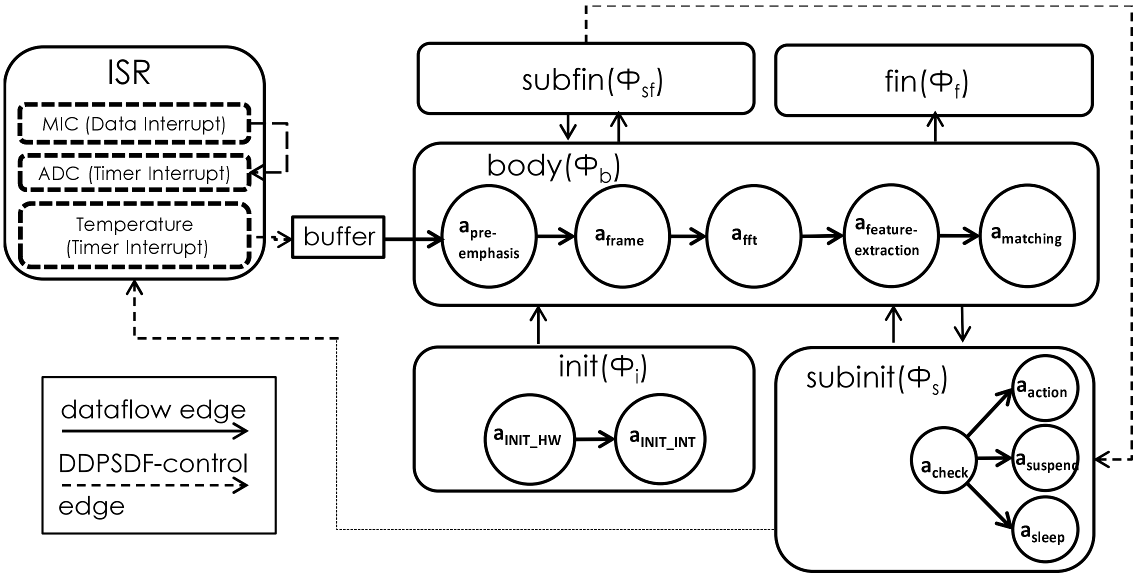


Figure 5.2: DDPSDF model for the experimental WSN application.

The speech signal is sampled at an 8Khz sampling rate, and a timer is initialized in Φ_i for triggering the sampling of speech signals whenever there is an input signal of sufficient magnitude detected at the acoustic sensor. Furthermore, an *analog to digital converter* (ADC) is initialized in Φ_i to convert the sampled speech signals into digital form. The temperature sensor is also enabled and configured with a timer in Φ_i. The subinit graph Φ_s is used to control enabling and disabling of timer and data interrupts based on dynamically determined priorities between speech and temperature sensing.

Signal processing in Φ_b for speech data begins by converting the sampled data into 12 or more overlapped frames. These frames are processed with a 64-point fast Fourier transform (FFT), and are filtered through a Mel-scaled filter bank. Filtered frames are used for extracting features from each frame. These features are matched with the given set of template words, and if a close enough match is detected, then the matched word is reported as the newly recognized word.

Speech processing is designated by default as having higher priority compared to temperature sensing in our experimental WSN application. This is because precise regularity in temperature sensing is typically not critical for an application, whereas dropping of speech recognition tasks will be immediately noticeable to the user. For energy and performance optimization, we can therefore disable interrupts for temperature sensing when speech processing tasks are being carried out. Such interrupt disabling and subsequent re-enabling can be integrated into the DDPSDF model specification, and carried out as part of the scheduling and optimization results that are derived from the model.

During system operation, the priority of speech processing can be lowered temporarily (e.g., if an excessively long time has elapsed since the most recent temperature reading). This is handled through re-assessment of interrupt priorities in the subfin graph of the DDPSDF specification. In our experiments, we do not actually utilize this possibility for dynamic change in priorities, and the subfin graph here is thus effectively a placeholder for handling dynamic switching between prioritizations. Experimentation with dynamic priority handling in our experimental WSN system and other DDPSDF application contexts is an interesting direction for further investigation.

Timer interrupts are enabled (through an ISR) only when a threshold-based interrupt (data driven interrupt) indicates to the system that there is a signal on the acoustic sensor input of sufficient magnitude to be part of a possible new spoken word (speech candidate). Once a speech candidate is detected, the ISR configures timer interrupts, which drive acquisition and processing of new speech samples. The associated (parameterized) timer value can be adjusted dynamically based on the desired sample rate, which in turn affects trade-offs between buffer memory requirements and speech processing accuracy through Φ_s.

The DDPSDF representation allows PSDF scheduling techniques to be applied for systematic derivation of efficient and correct-by-construction schedules for the targeted class of WSN applications. The schedule S for our experimental WSN system implementation begins by executing Φ_i, and iteratively executes Φ_b and Φ_s with transitions between Φ_b and Φ_s occurring as body graph iterations complete or as interrupts arrive for handling by Φ_s.

To validate the correctness and efficacy of our DDPSDF-based design, and evaluate the energy improvement provided by the disabling of redundant temperature interrupts (i.e., interrupts that will effectively be ignored because the concurrently executing speech processing tasks will have higher priority), we implemented our integrated speech- and temperature-sensing WSN system, and analyzed its performance and energy consumption. Our DDPSDF-based design is implemented by applying the lightweight dataflow design environment (LWDF), which provides a tool for experimenting with design and optimization techniques for dataflow-based signal processing systems [2]. We compare the performance and energy efficiency of the DDPSDF-based system with a conventional synchronous dataflow (SDF)-based implementation that does not apply dynamic enabling and disabling of interrupts.

As shown in Section 5.3, the energy consumption for each system was estimated, using the energy model discussed in Section 5.3, in terms of the processor power modes employed and the time spent in each mode. The times spent in the active and sleep processor modes (t_1 and t_2 in the model of Section 5.3) were measured using an oscilloscope based on execution on the targeted TI CC2530 platform. Additionally, we applied a number of energy-related device parameters (shown in Table 5.1) that are obtained from the TI CC2530 data sheet.

In this experiment, we first stored 4 pre-sampled 120ms-duration speech segments into memory. In the DDPSDF-based implementation, we configured timer interrupts to trigger temperature sensing every 200ms, and we set speech signal processing to have the highest priority, as described earlier in this section. Along with initiation of the body graph schedule for signal processing on each speech segment, timer interrupts for temperature sensing were disabled for the duration of processing

Table 5.1: Energy-related device parameters for the TI CC2530 microcontroller.

Parameter	Value
I_A	6.5mA
I_S	0.1μA
V_{ref}	3.3V

for the corresponding speech segment.

On the other hand, in the implementation that is based on a conventional SDF modeling approach, temperature sensing interrupts continue to operate during speech processing. Even though these interrupts are effectively ignored because of the higher priority given to speech, the system consumes redundant energy for the temperature sensing interrupts.

Table 5.2 shows the time required to process the pre-sampled speech segments for both the DDPSDF- and SDF-based implementations: t_D and t_S refer to the total execution times for the DDPSDF- and SDF-based implementations, respectively.

These results show that the DDPSDF-based implementation results in a 5.69% improvement in execution time. This execution time improvement can be translated into an energy consumption improvement if the same overall execution speed of the SDF-based version is maintained, and the processor is put into its sleep mode during the "slack" period ($t_S - t_D$) that is provided by the faster processing in the DDPSDF

Table 5.2: Measured execution times for the DDPSDF- and SDF-based implementations.

Measured value	Time (ms)
t_D	2840.10
t_S	3011.60

implementation. The energy consumption levels for both implementations can then be compared over identical processing windows of duration t_S.

Applying the energy estimation model of Section 5.3 to the DDPSDF-based implementation, we then have $t_1 = t_D$, $t_2 = (t_S - t_D)$, and the overall estimated energy computed as

$$E_1 = V_{\text{ref}} \cdot I_A \cdot t_1 + V_{\text{ref}} \cdot I_S \cdot t_2 = 60.021\text{mJ}. \tag{5.5}$$

On the other hand, the energy consumption (under the same level of overall processing speed) for the implementation without using DDPSDF modeling is given by:

$$E_2 = V_{\text{ref}} \cdot I_A \cdot t_S = 64.599\text{mJ}. \tag{5.6}$$

The energy improvement of the DDPSDF version, as determined by (5.4), can then be estimated as

$$G_E = \frac{E_2 - E_1}{E_2} = 5.69\%. \tag{5.7}$$

Thus, in this experiment, the implementation that uses DDPSDF modeling provides an estimated energy consumption improvement of 5.69% through disabling of timer interrupts during the processing of speech signals. Such interrupt-oriented optimization is not supported in an integrated way with conventional SDF or PSDF modeling techniques.

Furthermore, use of model-integrated interrupt- and data-driven processing techniques, as supported in DDPSDF-based design, allows designers to leverage optimized parameterized dataflow scheduling and buffer management techniques (e.g., for optimizing individual init, subinit and body graphs in the model), and systematic, model-based scheduling and integration of subsystems. These advantages of extending the power of dataflow-oriented, model-based design and implementation techniques are perhaps the most general and significant benefits of DDPSDF. Further development and demonstration of such benefits and application to other important areas of signal processing and DDDAS are useful directions for further investigation.

5.5 Conclusion

In this chapter, we have introduced the data-driven parameterized synchronous dataflow (DDPSDF) modeling technique and associated methods for design and implementation of wireless sensor network applications. DDPSDF can be viewed as a structured integration of parameterized synchronous dataflow (PSDF) modeling of signal processing applications, and the DDDAS paradigm for dynamic steering of system operation and instrumentation based on run-time data characteristics. DDPSDF provides model-based integration of optimized, interrupt-driven processing and interrupt control in energy-constrained sensor nodes. We have demonstrated the efficacy of our proposed DDPSDF techniques using a wireless sensor network application for integrated speech recognition and temperature sensing. Further development and demonstration of DDPSDF methods and application to other important areas of signal processing and DDDAS are useful directions for further investigation.

Chapter 6: Design and Implementation of FPGA-based DSP Systems using LWDF-V

6.1 Introduction

Development and demonstration of LWDF techniques has focused primarily so far on the use of LWDF for software implementations, e.g. through integration (API retargeting) of LWDF with C and CUDA [2, 41]. In this chapter, we study in depth the application of LWDF to digital system design, and present LWDF-V (*LWDF-V*), which is an integration of the lightweight dataflow programming model with the Verilog HDL. While such integration was studied in the preliminary form in [2, 41], those works, as mentioned above focused primarily on the application of LWDF to software implementation. In this chapter, we develop new extensions to the LWDF APIs for effective and flexible integration with HDLs, and demonstrate these extensions concretely using the Verilog HDL, and LWDF-V programming model.

In this chapter, we also present new methods in LWDF-V for supporting dynamic parameter manipulation (*DPM*) in dataflow-based applications and actors. By DPM, we mean the capability to flexibly reason about and change functional parameters of dataflow graphs and actors at run-time. Such capabilities for DPM are important for handling features such as adaptivity and multi-mode operation in signal processing systems, and for integrating LWDF-V-based actor implementations with higher-level methodologies for parameter management in dataflow graphs, such as change context analysis [42], parameterized dataflow [5], PiMM [43], and scenario-aware dataflow [44].

We demonstrate our proposed LWDF-V programming model and methods for managing dynamic parameters through design and implementation of actors for inner product computation.

6.2 Related Work

6.2.1 Model Based Design Methodologies for Digital Hardware

Research on model based design methodologies related to signal processing hardware design includes two major directions — process network models, and dataflow models. Process networks and dataflow graphs are related formalisms for model-based signal processing system design. relationship between these formalisms are discussed in [45].

Using *Kahn process networks* (*KPNs*), signal processing functionality can be described in terms of continuously executing computational processes and unbounded, first in first out (*FIFO*) connections between these processes. KPNs are deterministic, which means that the input to a KPN uniquely determines the outputs. Stefanov et al. [46] presented the Compaan and Laura tools for HDL code synthesis from KPN models. Compaan is used to translate MATLAB programs into KPN models from which Laura generates optimized HDL code.

Kienhuis and Deprettere [47] presented the *Stream-Based Functions* (*SBF*) model of computation, which is a specialized form of KPN. SBF models are composed of SBF objects and channels, where each object in turn is composed of a set of functions, a controller, and state. Methods for mapping SBF representations into FPGA platforms are developed in [48].

Various methods for implementing DSP dataflow graphs in hardware have also been developed. Eker and Janneck [49] introduced *CAL*, which is a language for dataflow programming that has been applied to a variety of DSP application domains, with some of the most interesting results presented in the area of reconfigurable video coding. The behavior of a CAL actor is defined in terms of a set of actions. Actions of a CAL actor define transitions that internal states of the actor can undergo. CAL is supported by the Eclipse integrated development environment (IDE) [50], and open source plugins called OpenDF and OpenForge [51], which allow CAL models to generate HDL code.

McAllister et al. [52] developed methods for optimized hardware synthesis of pipelined dataflow graph structures. This work presents a new modular signal flow graph synthesis method that systematically converts dataflow graphs into networks of pipelined components, which can then be mapped efficiently into FPGA implementations. This approach enables high level, hardware-oriented dataflow graph optimizations along with efficient design space exploration.

In contrast to previous work on model-based design methodologies for DSP hardware, our work on LWDF and HDL-targeted LWDF, such as LWDF-V, emphasizes the use of compact, retargetable APIs that are agnostic to specific platforms or languages. This facilitates cross-platform design, design for heterogeneous platforms, and systematic migration of actor implementations across different languages (e.g., porting from MATLAB-based simulation code to an HDL-based implementation version). With this different focus of our proposed LWDF design methods, there is significant potential to extend the design methods for application with some of the key directions of related work described above. For example, the optimizations developed in [52] can potentially be integrated to develop schedulers that control the execution of LWDF-V actors or actors in some other HDL variant of LWDF. Exploration of such extensions is a useful direction for further work that emerges naturally from the developments of this chapter.

Preliminary work on integration of Verilog and LWDF for actor design has been presented in [2, 41]. This chapter goes significantly beyond the preliminary work in a number of ways. First, the hardware execution status of LWDF-V actors is decomposed into explicit *idle* and *active* states, which are defined at actor interfaces, and allow tighter coordination between actors and any enclosing subsystem- or graph-level schedulers. Second, capabilities for DPM are integrated throughout the interface and design process for LWDF-V actors, which provides a flexible foundation for applying higher-level dataflow graph modeling, analysis, and synthesis techniques (e.g., see [5, 42–44]) involving dynamic management of actor- and graph-level parameters. Third, a comprehensive development of required module interfaces — for dataflow actors and edges — is provided for LWDF-V; this development is not dependent on any specialized features of Verilog, and is thus readily abstracted into module APIs that can be retargeted into other HDLs.

6.3 LWDF-V

In this section, we discuss the modeling and implementation of applications using LWDF-V. We emphasize in this section the different kinds of modules involved in the LWDF-V design methodology, and in the interface specifications for these modules.

Building on the LWDF operational context concept, as described in Section 2.1.4, we develop in this chapter an LWDF-based digital hardware design methodology for signal processing systems, called LWDF-V. The “V” in “LWDF-V” stands for Verilog, which is the specific HDL that we have applied for integrating hardware design capabilities with LWDF. However, the conceptual extensions developed in this chapter to provide hardware support in LWDF-V are relevant to HDLs in general, and we envision that they can be readily adapted to other HDLs, such as VHDL or System-C. Pursuing such adaptations for other relevant HDLs is a useful direction for future work.

In LWDF-V, the abstract enable and invoke functions of LWDF are implemented as two coupled Verilog *modules*, which are called the *actor enable module* (*AEM*) and *actor invoke module* (*AIM*). A key principle of LWDF is to impose minimal constraints on the design of actors. Accordingly, the AEM and AIM can contain arbitrary sub-modules (e.g., for modular design of complex actors), and can be described using any Verilog coding style, including behavioral, structural or mixed behavioral/structural coding [53], as long as the prescribed LWDF-V module interfaces (described below) are adhered to. Such flexibility facilitates evolutionary design of complex digital systems, where the functionality and level of abstraction associated with different subsystems can be adjusted as appropriate as the design progresses, and as more details are understood about the targeted implementation.

6.3.1 Actor Invoke Module

Figure 6.1 illustrates the high level operation of an LWDF-V AIM module. The required interface ports of an AIM module for an actor A can be divided into four groups:

- Dataflow-related inputs. Corresponding to each actor input port `xyz`, there is an AIM input port `in_xyz`. There is also Boolean input port called `invoke`. When this input signal is `true` and A in the idle state (see Figure 6.1), the next firing of A should be initiated in the next clock cycle/

- Dataflow-related outputs. Corresponding to each actor output port `abc`, there is an AIM output port `out_abc` (for writing output tokens), and another AIM output port `wr_abc` (for submitting write requests to the output edge connected to `abc`). Also, corresponding to each actor input port `xyz`, there is an AIM output port `rd_xyz` (for submitting read requests to the input edge connected to `xyz`).

- Platform-related inputs. A clock input and synchronous reset input are provided for relevant synchronization with interfacing circuitry. The synchronous

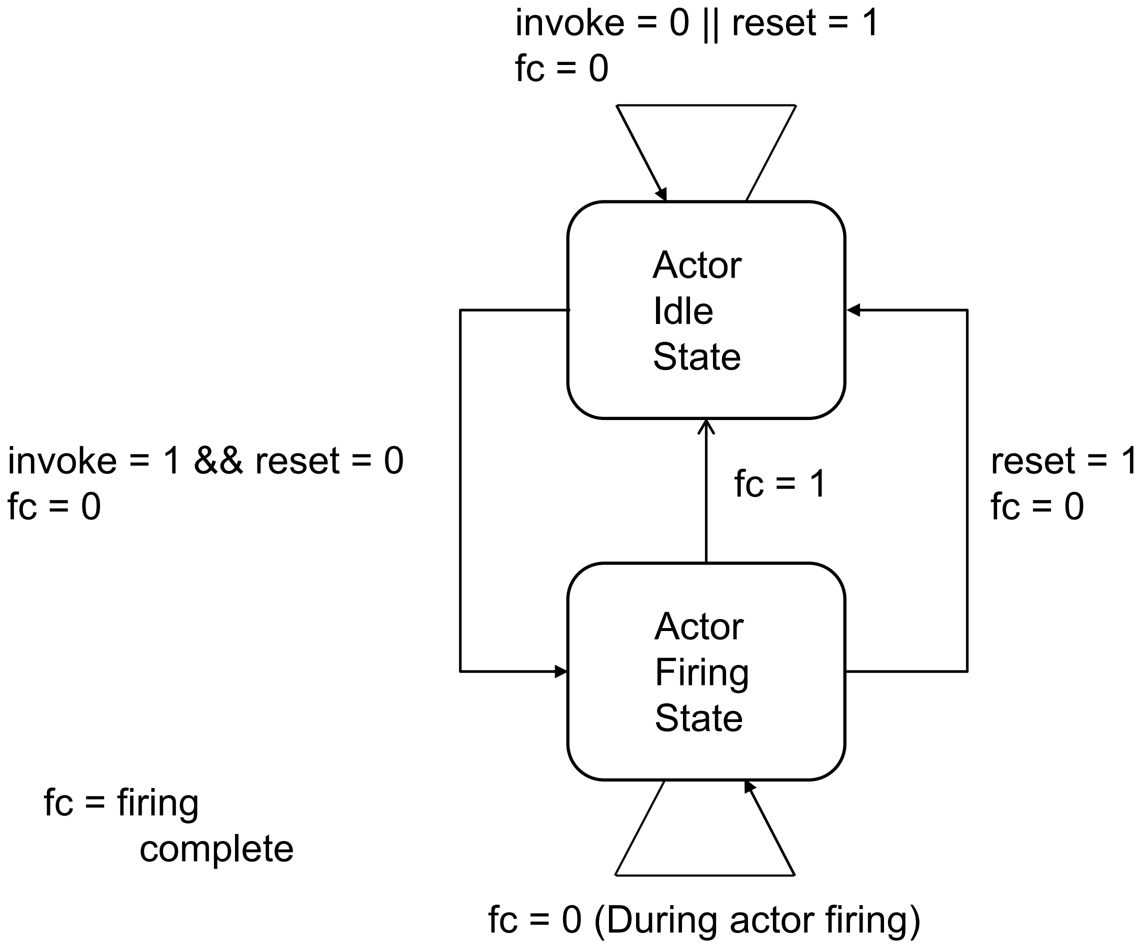


Figure 6.1: LWDF-V actor invoke module (AIM) design

reset, when asserted, brings the actor to its idle state on the next clock cycle even if it is in the middle of a firing.

- Control-related inputs and outputs. The `next_mode_out` signal (output) provides the next mode in which to invoke the actor as determined by the current firing. The `next_mode_in` signal (input) provides the next mode in which to invoke the actor as prescribed by the enclosing scheduler (external actor control). The `fc` signal (output) that is asserted (to the logic high value) when a firing completes during the current clock cycle.

Recall that LWDF is based on CFDF semantics so that each actor firing in LWDF corresponds to a specific CFDF actor mode, and results in a next mode value as a side effect of the firing. Typically, the `next_mode_in` signal will be set based on the `next_mode_out` from the most recent firing; however, by appropriate setting of the `next_mode_in` signal, the enclosing scheduler has the flexibility to "override" the `next_mode_out` value when invoking an actor — e.g., if some sort of actor-, subsystem- or application-level reset operation is being carried out. It is important also to note that any given firing may take one or more clock cycles; thus, actor modes do not necessarily correspond to individual clock cycles.

6.3.2 Actor Enable Module

The AEM provides the functionality associated with the `enable` function in CFDF semantics [8]. Analogous to the AIM operation diagram in Figure 6.1, Figure 6.2 illustrates the high level operation of the LWDF-V AEM module. The required interface ports for an AEM module are as follows.

- `enable` (output): indicates whether or not the associated actor has enough data on its input ports and enough free space on its output ports to fire in the given mode (i.e., as specified by the `mode` input, which is specified below).
- `pop_in_xyz` (input): one AEM input port `pop_in_xyz` is provided corresponding to each actor (input or output) port `xyz`. This AEM port is used to read the current buffer population (number of tokens) on the dataflow edge connected to actor port `xyz`.

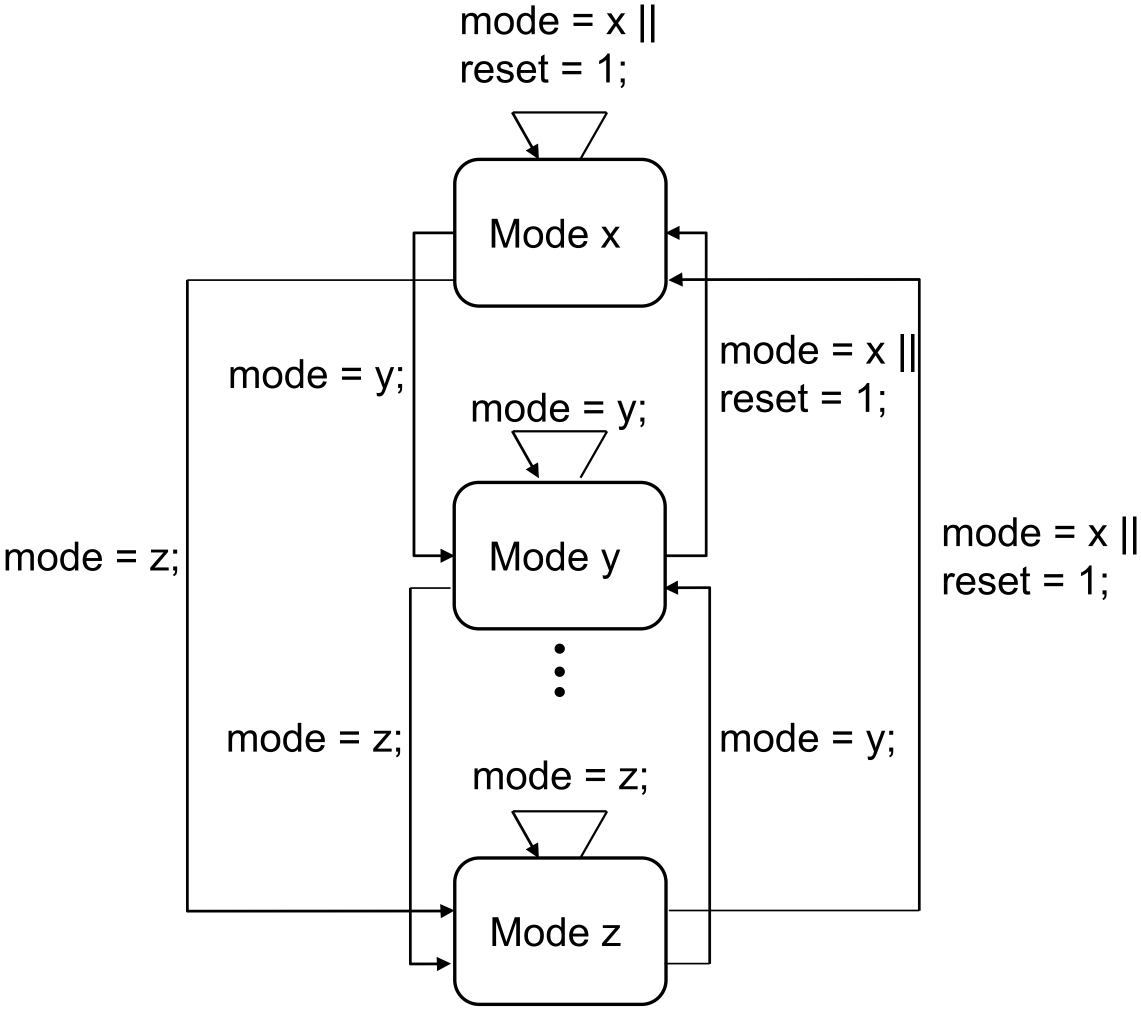


Figure 6.2: LWDF-V actor enable module (AEM).

- `mode`: an input port for specifying the CFDF actor mode in which the current (next) enable evaluation is to be computed for a combinational implementation (sequential implementation) of the AEM.

- `reset`: synchronous reset signal (active high).

As implied in the discussion of the `mode` port, interpretation of the interface ports for an AEM is different depending on whether the AEM is to be implemented as combinational or sequential logic. Sequential logic may be preferred for AEM implementation in cases where resources are constrained relative to the complexity of the CFDF enable functionality. For example, an actor with a large number of inputs or outputs requires a correspondingly large number of comparators under a straightforward combinational design for the enable logic associated with a given actor mode. By instantiating a limited number of comparators (less than the number of input ports) and time-multiplexing their use, a sequential implementation can be used to trade-off area cost reduction and latency increase through the AEM.

A sequential AEM implementation must provide the following additional module interface ports.

- `clock` (input): clock signal.
- `compute`: a single-bit, active high input signal used to initiate computation of the `enable` output based on the current value of the `mode` input.
- `valid`: a single-bit, active high output signal for specifying that the `enable` output is valid with respect to the most recent assertion of the `compute` input.

6.3.3 Dataflow Edge Module

A *dataflow edge module* (*DEM*) provides a Verilog realization for a CFDF graph edge in LWDF-V, and its instances provide communication channels for connections between actor output ports and corresponding actor input ports. As with the AIM and AEM interface specifications, the DEM interface specification is designed to impose minimal ("lightweight") constraints on module design so as to ensure the desired dataflow properties while providing flexibility to implement or simulate the modules according the designers specific interests and needs. For example, in the context of an FPGA design, DEM could be realized as using block RAM or distributed RAM, or as a purely behavioral (non-synthesizable) module for simulation purposes.

The following list summarizes the *input ports* in the DEM interface specification, as prescribed by LWDF-V.

- `wr_enable`: write enable (single bit) — an input enabling signal for storing data into the FIFO that is encapsulated by the DEM.
- `rd_enable`: read enable (single bit) — an output enabling signal for reading data from the FIFO.
- `data_in`: input data (directed from the actor at the source of the corresponding edge) to be written into the FIFO.

Similarly, the following list summarizes the *output ports* in the DEM interface specification.

- `data_out`: output data (directed to the actor at the sink of the corresponding edge) to be read from the FIFO.
- `population`: the number of tokens that are currently stored in the FIFO.
- `capacity`: FIFO size — the maximum number of tokens that can coexist on the edge at any given time.

Typically, in hardware implementations, the FIFO capacity associated with a DEM will be determined statically. However, this is not a requirement of LWDF-V. In particular, dynamically determined buffer capacities, such as capacities that are adjusted between “quiescent points” of a schedule [42], can be implemented as long as changes to the capacities are consistently reflected on the `capacity` output ports. Such flexibility is important to support adaptive scheduling and more powerful forms of dynamic reconfiguration, which are important directions for research and development in advanced signal processing systems.

6.3.4 Summary

In this section, we have presented the LWDF-V programming model for application of the lightweight dataflow design methodology to digital system design using the Verilog HDL. The emphasis in this section has been on describing the standard modules in LWDF-V — the AIM, AEM, and DEM modules — and discussing key aspects of the interface specifications for these modules.

We would like to emphasize that the interface specifications of the AIM, AEM, and DEM modules apply standard principles from the design of HDL design components and are not specific to the Verilog language; these specifications can therefore be retargeted to other HDLs. Thus, the developments in this section can be viewed as providing the basis for a general LWDF-HDL interface specification, with Verilog employed as a specific HDL with which to demonstrate the specification concretely.

6.4 Design Example

To illustrate the LWDF-V design approach presented in Section 6.3, we present in this section an example of an inner product actor that is developed using LWDF-V.

The inner product actor (IPA) targeted in this example, in terms of dataflow semantics, has two input ports, one output port, and a static parameter N, which gives the vector length of the inner product that is computed by the actor. This is illustrated in Figure 6.3(a). The IPA operates by reading (consuming) vectors of length N from its two input ports, where each input token is assumed to be a single vector element; computing the inner product of the two N-element vectors; and then producing the resulting inner product value on the actor output port. This process of reading, inner product computation, and production is repeated over and over on the input token streams as long is the actor continues to fire.

The operation of this actor is designed in terms of CFDF semantics, as illustrated in Figure 6.3(b), which shows the CFDF *mode transition graph* for our IPA actor implementation. Each vertex in this graph corresponds to a specific CFDF mode of the actor, and the edges specify transitions between modes. For a fixed value of the vector length N, this actor can also be implemented using CSDF semantics [54]; however, CFDF provides a natural extension to the case where N can vary dynamically — e.g., through a dynamically manipulated parameter (as discussed in Section 6.5) or through a separate input port in which a dynamic stream of vector lengths is read (consumed). For conciseness and clarity in the section, we focus only on the static parameter case. Extensions to handle dynamic parameter manipulation are discussed in Section 6.5, as mentioned above.

The mode transition graph of Figure 6.3(b) shows that the IPA computes one result (produces one new output token) on every third firing. The dataflow rates associated with the modes in the IPA are summarized in Figure 6.3(c).

Figure 6.4 illustrates a hierarchical design pattern that can be used to implement the AIM module for the IPA and also as a general approach to implement AIMs for a variety of different kinds of actors in LWDF-V. The top level (Figure 6.4(a)) in this design pattern specifies the coordination of overall actor operational status, as modeled in LWDF-V, and how it is controlled by the relevant signals in the AIM input interface.

The intermediate level (Figure 6.4(b)) illustrates the switching between CFDF modes, based on the `next_mode_in` signal at the AIM input interface. Such switching can be performed behaviorally or structurally, as desired by the designer, and intra-mode logic can be embedded inline, within the HDL code for the module illustrated in Figure 6.4(b). However, for more complex modes, it may be useful (e.g., for clarity and modularity) to instantiate mode-specific modules within some or all of the modes referenced in Figure 6.4(b). Such hierarchical embedding of intra-mode functionality is illustrated in Figure 6.4(c). This hierarchical approach, for example, allows designers to experiment efficiently with alternative implementations of individual modes — only the module instance for a given mode needs to be replaced to switch between alternative implementations, which the rest of the actor implementation remains intact.

Also, it should be noted that the top level of the AIM module, illustrated in Figure 6.4(a), is of general applicability to LWDF-V actors and can be reused to provide the top-level design across all LWDF-V actors in a design.

Figure 6.4(d) shows a Verilog code segment for the inner product mode of the IPA illustrated in Figure 6.3(b), where the intra-mode Verilog specification is instantiated through a Level 3 FSM, as illustrated in Figure 6.4(c).

6.5 Dynamic Parameter Manipulation in LWDF-V

As signal processing systems incorporate increasing levels of dynamic functionality, such as dynamics due to adaptivity or multi-mode operation, support for DPM is of great importance in processes and tools for design and implementation [3]. For example, 3GPP LTE wireless communication systems incorporate dynamic resource allocation capabilities that require run-time adjustments to relevant transceiver parameters [55, 56]. Similarly, in energy-constrained, real-time processing environments, filtering operations for image, speech, and video processing can benefit from run-time adjustments to filter size and other functional configurations to best mach the current state of the hardware.

Motivated by this need, we develop in this section methods in LWDF-V for integrating DPM support. This support is intended to help implement features such as adaptivity and multi-mode operation in dataflow-based system designs, and to help integrate LWDF-V-based actor implementations with higher-level methodologies for parameter management in dataflow graphs, such as change context analysis [42], parameterized dataflow [5], PiMM [43], and scenario-aware dataflow [44].

6.5.1 DPM-enabled LWDF-V Actors

A DPM-enabled actor in LWDF-V has three AIM inputs, called *DPM inputs*, in addition to the standard set of inputs defined for AIMs in Section 6.3. Similarly, one output, called `uc` (*update complete*) is added to the AIM output interface in support of DPM. Additionally, the top-level state diagram of the basic (statically parameterized) AIM (Figure 6.1) is extended with a third state, called the Actor DPM State, as illustrated in Figure 6.5.

Parameters for a DPM-enabled actor A are represented (indexed) by a set $paramset(A)$ of integers, which are referred to as *parameter IDs.* Thus, each actor parameter, such as a filter length value or a vector of filter coefficients, has a unique ID within $paramset(A)$ that can be used to reference the parameter at run-time. External, subsystem- or graph-level control, such as that incorporated within a quasi-static scheduler, can then be used to update actor parameters in a flexible manner through use of these parameter IDs.

The additional AIM inputs that are used to support DPM are labeled `param_update`, `param_ID`, and `param_value`. When the `param_update` input is high, and the actor is in its idle state, the actor DPM state is entered, as illustrated in Figure 6.5. In this state, the parameter referenced by parameter index `param_ID` is updated with the value received from the `param_val` input. Once the parameter update is complete, the actor asserts the `uc` output, and returns to the actor idle state, as illustrated in Figure 6.5. Note that a parameter update may involve arbitrarily complex changes to the state of an actor, and thus, there may be an arbitrarily long delay (number of clock cycles) between the initiation of a parameter update and the corresponding assertion of `uc`. For example, suppose that a sequence of n vector elements, such as filter coefficients, must be updated cycle-by-cycle as part of a single parameter update. Such an update would require at least n clock cycles. In this case, the parameter could, for example, represent an entire set of filter coefficients, or could correspond to a specific mode of operation, where each mode corresponds to a specific set of internal filters.

6.5.2 DPM-enabled Inner Product Actor

In this section, we extend the IPA example of Section 6.4 with a dynamically variable `size` parameter that replaces the static vector length parameter N defined in Section 6.4. In this case, the parameter update is a simple update of a scalar value, and this value in turn affects the consumption rate of the actor — in particular, the number of tokens consumed from each input during the reading mode in Figure 6.3.

We implemented this DPM-enabled IPA using the three-level hierarchical design structure. Figure 6.6 illustrates this design along with excerpts of relevant code blocks from the design. The code block on the left corresponds to the reading mode of the IPA, which now consumes a dynamically-determined number of tokens, and the code block on the right corresponds to the inner product mode, which carries out a dynamically-determined number of multiply-accumulate (MAC) operations to produce its single output token.

In addition to illustrating DPM support in LWDF-V, the example in this section provides a more detailed look at actor implementation using LWDF-V. Synthesis results based on this design are presented in Section 6.6.

6.6 Experiments

In this section, we demonstrate a synthesized implementation of the DPM-enabled, LWDF-V-based actor presented in Section 6.3, and we assess the performance and area efficiency of this design compared to a corresponding LWDF-V design that does not employ DPM. This assessment provides a quantitative example of the overhead (in terms of area and performance) for incorporating DPM features into an LWDF-V design for a practical actor example.

6.6.1 Overhead Evaluation Metrics

Our experiments in this section assume an FPGA (xc3s500efg320-4) as the target platform. We use the number of occupied FPGA slices in the targeted FPGA device as a measure of resource utilization. We employ the resource utilization efficiency metric O_R defined in [57]. The metric O_R measures the resource overhead of a given design D_i compared to that of another design D_i. The metric is defined by:

$$O_R(D_i, D_j) = (1 - \frac{R(D_j)}{R(D_i)}) \times 100(\%). \tag{6.1}$$

Here, for a given design X, $R(X)$ represents the number of FPGA slices used by the synthesized result for design X. Given two designs D_i and D_j such that $R(D_i) > R(D_j)$, we use the metric $O_R(D_i, D_j)$ to quantify the degree to which D_i utilizes more resources compared to D_j.

We also define $t_s(X)$ as the schedule completion time for a given design X. More specifically, suppose that X represents a hardware design for an SDF graph G. Then $t_s(X)$ gives the total execution time for a single iteration of a minimal periodic schedule for G when it is executed using design X. A *minimal periodic schedule* is a well defined unit of execution for an SDF graph where the amount of computation involved is defined by the repetitions vector for the graph. For background on the repetitions vectors and minimal periodic schedules for SDF graphs, we refer the reader to [3, 7]. In this context, a minimum periodic schedule is often referred to as an *iteration* of the associated SDF graph.

The schedule completion time $t_s(X)$ is defined by:

$$t_s(X) = \frac{N_c}{f_{max}(X)}, \tag{6.2}$$

where N_c is the number of cycles required to complete a graph iteration, and $f_{max}(X)$ is the maximum frequency of the synthesized design X. If $t_s(D_j) \leq t_s(D_i)$, then the *slowdown* (inverse of speedup) of design D_i compared to D_j is denoted by $S_d(D_i, D_j)$, and defined by:

$$S_d(D_i, D_j) = (1 - \frac{t_s(D_j)}{t_s(D_i)}) \times 100(\%). \tag{6.3}$$

Thus, the slowdown (under the stated assumption that $t_s(D_j) \leq t_s(D_i)$) is within the range $0 \leq S_d(D_i, D_j) < 1$ (i.e., $0\% \leq S_d(D_i, D_j) < 100\%$), and a slowdown value of zero indicates that the two designs have the same speed.

6.6.2 Performance of DPM-enabled LWDF-V Design

Providing a DPM-enabled actor implementation requires some amount of additional hardware resources, which can be viewed as an overhead associated with the flexibility of DPM. We have evaluated this overhead for the DPM-enabled LWDF-V inner product example (presented in Section 6.4) compared to a statically parameterized LWDF-V inner product actor. Both designs (DPM-enabled and statically parameterized) are synthesized under Xilinx ISE version 13.4 with the option of using the Digilent Spartan 3e platform, which is based in turn on a Xilinx FPGA device. Results obtained from our synthesis experiments are shown in Figure 6.7. For fairness in our comparison, we maintained consistency where possible between the configurations used for the DPM-enabled LWDF-V inner product actor and the corresponding configurations used for the statically parameterized version.

The x-axis of Figure 6.7 represents the inner product dimension (vector length). The left and right sides of the y-axis represent the O_R and S_d metrics defined in Section 6.4. These metrics are applied to measure the overhead, in terms of resource utilization and performance, of the DPM-based, LWDF-V inner product actor compared to the statically parameterized LWDF-V inner product actor. As shown in the results, the resource utilization of the DPM version is higher than the statically parameterized version. However, as the dimension of the inner product increases, the O_R (relative overhead) value decreases rapidly.

Figure 6.8 breaks down resource utilization in the DPM-enabled inner product actor design. The figure shows the resource utilization due to the core actor functionality (i.e., all portions except for the logic that implements DPM), and the utilization due to DPM. The horizontal axis in the figure again shows variation across different inner product dimensions. Again, we see a trend of decreasing overhead, in relative terms, of DPM as the inner product dimension increases. The results also show relatively low overhead for DPM considering the significant flexibility provided by DPM.

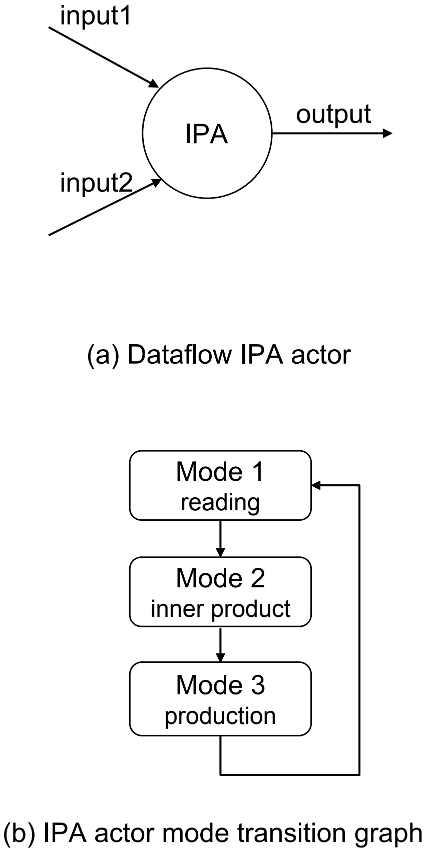
6.6.3 Summary

In summary, the experimental results in this section quantify the overhead of DPM for a practical LWDF-V actor of moderate complexity (i.e., a "medium granularity" actor). The overhead is assessed for both hardware resource utilization and run-time performance, with comparison being made between DPM-enabled and statically-parameterized versions of the same actor. The results demonstrate relatively low overhead for this example. In general, the relative overhead can be expected to vary based on specific characteristics and overall complexity of the actor being implemented, and the results in this section can be viewed as giving a concrete assessment of the overhead for a specific LWDF-V actor design.

6.7 Conclusion

This chapter has focused on the application of the lightweight dataflow (LWDF) design methodology to hardware implementation for signal processing systems. We have presented in depth *LWDF-V*, which is an integration of LWDF design principles with the Verilog hardware description language (HDL). We have developed new extensions to the application programming interfaces of LWDF that enable effective and flexible integration with HDLs, and we have demonstrated these extensions concretely using the Verilog HDL. We have also presented new methods in LWDF-V for supporting dynamic parameter manipulation (*DPM*) in dataflow-based actors. By DPM, we mean the capability to flexibly reason about and change functional parameters of dataflow actors at run-time. We have demonstrated that DPM can be supported systematically within the framework of LWDF-V with relatively low overhead in terms of hardware resource utilization and performance.

Chapter 7: Conclusions



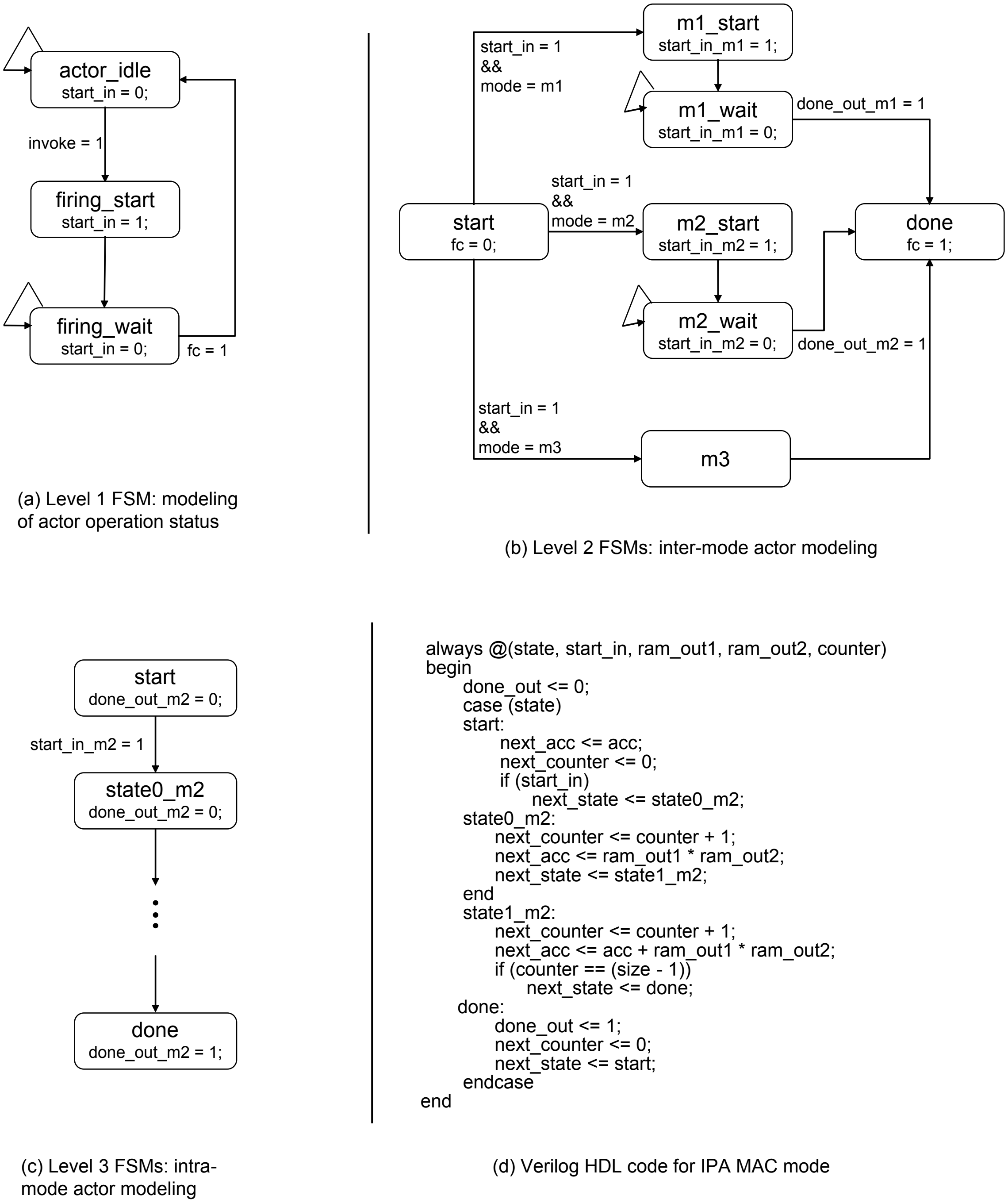
(a) Dataflow IPA actor

(b) IPA actor mode transition graph

	consumption rate		production rate
	input1	input2	output
reading mode	N	N	0
inner product mode	0	0	0
production mode	0	0	1

(c) IPA actor dataflow rate of modes

Figure 6.3: inner product actor (IPA)



(a) Level 1 FSM: modeling of actor operation status

(b) Level 2 FSMs: inter-mode actor modeling

(c) Level 3 FSMs: intra-mode actor modeling

```
 always @(state, start_in, ram_out1, ram_out2, counter)
 begin
       done_out <= 0;
       case (state)
       start:
             next_acc <= acc;
             next_counter <= 0;
             if (start_in)
                  next_state <= state0_m2;
       state0_m2:
            next_counter <= counter + 1;
            next_acc <= ram_out1 * ram_out2;
            next_state <= state1_m2;
       end
       state1_m2:
            next_counter <= counter + 1;
            next_acc <= acc + ram_out1 * ram_out2;
            if (counter == (size - 1))
                 next_state <= done;
      done:
            done_out <= 1;
            next_counter <= 0;
            next_state <= start;
       endcase
end
```

(d) Verilog HDL code for IPA MAC mode

Figure 6.4: Hierarchical specification of an AIM for IPA.

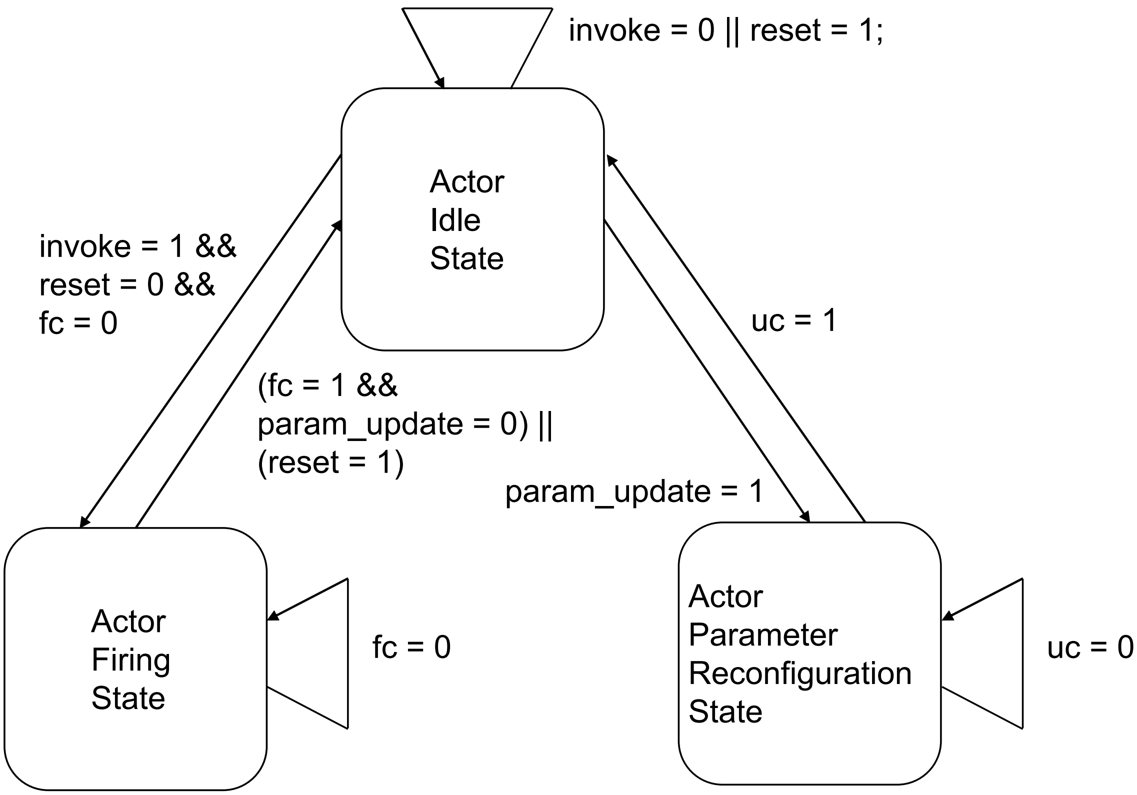
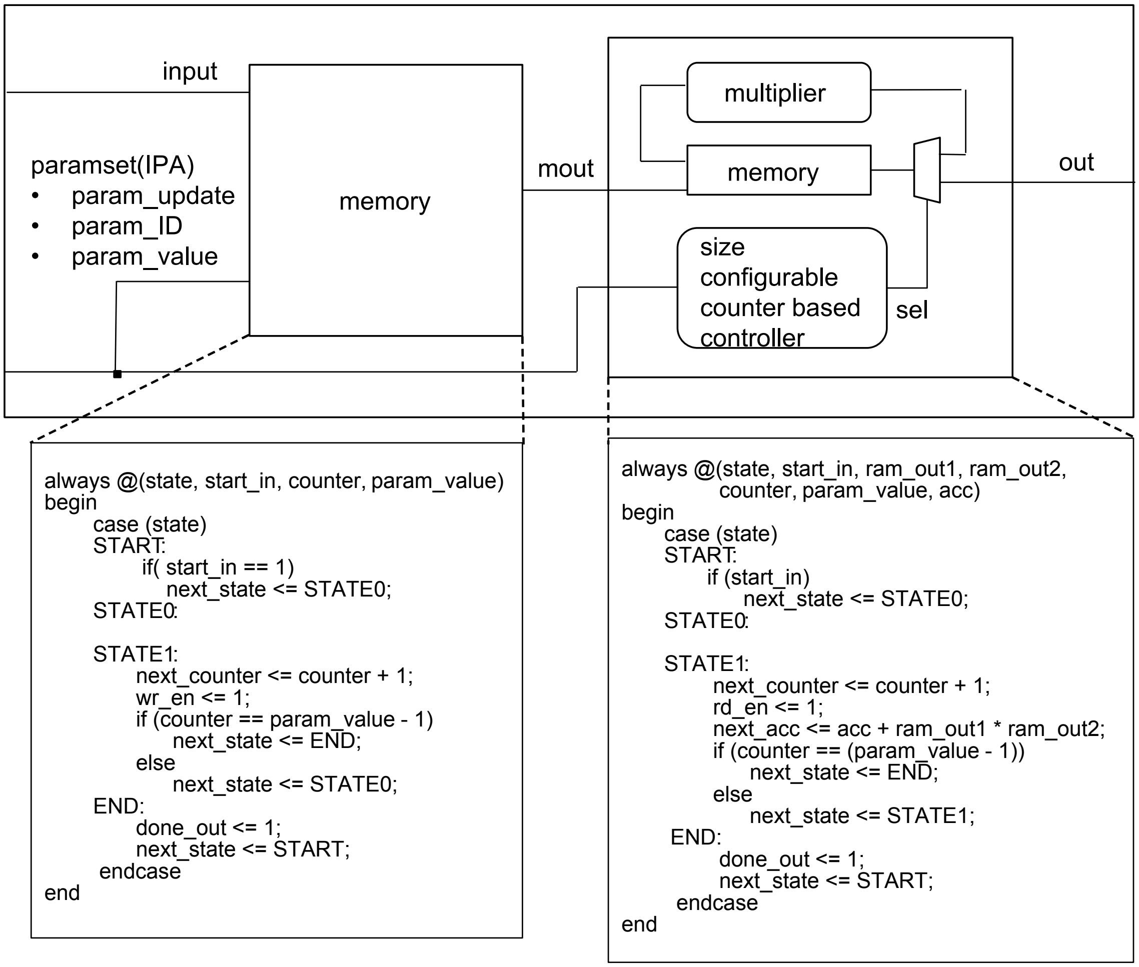


Figure 6.5: AIM design for a DPM-enabled LWDF-V actor.



```
always @(state, start_in, counter, param_value)
begin
        case (state)
        START:
                if( start_in == 1)
                    next_state <= STATE0;
        STATE0:

        STATE1:
               next_counter <= counter + 1;
               wr_en <= 1;
               if (counter == param_value - 1)
                     next_state <= END;
               else
                     next_state <= STATE0;
        END:
               done_out <= 1;
               next_state <= START;
         endcase
end
```

```
always @(state, start_in, ram_out1, ram_out2,
                counter, param_value, acc)
begin
       case (state)
       START:
              if (start_in)
                    next_state <= STATE0;
       STATE0:

       STATE1:
               next_counter <= counter + 1;
               rd_en <= 1;
               next_acc <= acc + ram_out1 * ram_out2;
               if (counter == (param_value - 1))
                     next_state <= END;
               else
                     next_state <= STATE1;
        END:
                done_out <= 1;
                next_state <= START;
         endcase
end
```

Figure 6.6: Illustration of DPM-enabled IPA using three-level hierarchical design structure.

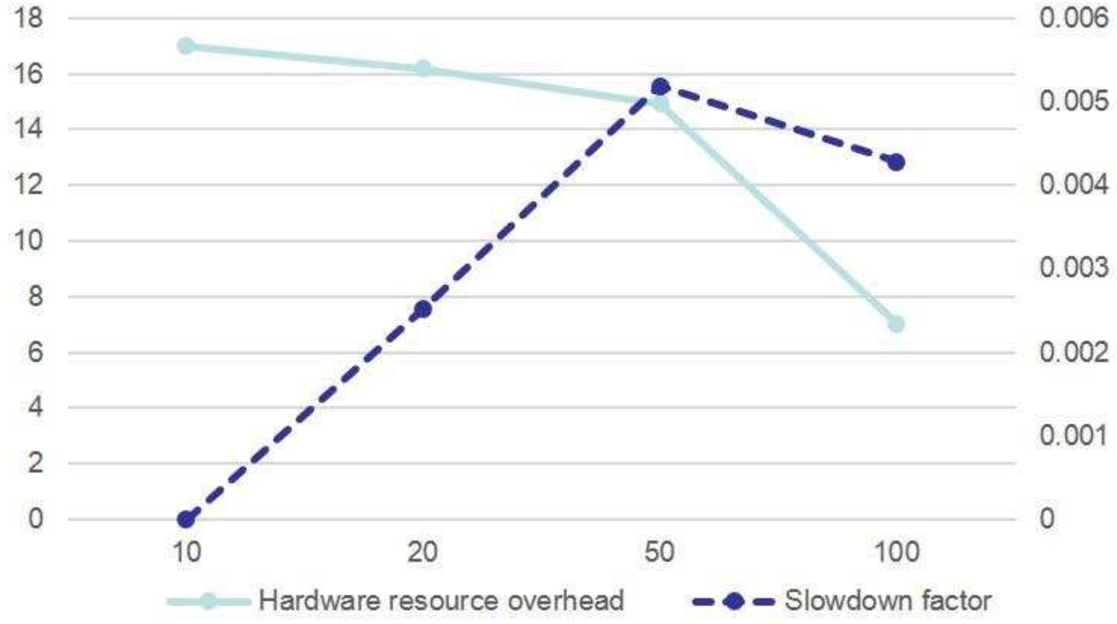


Figure 6.7: Resource and speed overhead of DPM-enabled LWDF-V inner product actor compared to statically parameterized LWDF-V inner product actor.

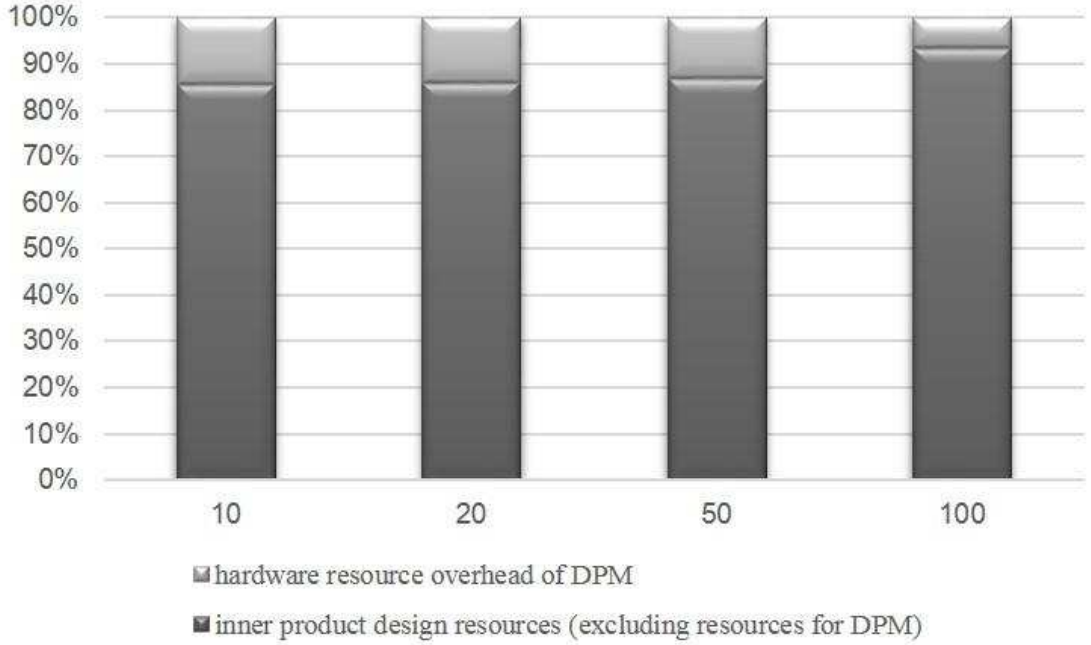


Figure 6.8: Overhead in resource of runtime parameter reconfigurability in inner product example

Optimization of energy and resource utilization efficiency is critical in design and implementation of signal processing systems in many application areas. In this thesis we have developed methods for helping DSP system designers meet stringent constraints on energy consumption and resource utilization. We have focused on the areas of wireless communication and wireless sensor networks, which often involve critical constraints on energy and resource utilization.

In Chapter 3, we presented a new energy management scheme that is targeted to maximizing end node lifetime in building energy monitoring systems. We first presented an energy analysis method for estimating the energy consumption and lifetime at network end nodes. Based on this analysis method, we formulated trade-offs between energy consumption and the reporting interval for building energy monitoring, and we presented an energy-optimized dynamic reporting control scheme. We validated our proposed energy optimization scheme through experiments on a fully functional building energy monitoring system that is equipped with Texas Instruments CC2530 Zigbee network-enabled microcontrollers.

In Chapter 4, we presented a resource-efficient fast Fourier transform (FFT) architecture for hardware implementation of orthogonal frequency division multiplexing (OFDM) communication systems. Our architecture is based on a serial input and output architecture. We have identified inefficiencies in memory and resource utilization exhibited by conventional FFT implementations for OFDM. Our proposed FFT architecture is designed to help overcome these limitations and achieve significantly improved area efficiency in OFDM system design. To assess the resource utilization of our FFT architecture, we applied relevant evaluation metrics, and compared the resource usage of our proposed architecture with a Xilinx FFT intellectual property (IP) core that is widely used as a commercial IP core for FFT implementation.

In Chapter 5, we proposed a novel dataflow modeling approach, called *data-driven parameterized synchronous dataflow* (*DDPSDF*), to support efficient model-based development of signal processing systems in a manner that provides integrated modeling and management of hardware interrupts. Our proposed design methodology based on DDPSDF graphs provides model-based integration of interrupt-driven processing and interrupt control in energy-constrained sensor nodes, and provides energy optimized control for digital signal processing. We validated and experimented with our proposed DDPSDF methods using a case study involving a WSN application that includes speech recognition and temperature sensing.

In Chapter 6, we presented LWDF-Verilog (*LWDF-V*), which is an integration of the lightweight dataflow programming model with the Verilog hardware description language. Use of LWDF-V facilitates the use of formal dataflow techniques in the design and implementation of hardware for signal processing systems. These techniques in turn are effective for exposing high level application structure, which can be exploited to optimize implementations in terms of key metrics, including energy and resource utilization. We demonstrated our proposed LWDF-V programming model and methods for managing dynamic parameters through design and synthesis of actors for inner product computation.

Various useful directions for future work emerge from the developments of this thesis. For example, the energy efficient monitoring scheme that we presented in

Chapter 3 is based on transmission interval control with moving averages. Depending on the specific sensors employed and the building environment, the optimized control scheme employed in this approach can be varied. We can employ sensor-specific parameters for controlling the transmission interval, and also provide independent controls for each of the sensors or for subsets of the sensors in the system. Such finer granularity control over the employed sensors can provide more precise control for the overall system, which can lead to higher energy efficiency.

Another useful direction for future work, related to Chapter 6 in this thesis, is the development of hardware synthesis techniques that operate on high-level dataflow graphs, and apply LWDF-V actors and DPM capabilities. Such methods could be developed to operate on libraries of LWDF-V actors together with characterizations of the actors (e.g., in terms of resource utilization and energy consumption or performance characteristics). The overall dataflow graph structure of an application can be derived easily from the connectivity between actors and edges in the LWDF-V representation, and can be used to provide the graphical input to this type of high-level, hardware synthesis approach.

Buffer minimization is an important problem in the mapping of dataflow graphs onto resource constrained architectures. Thus, another important direction for future work, related to the high-level, hardware synthesis methods described above, is development of methods in LWDF-V for efficient analysis and optimization of dataflow buffers.

Buffer minimization is closely related to dataflow graph scheduling, and has been studied in various bodies of prior work. Related prior work on buffer minimization has often focused on specialized forms of dataflow, such as synchronous dataflow (SDF) and homogeneous SDF [7], as well as cyclo-static dataflow (CSDF) [54]. Scheduling homogeneous SDF graphs to minimize buffer costs was proved to be NP-complete [58]. Cubric and Panangaden presented a heuristic approach to minimize buffer costs for a restricted set of acyclic, delayless SDF graphs [59]. Wiggers et al. presented an algorithm that determines minimum buffer sizes for CSDF graphs under throughput constraints [60]. Oh and Ha [61] presented a method for minimizing buffer costs based on representing production and consumption rates as fractional values [61]. Horstmannshoff and Meyr studied interactions between retiming and buffer minimization for dataflow graph implementation [62].

These approaches for buffer optimization are based on various types of analysis of dataflow graph properties. However, they do not address the integrated optimization of actor-level pipelining, overlapping of token processing with input token consumption, and buffer management for hardware implementation. Such integrated optimization is useful in efficiently mapping dataflow graph models into hardware implementations, and is a useful direction for future work on further enhancements to LWDF-V.

Bibliography

[1] F. Darema. Grid computing and beyond: The context of dynamic data driven applications systems. *Proceedings of the IEEE*, 93(2):692–697, 2005.

[2] C. Shen, W. Plishker, H. Wu, and S. S. Bhattacharyya. A lightweight dataflow approach for design and implementation of SDR systems. In *Proceedings of the Wireless Innovation Conference and Product Exposition*, pages 640–645, Washington DC, USA, November 2010.

[3] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, editors. *Handbook of Signal Processing Systems.* Springer, second edition, 2013. ISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online).

[4] J. Suhonen, M. Kohvakka, V. Kaseva, T. D. Hämäläinen, and M. Hännikäinen. Low-power wireless sensor network platforms. In S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, editors, *Handbook of Signal Processing Systems*. Springer, second edition, 2013.

[5] B. Bhattacharya and S. S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, 49(10):2408–2421, October 2001. DOI:10.1109/78.950795.

[6] J. B. Dennis. First version of a data flow procedure language. Technical report, Laboratory for Computer Science, Massachusetts Institute of Technology, May 1975.

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

[8] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya. Functional DIF for rapid prototyping. In *Proceedings of the International Symposium on Rapid System Prototyping*, pages 17–23, Monterey, California, June 2008.

[9] Texas Instruments, Inc. *CC2530F32, CC2530F64 CC2530F128, CC2530F256: A True System-on-Chip Solution for 2.4–GHz IEEE 802.15.4 and ZigBee Applications*, February 2011.

[10] I. Cho, C. Shen, S. Potbhare, S. S. Bhattacharyya, and N. Goldsman. Design methods for wireless sensor network building energy monitoring systems. In *Proceedings of the IEEE International Workshop on Practical Issues in Building Sensor Network Applications*, pages 974–981, Bonn, Germany, October 2011.

[11] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. Wireless multimedia sensor networks: Applications and testbeds. *Proceedings of the IEEE*, 96(10):1588–1605, October 2008.

[12] C. Shen, W. Plishker, and S. S. Bhattacharyya. Dataflow-based design and implementation of image processing applications. Technical Report UMIACS-TR-2011-11, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011. http://drum.lib.umd.edu/handle/1903/11403.

[13] L. Zhaohua and G. Mingjun. Survey on network lifetime research for wireless sensor networks. In *Proceedings of the IEEE International Conference on Broadband Network & Multimedia Technology*, pages 899–902, 2009.

[14] R. Zhang, Z. Jia, and D. Yuan. Analysis of lifetime of large wireless sensor networks based on multiple battery levels. *International Journal of Communications, Network and System Sciences*, 1(2):136–143, 2008.

[15] T. V. Padmavathy. Extending the network lifetime using optimized energy efficient cross layer module (OEEXLM) in wireless sensor networks. *Wireless Sensor Network*, 1(1):27–35, 2009.

[16] L. Wang and S. S. Kulkarni. Sacrificing a little coverage can substantially increase network lifetime. In *Proceedings of the Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, pages 326–335, 2006.

[17] R. Madan, C. Shuguang Cui, S. Lall, and A. Goldsmith. Cross-layer design for lifetime maximization in interference-limited wireless sensor networks. *IEEE Transactions on Wireless Communications*, pages 3142–3152, 2006.

[18] M. Kintner-Meyer, M. R. Brambley, T. Carlon, and N. Bauman. Wireless sensors: Technology and cost-savings for commercial buildings. In *ACEEE Summer Study on Energy Efficiency in Buildings*. European Council for an Energy Efficient Economy, 2002.

[19] W. S. Jang, W. M. Healy, and M. J. Skibniewski. Wireless sensor networks as part of a web-based building environmental monitoring system. *Automation in Construction*, 17(6):729–736, 2008.

[20] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri. Monitoring civil structures with a wireless sensor network. *IEEE Internet Computing*, 10(2):26–34, 2006.

[21] NXP Semiconductors. *I2C–bus specification and user manual*, October 2012.

[22] E. H. Wold and A. M. Despain. Pipeline and parallel-pipeline FFT processors for VLSI implementations. *IEEE Transactions on Computers*, C-33(5):414–426, 1984.

[23] W. Han, T. S. Arslan, A. T. Erdogan, and M. M. Hasan. Multiplier-less based parallel-pipelined FFT architectures for wireless communication applications. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages v/45–v/48, 2005.

[24] J. Palmer and B. Nelson. A parallel FFT architecture for FPGAs. In J. Becker and S. Vernalde M. Platzner, editors, *Field Programmable Logic and Application*, volume 3203 of *Lecture Notes in Computer Science*, pages 948–953. Springer Berlin Heidelberg, 2004.

[25] S. He and M. Torkelson. A new approach to pipeline FFT processor. In *Proceedings of the International Parallel Processing Symposium*, pages 766–770, 1996.

[26] A. Saeed, M. Elbably, G. Abdelfadeel, and M. I. Eladawy. Efficient FPGA implementation of FFT/IFFT processor. *International Journal of Circuits, Systems and Signal Processing*, 3(3):103–110, 2009.

[27] H. Jiang, H. Luo, J. Tian, and W. Song. Design of an efficient fft processor for ofdm systems. volume 51, pages 1099–1103, 2005.

[28] C.-P. Hung, S.-G. Chen, and K.-L. Chen. Design of an efficient variable-length FFT processor. In *Proceedings of the International Symposium on Circuits and Systems*, pages II–833–II–836, 2004.

[29] Y. H. Jung, H. I. Yoon, and J.-S. Kim. New efficient FFT algorithm and pipeline implementation results for OFDM/DMT applications. *IEEE Transactions on Consumer Electronics*, 49(1):14–20, 2003.

[30] S.-J. Huang and S.-G. Chen. A memory-efficient continuous-flow FFT processor for Wimax application. In *Proceedings of the International Symposium on Circuits and Systems*, pages 17–20, 2012.

[31] H. Rohling, editor. *OFDM: Concepts for Future Communication Systems.* Springer, 2011.

[32] F. Kristensen, P. Nilsson, and A. Olsson. Reduced transceiver-delay for OFDM systems. In *Proceedings of the Vehicular Technology Conference*, volume 3, pages 1242–1245, 2004.

[33] 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. *Proceedings of the IEEE*, January 2003.

[34] M. C. Johnson and K. Roy. Datapath scheduling with multiple supply voltages and level converters. *ACM Transactions on Design Automation of Electronic Systems*, 2(3):227–248, July 1997.

[35] 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*, November 1998.

[36] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, and J. Teich. A SystemC-based design methodology for digital signal processing systems. *EURASIP Journal on Embedded Systems*, 2007:Article ID 47580, 22 pages, 2007.

[37] M. Pelcat, S. Aridhi, J. Piat, and J.-F. Nezan. *Physical Layer Multi-Core Prototyping*. Springer, 2013.

[38] C. Shen, W. L. Plishker, D. Ko, S. S. Bhattacharyya, and N. Goldsman. Energy-driven distribution of signal processing applications across wireless sensor networks. *ACM Transactions on Sensor Networks*, 6(3), June 2010. Article No. 24, 32 pages, DOI:10.1145/1754414.1754420.

[39] J. Janneck, I. Miller, and D. Parlour. Profiling dataflow programs. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, June 2008.

[40] S. Phadke, R. Limaye, S. Verma, and K. Subramanian. On design and implementation of an embedded automatic speech recognition system. In *Proceedings of the International Conference on VLSI Design*, pages 27–132, 2004.

[41] C. Shen, W. Plishker, and S. S. Bhattacharyya. Dataflow-based design and implementation of image processing applications. In L. Guan, Y. He, and S.-Y. Kung, editors, *Multimedia Image and Video Processing*, pages 609–629. CRC Press, second edition, 2012. Chapter 24.

[42] S. Neuendorffer and E. Lee. Hierarchical reconfiguration of dataflow models. In *Proceedings of the International Conference on Formal Methods and Models for Codesign*, June 2004.

[43] K. Desnos, M. Pelcat, J.-F. Nezan, S. S. Bhattacharyya, and S. Aridhi. PiMM: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 41–48, Samos, Greece, July 2013.

[44] B. D. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Proceedings of the International Conference on Formal Methods and Models for Codesign*, July 2006.

[45] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, pages 773–799, May 1995.

[46] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprettere. System design using Kahn process networks: the Compaan/Laura approach. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, February 2004.

[47] B. Kienhuis and E. F. Deprettere. Modeling stream-based applications using the SBF model of computation. *Journal of Signal Processing Systems*, 34(3):291–299, 2003.

[48] C. Zissulescu, B. Kienhuis, and E. Deprettere. Expression synthesis in process networks generated by laura. In *Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors*, July 2005.

[49] J. Eker and J. W. Janneck. CAL language report, language version 1.0 — document edition 1. Technical Report UCB/ERL M03/48, Electronics Research Laboratory, University of California at Berkeley, December 2003.

[50] S. Holzner. *Eclipse*. O'Reilly & Associates, Inc., 2004.

[51] S. S. Bhattacharyya, G. Brebner, J. Eker, J. W. Janneck, M. Mattavelli, C. von Platen, and M. Raulet. OpenDF — a dataflow toolset for reconfigurable hardware and multicore systems. In *Proceedings of the Swedish Workshop on Multi-Core Computing*, pages 43–49, Ronneby, Sweden, November 2008.

[52] J. McAllister, R. Woods, R. Walke, and D. Reilly. Synthesis and high level optimisation of multidimensional dataflow actor networks on FPGA. In *Proceedings of the IEEE Workshop on Signal Processing Systems*, 2004.

[53] D.E. Thomas and P.R. Moorby. *The Verilog Hardware Description Language*. Kluwer Academic Publishers, fifth edition, 2002.

[54] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, February 1996.

[55] European Telecommunications Standards Institute. *ETSI TS 136 300 Technical Specification, V10.7.0*, 2012.

[56] LTE; feasibility study for further advancements for E-UTRA (LTE-Advanced). Technical Report 3GPP TR 36.912 version 10.0.0 Release 10, 2011.

[57] I. Cho, C. Shen, Y. Tachwali, C. Hsu, and S. S. Bhattacharyya. Configurable, resource-optimized FFT architecture for OFDM communication. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 2746–2750, Vancouver, Canada, May 2013.

[58] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.

[59] M. Cubric and P. Panangaden. Minimal memory schedules for dataflow networks. In *Proceedings of CONCUR '93*, pages 368–383, August 1993.

[60] M. H. Wiggers, M. J. G. Bekooij, and G.J.M. Smit. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proceedings of the Design Automation Conference*, pages 658–663, 2007.

[61] H. Oh and S. Ha. Fractional rate dataflow model for efficient code synthesis. *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 37:41–51, May 2004.

[62] J. Horstmannshoff and H. Meyr. Optimized system synthesis of complex rt level building blocks from multirate dataflow graphs. In *Proceedings of the International Symposium on System Synthesis*, November 1999.