# A CONTEXT-BASED SOFTWARE RECONFIGURABLE SYSTEM FOR WIRELESS SENSOR NETWORK

**BY**

**ERONU EMMANUEL MAJIYEBO**

**Ph.D/SEET/2010/333**

**DEPARTMENT OF COMPUTER ENGINEERING,
FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA**

**SEPTEMBER, 2014**

# A CONTEXT-BASED SOFTWARE RECONFIGURABLE SYSTEM FOR WIRELESS SENSOR NETWORK

**BY**

**ERONU EMMANUEL MAJIYEBO**

**Ph.D/SEET/2010/333**

**THESIS SUBMITTED TO THE POSTGRADUATE SCHOOL, FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA, NIGERIA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF DOCTOR OF PHILOSOPHY (Ph.D) IN COMPUTER ENGINEERING**

**SEPTEMBER, 2014**

# ABSTRACT

Wireless Sensor Network application entails deploying thousands of wireless sensor nodes in unreachable locations. The inability to reconfigure each node in order to take on new tasks poses a serious challenge to the continued operation of the entire system. Several attempts have been made to address these challenges, of interest is one that exploits design-time knowledge of the application scenario dynamics to construct and implements a proactive runtime reconfiguration paradigm. However, It suffers two defects: the possibility of capturing all anticipated reconfiguration needs can be challenging, and the scarcely available memory space might not be sufficient to accommodate codes written to address these needs. Moreover, even if it does, there is the likelihood of redundant codes written to handle anticipated changes, which might never occur, and invariably taking up scarcely available memory spaces. This research work explores the use of context information to improve upon wireless sensor networks reconfiguration processes. The research's aim is to develop a software system that dynamically reconfigures wireless sensor network operational functionalities optimally based on evolving application context. In order to demonstrate the benefits of the context based reconfiguration model, two contexts related input variables were used. The first variable is obtain using a metric tool (PDE) devised for extracting context information from the delta of two files (application related context). The second variable entails the battery energy level state of the sensor node taken as an operational-demand related context. A robust inference engine was developed based on the inferred expert knowledge on memory related energy consumption pattern during the reconfiguration process. The pattern studied and presented explains how delta size and its orientation can influence energy consumption while reprogramming sensor nodes. The resulting output from the fuzzy logic system controls when and which one of the reconfiguration approaches should be implemented in order to prolong the battery life. The model's performance was evaluated on an OMNet++ simulation platform using pilot data obtained from a testbed composed of Microchips' PIC32MX320F128H microcontroller and MRF24J40MB transceiver. In a network of six nodes, two were equipped with the developed model capability and the others were not. The overall energy expended as read, erase and write were obtained from each node for the purpose of comparison. Results obtained show that 65% of energy expended during the erasure procedure is saved in nodes that adopt the context based reconfiguration model. Similarly, 45% and 69% reduction in energy consumption were obtained for the read and write procedures respectively. The research work was able to emphasise the benefits of identifying, employing and managing the impact of contextual information (Application/operational related) during wireless sensor network reconfiguration procedure.

# TABLE OF CONTENTS

**CONTENT** **PAGE**

# LIST OF TABLES

**TABLE** **PAGE**

# LIST OF FIGURES

**FIGURE** **PAGE**

# LIST OF PLATES

# ABBREVIATIONS

| | |
|---|---|
| AI | Artificial Intelligence |
| ANN | Artificial Neural Networks |
| BER | Bit Error Rate |
| CAN | Controller Area Network |
| CCA | Clear Channel Assessment |
| CMOS | Complementary Metal Oxide Semiconductor |
| CSMA-CA | Carrier Sense Multiple Access-Collision Avoidance |
| DLM | Dynamic Loadable Module |
| DMA | Direct Memory Access |
| EEPROM | Electrical Erasable Programmable Read Only Memory |
| ELF | Execution Link File format |
| EOS | Embedded Operating System |
| FCL | Fuzzy Control Language |
| FIFO | First in First out |
| FIS | Fuzzy Inference System |
| FLC | Fuzzy Logic Controller |
| FPGA | Field Programmable Gate Array |
| FSK | Frequency-Shift Keying |
| GNU | Gnu’s Not Unix |
| GPRS | General Packet Radio Services |
| GPS | Global Positioning System |
| GSM | Global System for Mobile |

| | |
|---|---|
| GTS | Guaranteed Time Slots |
| HAA | Hardware Abstraction Architecture |
| HDL | Hardware Description Language |
| HEI | Hot Electron Injection |
| HIL | Hardware Independent Layer |
| HPL | Hardware Presentation Layer |
| I2C | Inter-Integrated Interface |
| MAC | Media Access Control |
| MIPS | Microprocessor without Interlocked Pipeline Stages |
| MLF | Micro Lead Frame |
| NFT | Nordheim Fowler Tunnelling |
| OS | Operating System |
| PDE | Precise Delta Extraction (scheme) |
| PHY | Physical Layer |
| PSK | Phase-shift keying |
| REOS | Real-time Embedded Operating System |
| RF | Radio Frequency |
| RFID | Radio Frequency Identification |
| RISC | Reduced Instruction Set Computer |
| RSSI | Received Signal Strength Indicator |
| SDR | Software Define Radio |
| SINR | Signal to Interference plus Noise Ratio |
| SNR | Signal to Noise Ratio |
| SPI | Serial Peripheral Interface |

| | |
|---|---|
| SRAM | Static Random Access Memory |
| UART | Universal Asynchronous Receiver/Transmitter |
| UWB | Ultra-Wide Band |
| USB | Universal Serial Bus |
| WIFI | Wireless Fidelity |
| WIMAX | Worldwide Interoperability for Microwave Access |
| WPAN | Wireless Personal Area Network |
| WSN | Wireless Sensor Network |

# CHAPTER ONE

## 1.0 INTRODUCTION

Wireless sensor network (WSN) is a collection of small-embedded devices interconnected with the sole aim of sensing, processing, sharing and remotely relaying data via known communication protocols. WSN applications are widespread and increasingly growing by the day. Examples of these applications entail: military sensing, physical security, air traffic control, traffic surveillance, video surveillance, industrial and manufacturing automation, distributed robotics, health care monitoring and delivery.Others are environmental monitoring, observatory purposes in weather and earthquake monitoring, building and structural monitoring (Chong and Kumar, 2003).

The small embedded devices commonly referred to as wireless sensor nodes share or relay data to the base station by employing various communication models. A number of communication models exist as follows: direct, multi-hop and clustering. The node consists of sensors, processing elements (microcontrollers), radio communication interface and a power source (battery and solar). The sensors detect and measure physical phenomena such as temperature, light, magnetic field, pressure, acceleration, current and ultrasound.

A typical WSN application entails deploying hundreds or thousands of wireless sensor nodes in unreachable locations. Examples of these applications are as follows: surveillance, environmental monitoring, oil and gas pipeline monitoring (Misra and Eronu, 2012). When there is a change in the operational needs of the system or new functionalities are required in such application, reconfiguration of either the entire network or individual sensor nodes become inevitable. The inability to effect these changes could pose a serious challenge to the continued operation of the entire system

(Misra and Eronu, 2012). Other issues that could warrant the need for a reconfigurable WSN are bug fixes (Hinkelmann, Reinhardt and Glesner, 2008) , regular code updates (Kulkarni, Sanyal, Al-Qaheri and Sanyal, 2009), security challenges (Portilla, Otero, De la Torre, Riesgo, Stecklina, Peter and Langendorfer, 2010), RF communication link (Ramamurthy, Prabhu and Gadh, 2004) and efficient energy management.

Altering system functionality in both real-time or design time involves making changes to either the hardware component or software component or both components. The altering process could in some cases (Krishna, Bagchi and Khalil, 2009) be referred to as 'Reprogramming' , and in some other cases (Muralidhar and Rao, 2008) it is considered as 'Reconfiguration'. When only the software component is involved, it is termed 'Reprogramming'. Likewise, the term "Reconfiguration" is used when the Hardware components are involved. Probably in agreement with this proposition, Compton and Hauck (2002) described reconfigurable systems as devices that incorporate some form of hardware programmability. However, in this thesis, both terms are used interchangeably. Both terms refer to 'an act or process of effecting a change' to the system's underlying codes or instructions (high-level or low-level languages and Hardware Description Language (HDL)). The aim is to alter its initial functions. Some other words often used to connote reconfiguration in certain literature are 'updating' , 'adaptation' or 'Adapting' (Brown and Sreenan, 2006; Han, Kumar, Shea and Srivastava, 2005). Stating these definitions clearly prevent misapprehension due to the use of different words or terms meant to explain the same concept.

Good design criteria demand that for a system to be cost-effective, it should possess attributes that enable it take cognisance of the resources around its immediate and remote environments. It should autonomously or remotely be directed to perform new tasks or implement existing task more efficiently. The adoption of Context-driven and context-aware paradigm in distributed systems is on the increase, as such WSN should not be an exception (Silva and Vuran, 2010).

Sensor network application can be expensive to implement, especially when large-scale projects are involved. Being able to manage network resources and tailor their use towards several other applications other than what they were initially designed for can be a daunting task. Application objectives, anticipated constraints, resource managerial strategies and other surrounding factors, when well spelt out in the design model, simplify the complexity arising from adapting WSN to newer applications. Identifying these factors requires a careful analysis of the entire WSN operational environment. When these factors are considered as a source of relevant contextual information, then reconfiguring WSN becomes much easier. In this perspective, the intent is to understudy context-related approaches as they relate to reconfiguration computing and by extension reconfigurable WSN.

Baldauf, Dustdar, and Rosenberg (2007) described context-aware systems as systems that can alter their mode of operation to suit the current context without explicit user intervention thereby increasing the systems usability and effectiveness. Context awareness is commonly used in systems whose operation or responses are influenced by certain defined surrounding factors. The concept of context-aware systems allows applications to gather context data and adapt their operational behaviour accordingly. These applications can function without explicit intervention and thereby increase their usability and effectiveness within the context of the environment where they operate (Baldauf *et al.,* 2007). Context-driven allows a system to assign resources on current and relevant tasks, rather than just processing predefined applications. Equipping the node with relevant context sensing capabilities enables it to estimate future context requirements. When these requirements are used appropriately, the network can be configured to perform more optimally. Management systems can guess about what kind of tasks will be required in the near future and consider it when allocating resources. Hence, effecting the sensor nodes' reconfiguration processes based on contextual information can be helpful in several ways. For example, deciding on when and how to effect a reconfiguration process can result in reducing the system's operational cost. This cost invariably entails energy consumed and memory size utilised by the nodes during the reconfiguration process.

### 1.1 Motivation

In order to appreciate the benefit of such a context-based reconfigurable paradigm, consider a WSN application scenario as depicted in Figure 1.1. The application is intended to be deployed and utilised in an urban setting, and the nodes have the capability to reconfigure themselves autonomously. In addition, they can as well be remotely reconfigured to use any desired particular communication standard (RFID, Bluetooth, UWB, Zigbee, GSM, GPRS, WIFI or WiMax). Taking into consideration also that in most urban settings, fully installed and operational communication infrastructure supporting known communication standards is virtually everywhere. If the earlier mentioned considerations are viable, then the WSN can be remotely reconfigured to take advantage of the available infrastructure (gateways and base stations) already on the ground instead of setting up new ones. Adopting the intended model reduces the cost of deploying and installing new gateways and possibly new base stations.

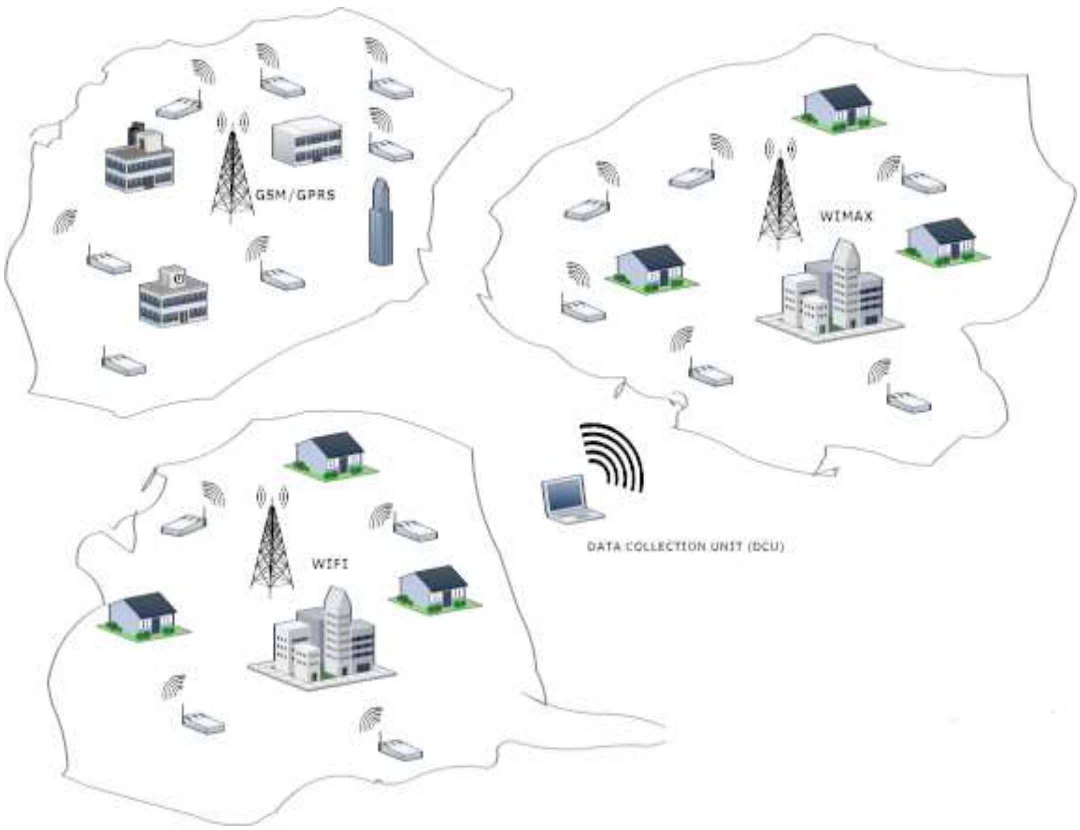


Figure 1.1: Exemplified scenario of context-aware inclined reconfigurable WSN

The nodes can easily adopt future communication standards whenever they become available. Instead of retrieving older nodes and replacing them with newer ones, the older ones can simply be reconfigured on the fly thereby enabling them to function within evolving context requirements.

Capturing and using context information during reconfiguration processes can be helpful in intelligently managing the node's resources. This guaranty optimal performance and efficient use of scarce available resources (energy and memory space)

In view of these needs and observed deficiencies in existing approaches, the research work is intended to address the following:

- Reduce the presence of redundant codes, thereby lessening the size of the firmware deployed to the wireless sensor nodes;
- Enhance the flexibility of reuse; allow real-time user input during reconfiguration processes and autonomous reconfiguration using fuzzy logic in decision making;
- Establish a two-way interactive platform between the reconfiguring agent (user via base station) and the reconfigured (sensor node) and by extension the entire wireless sensor network. The two-way interactive platform enables the base station to assess the state of the sensor node through the contextual information it relayed. In addition, coupled with other relevant information (operational related contextual information), the system then decides when and how or what manner of reconfiguration should be employed. The aim is to ensure that the entire network performs efficiently and optimally manages the available resources (memory usage and energy consumption)
- Include artificial intelligence techniques (fuzzy logic) in reconfiguration processes to enable the entire system autonomously respond to evolving changes especially in unfriendly environments.

The benefits of the model are to reduce energy consumption rate and effect a reduction in the amount of memory-space used while reprogramming a wireless sensor node. The inclusion of artificial intelligence techniques (fuzzy logic) in reconfiguration processes enables the entire system to respond to evolving changes in an unfriendly environment.

## 1.2 Problem Statement

Steine, Ngo, Oliver, Geilen, Basten, Fohler and Decotgnie (2011) introduced an approach that exploits design-time knowledge of the application scenario dynamics to construct and implements a proactive runtime reconfiguration paradigm. However, two challenging issues are apparent here: , the possibility of capturing all anticipated reconfiguration needs can be challenging, and the scarcely available memory space might not be sufficient to accommodate codes written to address these needs. Moreover, even if it does, there is the likelihood of redundant codes written to handle anticipated changes, which might never occur, and invariably taking up scarcely available memory spaces. A review of existing reconfiguration approaches and related challenges (energy consumption rate and memory space ) is reported in Eronu, Misra and Aibinu (2013).

In addition, implementing WSN reconfiguration may depend on whether it is needful, urgent, or sustainable. For example, instead of effecting reconfiguration procedure during unfavourable weather conditions, it may be needful to delay the process and then resume when the conditions become favourable. In extreme cases, it is advisable to stop the process completely when the available energy in the node cannot sufficiently sustain the reconfiguration process. Where the second option is the norm, the sensor node might not be able to implement new functionalities but it can still be utilised for other purposes not dependent on the update. The ability to take decisions of this nature is largely confined to the human domain. However, Artificial Intelligence (AI) techniques like the Fuzzy Logic and Artificial Neural Network allow machines to mimic human cognitive capabilities. Importantly, the problem needs to be presented as defined input variables and the output variables make-up the solutions. Solutions are obtained from the analyses of processed input variables in conformity with a set of rules that are based or derived from expert knowledge.

## 1.3 Aim and Objectives

The aim of this research is to develop a software system that dynamically reconfigures wireless sensor network operational functionalities optimally based on evolving application context. In order to realise the aforementioned aim, the under listed set of objectives were actualised and used to devise result-oriented procedures:

I. To devise a WSN context based reconfiguration model;
II. To design and implement a metric utility for measuring the degree of changes made in modified application source codes and relaying the exact changes;
III. To integrate and use fuzzy logic controller in deciding the most appropriate reconfiguration approach to adopt in response to evolving application or operational context; and
IV. To evaluate the performance of the developed system.

## 1.4 Limitation of Study

The Execution Link File (ELF) format adopted for developing the Precision Delta Extraction (PDE) tool in this work is not implemented in certain operating systems like the TinyOS. Hence, this limitation has constrained most of the work to only sensors nodes that employ the ELF format in their firmware generation and deployment.

## 1.5 Scope of Study

Several reconfiguration approaches are currently being implemented at various layers of the sensor node architecture. Majority of these approaches are still under development; that is, research are still on and their possible adoption in real life application scenario appears remote. For example, the use of field programmable gate arrays (FPGA) to actualise reconfigurable processors or rather soft-processors for wireless sensor nodes is not feasible now. More detailed information on the implementation of selected reconfiguration approaches at four layers is presented in Chapter Two. However, in this research work, the design and implementation processes are confined to the operating system platform.

## 1.6 Thesis Outline

The general introduction, statement of the problem, the aim, objectives and justification of the work were presented in this Chapter. Chapter Two presents a review of several wireless sensor reconfiguration research works from the following perspective: the driving factors necessitating reconfiguration needs, previous and current reconfiguration approaches at some selected layers of the sensor node. The four selected layers are namely: the application, middleware, processing elements and the operating system layers. In addition, challenges and lapses associated with these approaches as implemented in the various layers were also presented. Also, further discussion on how these lapses can be addressed using surrounding contextual information presented. In Chapter Three, a detailed description of the research methodology presented. The description spans over the design and development of the context based reconfiguration software system for wireless sensor network model. The formulation and application of two additional subcomponents namely the precise delta extraction tool and a fuzzy logic controller were discussed. In addition, the testbed composition and setup for evaluating the model's pilot data, and the simulation tool employed to evaluate the model on a larger scale are presented. Chapter Four presents the results and discussion of the research. Finally, Chapter Five presents some concluding remarks and recommendations for future works.

# CHAPTER TWO

## 2.0 LITERATURE REVIEW

Research efforts towards devising the most efficient and appropriate approach in realising WSN whose operational and functional capabilities can be altered on-the-fly have been on for quite a long time now. This chapter presents a review of previous and current reconfiguration approaches at some selected layers of the sensor node. The four selected layers are namely: the Processing Elements, Radio Frequency Transceiver, Application, Middleware, and the Operating System layers. Brief background information on the impact of reconfiguration processes on WSN operations was conveyed. In addition, challenges and lapses associated with these approaches as implemented in the various layers were also presented.

Much work is concentrated on the operating system layer, and its related reconfiguration approaches because of its widespread adoption as reported in most literatures. The different paradigms employed by some selected number of operating systems tailored for the WSN application were reviewed. Comparative studies of the energy cost of implementing the various approaches reviewed are also presented in this chapter.

Attempts to use context information in WSN applications were reviewed and subsequently reported. Studies indicate that limited efforts were directed towards the use of contextual information in addressing WSN reconfiguration issues especially those related to its resource management. In addition, the use of Artificial Intelligence (AI) to manage WSN related resource-constraint problems, which were mainly at experimental stages were presented. The review highlighted some of the milestones archived from previous attempts to employ contextual information and AI in addressing WSN resource-constrained issues. The findings indicate that not much work has been concentrated in WSN related reconfiguration problems

**2.1 Wireless Sensor Network (WSN)**

The WSN (see Figure 2.1) is built of few to several hundred or even thousands of sensor nodes. Each node is connected to one or more sensors. Each sensor network node consist of several parts: a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, an electronic circuit for interfacing with the sensors and an energy source, usually a battery or an embedded form of energy harvesting. The topology of the WSNs can vary from a simple star network to an advanced multihop wireless mesh network.

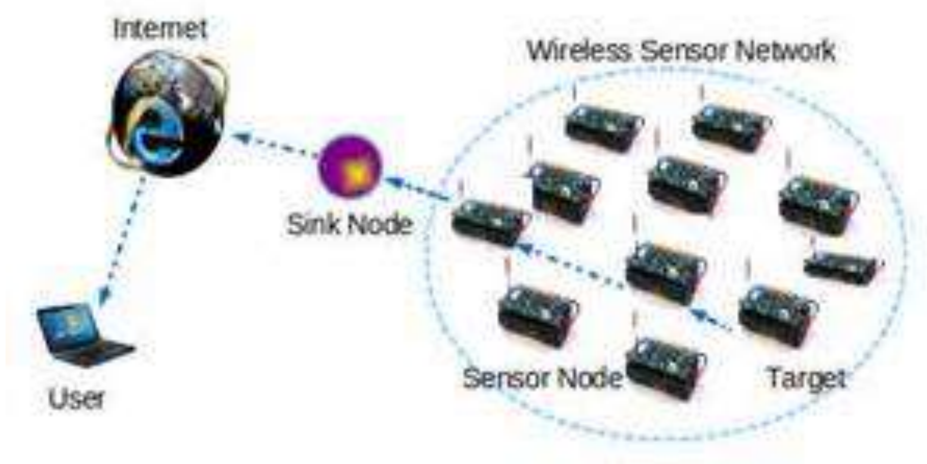


Figure 2.1: A typical example of a Wireless Sensor Network (www.virtual-labs.ac.in)

The propagation technique between the hops of the network can be routing or flooding

The main characteristics of a WSN include:

- Power consumption constraints for nodes using batteries or energy harvesting
- Ability to cope with node failures (resilience)
- Mobility of nodes
- Heterogeneity of nodes
- Scalability to large scale of deployment
- Ability to withstand harsh environmental conditions
- Ease of use
- Cross-layer design

Cross-layer is becoming an important studying area for wireless communications. In addition, the traditional layered approach presents three main problems:

- Traditional layered approach cannot share different information among different layers, which leads to each layer not having complete information. The traditional layered approach cannot guarantee the optimization of the entire network.
- The traditional layered approach does not have the ability to adapt to the environmental change.
- Because of the interference between the different users, access confliction, fading, and the change of environment in the wireless sensor networks, traditional layered approach for wired networks is not applicable to wireless networks.

### 2.1.1 Hardware and Software components of WSN

One major challenge in a WSN is to produce low cost and tiny sensor nodes. Many of the nodes are still in the research and development stage, particularly their software. Also, inherent to sensor network adoption is the use of very low power methods for radio communication and data acquisition.

In many applications, a WSN communicates with a Local Area Network or Wide Area Network through base stations or a gateway. The base stations are one or more components of the WSN with much more computational, energy and communication resources. They act as a gateway between sensor nodes and the end user as they typically forward data from the WSN on to a server. Other components in routing based networks are routers, designed to compute, calculate and distribute the routing tables. The Gateway acts as a bridge between the WSN and the other network. This enables data to be stored and processed by devices with more resources, for example, in a remotely located server.

Energy is the scarcest resource of WSN nodes, and it determines the lifetime of WSNs. WSNs may be deployed in large numbers in various environments, including remote and hostile regions, where ad hoc communications are a key component. For this reason, algorithms and protocols need to address the following issues:

- Lifetime maximization
- Robustness and fault tolerance
- Self-configuration

Lifetime maximization: Energy/Power Consumption of the sensing device should be minimised and sensor nodes should be energy efficient since their limited energy resource determines their lifetime. To conserve power the nodes normally turn off the radio transceiver when not in use. Some of the important topics in WSN software research are:

- Operating systems
- Security
- Mobility
- Usability
- Maintenance

Operating systems for wireless sensor network nodes are typically less complex than general-purpose operating systems. Wireless sensor nodes strongly resemble embedded systems, for two reasons. First, wireless sensor networks are typically deployed with a particular application in mind, rather than as a general platform. Second, a need for low costs and low power leads most wireless sensor nodes to have low-power microcontrollers ensuring that mechanisms such as virtual memory are either unnecessary or too expensive to implement.

TinyOS is perhaps the first operating system specifically designed for wireless sensor networks. TinyOS is based on an event-driven programming model instead of multithreading. TinyOS programs are composed of event handlers and tasks with run-to-completion semantics. When an external event occurs, such as an incoming data packet or a sensor reading, TinyOS signals the appropriate event handler to handle the event. Event handlers can post tasks that are scheduled by the TinyOS kernel some time later. Contiki uses a simpler programming style in C while providing advances such as 6LoWPAN and Protothreads.

### 2.1.2 Simulation of WSNs

At present, agent-based modelling and simulation are the only paradigm, which allows the simulation of complex behaviour in the environments of wireless sensors (such as flocking). Agent-based simulation of wireless sensor and ad hoc networks is a relatively new paradigm. Network simulators like OPNET, OMNeT++, NetSim, and NS2 can be used to simulate a wireless sensor network.

## 2.2 Reconfigurable Computing

Reconfigurable computing is a computer architecture combining some of the flexibility of software with the high performance of hardware by processing with very flexible high-speed computing fabrics like field-programmable gate arrays (FPGAs). The principal difference when compared to using ordinary microprocessors is the ability to make substantial changes to the datapath itself in addition to the control flow. On the other hand, the main difference with custom hardware, i.e. application-specific integrated circuits (ASICs) is the possibility to adapt the hardware during runtime by "loading" a new circuit on the reconfigurable fabric.

Reconfigurable computing technologies offer the promise of substantial performance gains over traditional architectures via the customizing, even at run-time, the topology of the underlying architecture to match the specific needs of a given application.

Contemporary configurable architectures allow for the definition of architectures with functional and storage units that match in function, bit-width and control structures the specific needs of a given computation. For example, one can define a numerically intensive architecture for digital signal processing with specific number of input/output channels meeting specific timing requirements and/or organise internal RAM modules with a given bandwidth to match the processing rate of the functional units. The flexibility enabled by reconfiguration is also seen as a basic technique for overcoming transient failures in emerging device structures.

There are two primary methods in traditional computing for the execution of algorithms. The first is to use an Application Specific Integrated Circuit (ASIC), to perform the operations in hardware. Because these ASICs are designed specifically to perform a given computation, they are very fast and efficient when executing the exact computation for which they were designed. However, after fabrication, the circuit cannot be altered. Microprocessors are a far more flexible solution. Processors execute a set of instructions to perform a computation. By changing the software instructions, the functionality of the system is altered without changing the hardware. However, the downside of this flexibility is that the performance suffers and is far below that of an ASIC. The processor must read each instruction from memory, determine its meaning, and only then execute it. This results in a high execution overhead for each operation. Reconfigurable computing is intended to fill the gap between hardware and software, and to achieve much higher performance than software potentially while maintaining a higher level of flexibility than hardware.

## 2.3 Impact of Reconfiguration Approaches

Reconfiguration processes, though intended to improve upon the services and operation of WSN, unfortunately, contribute to the system's performance impediment. This notably poses many reconfiguration challenges at all layers/platforms.

A measurement of performance related issues for the purpose of comparison can be complicated. Several factors not directly related to the reconfiguration process can impede a wireless sensor network performance. For example, propagation delays resulting from Multipath phenomenon, especially in extreme cases of nulling**.** Nulling refers to the cancellation of RF signal. The cancellations results in retransmission attempts (Moerschel *et al.,* 2007), which, invariably affects how long it takes for the updates or reconfiguration process to be completed.

An assessment of these challenges confines this impediment to the following: Energy demands of key active components of the sensor nodes and memory space required when carrying out the reconfiguration process.

### 2.3.1 Memory space

Depending on the reconfiguration method employed, it is possible for a scheme to consume a large portion of memory while storing an image or related patches. A memory overlap could occur, thereby limiting the overall performance of the network. It gains more significance in Operating Systems like Sensor Operating System (SOS) where it is allocated dynamically at runtime (Balani, Han, Rengaswamy, Tsigkogiannis and Srivastava, 2006).

### 2.3.2 Energy consumption

Reprogramming requires the transmission of new images (complete, patches, modular) as updates from the base station to the individual nodes. In the course of implementing this process, certain sections of the nodes' memories (EEPROM or Flash memory) are read from and written to. Sometimes, the whole process is repeated several times because of erroneous transmission and reception of data via noisy communication channel. Subsequently, this leads to an increase in the nodes processing power an appreciable demand in consumption of scarce energy resources.

## 2.4 Overview of Reconfiguration Approaches on Selected Enabling Platforms

Similar Survey works on reconfigurable WSN seems to concentrate on software updates alone (Chong and Kumar, 2003; Han, Kumar, Shea, and Srivastava, 2005; Kulkarni, Sanyal, Al-Qaheri, and Sanyal, 2009; Yick, Mukherjee, and Ghosal, 2008). However, this research work spans over reconfiguration approaches involving hardware components as well. The reconfiguration approach viewed from two perspectives involves those categorised under software and hardware groups. The software entails the application layer, middleware and the operating system whereas hardware comprise of the Processing element and the RF communication platform (Figure 2.2). Reconfiguration-related issues and challenges in both the hardware and software subcomponents are presented in this subsection.

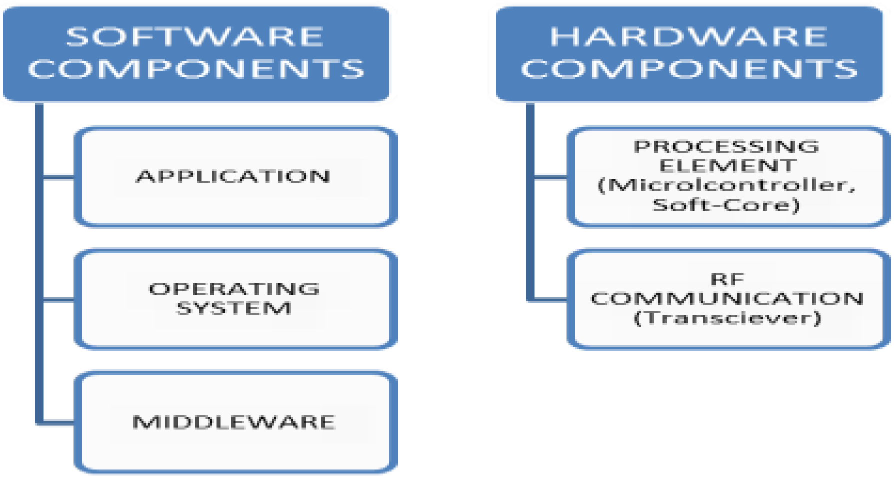


Figure 2.2: Reconfigurable software and hardware

### 2.4.1 Processing Element

The Processing Element consists of hardware platforms that handle the execution of instructions within the sensor nodes. Reconfigurable WSN system nodes consisting of microcontrollers interfaced with detachable sensor and RF communication modules makeup the bulk of commercial wireless sensor nodes available in the market. WSN systems built around these processing elements are much easier to design and implement (Leligou, Redondo, Zahariads, Retamosa, Karkazis, Papaefstathiou and Voliotis, 2008). The Libelium waspmote (www.libelium.com) is one clear example of a sensor node built around microcontrollers. The processing element in use is the ATMEGA 128 microcontroller. The waspmotes architecture is modular in design. The intention is to integrate only modules needed for a particular application in each device. These modules can be changed or expanded to accommodate the WSN application's goal. Some examples of the services provided by these modules entails: providing an enabling platform for interfacing an array of sensors; acquisition and storage of data and providing platforms that allow the sensor node to effect transmission and reception services by selectively adopting any of the available RF communication standards (ZigBee/802.15.4, GSM/GPRS and GPS module)

Microcontrollers are most flexible, but they exhibit shortcomings in energy efficiency. Field programmable gate arrays (FPGA) strike an optimal balance between computing power, energy demands and flexibility (Tanaka, Fujita, Yanagisawa, Terada, and Tsukamoto, 2008). As a result, FPGA based processors are emerging as a better option for implementing reconfigurable WSNs at the processing element layer. These types of processors also referred to as soft-core processors make up a class of software defined and alterable processors. Typical examples of soft-core processors are the Microblaze and NIOS II, which are products of Xilinx (www.xilinx.com) and Altera (www.altera.com) respectively. Much work in this direction has been mostly experimental (Compton and Hauck, 2002; Muralidhar and Rao, 2008). However, a good number of commercial products have employed this option though in combination with other processing platforms like Digital Signal Processors (www.libelium.com). Muralidhar and Rao (2008) used FPGA (Cyclone II) from Altera (www.altera.com) to implement a soft-processor, the NIOS. The soft-core characteristic of the NIOS II processor enables the system designer to develop a custom processor core, to handle intended WSN application's requirement. The addition of predefined memory management unit to the NIOS II soft processor allows its basic functionality to be extended. Using the aforementioned technique, the designer can also define new custom instructions and peripherals. Muralidhar and Rao (2008) employed the soft-processor concept in achieving some level of hardware reconfiguration that increases the target system's efficiency and ease of adaptation, notwithstanding the wireless sensor node's small size (Muralidhar and Rao, 2008). In a related work, Khan and Vemuri (2005), using the FPGA processing platform, devised a paradigm that prolongs the battery life of a sensor node by ensuring that the rate of energy usage in conjunction with the task being implemented is efficiently managed.

### 2.4.2 Radio Frequency Transceiver

The use of reconfigurable platforms like Field Programmable Gate Arrays and software-defined radio technology allows transceivers that previously operate on a single radio spectrum to operate on several other spectrums. Software Defined Radios (SDR) involves the software implementation of hardware constituents of a communication system (for example modulators, demodulators, detectors, filters and amplifiers) with an implicit assumption of an analogue to digital conversion close to the antenna (Tuttlebee, 2002).

The term "Software Defined Radio" was used by Joseph Mitola, in his first publication on the topic in 1999 (Yick, Mukherjee, and Ghosal, 2008; Dong, Chen, Liu and Bu, 2010). Software defined radios early development can be traced to the defense sector of both the U.S. and Europe during the 1970s (Tuttlebee, 2002). SDR realisation is largely attributed to the evolution and convergence of digital radio and innovations in software technologies.

In SDR each of the major functions of the radio as depicted in Figure 2.3, which includes the RF transceiver, contain reconfigurable features that can be altered on-the-fly. This reconfiguration process is made possible by a blend of field-programmable gate arrays (FPGAs), digital signal processors (DSPs) and general-purpose processors (GPPs). The suitability of using an ASIC, FPGA, or DSP depends largely on the following: Programmability, Level of integration, Development cycle, and Performance and Power utilisation (http://www.sdrforum.org).

The benefits of SDR taken from SDRforum (http://www.sdrforum.org) are summarily listed below:

- It allows new functionalities to be added to the existing communication infrastructure with ease and at reduced cost;

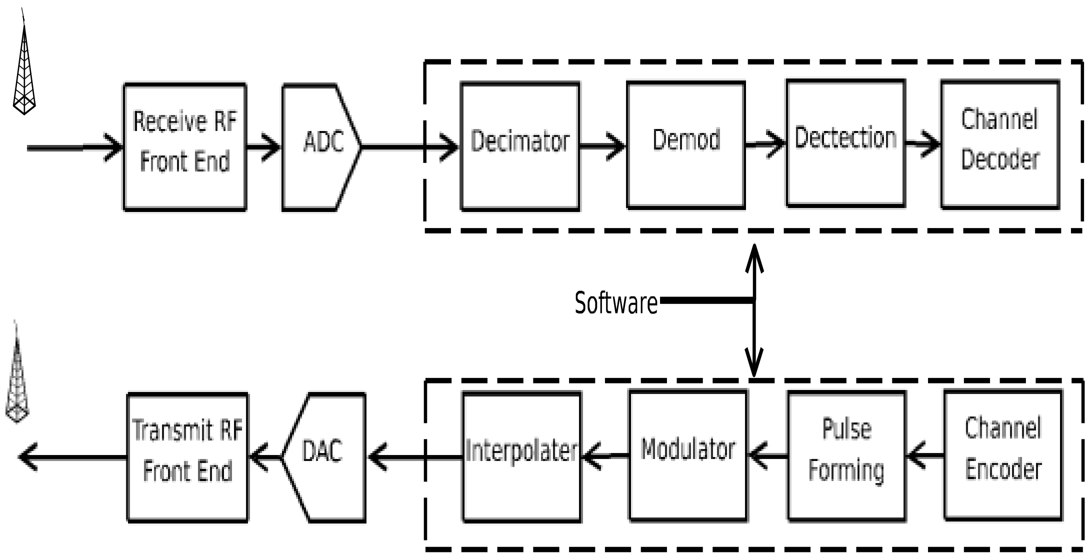


Figure 2.3: A Typical SDR Architecture (http://www.sdrforum.org)

- Service capacity is enhanced via capability upgrades. This is made possible through remote software download;
- Enables end-users have access to ubiquitous wireless communications; allowing ease of communication to any spectrum, whenever and in whatever mode thereby reducing costs;
- Operational and maintenance (real-time debugging via over-the-air remote reprogramming/reconfiguration) time as well as their associated cost can be reduced significantly; and
- It enables a family of radio "products" to be implemented using common platform architecture. This invariably facilitates speedy production to market scenario.

A typical wireless sensor node is characterised by its small size and in most WSN applications the smaller it is, the better. This explains why adopting SDRs for the wireless sensor communication interface can become a challenging task. The challenges stem from application requirements that span over small size and weight, limited power consumption to long battery lifetime. Recent research efforts (Balani, Han, Rengaswamy, Tsigkogiannis, and Srivastava, 2006; Fuentes and Gámez, 2011; Tanaka, Fujita,Yanagisawa, Terada, and Tsukamoto, 2008; Linn, 2009) can accelerate the developing of SDR wireless sensor node. Reviewing some of these works, equally lead to new and inspiring research questions. Cafaro, Gradishar, and Guimaraes (2009) reported a flexible integrated circuit transceiver operating from 10MHz to 4 GHz and having a dimension of 5.0 mm x 5.4 mm in 90nm CMOS and housed in a 10mm x 10mm 132-pin dual row Micro Lead Frame (MLF) package. The reported transceiver can handle as many protocols as possible. Considering its relatively small size, it can effectively be adopted for SDR application tailored for wireless sensor node RF front end.

The option of identifying and reducing software overheads in such a way that SDR algorithm can be adopted in wireless sensor nodes was proposed by Linn (2009). Using this approach, Linn (2009) invented an extremely efficient Verilog programming technique that allows the cramming of SDR algorithms into the FPGA.

A good number of SDR development platforms and software tools are now readily available for rapid development of SDR applications. Some of these tools (Universal Software Radio Platform and GNU (Gnu's Not Unix) Radio) have been largely used by both the educational and commercial research bodies in conducting research in this field. The "Lightweight Communications Architecture" or LCA is also being proposed for use on smaller commercial platforms, with land-mobile radio (LMR) systems, which can easily be adopted for wireless sensor nodes (Cafaro, Gradishar, and Guimaraes, 2009).

### 2.4.3 Application

Basic changes at the application layer to suit application needs involve the addition, removal or editing of constants, variables and functions. In some cases the use of compiler directives like '#define' and '#if' are used to selectively bypass the compilation of program codes, modules or library functions in line with the application specifications and requirements. Microchip wireless application programming interface (Miapp) is one good example of a framework that provides an enabling platform for reconfiguring WSN at the application layer (Yang, 2009). Similar other frameworks also exist, and they are referred to as Application Programming Interface (API) (www.libelium.com and www.digi.com).

One of the primary objectives of the MiApp is to provide a communication-programming interface through which the application developer can adopt or implement different WSN communication protocol using appropriate RF transceivers without the need to understand in details the workings of the physical layer or Media Access Control layer (MAC).

The MiApp specification benefits WSN reconfiguration in a number of ways (Han, Kumar, Shea, Kolher, and Srivastava, 2005; Yang, 2009):

- It allows developers to select wireless protocol at any phase of application development with ease;
- As depicted in Figure 2.4, MiApp indirectly communicates with Microchip RF transceivers through the Microchip Wireless Media Access Controller (MiMAC) interface. MiMAC controls the lower interface of the Microchip propriety wireless protocols, while MiApp regulates the higher interface of Microchip propriety wireless protocols. Combined use of both MiApp and MiMAC gives the application developer the flexibility of using different RF transceivers. Each RF transceiver has varied capability in handling known wireless communication protocols.

Support for WSN reconfiguration in MiApp is in two parts. First, it involves the definition of configuration parameters (CONFIG_PARAMETER) within a configuration file (using "#if define (CONFIG_PARAMETER)", "#define C CONFIG_PARAMETER") and secondly, the inclusion of signatures of functions calls to the Microchip proprietary wireless communication protocols. The configuration parameters stipulate among others the requirements (the microntroller hardware resources, peripheral and RF transceiver control pins) to be used and specifies or decides what sections of the entire application source code should be compiled into the firmware hex file using the appropriate compiler directive. Table 2.1 shows selected samples of some configuration parameters.

Reconfiguration at the application layer is only possible at design time. In addition, these changes cannot take effect except the source code is recompiled and redeployed. The flexibility of reuse at this layer during run time is limited.

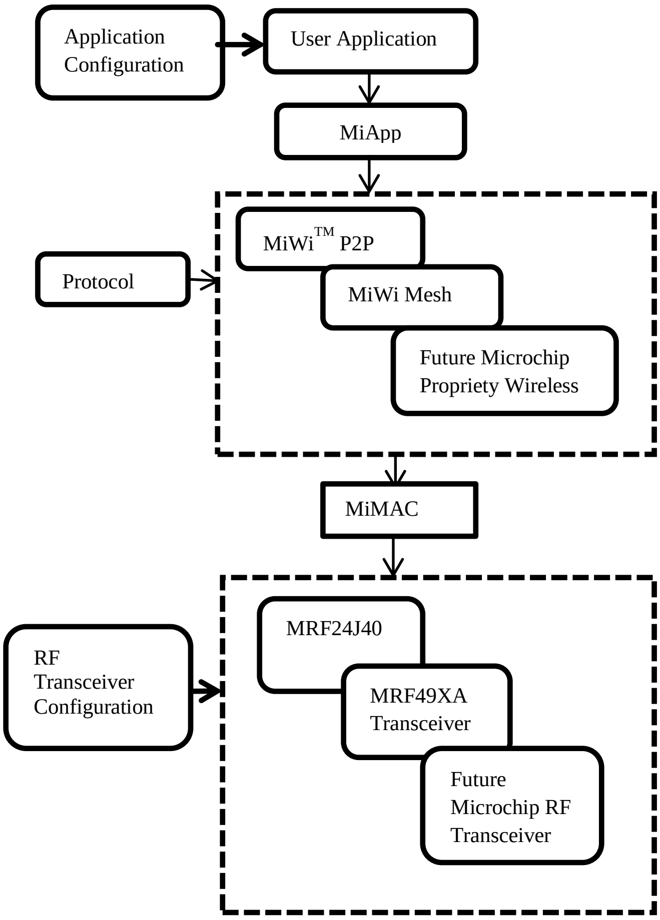


Figure 2.4: Block Diagram of Microchip Wireless (Miwi) Stack (Yang, 2009)

However, this limitation can be overcome if the user can capture the application's real time field experiences in the source code's design and implementation. Steine, Ngo, Oliver, Geilen, Basten, Fohler, and Decotgnie (2011) introduced an approach that exploits design-time knowledge of the application scenario dynamics to construct and implement a proactive runtime reconfiguration paradigm. The issues with this approach are: First, the possibility of capturing all anticipated reconfiguration needs can be challenging and secondly, the scarcely available memory space might not be sufficient to accommodate codes written to address these needs. Moreover, even if it does, there is the likelihood of redundant codes written to handle anticipated changes, which might never occur, and invariably taking up scarcely available memory spaces.

Table 2.1: Selected software definition in the configuration file (Yang, 2009)

| Example of Definition | Functionality | Comments |
|---|---|---|
| #define PROTOCOL_MIWI<br>#define PROTOCOL_P2P | Selects the Microchip protocol to be used in wireless application | Only a single protocol is allowed |
| #define MRF24J40<br>#define MRF49X | Specifies what type of Microchip RF transceiver to employ. | Only a single protocol is allowed<br>The MRF24J40 definition is a related transceiver |
| #define ENABLE_SLEEP | This enables the RF transceiver's sleep mode capability. It is meant to reduce power consumption whenever the system is in an idle state. | The type of transceiver in use determines whether the sleep mode can be activated or not. |
| #define ENABLE_SECURITY | Enables Microchip's propriety protocol which ensures that packets reliability is guaranteed | The main components (security engine) and attributes(security mode and keys) are defined within a specification file meant for every RF transceiver, |

### 2.4.4 Middleware

A middleware is a software abstraction layer that exists between operating systems and applications. It is meant to simplify operations, enable heterogeneity and masks the basic hardware or software layers of sensor nodes (Graziosi, Pomante, and Pacifico, 2008). Also, they provide some degrees of abstraction of communication networks, operating systems, programming languages and management of distributed applications, via the use of API that encapsulates the access to the underlying mechanisms (Alkhawaja, Ferreira and Albano, 2012). Middleware implementation in distributed systems entails (Graziosi, Pomante, and Pacifico, 2008; Puder, Romer, and Pilhofer, 2006; Myerson, 2002) the following:

- Application manageable data representation and codification;
- Remote processing and monitoring;
- Open system interconnect (OSI) protocol compliant; and
- Location transparency (effect communication with distributed systems devices by using the middleware capabilities, and suited to offer Quality of Service to the application layer).

These implementation paradigms allow mobile distributed systems to have context-aware capabilities. Implementing these traditional middleware functionalities in WSN can be challenging because of the constraints (limited processing and energy resources) associated with sensor nodes. As such, middleware implementation for reconfigurable WSN is required to be of a lightweight type (Graziosi, Pomante, and Pacifico, 2008). Few work done in this area has been published in notable literatures (Gámez, Cubo, Fuentes and Pimentel, 2012; Graziosi, Pomante, and Pacifico, 2008; Hu, Ndulska, and Robinson, 2006; Kjær, 2007). Graziosi, Pomante, and Pacifico (2008) presented a middleware-based approach for WSN, which enables WSN to transport data across heterogeneous networks. It also offers a homogenous API (*Application Programming Interface*) for the related applications development.

Hu, Ndulska, and Robinson (2006) implemented a dependable context management system at the middleware layer that dynamically locates and replaces failed sensors or network based on context information derived from context sensing sources. The system, as illustrated in Figure 2.5, is composed of the following layers:

- *Context-aware applications layer* - context information are obtained, analysed and appropriate decisions taken at this layer to adapt the system to evolving context.
- *Reconfigurable context management layer* – composed of several components meant to store and evaluate context information according to the context models and broadcasts this information through responses to queries and/or context changes; and
- *Context sensing layer* – made up of context sources (sensors) and possibly related processing components to transform acquired context data into context information required by the application.

The intentions of Gámez *et. al.,* (2012) were tailored towards implementing a context aware architecture that can easily be adapted to several platforms via the use of a model-driven configuration process approach. The model is designed to integrate new contexts to the FamiWare family (Fuentes and Gámez, 2011) by producing context-aware versions of the middleware for every application. The FamiWare, a family of middleware for Ambient Intelligence is designed to be aware of contexts in sensor and smartphone devices. It provides several monitoring services capable of acquiring contexts from devices and users alike. In addition, it integrates a context-awareness service that analyses and detects context changes as well.

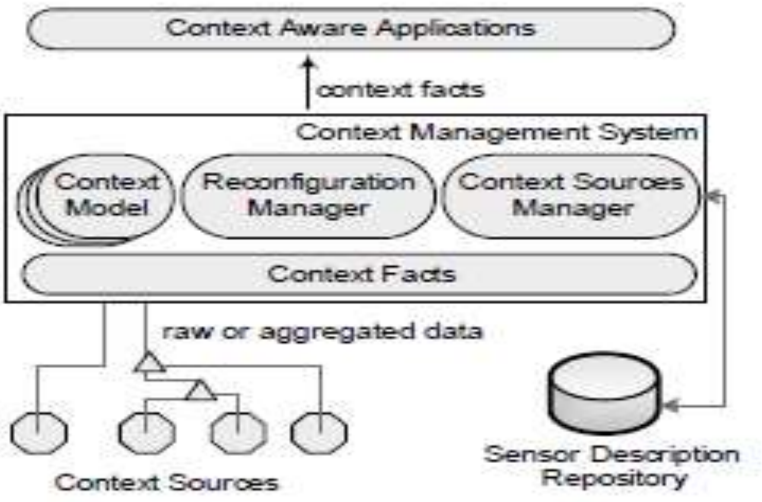


Figure 2.5: Reconfiguration Architecture (Hu, Ndulska, and Robinson, 2006 )

### 2.4.5 Operating System

Jun-Zhao (2010) categorises WSN reconfiguration paradigm in operating system (OS) in terms of how much alteration is done to the original source code. The phrase 'alteration' refers to the addition of new codes, removal of existing codes or editing of existing codes. This invariably means the introduction of a new task, the removal of no longer needed task or an enhancement of an existing task respectively. Jun-Zhao (2010) classified the OS reconfiguration approaches into three groups: the *complete code image replacement* scheme, *loadable module* scheme and the *difference* scheme. The complete *code image replacement* scheme involves overwriting the entire code memory of a wireless sensor node with a new firmware. Examples of these implementations are XNP (Jeong, Kim and Broad, 2012), Trickle (Levis, Patel, Culler and Shenker, 2004), Deluge (Hui and Culler, 2004), Stream (Krishna, Bagchi, and Khalil, 2009) and Mate (Levis, and Culler, 2002). The *Loadable module* approach effects changes at the modular level; this also means the OS framework is modular in setup. It also allows for the addition and removal of new application task packaged in modular form. However, use of large memory space and demand for more processing time (which invariably translates to higher power consumption and slows system execution) are drawbacks associated with the loadable module based approach. The *Difference-Based* approach specifically overwrites the identified difference between the original and the modified file. In addition, only the delta (the difference between the old and updated program) generated at the base station is transmitted to the terminal nodes in the field. This invariably reduces the amount of data needed to be transferred especially when only small changes are involved.

Each paradigm has its advantages and disadvantages. The reconfiguration paradigms and their related challenges as implemented in four selected OS (namely TinyOS, Contiki, Sensor Operating System and MANTIS) are presented in Tables 2.2 and 2.3.

#### 2.4.5.1 TinyOS

Reconfiguration and dissemination schemes implemented using the TinyOS component architecture exists. Some of the paradigms implemented adopt either the *Entire Image replacement* approach (Hill *et. al.,* 2005; Jeong, Kim and Broad 2012; Levis, and Culler, 2002) or the *Difference-based* approach (Krishna, Bagchi, and Khalil, 2009).

The various successive schemes implemented on the TinyOS platform over time stem from attempts to improve upon the challenges associated with their predecessors. Some of these challenges span over performance issues, memory and energy management related issues. Some of the key improvements inferred while reviewing the trend of development and implementation of the various schemes are relayed thus: Starting with the XNP (Jeong, Kim and Broad, 2012), this scheme was primarily designed to function as a single-hop reprogramming protocol. XNP performance suffers some defects resulting from overheads when making request directly from the base station. However, Trickle (Levis, Patel, Culler and Shenker, 2004) addressed this defect by implementing the first multi-hop code dissemination protocol. Trickle has a limitation of only being able to transmit the update-codes in small size patches. This shortcoming again was addressed by the introduction of Deluge (Hui and Culler, 2004). Deluge, an extension of the Trickle Protocol improved upon its predecessor by being able to effect bulk transfer at a reduced transmission time using pipelined data transfer technique.

Deluge employs the *complete image replacement* approach and it transmits the actual binary codes (firmware) during every code update. Thereby causing a large number of energy hungry memory (EEPROM and or flash program) writing to transpire. Concerns about energy demands by the Deluge protocol subsequently leads to the evolution of newer OS-based reconfiguration paradigm. Few examples like the Contiki and SOS were fashioned after the *Loadable module* approach while others like the Zephyr (Krishna, Bagchi, and Midkiff, 2009) and FlexCup (Marron, Gauger, Lachenmann, Minder, Saukh, and Rothermal, 2006) implemented the *Difference-based* approach. More discussions on the *loadable module* approach were presented in section 2.4.5.3 while discussing the contiki and SOS OS platforms.

Under the *Difference-based* approach, most algorithms employed to detect and construct deltas for dissemination and reconstruction within wireless sensor nodes differ in their mode of operation. A typical algorithm in use is the Rsync and its variants. Rsync and the corresponding RDIFF algorithm (Tridgell, 1999) use non-overlapping fixed-sized blocks for matching indistinguishable data between the modified and original files. Both files are segmented into blocks, and for each one, a rolling-checksum and an MD5 (a message-digest algorithm based on a cryptographic hash function that produces a 128-bit hash) checksum are computed. Using these checksums, the delta is constructed of either reference to blocks that already exist in the old version, or the entire content of new or changed blocks. While the rolling checksum is implemented to be as fast as possible, an MD5 checksum is not appropriate for sensor nodes. The apparent flaw of the algorithm is that if two blocks differ in even one byte, the entire block content has to be present in the delta. The sensor nodes perform expensive MD5 computation for each block of the binary image when the algorithm is utilised for differential reprogramming. In addition, a study on the limitations of the MD5 for which most variants of the Rsync are based on reveals the following:

i. Xiaoyun and Hongbo (2005) show that MD5 is not collision resistant.

ii. A group of researchers created a pair of files that share the same MD5 checksum (Black, Cochran and Highland, 2008).

iii. CMU Software Engineering Institute reportedly declared that the MD5 should be considered cryptographically broken and unsuitable for further use (‘‘CERT Vulnerability Note VU#836068’. Kb.cert.org. Retrieved 9 August 2010.)

iv. The Flame malware exploited the weaknesses in MD5 to fake a Microsoft digital signature (‘NIST.gov-Computer Security Division-Computer Security Resource Centre’. Csrc.nist.gov. Retrieved 9 August 2010).

Milosh, Cuijipers and Lukkien (2013) modified Rsync such that all the expensive operations regarding delta script generation are performed on the host computer and not on the sensor nodes. In addition, it ensures that the expensive MD5 computation is done only when the inexpensive checksum matches between the two blocks (Milosh, Cuijipers and Lukkien, 2013). If no matching block is found then the algorithm moves to the next byte in the new image and the same process is repeated until a matching block is found. While the probability of collision is not negligible for two blocks having the same checksum, with MD5 the collision probability *is* negligible (Milosh, Cuijipers and Lukkien, 2013). To ensure the correctness of the scheme in the rare case when two different blocks have the same MD5 hash, Zephyr (Krishna, Bagchi, and Midkiff, 2009) performs a byte-by-byte comparison when MD5 hashes match (Milosh, Cuijipers and Lukkien, 2013). A byte–by-byte comparison is deficient when dealing with machine codes generated for execution on a microcontroller. Physical addresses of data locations always differ whenever changes occur in the new image file. Having a common reference point for the purpose of comparison becomes a problem

#### 2.4.5.2 Sensor Operating System

Sensor Operating System (SOS) is composed of dynamically-loaded modules and a common kernel that implements messaging, dynamic memory and module loading and unloading. SOS improves on the XNP energy usage by using modular updates instead of full binary system image and does not require rebooting the node after installing an update (Han, Kumar, Shea, and Srivastava, 2005). It also installs updates directly into program memory without costly external flash access.

#### 2.4.5.3 Contiki

Contiki is designed to support dynamic loading and replacement of individual application programs and services. It is developed around an event-driven kernel with optional support for pre-emptive multithreading. Implementing basic routines as services allow the system to effect reconfiguration at run time. Very important services like the communication routines, which exist in stacks, can be loaded simultaneously.

Dynamic loading is an effective way to make sensor nodes take up new functionalities. The approach disseminates loadable modules, which are relatively much smaller compared to entire application image. The modular design approach can effectively reduce the transferred code size, thereby reducing the amount of energy expended during network reprogramming. The files are loaded in Execution Linking Format (ELF). The ELF ranks among the most widely used object code format for dynamic linking. It is composed of program code, data and supplementary details such as a symbol table, the names of all external unresolved symbols, and relocation tables. The relocation tables provide information on where the program code and data can be placed in memory other than where they were originally meant to be during assembly. One problem with the ELF format is the overhead in terms of bytes to be transmitted across the network when compared to pre-linked modules. Modular design has other benefits (other than reduce data size for efficient reprogramming) like making code reuse easier to handle.

#### 2.4.5.3 Mantis

The Mantis OS employs the traditional concept of preemptive multi-threaded model. Reprogramming of the entire operating system and parts of the program memory is feasible. It employs the locking mechanism, which mutually excludes shared variables while allocating stack spaces to its program (Dong, Chen, Liu and Bu, 2010). The dynamic reprogramming capability of the Mantis OS is implemented as a system of call library that are built into the Mantis OS kernel (Bhatti *et. al.*, 2005) Applications can make changes to the new code image via the library. These changes are then implemented on system reset using a bootloader and a called function *Commit* (Bhatti *et. al.*, 2005).

Table 2.2: Reconfiguration features, approaches, impact and comparative advantages for TinyOS and SOS

| Operating System | Scheme/ Protocol | Sensor Nodes where deployed | Energy Management Related Issues | Performance/Memory space related Issues | Comments: Comparative Advantage Recommendation |
|---|---|---|---|---|---|
| TinyOS | Deluge<br>Disseminates large data objects (binaries) to many nodes in WSN using multi-hop dissemination protocol.<br>Combining the above mechanism with a bootloader and command dissemination it<br>Build around an event-driven kernel | Mica2, Mica2-dot ,MicaZ,Telos, Tmote Sky, Eyes, Tinynode, IRIS | Much energy required for transmitting entire image Hence, much processing needed for flash writing. | Its transmission time is much faster because it uses pipelined data transfer | Comparatively efficient when an entire code or application needs changed completely.<br>Not suitable for updating small changes. |
| | Zephyr<br>Implements incremental/Differential reprogramming.<br>**The goal is to transfer small details (difference between the old and the new software), thereby minimising reprogramming time and energy.** | Mica2 | less energy required for transmitting patches therefore<br>Less processing needed for flash writing. | Depending on the algorithm employed, the transmission of large number of small differences spread over the entire code can be disadvantageous.<br>Transmission cost resulting from overheads is very high | less energy required for transmitting patches<br><br>Less processing needed for flash writing. |
| Sensor Operating System [SOS] | Uses modular approach. Each module has a defined entry and exit point.<br>The modules are designed in a loosely coupled manner.<br>Interactions between modules are effected via message passing, direct calling of registered functions within modules or kernel system's calls.<br>Build around and event-driven kernel | Mica2, MicaZ, TelosB, Tmote Sky | Moderate in comparison to TinyOS | Less safety features to address missing and updated modules. | Remotely insert binary modules into running kernel without interrupting system operation. Reboots not needed as in differential patching. |

Table 2.3: Reconfiguration features, approaches, impact and comparative advantages for Contiki and Mantis.

| Operating System | Scheme/ Protocol | Sensor Nodes where deployed | Energy Management Related Issues | Performance/Memory space related Issues | Comments: Comparative Advantage Recommendation |
|---|---|---|---|---|---|
| Contiki | First to support modular update and consists of two main components: system core and loaded program<br><br>Build around and event-driven kernel<br><br>Implements a dynamic linker that links, relocate and load either standard ELF files or CELF (compact ELF) files. | ESB,TelosB, Tmote Sky | Less energy as only specified module are transmitted<br><br>Processing overhead arising from a number of book keeping tasks to resolve cross referenced symbols required to link and load new module | The modules are designed in a loosely-coupled manner and communicating only via the kernel.<br><br>Dynamic linking and loading causes performance degradation<br><br>. | Scheme should be able to estimate the percentage of code that need to be modified. And if more than a specified threshold (suggesting near entire image size) then opt out of loadable modular approach<br><br>Extra storage needed for keeping track of the symbol table. |
| Mantis | Achieves dynamic reprogramming on several granularities<br><br>Re-flashes the entire OS.<br><br>Able to reprogram a single thread and make changes to variables within a thread. | Mica2, Eyes, Telos, Mantis nymph | Employs a power efficient scheduler that puts the microcontroller to sleep in response to reconfiguration handling-threads calls to the sleep() function. Thereby reducing current consumption to the micro-ampere range. | It maintains two logically distinct sections of RAM: Global variables that are allocated at compile time while the rest of the RAM is managed as a heap.<br><br>It implements dynamic memory management scheme. However, it also results in a lot of overheads | Its multi-thread driven capability allows for priority-based scheduling and preempting of task execution |

## 2.5 Application of Artificial Intelligence to WSN related Issues

Artificial Intelligence (AI) is the study of adaptive mechanisms that enable or facilitate intelligent behaviour in complex and changing environments (Venayagamoorth, 2009; Engelbrecht, 2007). These mechanisms involve paradigms that exhibit the capacity to learn or adjust to new situations, to generalize, abstract, discover and associate (Kulkarni, Forster and Venayagamoorthy, 2011). AI encompasses paradigms such as Artificial Neural Networks (ANN), Reinforcement Learning (RL), Swarm Intelligence (SI), Genetic Algorithms (GA), Fuzzy Logic (FL) and Artificial Immune systems (Kulkarni, Forster and Venayagamoorthy, 2011). Brief descriptions of popular AI paradigms applied to WSN problems are concisely presented in the following subsections. In some cases, hybrids of these paradigms do exist. Notable examples of these combinations are neuro-fuzzy systems and fuzzy-immune systems.

### 2.5.1 Artificial Neural Networks

The Artificial Neural Networks (ANNs) is modelled after the human brain known to have an astonishing capacity to learn, remember and simplify complex issues. It is a network of more than ten billion neurons; each neuron is joined to approximately ten thousand other neurons. The neuron receives signals through synapses. The synapses regulate the effect of the signals on the neuron thereby playing an important role in the performance of the brain (Haykin, 1994). Figure 2.6 and 2.7 shows an artificial neuron and a popular ANN architecture respectively. It is made up of three constituents: one, the links that provide weights $W_{ji}$, to $n$ inputs of $j^{th}$ neuron $x_i$, $i = 1, \dots, n;$ two, an aggregation function that produces $u_j$, a summation of $\Theta_j + \sum_{i=1}^{n} x_i W_{ji}$, where $\Theta_j$ is the bias; and thirdly, an activation function $\Psi$ that maps the output $\Psi(u_j)$ to $u_j$.

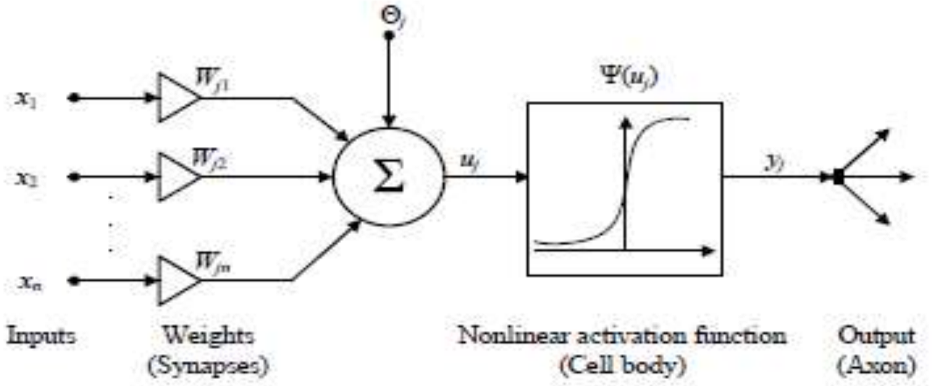


Figure 2.6: Structure of an Artificial Neuron. (Kulkarni *et al.*, 2011)

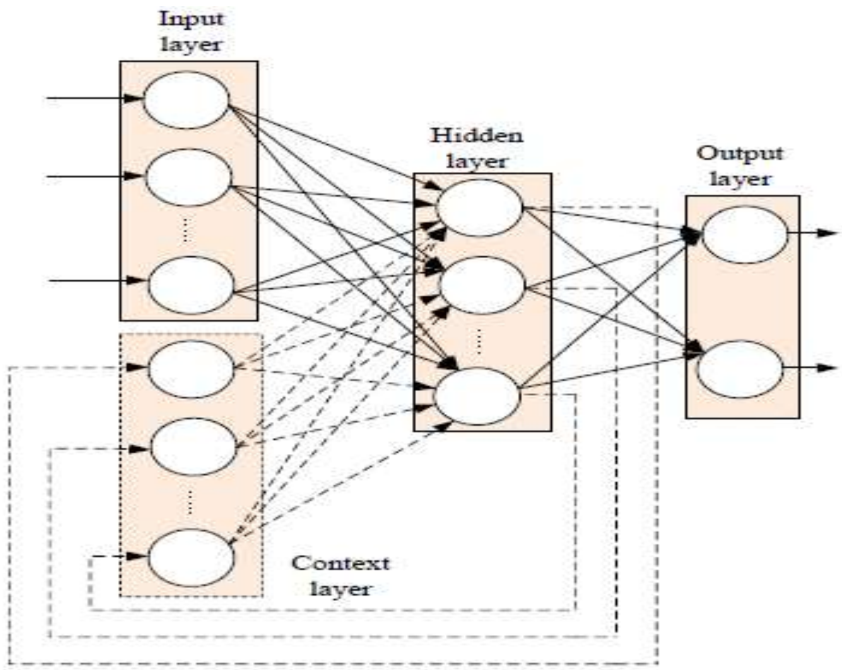


Figure 2.7: Popular ANN architectures: The connections shown in solid lines and the context later make up a feedforward NN. Addition of the connections shown in dotted lines converts it into a recurrent neural network. (Kulkarni *et al.*, 2011)

ANNs learn the facts characterised by patterns and deduce their inter-relationships. Learning approaches are via supervised learning, unsupervised learning, or reinforcement learning. Successful applications of ANNs are found in power system stabilization, image processing, speech recognition, and prediction related problems.

### 2.5.2 Genetic Algorithm

Genetic algorithm (GA) implementation is based on a search algorithm that tends to proffer solutions to AI problems using the natural selection approach. It starts with a simple potential solution and evolves toward a set of more ideal solutions. In the course of progressing toward the best solution, it excludes those solutions that are less result oriented, while superior solutions are combined and their beneficial traits proliferated, thereby allowing more solutions into the set, which subsequently facilitate better potentials. In order to avoid stagnation occurring in the process, random mutation are carried out to replace the several replicas of identical solutions. In order to use genetic algorithms efficiently, the under listed conditions need to be met:

- The system should be able to appraise how 'good' a prospective solution is relative to other would-be solutions with ease.
- The system should be able break a potential solution into separate portions ('genes') that can vary independently.
- Lastly, genetic algorithms are well-matched for situations where a 'good' solution is viable and might not necessary be the absolute best solution.

The operation of the genetic algorithm entails the following:

- **Reproduction**: The process of duplicating a prospective solution;

- **Crossover**: The process of exchanging gene values between two prospective solutions, mimicking the "mating" of the two solutions; and,
- **Mutation**: The process of arbitrarily varying the value of a gene in a prospective solution.

### 2.5.3 Fuzzy Logic system

The Fuzzy Logic (FL) model is empirically-based. It relies on operator's know-how and little attention is given to the working details of the system. Fuzzy Logic System is compose of four components: These are namely the fuzzifier, adopted rules based on expert knowledge, an inference engine, and defuzzifier.

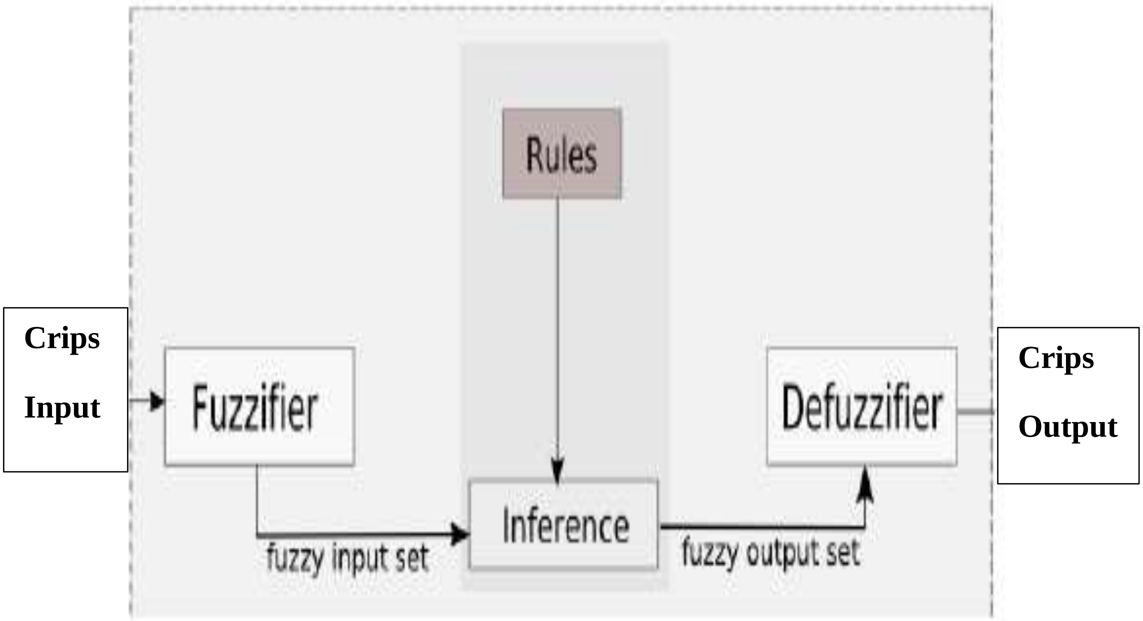


Figure 2.8: A Fuzzy Logic System

FL is inherently robust because it does not need precise, noise-free inputs. It produces a smooth output control even when the input variations are notably wide. Interestingly, it allows the designer to implement a fail-safe option should in case a major critical component of the system fails. FL controller processes user-defined rules that can be modified and tuned easily to improve system performance. It allows for easy integration of new sensors and subsequent modification of existing rules to accommodate the update.

The rule-based operation allows any rational number of inputs to be processed, and copious outputs generated. However, an increase in the number of input and output could result in a complicated rulebase formation. Fuzzy Logic system has been found very useful in controlling nonlinear systems that are mathematically demanding to model.

To use fuzzy logic approach, the following steps is recommended:

- The control objectives and criteria need to be defined.
- Infer the number of input and output requirement and their relationship within the context of the system's goals.
- The control problem should be broken down to a chain of ' IF X AND Y THEN Z' rules that define the anticipated system output response for given system input settings.
- Produce Fuzzy Logic membership functions that state the values of Input or Output terms employed in the rules.
- Generate the needed pre- and post-processing Fuzzy Logic functions for Software or Hardware implementation.

- Lastly, setup a Testbed to examine the system, appraise the results, alter the rules and membership functions, and retest the system again until suitable

## 2.6 Choice of AI Solution for WSN related Issues

AI provides adaptive mechanisms that exhibit intelligent behaviour in complex and dynamic environments like WSNs. AI brings about flexibility, autonomous behaviour, and robustness against topology variations, communication failures and scenario changes (Kulkarni, Forster and Venayagamoorthy, 2011).

Artificial intelligence (AI) Paradigms have been employed as tools to handle several WSN problem areas. Notable among these areas are efficient management of data collection and fusion activities, optimal localization and energy ware routing. However, not much has been reported in WSN reconfiguration related issues. Many AI methods have outperformed or complimented conventional methods under uncertain environments and severe limitations in power supply, communication bandwidth, and computational capabilities.

Kulkarni, Forster and Venayagamoorthy (2011) surveyed some WSN application areas where some of the AI techniques earlier mentioned in the preceding section were used. The outcome of their findings is shown in Figure 2.7. The findings are intended to serve as a guide for selecting the most appropriate AI approach to explore or adopt when solving WSN related problems. Though WSN reconfiguration related issues were not mentioned, some of the problem areas surveyed (for example Deployment, Routing, Data Aggregation, Fusion and Quality of Service management (QoS)) share some operational characteristic with it.

Figure 2.9 depicts a table that is composed of columns and rows representing the surveyed WSNs application areas and the main AI techniques employed respectively. The number of articles surveyed for a particular combination of WSN problem and the adopted AI approach was symbolically represented by the size of black circles. Moreover, the cells were equally hashed to indicate which AI is most suitable and applicable for the problem in question. The evaluation is rather an estimate, since the actual outcomes depend on the nature of the problem, the AI algorithm employed, and the parameters used. Also, most researchers rarely evaluate their algorithms under real WSN environments like test-bed or in the field.

The findings presented by Kulkarni *et. al.,* (2011) indicates that Design and deployment is usually a centralized problem, where an optimal architecture for the WSN to be deployed is determined. AI models like ANNs, and GAs are very well suited for that purpose. They can produce optimal results from large datasets where memory and processing restrictions do not apply. For localization, it looks like ANNs and GAs are the best suited techniques, although they need to be used in a centralized manner. The problem is the high variance of the localization data, for example, using RSSI values to compute distances between nodes. Fuzzy logic is well suited for security and QoS problems. It is able to compute general non-optimal rules that can accommodate larger variance of the data, as in case of security applications. Routing and clustering seems to be the most popular WSN problem for applying AI methods (in fact, it is also a very active research area in general). However, not all AI methods are equally suited. ANNs and GAs have very high processing demands and are usually centralized solutions. In the case of ANNs, learning can also be conducted online at each of the nodes, but is slow and has high memory requirements. These two AI approaches are slightly better suited for clustering when the clustering schemes can be pre-deployed. Fuzzy logic is very well suited for implementing routing and clustering heuristics and optimizations, like link or cluster head quality classification. However, it generates non-optimal solutions, and fuzzy rules need to be re-learnt upon topology changes.

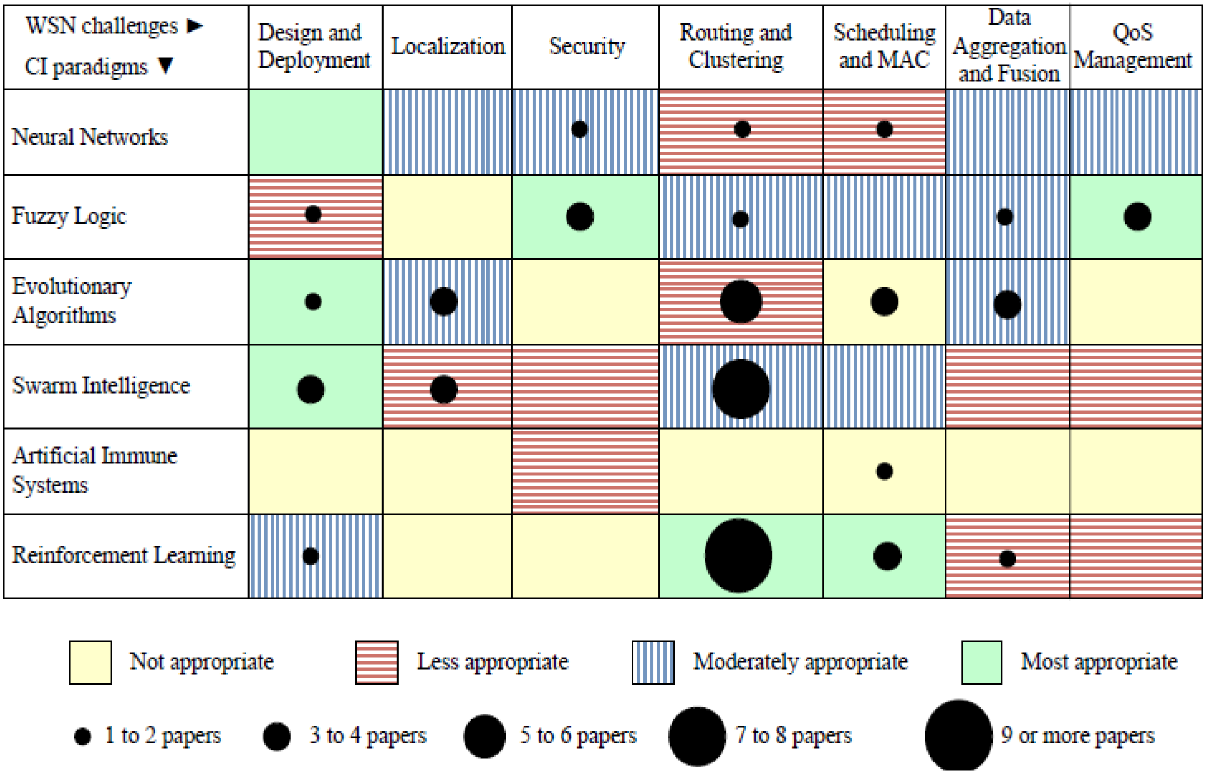


Figure 2.9: An overview of WSN challenges and the AI paradigms applied to address them (Kulkarni *et al.*, 2011)

When dealing with data aggregation and fusion, the best suited AI methods are fuzzy logic, evolutionary algorithms and neural networks. It is interesting to note that two AI techniques, ANNs and AIS, have been rarely applied to WSNs. This is predominantly awkward in the case of ANNs, because this paradigm is very well studied and there exist many different ANN models with different properties.

Based on previous studies on OS related reconfiguration approaches, reconfiguration paradigm are very likely to thrive more when the algorithm in use can handle routing efficiently and places less demand on complex computation and high demand for memory space. Hence, the choice of AI technique to use must conform to these requirements.

## 2.7 Summary

In this chapter, various research approaches adopted towards realising fully reconfigurable WSNs under severely constrained resources were discussed. Reconfiguration at the application layer is only possible at design time. In addition, these changes cannot take effect except the source code is recompiled and redeployed. The flexibility of reuse at this layer during run time is limited.

A review of the existing reconfiguration schemes under the OS approach shows that the difference approach method is more promising when compared to others (Misra and Eronu, 2012). A review of the Difference based approach reveals that in some cases, instead of smaller deltas being generated, larger ones were rather produced (Misra and Eronu, 2012). A problem largely attributed to the use of traditional differential utilities like Rsync employed in (Bert and Weiss, 2009), Longest Common Sub-sequence (LCS) employed in (Apostolio, 1986) and Clone Detection in (Burd and Bailey, 2002). These utilities were inherently not designed to handle file structures well-matched for sensor network data transmission and dissemination. In order to mitigate the aforementioned shortcomings, this research work has devised a Precise Delta Extraction model for use in reprogramming or reconfiguring wireless sensor nodes. The scheme is intended to reduce energy consumption rate, as well as effect a reduction of memory space used during reprogramming processes. In addition, it also serves as a metric utility software for measuring the degree of changes made in modified application source codes and relaying the exact changes. This information can also be fed as input to a fuzzy logic controller, which can guide a WSN in deciding the best reconfiguration approach to adopt under certain defined application or operational context

Among the three AI algorithm understudied, the Fuzzy logic is adjudge to be the most suitable to adopt. A critical look at the other two approaches (ANN and GA), suggest a high degree of complication could arise during real life application or implementation. The reasons as reported in (Kulkarni *et al.*, 2011) clearly discourages the adoption of the ANN and GA in most WSN applications. In WSN reconfiguration scenarios, memory space, high computational processing demands as well as the associated energy consumption is not that readily available. Hence, Fuzzy Logic is the most appropriate AI technique to use. It is inherently robust and it processes user-defined rules that can be modified and tuned easily to improve system performance. It allows for easy integration of new sensors and subsequent modification of existing rules to accommodate the update.

# CHAPTER THREE

## 3.0 RESEARCH METHODOLOGY

This chapter discusses in detail the design, development and evaluation procedures employed in realising the Context-based WSN reconfiguration software system. Two software components were designed and developed using software modelling tool. The two components are the Precise Delta Extractor (PDE) and the Fuzzy logic Controller. The first provides information on the degree of changes resulting from the modification done to the application firmware as well as relaying the exact changes in bytes form. In addition, it provides fuzzified member set inputs (application context) to the fuzzy logic controller. The second component developed is based on expert knowledge of the energy consumption constraints associated with reprogramming procedures of wireless sensor node's program memory. In addition, background information on program memory reprogramming constraints and a devised algorithm to address these constraints were presented.

A testbed made-up of an ad hoc network of three 32-bit processor based sensor nodes (PIC32MX320F128H architecture and the MRF24J40B Zigbee based transceiver) was used to provide some pilot data. The test applications were developed in the Contiki operating system. The pilot data were then used to test the efficacy of the context based reconfiguration software at a much larger scale using the OMNeT++ and Castalia WSN simulation platform. Details of these procedures are presented in subsequent sections of this chapter. The order and nature of results to be obtained and analysed were also relayed in each subsection where appropriate.

### 3.1 Context-Based Reconfiguration System Design

This section presents the design of a model that utilise context information to improve upon WSN reconfiguration processes.

#### 3.1.1 Deriving a Context-Based Reconfigurable Model

A well-developed context information model, which entails gathering, evaluation and maintenance of context information, can be expensive. Hence, provision for context information re-use and sharing should be part of the application's planning phase (Bettini *et al.,* 2010). Based on the survey work carried out on the various identified WSN reconfigurable components, context related information relevant to WSN reconfiguration can be classified into two main categories. These entail the following: Application related context information and Operational-demands related context information. The model allows the user to associate selected context information to sensed application data. Thereby allowing every sensed datum have a history of related surrounding activities (defined contexts) for further analysis. Reconfigurable WSN implementation at the various layers earlier highlighted in the preceding sections can easily be maintained and improved upon when relevant contextual information are modelled into the target system's design and development processes.

The essence of the model is to associate selected context information with sensed application data. In addition, it allows every node in the network to build up a history of related surrounding activities (defined contexts). This information is further analysed and used to guide the entire system in taking decisions that are beneficial to its operation.

##### 3.1.1.1 Application related Context Information

WSN application related contextual information are primarily the key driving factors behind reconfiguration needs. They determine what other contextual information will be needed, acquired, analysed, evaluated or probably stored in the systems database. The selection of sensor types is a function of the WSN application goals.

WSN applications' source codes and the resulting firmware when compiled can be viewed as a set of bytes/words. Hence, it can be argued that the composition of any sensor node's application firmware is a reflection of the type of sensor it employs and the related functions assigned to it. Any change in sensor type or mode of usage will also translate into a corresponding change in the number and orientation of bytes contained in the firmware. In line with this proposition, it is feasible to measure changes reflected in modified firmware (result of the reconfiguration process) and relayed them as a source of the application-related context information. The metrics derived for extracting application-related context information is presented in section 3.1.2.

Indulska and Sutton (2003) classified sensor types into three categories, namely: physical sensors, virtual sensors and logical sensors. Table 3.1 shows the context types, sections and reconfiguration layers classified under the aforementioned categories.

Table 3.1: Classification of Sensor types (Indulska and Sutton, 2003)

| Categories | Context Type | Available sensors | Layers where Reconfiguration is feasible |
|---|---|---|---|
| Physical | Temperature<br>Light<br><br>Visual<br>Audio<br>Motion acceleration<br>Position/Location | Thermometers<br>Photodiodes, colour sensors, IR and UV-sensors<br>multimedia cameras<br>Microphones<br>Accelerometers, motion detectors<br><br>Outdoor: Global Positioning System (GPS), Global System for Mobile Communication(GSM)<br>Indoor: Radio Frequency Identification(RFID), Received Signal Strength Indicator(RSSI) | Application Layer |
| Virtual (Use of software at various reconfigurable layers to deduce impact of ) | Energy | - Inbuilt mechanism to measure energy consumption at various layers via experimentation. | Processing Element Application via Communication Layer |
| | Memory | - Space measured as memory space taking over | Operating System |
| | Performance | - Code execution timing | Processing Element Application via Communication Layer |
| | Reliability | - Transmission issues | Processing Element Application via Communication Layer |
| Logical | Use of logical operators (AND, OR) state inputs of either physical or virtual sensed data, decisions resulting | - Software based | Application Layer |

### 3.1.1.2 Operational-Demands related Context Information

Context classification of this type is rarely mentioned in the literature. Operational-demands refer to issues or factors arising from reconfiguration approaches that can affect the performance or efficiency of a WSN application. In most cases, they also constitute the metrics for evaluating the effectiveness of the reconfiguration approaches employed. Examples of this context information types are:

i. Energy usage and management issues;

ii. Memory utilisation; and

iii. Performance related issues (speed, reliability, efficiency and others)

When designing or implementing reconfiguration processes, it is expedient that energy consumption and related issues are managed effectively in order to enhance the operational life span of the individual sensor nodes, as well as the entire network.

There is a need to strike a balance between the operational constraints or demands and application goals in a dynamic way. Hence, a constant feed of operational context information is necessary. Likewise, a measure of its impact (whether positive or negative) on application goals can be helpful in optimising WSN performance.

#### 3.1.1.3 The Model Description and Implementation

The Context-based WSN reconfiguration model as depicted in Figure 3.1 is composed of three key main layers or levels: the context sensing layer, lower context management layer and higher context management layer. The model provides a platform for deriving Data Frames (Structural and Descriptive Metadata) that associates all relevant context information with the primary context data for the purpose of analysis, control, and storage purposes. In addition, it empowers the WSN with autonomous capability to respond to evolving application changes using any known artificial intelligent technique. Table 3.2 summarily describes the parameters and the associated symbols used in the model.

**Context sensing layer:** At this level, the following: context identification, definition, sensing are implemented. In some cases, there might be need to derive context information by processing non-quantifiable context data. These contexts could be any of the two types earlier mentioned (Application related or Operational-demand related). The layer acts as a presentation layer by making available usable context information to the next higher layer "Low Context Management level."

Table 3.2: Description of proposed Models parameter and associated symbols

| Symbol | Description | | |
|---|---|---|---|
| $\boldsymbol{x_j}$ | Application related context information - physical, virtual(goals, objectives, ) | | |
| $\boldsymbol{y_i}$ | Operational-demands related contextual information | Energy<br>Memory<br>Performance | |
| $\boldsymbol{D^L_{n,i}}$, | Determinant used at **Lower Context Management Level** to decide whether a context information should be used in this layer or not. The "L" superscript denotes the determinant's level of application | | |
| $\boldsymbol{D^H_m}$ | Determinant used at **Higher Context Management Level** to decide whether a context should be used at this level. The "H" superscripts denote the determinant's level of application | | |
| $\boldsymbol{Z_s}$ | Logical context derivation used as switching element at context sensing level with the aid of $D^L_{n,i}$ | | |
| $\boldsymbol{Z_n}$ | Logical context derivation used as switching element at lower context management level with the aid of $D^L_{n,i}$ | | |
| $\boldsymbol{T}$ | Final collection of data and associated Contexts | | |
| $\boldsymbol{T(t,p)}$ | The final collection of data and associated Contexts expressed as function of time and location | Time($t$) | - Synchronization<br>- Aid metrics assessments<br>• Qualities<br>• Quantity |
| | | Location/Position ($p$) | - Adaptation<br>- Performance per location assessment |

**Lower context management layer**: This layer selectively (via the use of switching elements $D^L_{n,i}$ as depicted in Figure 3.1 and Figure 3.2) accepts raw data ($x_j$) or processed context information ($y_i$) from the sensing layer and logically combines them in consonance with a defined operational-demand related context. The selection process can be done either manually or autonomously via the use of artificial intelligence (Fuzzy logic or neural networks). The collective impact of the combined related context information (considered as secondary context information) is assessed, analysed and then passed onto the next higher level. In addition, based on changes in context value/parameter occasioned by evolving application scenario, the Lower Context management level can make demands to a Higher Context Management level to tune certain reconfigurable components in order to optimise the overall system performance.

To illustrate this, assuming retransmission occurs too often thereby consuming scarce energy resource in the process, the energy context manager (see Figure 3.2) can then inform the Coordinating Context manager (located in the Higher Context management Level) about this development. Moreover, possibly advise it to suspend transmission activities pending when contending issues are eventually resolved.

$$Z_s = \{D^L_{(s,0)} * y_0 , D^L_{(s,1)} * y_1 , D^L_{(s,2)} * y_2 , D^L_{(s,3)} * y_3 , \ldots D^L_{(s,k)} * y_k \} \qquad (3.1)$$

Each combination is relayed as a Set $Z_s$ (Equation 3.1) and managed appropriately to produce an output designated as $Z_n$ in Equation 3.2.

$$Z_n = \bigcup_{n=0}^{n=s} Z_s = \bigcup_{i=0,n=0}^{i=k,n=s} [D^L_{(n,i)} * y_i ] \qquad (3.2)$$

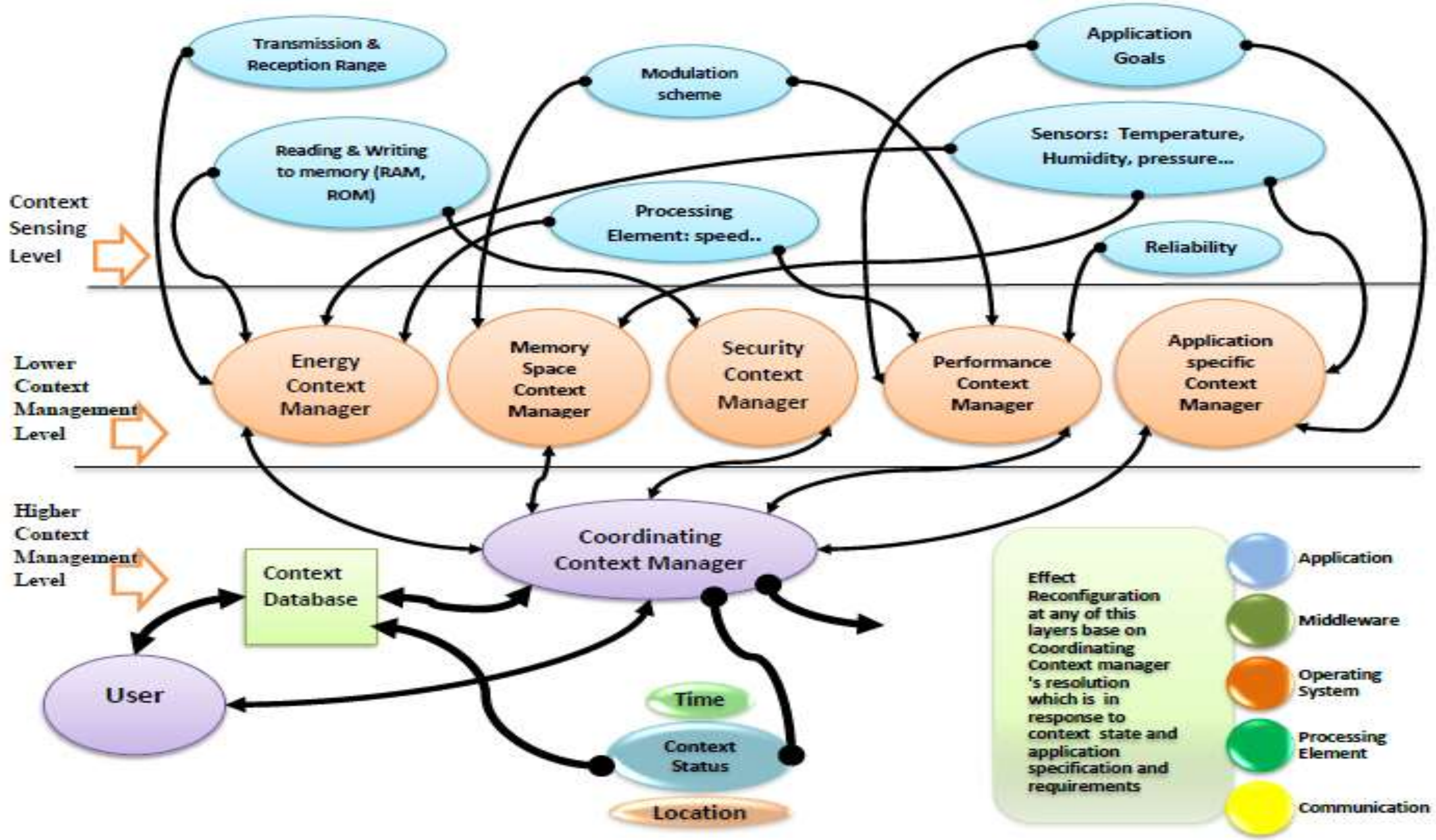


Figure 3.1: Program Flow representation of Proposed Context based Reconfiguration **Model**

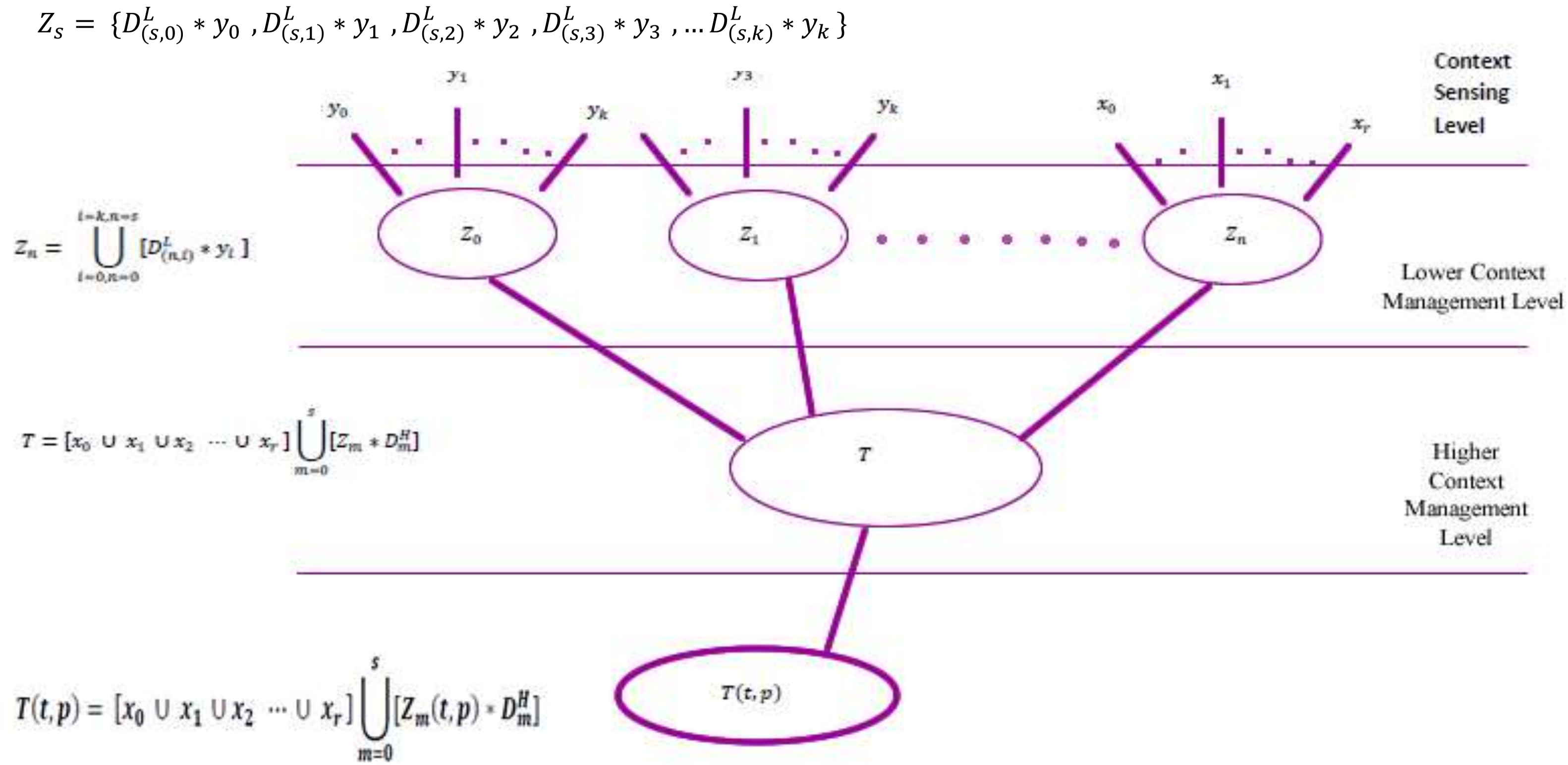


Figure 3.2: Context based reconfigurable WSN model

$$Z_n = \begin{cases} relevant\,, & D^L_{(n,i)} = 1, for\ i = 0\ to\ k \\ not\ relevant\,, & D^L_{(n,i)} = 0, for\ i = 0\ to\ k \end{cases} \quad (3.3)$$

Taking decision on when and how to effect changes can be a complex task. However, this can be simplified if all the identified or relevant applications and operational context information are quantifiable. Suitable metrics can be derived and utilised in this layer. For example, examining the rate of energy consumption within the node in relation to each instruction code execution can aid in devising a more energy to source-code software architecture. Hence, deriving a source-code-execution to energy consumption metrics will be relevant in establishing a relationship between the rate of energy consumption within the node to the system's application goals. An established relationship will invariably aid in the predicting the life span of nodes' energy sources (batteries)

**Higher Context management layer:** Much like its predecessor, the higher context management layer is designed to combine selected contextual information $Z_n$with the application key context information$'x_r'$. The coordinating context manager at this layer in response to the directive given by an end user selectively implements the combination via the use of switching elements denoted as $D^H_m$ (Refer to Equations 3.4 – 3.7).

$$T = \{x_0 * Z_0 * D^H_0, x_0 * Z_1 * D^H_1, x_0 * Z_2 * D^H_2, x_0 * Z_3 * D^H_3, \cdots x_0 * Z_s * D^H_s\} \cup \{x_1 * Z_0 * D^H_0, x_1 * Z_1 * D^H_1, x_1 * Z_2 * D^H_2, x_1 * Z_3 * D^H_3, \cdots x_1 * Z_s * D^H_s\} \cdots \cup \{x_r * Z_0 * D^H_0, x_r * Z_1 * D^H_1, x_r * Z_2 * D^H_2, x_r * Z_3 * D^H_3, \cdots x_r * Z_s * D^H_s\} \quad (3.4)$$

$$Z_n = \begin{cases} relevant\,, & D^H_{(m)} = 1, for\ i = 0\ to\ k \\ not\ relevant\,, & D^H_{(m)} = 0, for\ i = 0\ to\ k \end{cases} \quad (3.5)$$

$$T = x_0 \bigcup_{m=0}^{s} [Z_m * D_m^H] \cup x_1 \bigcup_{m=0}^{s} [Z_m * D_m^H] \;\; x_2 \bigcup_{m=0}^{s} [Z_m * D_m^H] \cdots \cup x_r \bigcup_{m=0}^{s} [Z_m * D_m^H] \qquad (3.6)$$

$$T = [x_0 \cup x_1 \cup x_2 \cdots \cup x_r] \bigcup_{m=0}^{s} [Z_m * D_m^H] \qquad (3.7)$$

The combined information is tagged with timing and location context information $'T(t,p)'$ (Equation 3.8) so that at every instant, detail historical data are easily constructed and a viable database is dynamically built and maintained.

$$T(t,p) = [x_0 \cup x_1 \cup x_2 \cdots \cup x_r] \cup_{m=0}^{s} [Z_m(t,p) * D_m^H] \qquad (3.8)$$

Time ($t$ ) can be specified or implemented in intervals,

Position (*p)* can be derived from any of the following: GPS, RSSI and RFID.

Appropriate reconfiguration processes can be initiated or implemented in any one of the five reconfigurable components in response to any of the lower level context manager's recommendations**.** However, the user ultimately decides what actions are to be taken and in setups where some form of artificial intelligence is involved, the decision and the nature of reconfiguration processes are automated. WSN application users can also avail themselves with the system's activity and performance history.

In realising the proposed concept, a few challenges are to be expected. One key challenge is how to identify and develop appropriate metrics for certain contexts (for example, performance, and reliability related issues). This can be a complicated venture requiring comprehensive experiments. Others might span over the development and optimisation of execution-codes that maintain a balance between performance and memory size requirement.

Some of the benefits of realising the model are highlighted below:

- Serves as a framework for developing an all-encompassing context-based reconfigurable WSN, in addition, prompting the exploration of other system-related contexts and the development of appropriate metrics.
- Encourage research work that explores the inclusion of artificial intelligence techniques at higher context level management. This allows the application to respond to evolving changes especially in unfriendly environments.
- If properly implemented, system performance and the rate of resource depletion can easily be managed and optimised. For example, predictions or estimation of the energy depletion rate is attainable at much higher precision.

### 3.1.2 Software Component for Application Context Extraction

#### 3.1.2.1 Precise Delta Extractor (PDE) Design and Implementation

The PDE implementation serves the following two purposes:

a. A precise delta extraction tool that can be used with the different methods

b. To provide a measure of change/ modification or indirectly a measure of changes in the application context information, this also serve as input for the fuzzy logic controller.

Program modification can occur in any of the ways below listed:

- Adding new functionalities or data (for example, constants, variables, program constructs)

- Removing no longer needed functionalities and related data.
- Updating existing functions or data content.

#### 3.1.2.2 PDE Design Concept

$Let\ A = \{x | all\ bytes\ making\ up\ the\ firmware\ of\ original\ source\ code\}\ and$

$B = \{x | all\ bytes\ making\ up\ the\ firmware\ of\ modified\ \ source\ code\}$

$Now\ \ \Delta^{+} = B \backslash A :\ \ \Delta^{+} \Rightarrow Added\ set\ of\ code\ with\ significant\ increase\ in\ |B|$

$Also,\ \ \Delta^{-} = A \backslash B :\ \Delta^{-}\ \ \Rightarrow Removed\ \ set\ of\ codes\ with\ significant\ decrease\ in\ |A|$

However, modification could take place without a significant change in the number of elements contained in either A or B. Such occurrences can be represented as $\Delta^{\mp}$

**Extracting** $\Delta^{+}$ , $\Delta^{-}$ $\boldsymbol{and}$ $\Delta^{\mp}$

Descriptions of the symbols used in the mathematical modelling of the PDE scheme are given in Table 3.3 and Table 3.4 respectively. The symbols used were based on the structure of the Execution Link File (ELF) format as highlighted in Appendix A.

Table 3.3: Description of ELF membership type symbols

| Member Types | Description |
|---|---|
| $\boldsymbol{PH.ptype}$ | Type of segment this array element describes |
| $\boldsymbol{SH.sh_{addr}}$ | Section's physical address |
| $\boldsymbol{PH.p_{addr}}$ | Segment's physical address |
| $\boldsymbol{PH.p_{filez}}$ | The number of bytes in the file image of the segment |
| $\boldsymbol{SH.sh_{filsize}}$ | The number of bytes in the file image of the section |
| $\boldsymbol{SH.sh_{flags}}$ | Flags relevant to the segment |

Table 3.4: Description of ELF memberships' attributes type symbols

| Attributes | Description |
|---|---|
| $\boldsymbol{PH_{T\_LOAD}}$ | The array element specifies a loadable segment |
| $\boldsymbol{SHF_{Alloc}}$ | The section occupies memory during process execution |
| $\boldsymbol{SHF_{EXECINSTR}}$ | The section contains executable machine instructions |

*Let SEG* = a collection of $seg_j$with the *PH.p_type* attributes =$PH_{T\_LOAD}$*:*

$$SEG = \left\{ \bigcup_{j=0}^{n-1} seg_j \middle| PH.ptype = PH_{T\_LOAD} \right\} \tag{3.9}$$

Where PH ⇒ Program Header and *n* = *number of segments:*

And let

$$A = sec_i.SH.sh_{addr} \in \left[ seg_{j.}PH.p_{addr}, seg_{j.}PH.p_{addr} + seg_{j.}PH.p_{filez} \right] \tag{3.10}$$

$$B = sec_i.SH.sh_{filsize} \neq 0 \tag{3.11}$$

$$C = sec_i.SH.sh_{flags} = SHF_{Alloc} \tag{3.12}$$

$$D = sec_i.SH.sh_{flags} = SHF_{EXECINSTR} \tag{3.13}$$

Where *SH* ⇒ Section Header and m = number of sections then the elements of $seg_j$ consists of a collection of sections $sec_i$ expressed thus:

$$seg_j = \left\{ \bigcup_{i=0}^{m-1} sec_i \mid A \,\&\, B \vee C \vee D \right\} \tag{3.14}$$

From each section contained in $seg_j$, a unique address value ($Uaddr_k$) is derived for each instruction code/data by concatenating values of segment number ($j$), section number ($i$) and the position ($p$) of each instruction/data ($D_k$).

$$Uaddr_k \quad = j + i + p \quad for\ k = 0 \rightarrow \sum_{j=0}^{m-1} |seg_j| \tag{3.15}$$

The addressing scheme uniquely identifies an associated instruction code/data contained in the entire loadable file. In order to identify changes ($\Delta^+$, $\Delta^-$ $\boldsymbol{and}$ $\Delta^\mp$) resulting from reprogramming or reconfiguration processes, $seg_j$ are obtained for the original file's ELF ($F_{orig}$) and the modified version ($F_{mod}$) respectively. Subsequently, while using $Uaddr_k$ as a reference, each $D_k$ within $sec_i$ of respective $seg_j$ are compared and where there are differences, they are reported as either modified set of codes ($\Delta^\mp$), added set of codes ($\Delta^+$) or removed set of codes ($\Delta^-$) appropriately. Algorithm 1 listing shows the algorithm employed for the PDE.

---

**Algorithm 1: Precision Delta Extraction (PDE) Implementation**

---

From SEG obtain a collection of seg
{
For each seg, obtain a collection of sec
{
For each sec collection
{
Compare associated contents ($D_k$) of $F_{orig}$ and $F_{mod}$ as addressed by unique address value ($Uaddr_k$)
{
Case (**contents = equal**) : ignore
Case (**contents = different**) : report as modified, note address, count number of occurrence(s)
Case (**$Uaddr_k$ contained in $F_{orign}$ does not exist in $F_{mod}$**) : a deletion of code(s) has taken place , note address, count number of occurrence(s)
Case (**$Uaddr_k$contained in $F_{mod}$ does not exist in $F_{orig}$**) : an addition of code(s) has taken place, note address, count number of occurrence(s)
}
}
}
}

---

**Measuring the degree of $\Delta^{+}$ , $\Delta^{-}$ $\boldsymbol{and}$ $\Delta^{\mp}$ in relation to the original firmware size (Distortion Metrics)**

*Let m, n* and *p* represent the total number of segments, sections and bytes/words respectively, Likewise:

$Tsec_i$ = Total Number of bytes /words contained in a section.

$Tseg_j$= Total Number of bytes /words contained in a segment

$T_f$ = Total Number of bytes /words contained in the file.

These terms can be obtained thus:

$$\mathrm{Tsec_i} = |sec_i| \tag{3.16}$$

$$\mathrm{Tseg_j} = \sum_{i=0}^{n-1} |sec_i| \tag{3.17}$$

$$\mathrm{T_f} = \sum_{j=0}^{m-1} |seg_j| \tag{3.18}$$

Where $\delta$ represents the degree of changes effected, the value $\delta$ can be obtained thus:

$$\delta = \left( \frac{T_f\left(F_{orig}\right) - T_f\left(F_{mod}\right)}{T_f\left(F_{orig}\right)} * 1 \right) \tag{3.19}$$

Based on the value of $\delta$ , the following can be inferred:

i. When($\delta < 0$ ) , it implies that a set of codes has been added and possibly some of the original codes could have been modified as well.

ii. When($\delta > 0$ ) , it implies that a set of codes has been removed and possibly some of the original codes could have been modified as well.

iii. When($\delta = 0$ ) , it implies that no change has taken place, however, it is possible that some of the original codes could have been modified as well.

### 3.1.2.3 PDE Evaluation

The roles of PDE as earlier discussed entails providing information on changes occurring in the application context. The information being a function of the size of bytes involved in comparison to the total size of the application.

The acquisition of Application Context information for the purpose of system evaluation was achieved as follows: Sample application source codes' ELF files were obtained using the GNU C compiler customised for the Contiki operating system. Each of the sample files' source codes were altered or modified in response to changes emanating from evolving application needs. Typically, these changes could involve or span over variables, constants, function names, libraries and other source code constructs. However, in this work the changes were confined to variation involving constants, variables and Function names only.

Having implemented these changes, the modified files were then recompiled to obtain new ELF files. Each pair of generated ELF files (original and modified) were further processed using the PDE. The PDE, by design, outputs a dataset, which contains a collection of delta (the data difference(s) between the original and modified files) and their respective address or addresses where applicable. In addition, the PDE produces three reports: the first and second reports are printouts of ELF constituents (available sections, data contents and their respective addresses) of both the original and modified files respectively. The third report relays the changes detected in the two files. Samples of the relevant extract of these printouts can be found in appendix C. Figure 3.3 shows the front end of the PDE application developed using C-sharp programming tools, while Figure 3.4 shows an additional form that displays ELF profile information of application firmware. As indicated in the Figure 3.3, the original and modified application's ELF constituents (generated unified address, physical address, data, list of loadable segments and segments related addresses and size) as well as the generated delta are displayed using the list view object components labelled as 'Original' , 'Modified' and 'Delta' respectively. The benefits of PDE are listed below:

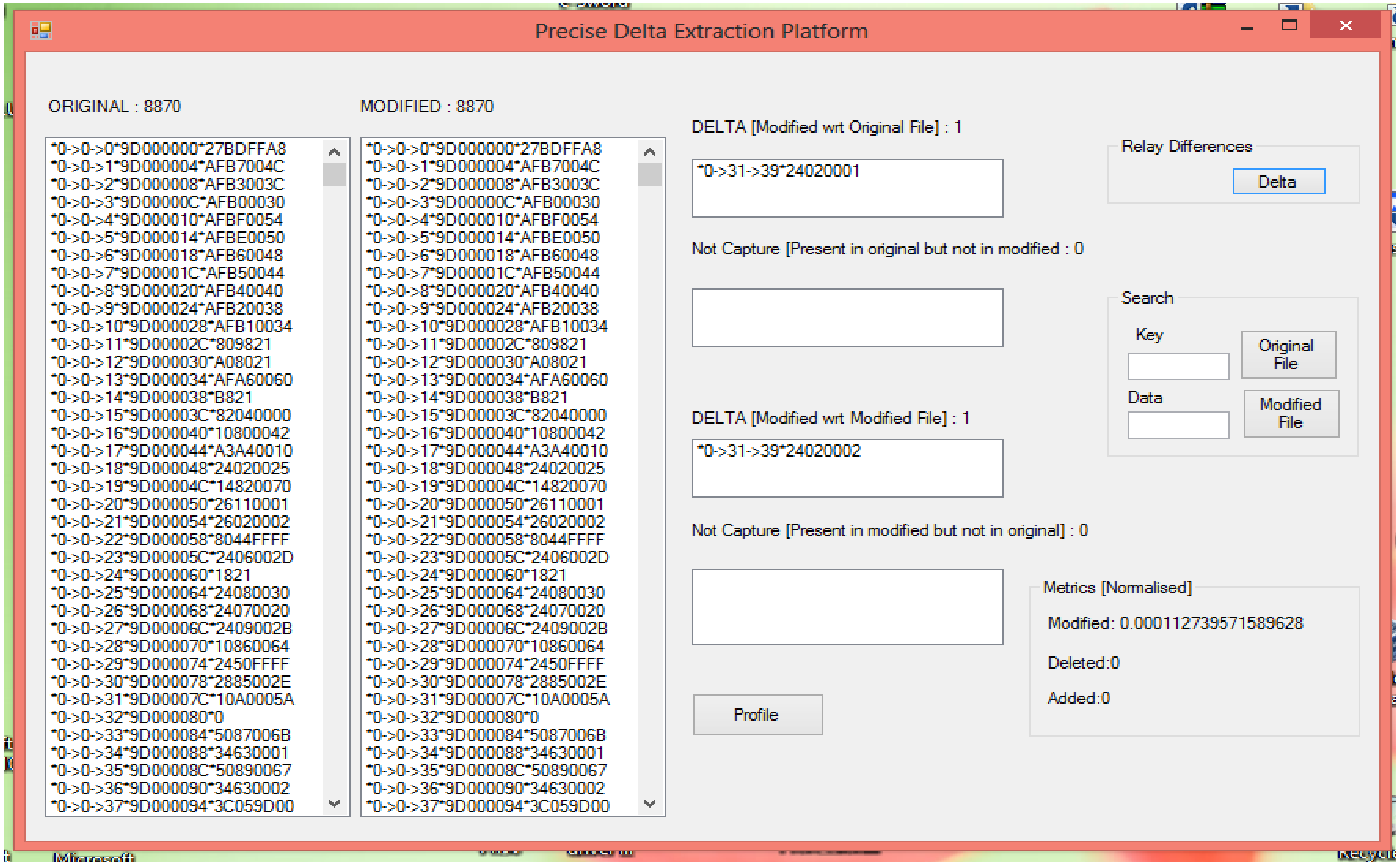


Figure 3.3: Delta Extraction Front end.

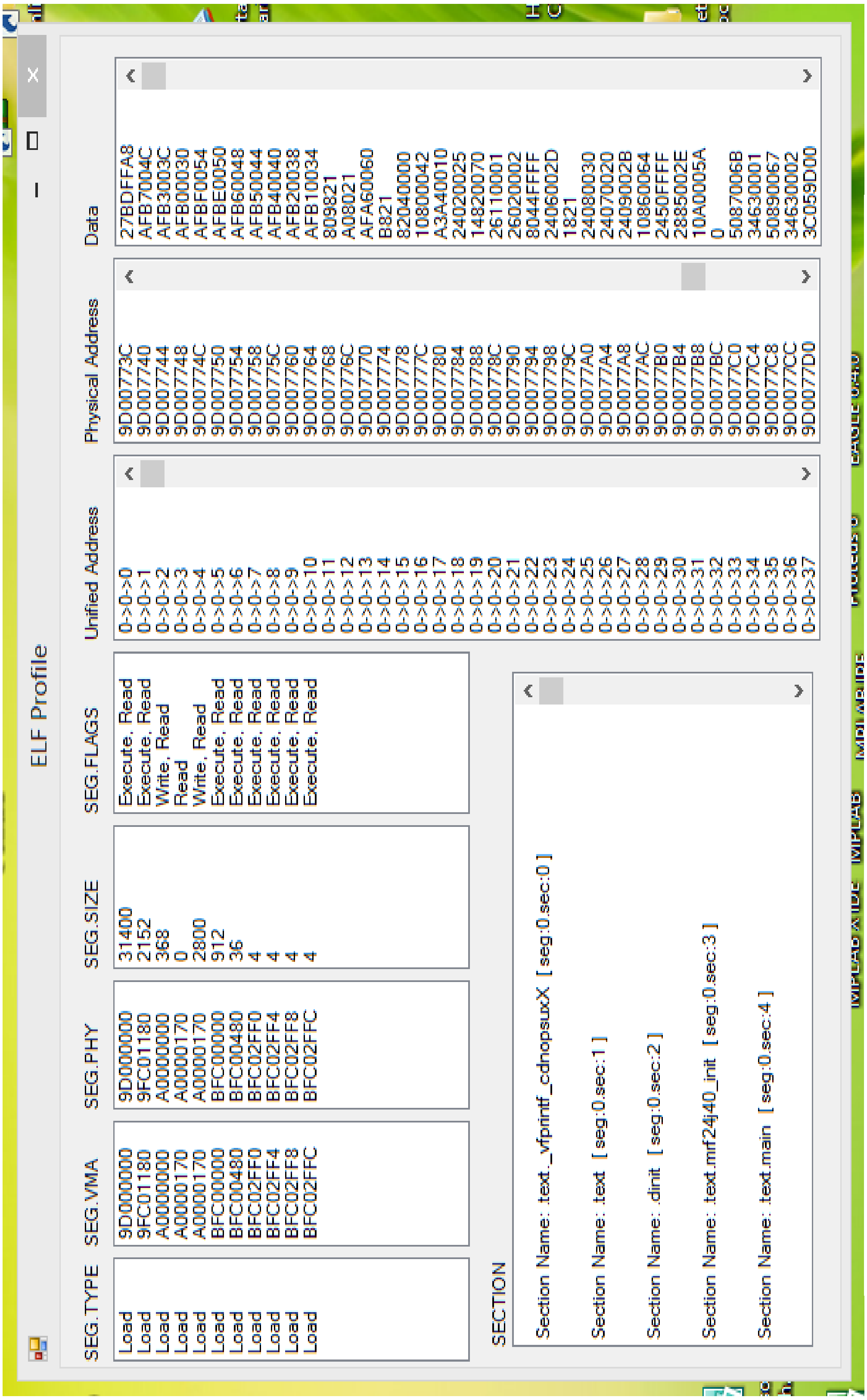


Figure 3.4: ELF Profile Display Front end.

i. It is used to measure the extent of firmware modification resulting from the addition of new functions, removal, or an update of existing functions.
ii. The normalised delta output is passed as an input to the context-based WSN reconfiguration’ s fuzzy controller, which aids in deciding the most appropriate reconfiguration scheme to use.
iii. It is useful in detecting firmware cloning.

### 3.1.3 Flash Memory Energy Consumption Modelling

One very principal factor worth considering when evaluating the impact of reconfiguration processes on entire WSN performance and energy sustenance is the knowledge of the characteristics of the memory technologies. In practice, no memory technology reads and writes in negligible time, retains its stored value indefinitely, occupies negligible space and consumes negligible power. Available memory technologies have varied advantageous capabilities: some are stronger in one or more of the aforementioned characteristics and weaker in others.

These technologies entail: Static RAM (SRAM), Electrically Erasable Programmable Read-Only Memory (EEPROM) and Flash. The Flash is a product of advancement in the floating gate technology using two known techniques, the hot electron injection (HEI) and the Nordheim Fowler tunnelling (NFT) technologies. It consists of a single transistor per memory cells. Unlike EEPROM, it can only erase in blocks. It has a wear-out mechanism that limits the number of erase and write operations. Flash technology is amazingly powerful, and it is mainly used currently in microcontroller program memory.

Most microcontrollers in use by wireless sensor nodes employ at least two to three of the aforementioned memory technologies. Some examples of these microcontrollers are the MSP430 and the PIC32MX320F128H. The second microcontroller is used in the evaluation testbed.

MSP430 flash memory is segregated into segments. Single bits, bytes, or words can be written to flash memory, but the segment is the minimum size of flash memory that can be erased (www.ti.com/product/msp430f123.pdf). The segments are further divided into blocks. A block is 64 bytes, starting at 0xx00h, 0xx40h, 0xx80h, or 0xxC0h, and ending at 0xx3Fh, 0xx7Fh, 0xxBFh, or 0xxFFh. Figure 3.5 shows the flash segmentation using an example of 4-KB flash that has eight main segments and both information segments.

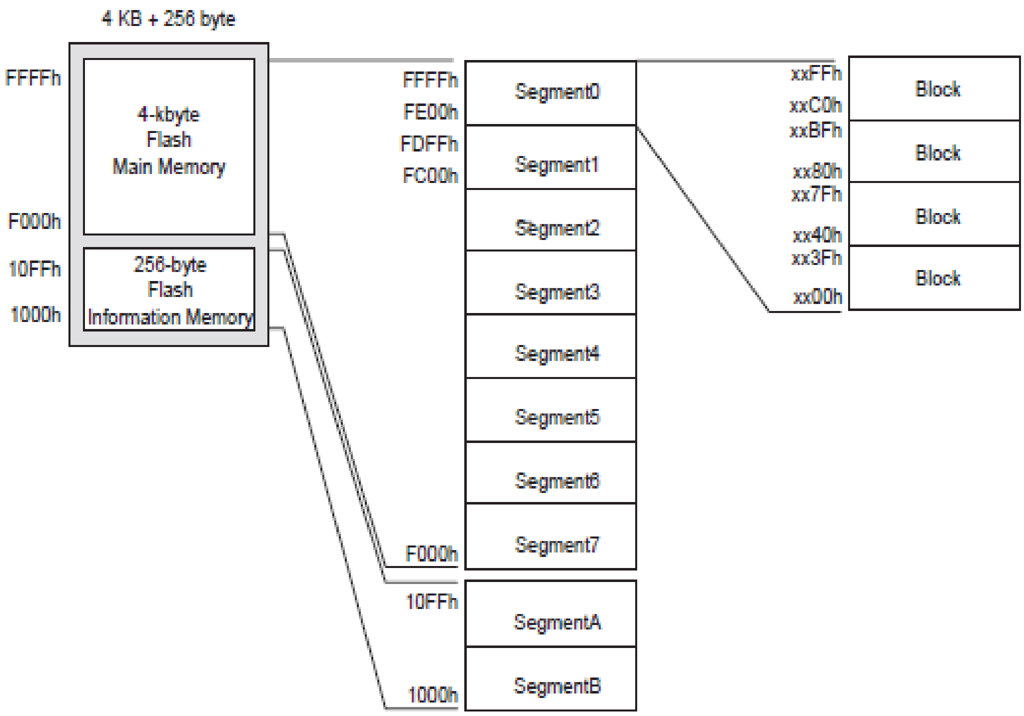


Figure 3.5: Flash Memory Segments, 4KB Example
(www.ti.com/product/msp430f123.pdf)

The program Flash array for the PIC32MX320F128H device is built up of a series of rows. A row contains 128 32-bit instruction words or 512 bytes. A group of eight rows/blocks compose a page; which, therefore, contains 8 × 512 = 4096 bytes or 1024 instruction words. A page/segment of Flash is the minimum unit of memory that can be erased at a single time. The program Flash array can be programmed by Row/Block programming (128 instruction words at a time), Word programming (one instruction word at a time) or both.

#### 3.1.3.1 Related Memory re-Flashing Constraints

Three possible reconfiguration scenarios are highlighted in Figure 3.6, Figure 3.7 and Figure 3.8. In each Figure, two columns of a set of blocks designated as '$SegO_0$…$SegO_n$' and '$SegN_1$…$SegN_n$' represent original and reconfigured contiguous segments of flash memory respectively. Reconfigured data are represented by a strip of filled rectangular blocks. As shown in Figure 3.6, the first scenario describes a situation where the number of reconfigured data bytes is confined to a single segment '$SegN_1$'. In such a scenario, erasure and rewriting procedures should naturally be limited to a single segment. However, in practice, this is not always the case; the entire flash memory is always erased, and the new firmware rewritten all again. The repeated occurrence of the erasure and rewriting procedures will eventually accelerate energy consumption at a higher rate.

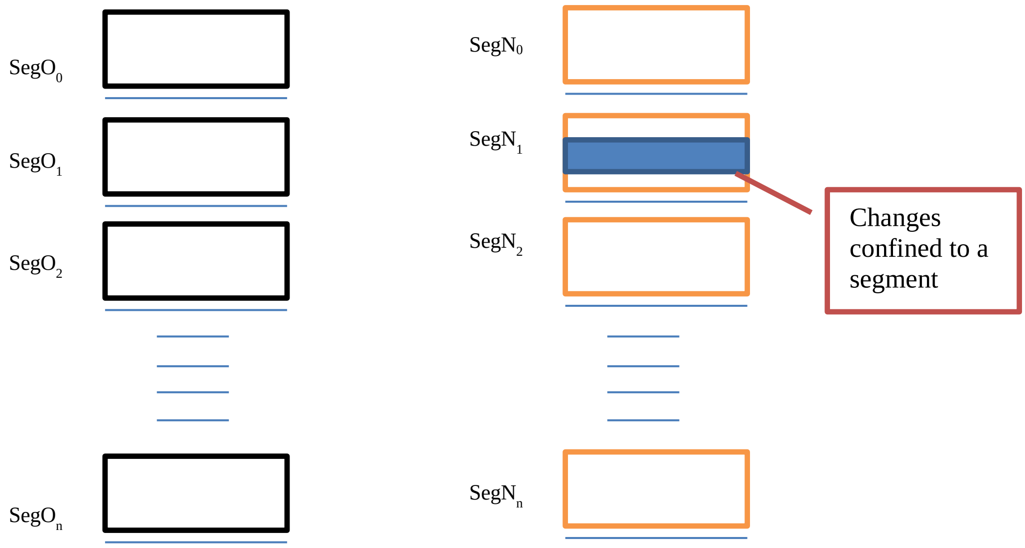


Figure 3.6: Reconfigured data confined to a single segment

The second scenario as depicted in Figure 3.7 illustrates the space taken in memory by the reconfigured data.

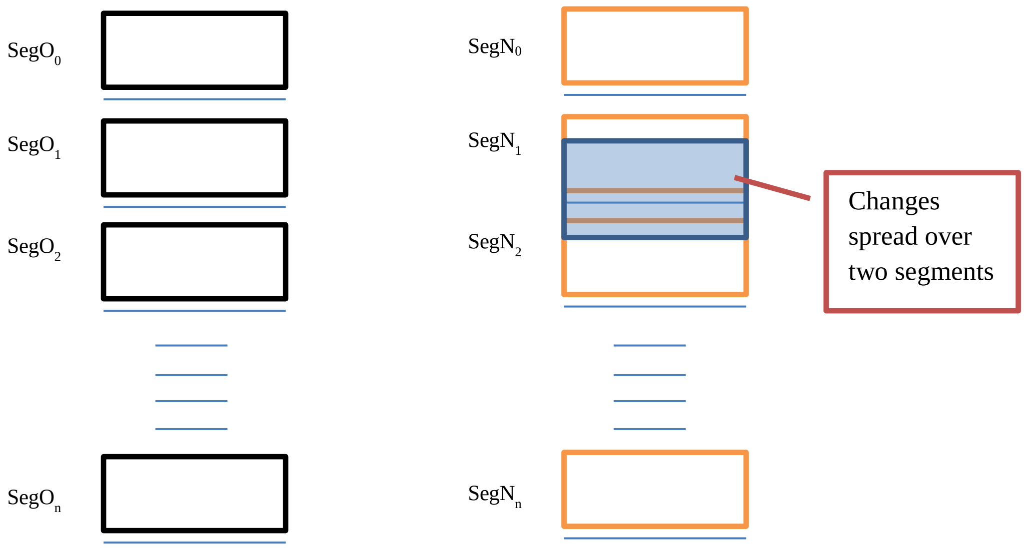


Figure 3.7: Reconfigured data spread over adjoining segment

The space overlaps adjoining segments and being able to handle erasure and writing operations within these two segments will invariably result in consumption of much less energy.

The third scenario, shown in Figure 3.8, depicts changes in the new firmware that are unevenly distributed all over the memory space. This is attributed to changes resulting from the addition, removal or renaming of functions within application source codes. These can be more complex when the functions are referenced in several places inside the application source code. Similar problems exist for global data variables (Dong, Liu, Chen, Bu, Huang and Zhao, 2011).

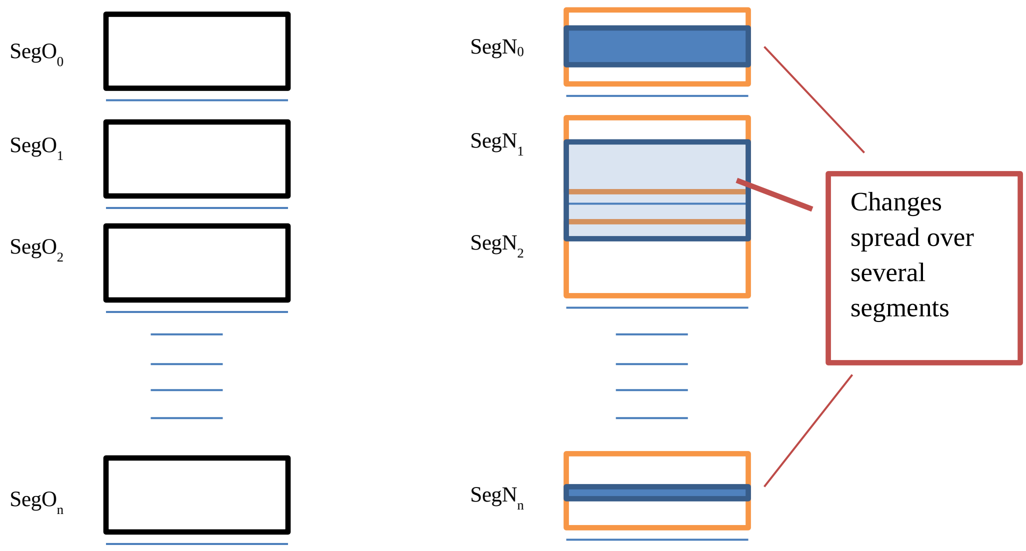


Figure 3.8: Changes spread over several segments

### 3.1.3.2 Firmware Reconstruction Algorithm

In cases where changes occur within a single segment and considering the 'erase before rewriting' constraint associated with Flash memories, it is economical to ensure that the reconstruction procedures are confined to that segment. Based on the data obtained from earlier works (Han-Lin, Chia-Lin and Hung-Wei, 2008; Gaurav, Peter, Deepak and Prashant, 2006), it can be inferred that the cost of erasing an entire memory is far less than erasing individual segments. Likewise, writing to a segment is much cheaper than writing to each word that makes of a segment.

The norm in practice has been to erase the entire Flash memory and then reprogram it with the new update. Based on the analysis highlighted in the preceding section, three delta-orientations were inferred. These are namely 'Segment-confined', 'Adjoint-Segments' and 'Disjoint-Segment'. In practice, only the first and the last are more pronounced.

The PDE presented in section 3.1.2 provides the address of every delta detected, which invariably can be helpful in pinpointing the exact segment where they occur. This information allows for erasure and rewriting operations to be carried out within only selected segment(s) of relevance.

Algorithm 2 listing in highlights the re-flashing algorithm developed and employed in the context-based WSN reconfiguration software system. $a_k$ and $d_k$ represent the address and data of delta extracted by the PDE utility where $k$ signify the index or position of each member in the set with cardinal value of $m$. Let $SO_i$ and $SN_j$ denote segments containing the original and modified firmware in flash memory where $i$ and $j$ are their respective locations within a set of $n$ segments contained in the flash memory. $T(r)$ connote an array for storing the index or indices of segment(s) affected by the modifications or reconfigurations.

**Algorithm 2:** Flash Program Memory re-flashing

```
r = 0; j = 1; k = 1
While ( j<=n) do
While (k<=m) do
If (a_k => start address of SN_j & a_k <= start address of SN_j)
T( r ) = j
end if
k++ ; r++ ;j++
 end while
end while
Select |T( r )|
Case 1:
Erase and reprogram within SO_T(0)
Case 2:
Erase and reprogram SO_T(0) and SO_T(1)
Case >2:
Erase and reprogram entire memory space
end select
```

### 3.1.4 Adoption of Fuzzy Logic Controller

This subsection discusses the adoption of Fuzzy Logic controller earlier introduced at the beginning of this chapter. In order to demonstrate the benefits of the context based reconfiguration model, two contexts related input variables were used. The delta-orientation obtained from the ELF profile of the modified code served as the application related context and the Battery energy level state was taken as an operational-demand related context. A robust inference engine was developed based on the inferred expert knowledge on memory related energy consumption pattern during the reconfiguration process. The pattern studied and presented in section 3.1.3 explains how delta size and its orientation can influence energy consumption during reprogramming operations. The resulting output from the fuzzy logic system controls when and which one of the reconfiguration approaches should be implemented in order to prolong the battery life. Figure 3.9 shows the fuzzy logic controller's flow diagram.

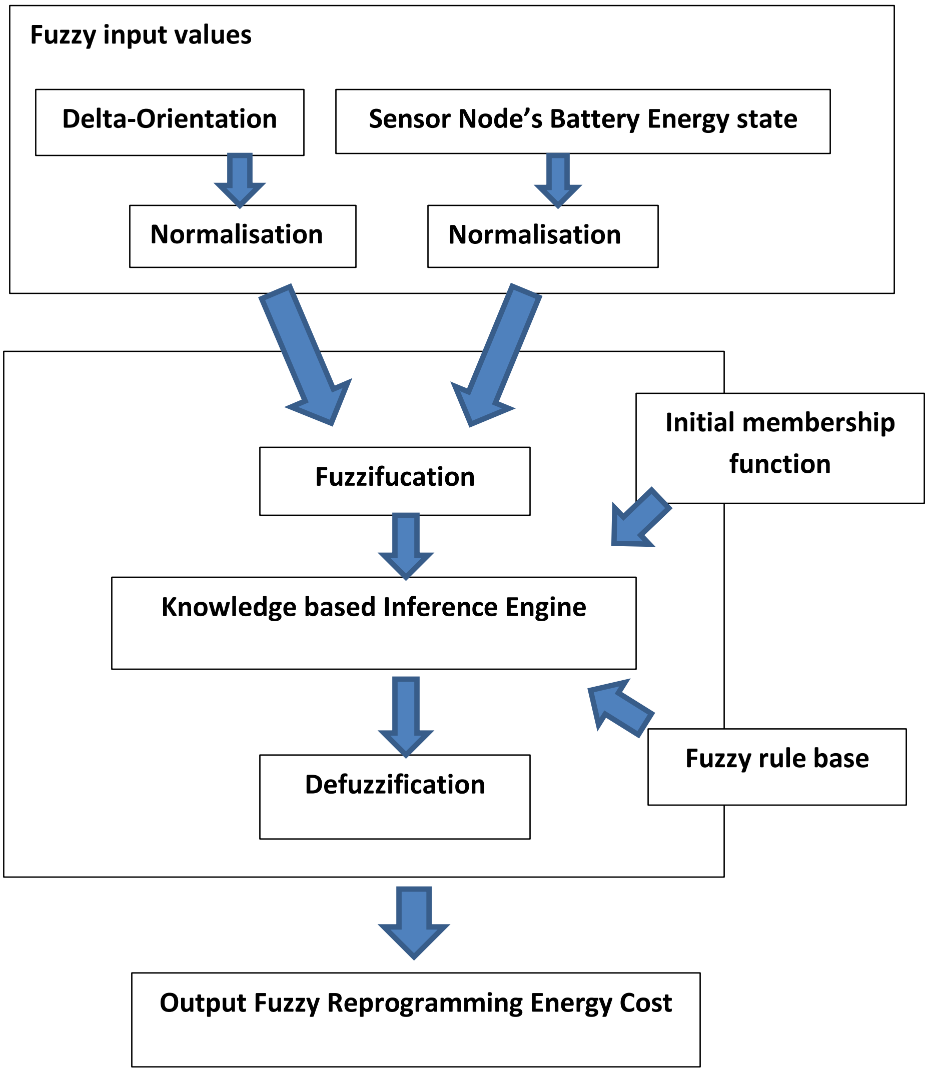


Figure 3. 9: Computation of Fuzzy Reprogramming Energy Cost

### 3.1.4.1 Fuzzy Logic Controller

A Fuzzy Logic Controller (FLC) is a software component that controls the output variables of a system according to its inputs and a set of rules expressed with the uncertainty of human terms (Rada-Vilela, 2013).

The fuzzy system designed and employed in this research work is composed of four main parts. These parts are namely a fuzzifier, a knowledge base, an inference engine, and a defuzzifier.

The fuzzifier transforms the real crisp inputs' into fuzzy functions, therefore determining the 'degree of membership' of the inputs to a vague concept. The values of the input variables are mapped to the range of values of the corresponding universe of discourse. The range and resolution of input fuzzy sets and their effect on the fuzzification process are considered as a factor affecting the overall performance of the controller.

The knowledge base comprises the knowledge of the application domain and the related control goals. It can be splitted in a database of definitions and used to express linguistic control rules in the controller, and a rule base that describes the knowledge held by the experts of the domain. Intuitively, the knowledge base is the core element of a fuzzy controller as it will contain all the information necessary to accomplish its execution tasks. Extensive research has been carried out in order to fine-tune a fuzzy controller's knowledge base, many using other Artificial Intelligence (AI) disciplines such Genetic Algorithms or Neural Networks.

The Inference Engine provides the decision-making logic of the controller. It deduces the fuzzy control actions by employing fuzzy implication and fuzzy rules of inference. In addition, it is viewed as an emulation of human decision making. In Mamdani systems, the antecedents and consequents of a **fuzzy** rule are **fuzzy** sets. Inferences are based on Generalised Modus Ponens, which states that the degree of truth of the consequent of a **fuzzy** rule is the degree of truth of the antecedent. In the case where more than one antecedent clause is present, the individual degrees of membership are joined using a min t-norm operator. If the **fuzzy** set contains several rules, their output is combined using a max s-norm operator.

The defuzzification process converts fuzzy control values into crisp quantities; that is, it links a single point to a fuzzy set, given that the point belongs to the support of the fuzzy set. The defuzzification stage consists of converting the fuzzy outputs from each variable into crisp values, which are computed with a defuzzifier. Many defuzzifiers have been suggested in the literature (Leekwijck and Kerre, 1999), but the most common ones are the centroid and maxima defuzzifiers for Mamdani controllers (Mamdani and Assilian, 1975). Others are the weighted average and weighted sum for Takagi-Sugeno or Tsukamoto controllers (Takagi and Sugeno, 1985). The centroid computes the $u$ value of the centre of mass of the fuzzy set (Equation 3.20). A maximum defuzzifier returns the smallest, mean or largest $u$ value for the maximum membership function (Equation 3.21). The weighted average and weighted sum are computed on the modified functions utilising their activation degrees as weights. In the case of Tsukamoto, the defuzzifiers utilise the activation degrees as weights, and the membership functions of the activation degrees as values.

$$\bar{u} = defuzz(D) = \frac{\int uD(u)du}{\int D(u)du} \tag{3.20}$$

$$\bar{u} = defuzz(D)|\ D(u) is\ maxmum \tag{3.21}$$

### 3.1.4.2 Design and implementation of the fuzzy logic controller

The design and implementation of the fuzzy logic controller (FLC) consist of modelling the system inputs and outputs as linguistic variables, and creating the necessary inference rules that will control the system.

**Choice of Fuzzy Logic Control Library**

The design and implementation of fuzzy logic controllers are centred on the use of Fuzzy logic control libraries. The Matlab Fuzzy Logic Toolbox (http://www.mathworks.com.au/products/fuzzy-logic/index.html, accessed on July, 2013) is perhaps the most widely known library for designing FLCs. It is built on top of the Matlab computing environment and bundles Mamdani and Takagi-Sugeno controllers, four types of hedges, four fuzzy logic operators, seven defuzzifiers, over eleven linguistic terms, and FLCs can be imported and exported utilising the Fuzzy Inference System (FIS) format. Matlab and its toolbox are sold separately under restrictive and costly proprietary licenses. The toolbox has not been updated since 2005 (Rada-Vilela, 2013). Other FLC libraries are the Octave Fuzzy Logic Toolkit (Markowsky and Segee, 2011) and the jFuzzyLogic (Cingolani and Alcala-Fdez, 2012). These state-of-the-art libraries to model fuzzy logic controllers have strong limitations in terms of licensing, cost, design and implementation, all of which have been recently addressed in a free open-source fuzzy logic control library named fuzzylite (Rada-Vilela, 2013).

The fuzzylite fuzzy logic controller (Rada-Vilela, 2014) was adopted in this research work because it is much easier to configure and use. In addition, it possesses the under listed features:

i. Controllers: Mamdani, Takagi-Sugeno and Tsukamoto
ii. Linguistic terms: rectangle, triangle, trapezoid, bell, pi-shape, sigmoid difference and sigmoid.
iii. T-Norms: minimum, algebraic product, bounded difference, drastic product, Einstein product and hamacher product.

iv. S-Norms: einstein sum, bounded sum, normalised sum, drastic sum, algebraic sum, maximum, and hamacher sum.

v. Defuzzifiers: centroid, bisector, smallest of maximum, largest of maximum, mean of maximum, weighted average and weighted sum.

vi. Import and export controllers utilising the FCL and FIS formats

Fuzzylite is a fuzzy logic control library that is programmed in C++ and it is free open-source. It has a cross-platform capability. Its goal is to provide the design and operation of FLCs with an object-oriented approach such that controllers can be incorporated into any application in just a few steps without requiring any third-party libraries. Additionally, it comes with an application named qtfuzzylite to visually design FLCs and interact with their operation in real time.

#### 3.1.4.3 Modelling the system Inputs and Output

**Project Definition in Development Tool**

The first step is to use the qtfuzzylite to define the structure of the controller via its editor. The project editor (Figure 3.10) displays the controller structure and allows the designer to access linguistic variables and rule definitions directly.

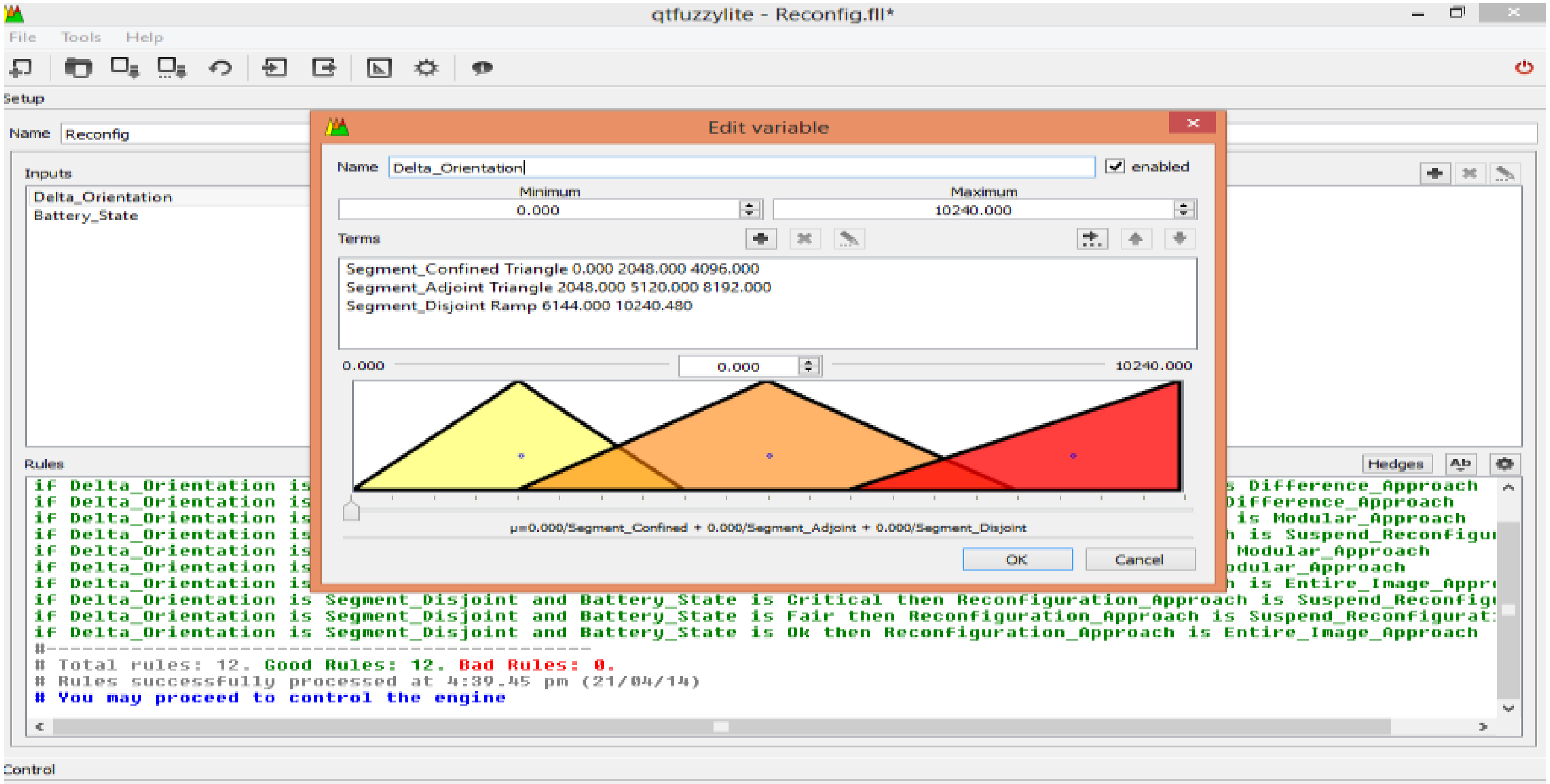


Figure 3.10: The qtfuzzylite designer editor

**Linguistic Variables Definitions**

The next step involves the use of qtfuzzilite graphic interface to create the most suitable linguistic variables and membership functions for the application.

The triangular functions are used as a membership function because they have been used extensively in real-time applications due to their simple formulas and computational efficiency (Sadiq, Abu, and Ghafoor, 2010).

The Delta-orientation obtained via the PDE and the sensor node's battery energy state served as input into the fuzzy logic system. The delta-orientation and the battery energy state were meant to represent the application and operational-demand context respectively.

The input membership functions shown in Figure 3.11 are defined for the Delta orientation input. It takes into account the three delta-orientation outlined in section 3.1.3.4. The delta-orientation is covered with three membership functions spread over a range of 2.5 * number of bytes contained in the segment of program memory (for the PIC32MX320F128H, each segment contains 4096 bytes). The three membership are Segment-confined, Segment-Adjoint and Segment-Disjoint.

The second input value for the fuzzy-logic system is the battery energy state expressed in terms of joules. As shown in Figure 3.12, the range of this input value is spread over the values of 0 to 18720 Joules, where 18720 Joules is the typical energy of two AA batteries (http://castalia.npc.nicta.com.au). The range maps the sensor node's battery energy level between when it is in a virtually depleted state to a fully charged state.

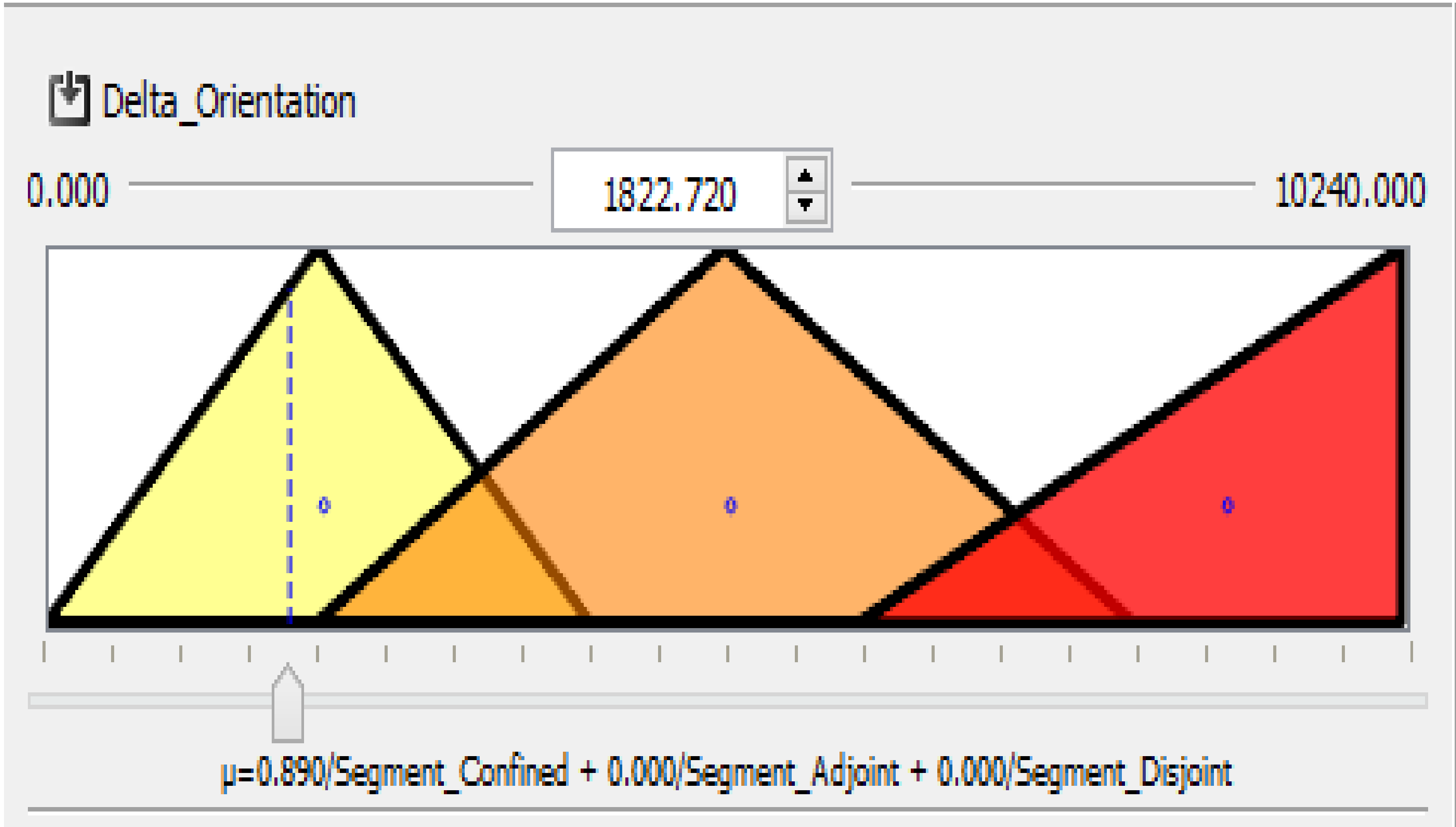


Figure 3.11: Membership function for Delta Orientation input value

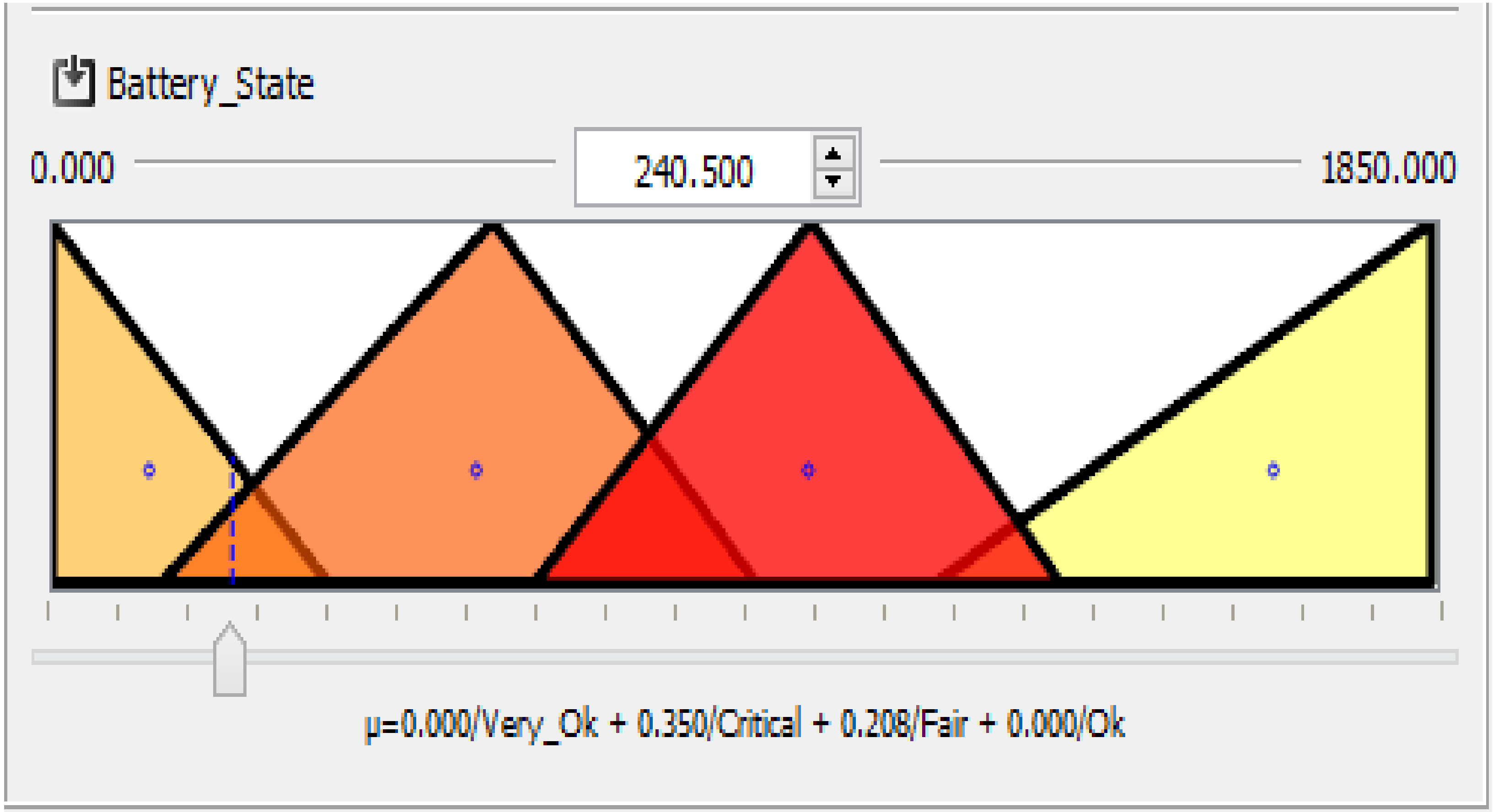


Figure 3.12: Membership functions for Battery energy state input value

The membership functions of the battery energy state input were distributed as follows:

i. Critical: Cannot support reconfiguration for any delta size or orientation; energy should rather be conserved for application's basic task

ii. Fair: Can support reconfiguration if delta size is within an acceptable size range- most probable a segment.

iii. OK: Can support whole or any delta size of reconfiguration. However, should be used with caution.

iv. Very OK: More than sufficient energy is available to handle any delta size or orientation.

The fuzzy system calculates the fuzzy-cost for each Delta and battery energy state input values. The ensuing output membership function intended to guide each sensor node in adopting the most appropriate reconfiguration approach while considering the available battery's energy level is shown in Figure 3.13. The distribution is spread over four options: Difference-Approach, Modular-Approach, Entire-Image-Approach, and Suspend-Reconfiguration. This process applies to each wireless sensor node while in the field. This ensures that the battery's energy level in every node is optimally managed during every reconfiguration process.

#### 3.1.4.4 Fuzzy Inference Engine

The fuzzy inference engine is composed of rules developed using expert knowledge. The design of the knowledge-based rules that connect the inputs, and the outputs is based on the philosophy behind reprogramming of Flash memory. This philosophy has been presented in section 3.1.3.

The fuzzy inference system is designed based on twelve rules listed below:

i. if Delta_Orientation is Segment_Confined and Battery_State is Very_Ok then Reconfiguration_Approach is Difference_Approach;

ii. if Delta_Orientation is Segment_Confined and Battery_State is Critical then Reconfiguration_Approach is Suspend_Reconfiguration;

iii. if Delta_Orientation is Segment_Confined and Battery_State is Fair then Reconfiguration_Approach is Difference_Approach;

iv. if Delta_Orientation is Segment_Confined and Battery_State is Ok then Reconfiguration_Approach is Difference_Approach;

v. if Delta_Orientation is Segment_Adjoint and Battery_State is Very_Ok then Reconfiguration_Approach is Modular_Approach;

vi. if Delta_Orientation is Segment_Adjoint and Battery_State is Critical then Reconfiguration_Approach is Suspend_Reconfiguration;

Vii if Delta_Orientation is Segment_Adjoint and Battery_State is Fair then Reconfiguration_Approach is Modular_Approach;

Viii if Delta_Orientation is Segment_Adjoint and Battery_State is Ok then Reconfiguration_Approach is Modular_Approach

ix. if Delta_Orientation is Segment_Disjoint and Battery_State is Very_Ok then Reconfiguration_Approach is Entire_Image_Approach;

x. if Delta_Orientation is Segment_Disjoint and Battery_State is Critical then Reconfiguration_Approach is Suspend_Reconfiguration;

xi. if Delta_Orientation is Segment_Disjoint and Battery_State is Fair then Reconfiguration_Approach is Suspend_Reconfiguration; and

xii. if Delta_Orientation is Segment_Disjoint and Battery_State is Ok then Reconfiguration_Approach is Entire_Image_Approach.

The qtfuzzylite development tool’s rule text editor (Figure 3.14) offers an easy way to examine and define the set of rules. Using these features, one can verify that all the defined rules are necessary, that no important rules are missing, and that the variations of the output variable are consistent with the designed system requirements. Optimising the entire system (Figure 3.15) behaviour is done easily and quickly by changing the set of rules, modifying the membership functions definitions, or selecting from the available defuzzification options.

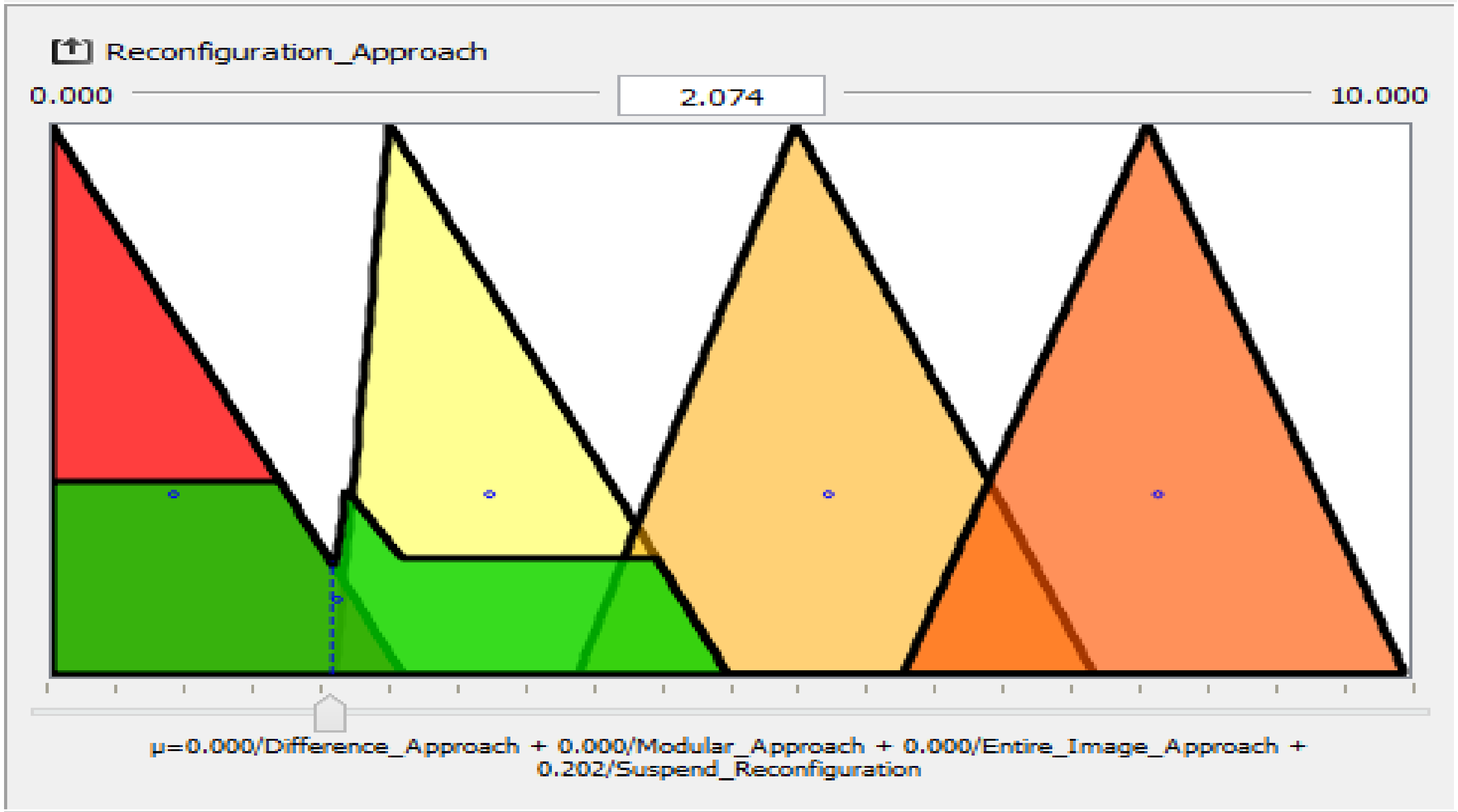


Figure 3.13: Membership function for the Reconfiguration Approach Output

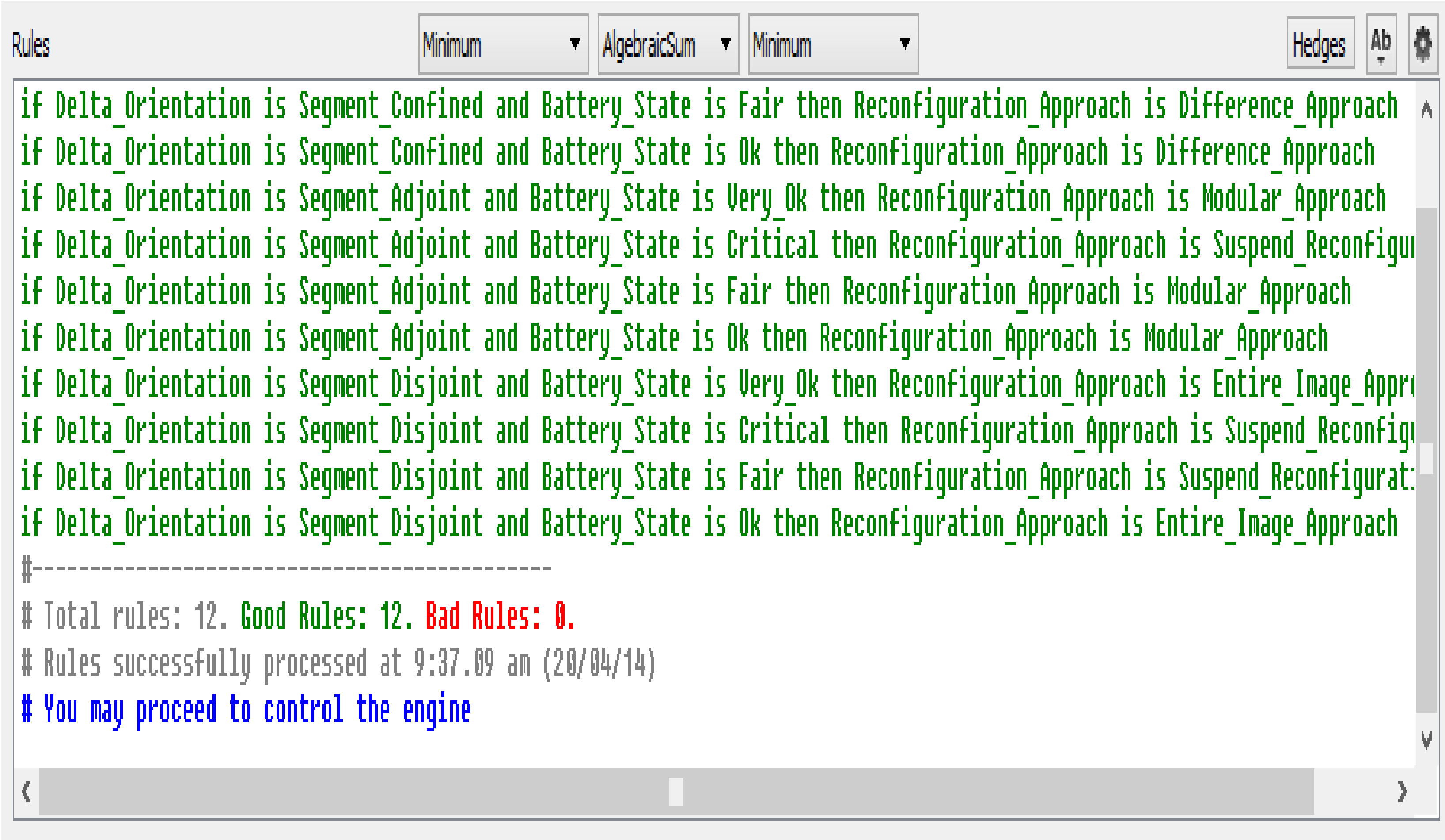


Figure 3.14: Qtfuzzylite development tool's rule text editor

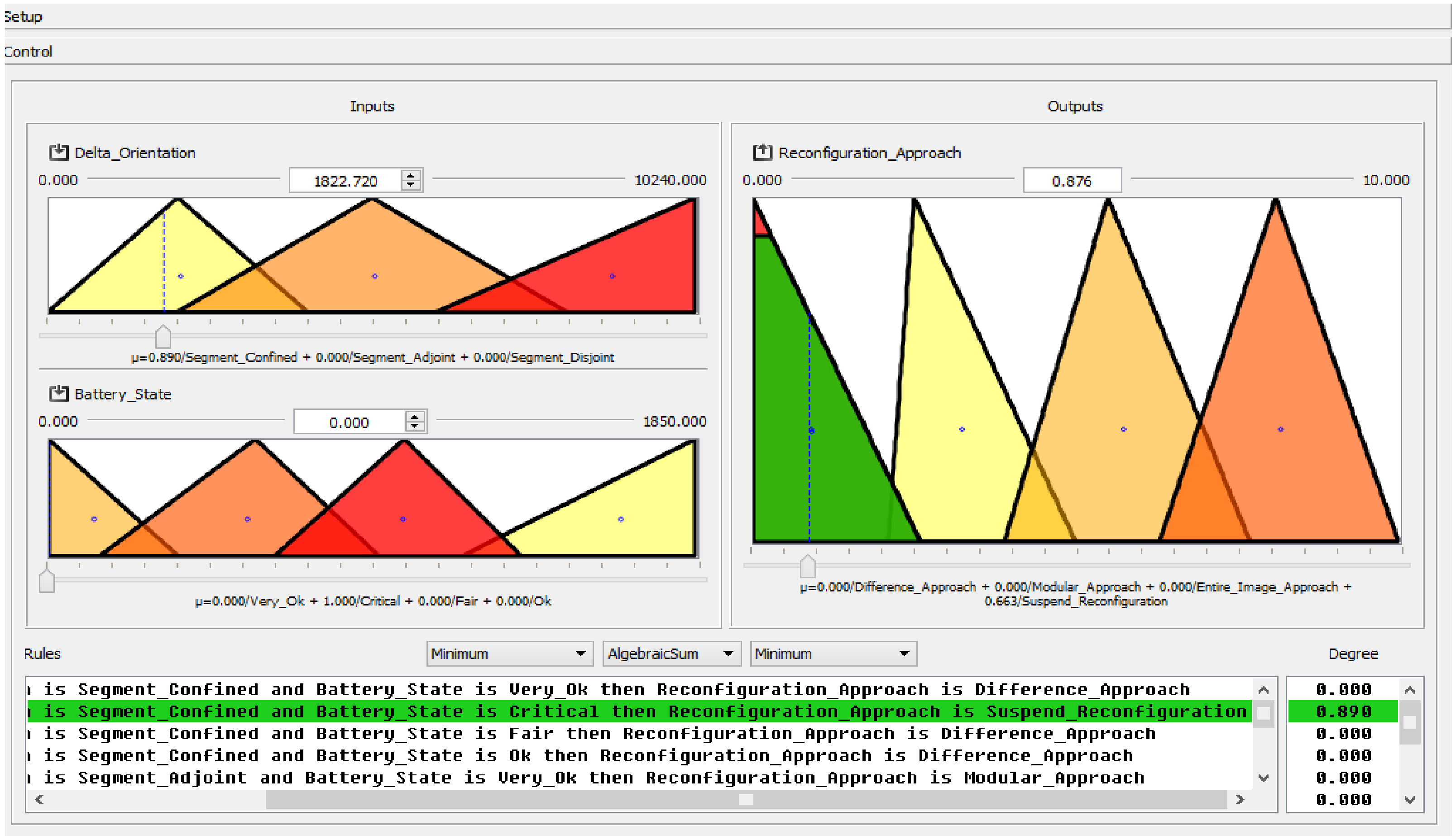


Figure 3.15: Complete Fuzzy control Platform

## 3.2 Context-Based Reconfiguration System Evaluation

### 3.2.1 Testbed Hardware Composition

The Testbed features a powerful Microchip PIC32MX320F128H microcontroller and a Microchip MRF24J40MB transceiver for implementing low-cost Wireless Sensor Network. The Microchip PIC32MX320F128H adopted is an extremely powerful microcontroller support implementing a Microprocessor without Interlocked Pipeline Stages (MIPS) architecture can provide up-to 80 *MIPS* of computational power. The microcontroller implements in hardware the following: Serial Peripheral Interface (*SPI),* Inter-Integrated Circuit (*I2C),* Universal Asynchronous Receiver/Transmitter (*UART),* Controller Area Network (*CAN)* and Universal Serial Bus *(USB)* communication protocols easing the connection with external units. It implements a reduced instruction set computer (RISC) instruction set. The Memory can be fully addressable by Direct Memory Access (*DMA)* controllers and IEEE802.3 Media Access Control *(MAC) layer* is implemented on chip (http://ww1.microchip.com/). The TestBed Board comes with Full software support, including porting for Contiki OS. Plates 3.17, 3.18 and 3.19 illustrate the hardware composition of the complete base station and the wireless sensor nodes respectively.

The Microchip MRF24J40MB transceiver (see Plates 3.20) is used for accessing the IEEE802.15.4 channel. This transceiver was chosen for its extremely high coverage (up to 100m in open space at max power) and for its high configurability. The MRF24J40 is an IEEE 802.15.4™ Standard compliant 2.4 GHz RF transceiver (http://ww1.microchip.com/). It integrates the PHY and MAC functionality in a single chip solution. The MRF24J40 creates a low-cost, low-power, low data rate (250 or 625 kbps) Wireless Personal Area Network (WPAN) device (http://ww1.microchip.com/).

The MRF24J40 interfaces to Microchip PIC microcontrollers via a 4-wire serial SPI interface, interrupt, wake and Reset pins.

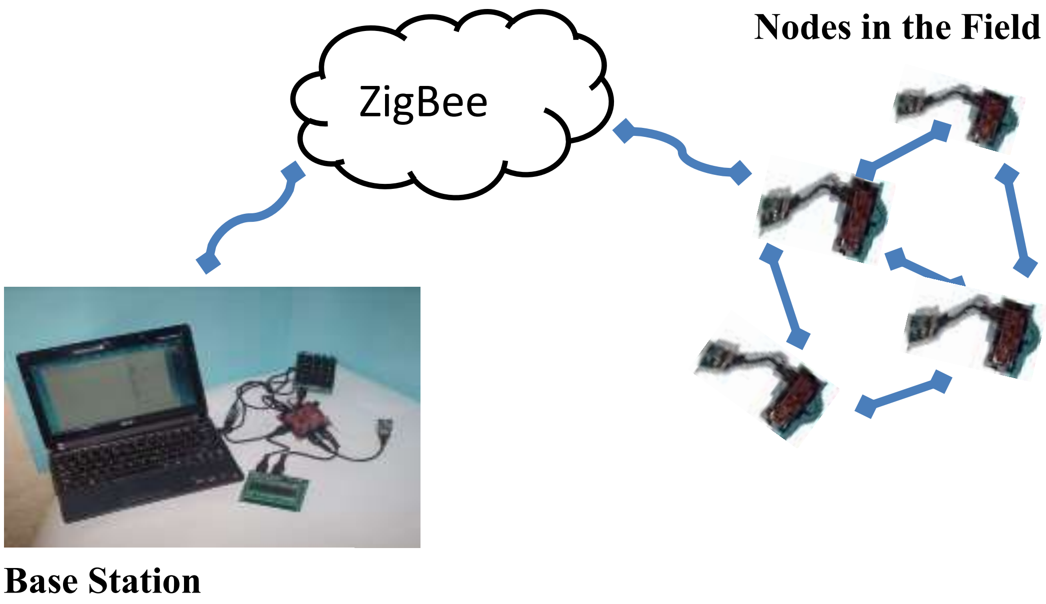


Plate I: The Tesbed Hardware Composition

Plate II: Base station Hardware Composition

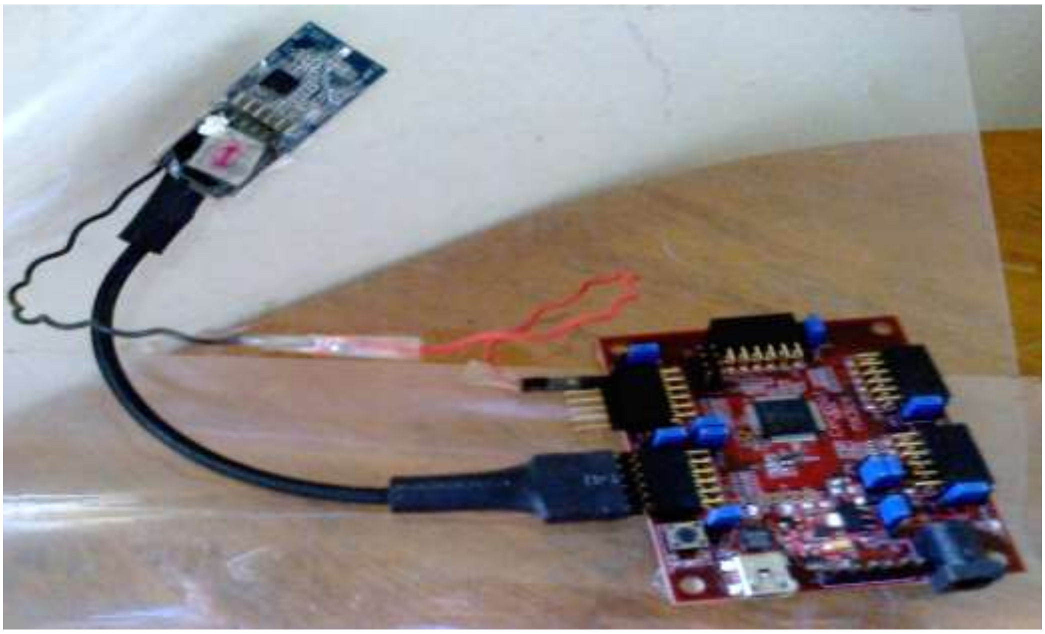

Plate III: Wireless Sensor Node Hardware

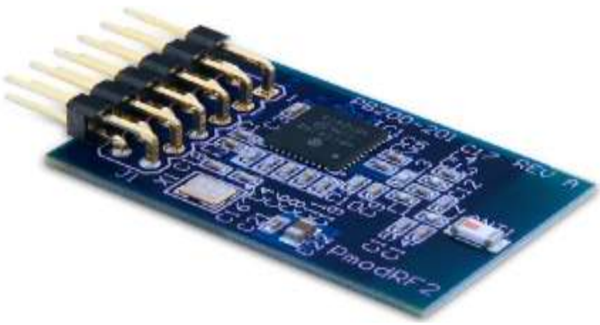


Plate IV: The Microchip MRF24J40MB transceiver (http://ww1.microchip.com/)

The MRF24J40 provides hardware support for:

•Energy Detection

•Carrier Sense

•Three Clear Channel Assesement (CCA) Modes

• Carrier Sense Multiple Access – Collision Avoidance (CSMA-CA) Algorithm

•Automatic Packet Retransmission

•Automatic Acknowledgment

•Independent Transmit, Beacon and Guaranteed Time Slot – First in First out (GTS-FIFO) Buffers

•Security Engine supports Encryption and Decryption for Media Acess Control (MAC) Sub layer and Upper Layer

These features reduce the processing load, allowing the use of low-cost 8-bit and 32-bit microcontrollers.

### 3.2.2 Testbed Software Composition

The embedded software system implemented in each source code runs on the Contiki operating system platform. Codes in Contiki run in either of two execution contexts: cooperative or pre-emptive. All Contiki programs are processes, which run in the cooperative context, whereas interrupts and real-time timers run in the pre-emptive context. A process is a piece of code that is run repeatedly by the OS. They are typically started when the system boots, or when a module that contains a process is loaded into the system. Processes run when something happens, such as a timer firing or an external event occurring.

Code running in the cooperative execution context is run sequentially with respect to other code in the cooperative context. Cooperative code must run to completion before other cooperatively scheduled code can run. Pre-emptive code may stop the cooperative code at any time. When pre-emptive code stops the cooperative code, the cooperative code will not be resumed until the pre-emptive code has completed. The pre-emptive context is used by interrupt handlers in device drivers and by real-time tasks that have been scheduled for a specific deadline.

The TestBed board support is fully integrated in Contiki build system. The Contiki system is designed to make it easy to compile Contiki applications either to a hardware platform or into a simulation platform by simply supplying different parameters to the *make* command, without having to edit makefiles or to modify the application code.

## 3.3 Overall System Performance Evaluation

### 3.3.1 Choice of Simulator

The simulator adopted for the purpose of evaluation is the Castalia based on the OMNeT++ platform. Castalia is a simulator for WSN and networks of low-power embedded devices. It is used to test their distributed algorithms and/or protocols in realistic wireless channel, with a realistic node behaviour especially relating to access of the radio. The main features of Castalia are:

- Advanced channel model based on empirically measured data.
    - Model defines a map of path loss, not simply connections between nodes
    - Complex model for temporal variation of path loss
    - Fully supports mobility of the nodes
    - Interference is handled as received signal strength, not as a separate feature
- Advanced radio model based on real radios for low-power communication.
    - Probability of the reception based on Signal to Inference plus Noise Ratio (SINR), packet size, Phase-Shift Keying (PSK) modulation type.
    - Frequency-Shift Keying (FSK) supported, custom modulation allowed by defining Signal to Noise – Bit Error Rate (SNR-BER) curve.
    - Multiple transmitter power levels with individual node variations allowed
    - States with different power consumption and delays switching between them

    - Realistic modelling of Received Signal Strength Indicator (RSSI) and carrier sensing
- Extended sensing modelling provisions
    - Highly flexible physical process model.
    - Sensing device noise, bias, and power consumption.
- Node clock drift
- MAC and routing protocols are available.
- Designed for adaptation and expansion.

The Castalia architecture as depicted in Figure 3.16 indicates the interconnections between the sensor nodes. The nodes are linked via the wireless channel module. The arrows indicate message passing from one module to another. Each node sends its packet to the wireless channel, which then selects the appropriate node(s) that should receive the packet. The nodes are also linked through the physical processes that they monitor.

There can be multiple physical processes, representing the multiple sensing devices (multiple sensing modalities) that a node has (http://castalia.npc.nicta.com.au). The node module is a composite one. Figure 3.17 shows the internal structure of the node composite module. The solid arrows signify message passing and the dashed arrows signify simple function calling. The application module is altered to simulate the Context based WSN reconfiguration model.

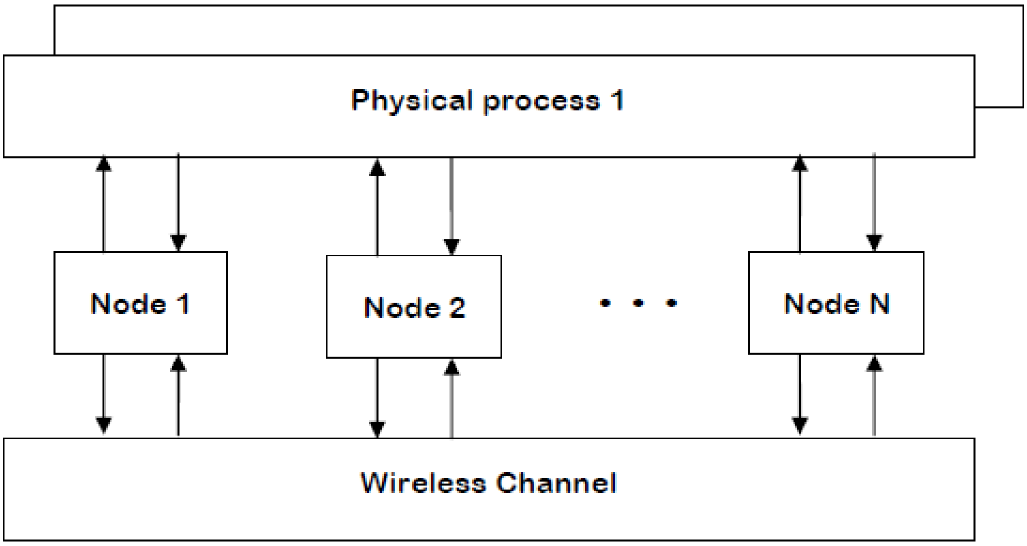


Figure 3.16: The Modules and their connections in Castalia (http://castalia.npc.nicta.com.au)

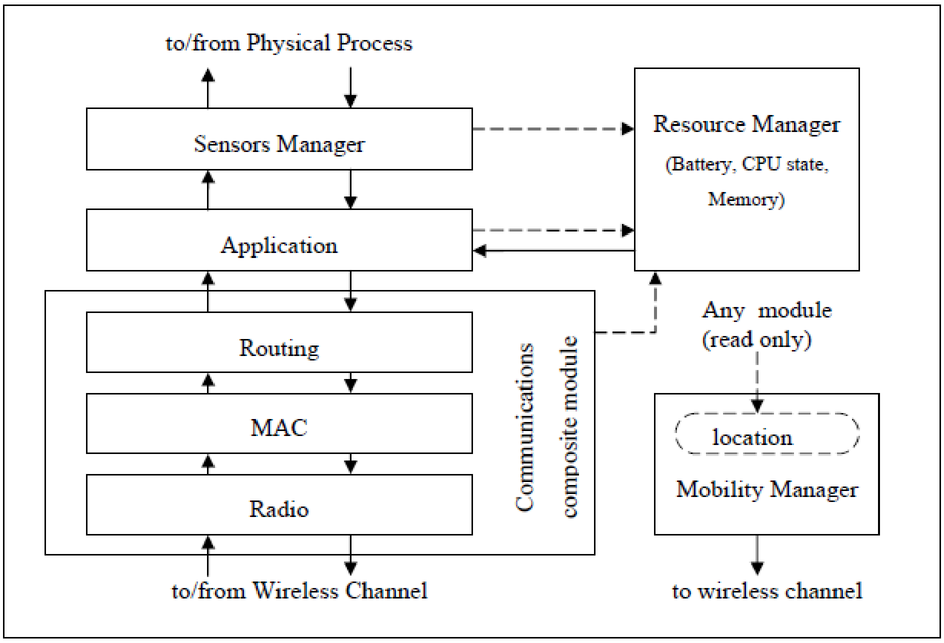


Figure 3.17: The node composite module (http://castalia.npc.nicta.com.au)

This structure depicted in Figures 3.16 and 3.17 were implemented in Castalia with the use of the OMNeT++ NED language.

### 3.3.2 Using OMNeT++ and Castalia Debugging and Reporting tool

The OMNeT++ simulation kernel records the message exchanges during the simulation into an *event log file*. This log file can be analysed later with the Sequence Chart tool. The Sequence Chart tool, and shows how the message is routed between the different nodes in the network. The sequence chart is valuable for debugging, exploring or documenting the complex model behaviour.

### 3.3.3 Simulation Setup

Adopting the Castalia framework, a network of six wireless sensor nodes was setup on the OMNeT++ platform. One of the nodes *SNode[0]* serves as the agent that links all the other sensor nodes to the base station, three of the nodes *SNode[2]* , *SNode[3] and SNode[5]* were programmed with context based reconfiguration capabilities and the remaining other two nodes *SNode[1]* and *SNode[4]* take on default reconfiguration paradigm.

The dataset obtained from the PDE and Fuzzy controller sub components were used to run the simulation platform. Erasure and writing energy, as well as the total energy consumption resulting from both erasure and writing operation were compared for the two sets of nodes. These results and the ensuing discussion are presented in chapter four.

### 3.4 Summary

The design, development, and evaluation procedures used to achieve the research work aim and objectives were outlined and discussed in this chapter.

The design process outlined the use of contextual information in reducing the cost of overheads during reconfiguration processes. Two categories of context related information were presented and discussed. These are the Application related context and the operational-demand related context. A design flow of how the two sets of information can be used to improve upon existing reconfiguration approaches were discussed in section 3.1.1.3. In order to use this information intelligently, the context information must be measurable and presented in a form that is concise so that appropriate decision on how and when best to effect a reconfiguration can be taken. Some of these information are generally imprecise, hence the selected decision-making system must have the capability to handle them. Two subcomponents devise to handle these requirements are the Precision Delta Extraction tool and a fuzzy logic controller. Details of these subcomponents' specifications, requirement, design, development and evaluation procedures were presented in this chapter.

# CHAPTER FOUR

## 4.0 RESULTS AND DISCUSSION

The results obtained while evaluating the context based WSN reconfiguration system were presented. First, the system's sub components comprising of the Precision Delta Extraction module and the Fuzzy logic Controller related results were presented in section 4.1 and 4.2 respectively. The overall system's performance is relayed in section 4.3. Related discussions of the results were presented in subsequent subsections 4.4, 4.4.1 and 4.4.2.

### 4.1 Precise Delta Extraction Tool

To evaluate the performance of the PDE, an application sample 'remotepowerswitch.c' built on the Contiki OS was used. The content of this sample application and other support files are listed in Appendix D. Changes effected at various source code' program structure were applied to each application's source code, each of the ensuing modified files paired with the original was compiled and their subsequent ELF files fed into the PDE. The Delta obtained, and other relevant information provided on the ELF profile form, are presented under related subsections 4.1.1.1, 4.1.1.2 and 4.1.1.3.

#### 4.1.1 ELF Profile of the 'Remote Power Switch' Sample Application

Using the ELF profile front end of the PDE, the constituents of the generated 'remotepowerswitch.elf' form in its original state (without any modifications) are presented in Table 4.1. The Profile's front end as shown in Figure 4.1 indicates where these constituents were obtained from. In addition, The ELF profile front end provides the following information:

i. A list of all loadable segments contained in the file. The information is obtained using Equation (3.9) as presented in chapter three.

ii. The virtual and physical start address of each segment.

iii. The total byte size of each segment

iv. Whether 'Execute', 'Read' or 'Write' operations are allowed in the listed segments.

v. It also indicates the unified Addressing scheme obtained to uniquely identify each data content within the ELF file.

Table 4.1: List of 'remotepowerswitch.elf' ELF constituents

| **Segment Number** | **Number of Sections** | **Segment Byte size** | **Segment Flags** |
|---|---|---|---|
| 0 | 465 | 79, 988 | Execute , Read |
| 1 | 65 | 2,152 | Execute , Read |
| 2 | 4 | 1,788 | Write, Read |
| 3 | 1 | 0 | Read |
| 4 | 15 | 8,344 | Write, Read |
| 5 | 2 | 912 | Execute , Read |
| 6 | 1 | 36 | Execute , Read |
| 7 | 1 | 4 | Execute , Read |
| 8 | 1 | 4 | Execute , Read |
| 9 | 1 | 4 | Execute , Read |
| 10 | 1 | 4 | Execute , Read |
| **Total bytes contained in File** | | 93,236 | |

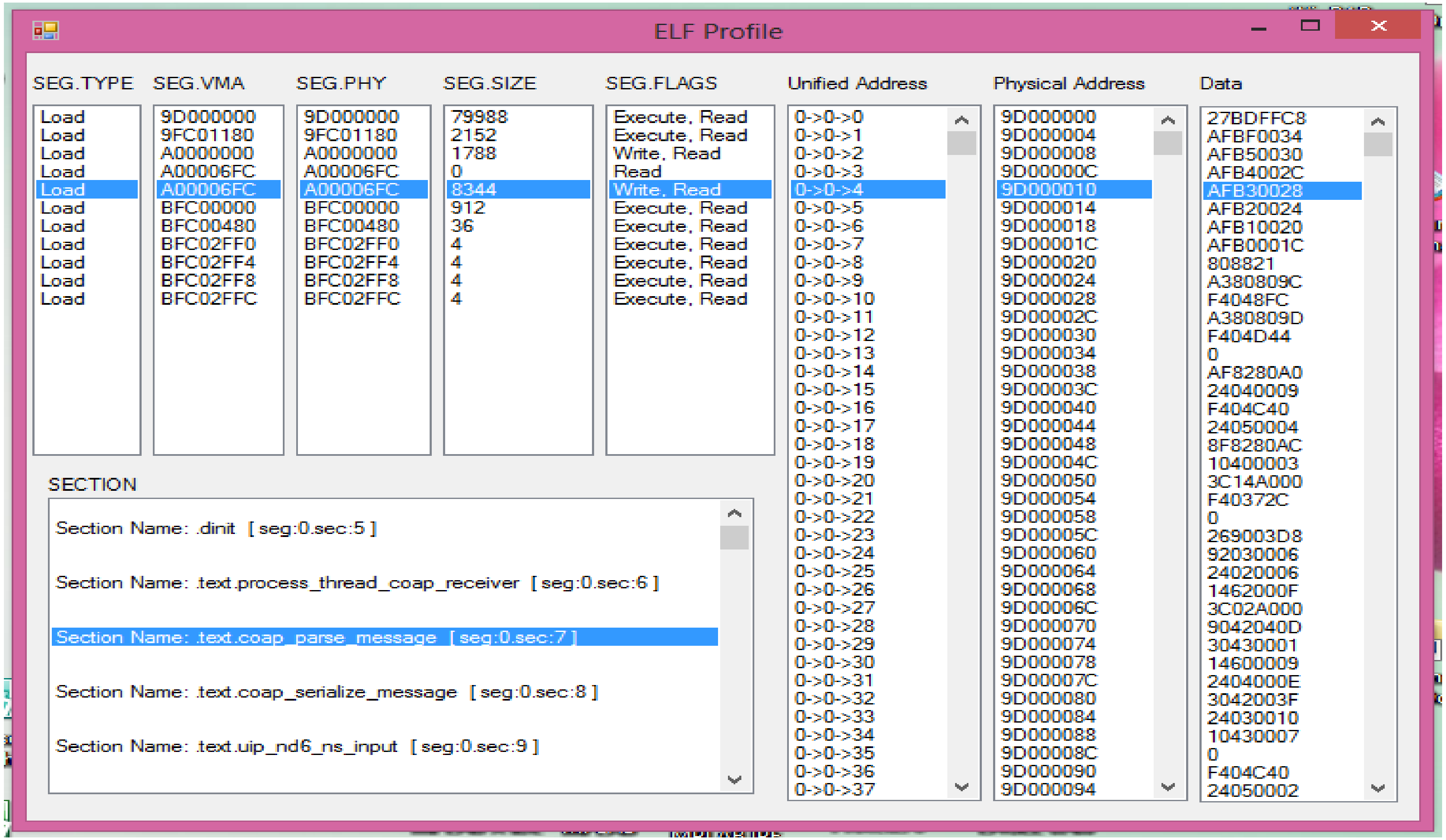


Figure 4.1: ELF profile of the ‘remotepowerswitch.elf’ file

4.1.1.1 **Case Study 1: Effecting Changes to 'Constant Data '**

Program Code Listing 1 and 2 show the highlighted section of the 'Led.c' source code where the change was made. In this case, the label definition 'LEDS_RED' used in the original source code has a value of '#2' as indicated in the header file 'Led.h' in Program Code Listing 1. The label definition was altered to take on a new value of '#4' represented by 'LEDS_YELLOW'. The two source codes (the original and the altered) were compiled and their generated ELF fed into the PDE. The delta obtained are illustrated in Figures 4.4, 4.5 and 4.6.

Program Code Listing 1: Extract from 'Led.h' showing values assigned to constant definitions used

```
#ifndef LEDS_GREEN
#define LEDS_GREEN  1
#endif /* LEDS_GREEN */
#ifndef LEDS_YELLOW
#define LEDS_YELLOW  2
#endif /* LEDS_YELLOW */
#ifndef LEDS_RED
#define LEDS_RED  4
```

Program Code Listing 2: Extract from 'Led.C' file showing original Constant Assignment (Case study1)

```
void
toggle_handler(void* request, void* response, uint8_t *buffer, uint16_t
preferred_size, int32_t *offset)
{
  leds_toggle(LEDS_RED);

  PORTEbits.RE0 = !PORTEbits.RE0;
}
```

Program Code Listing 3: Extract from ‘Led.C’ file showing modified Constant Assignment (Case study 1)

```
void
toggle_handler(void* request, void* response, uint8_t *buffer, uint16_t
preferred_size, int32_t *offset)
{
  leds_toggle(LEDS_YELLOW);
  PORTEbits.RE0 = !PORTEbits.RE0;
}
```

The delta listing in Figure 4.4 was obtained from the ‘modifiedRpt.txt’ and the initial values as presented in the ‘originalRpt.txt’ file is shown in Program Code Listing 4. Program Code Listing 4 depicts the alteration in the data content to be exactly one byte in size. The change occurs at unified address location ‘0->276->3’ and has a physical address value of ‘9D012014’. The extent of change does not affect the size of the entire firmware, and it is confined to just a segment in the program hence its orientation is of the segment confined type.

```
SECTION NO.: 276; SECTION NAME: .text.toggle_handler; NAME INDEX.:
2175 TYPE: ProgBits; LOAD ADDRESS: 9D012008; SIZE: 38

 |  0->276->0 |  9D012008 |  27BDFFE8
 |  0->276->1 |  9D01200C |  AFBF0014
 |  0->276->2 |  9D012010 |  F404C8F
 |  0->276->3 |  9D012014 |  24040004
 |  0->276->4 |  9D012018 |  3C02BF88
 |  0->276->5 |  9D01201C |  8C446110
 |  0->276->6 |  9D012020 |  30840001
 |  0->276->7 |  9D012024 |  2C840001
 |  0->276->8 |  9D012028 |  8C436110
```

Original Data value

Figure 4.2: Original Data value of the file before effecting changes (Case study1)

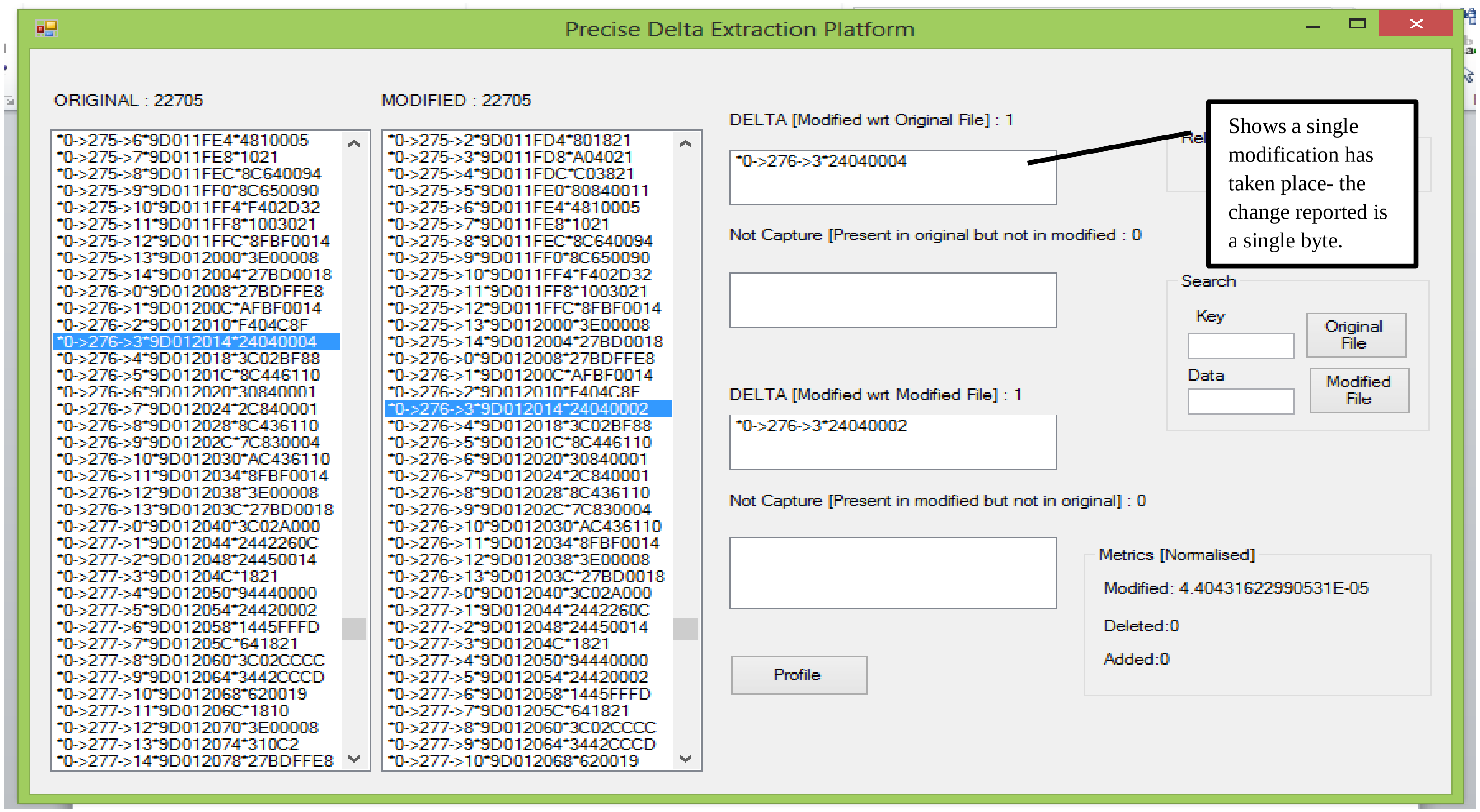


Figure 4.3: PDE display delta results obtained from Case study 1

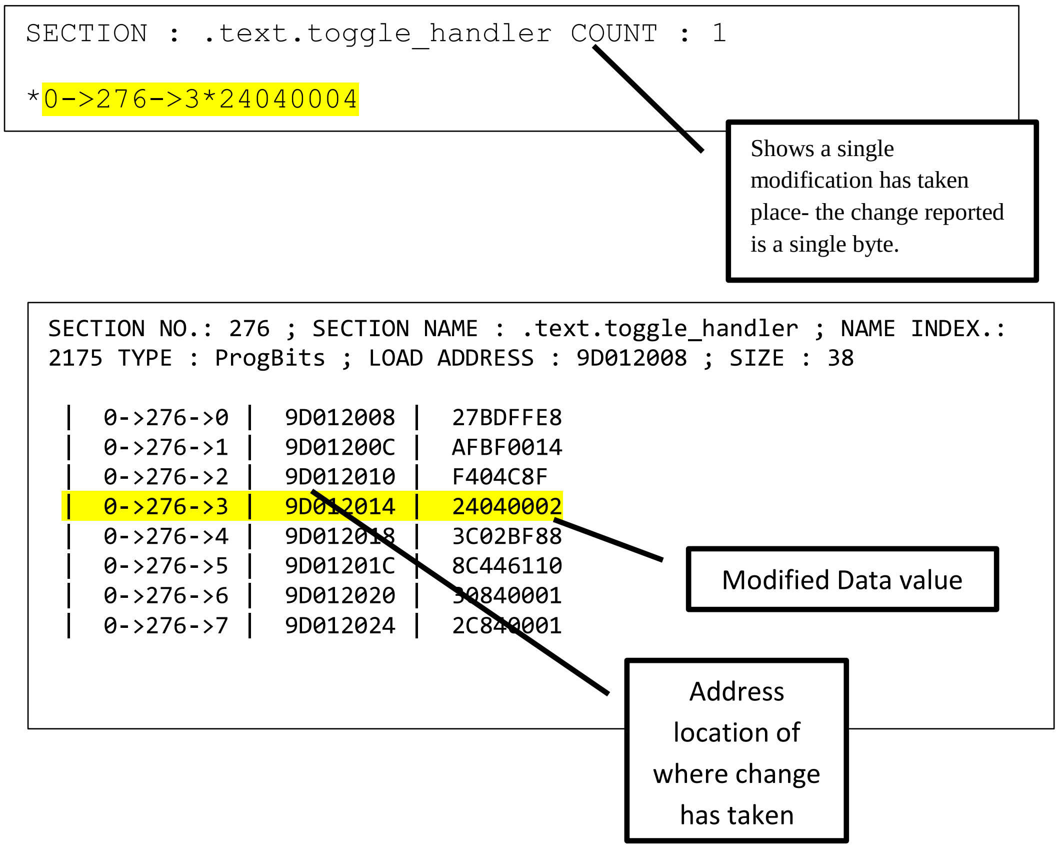


Figure 4.4: Modified value of Data the file after effecting changes (Case study 1)

#### 4.1.1.2 Case Study 2: Effecting Changes to 'Flow of Control'

Similar procedures carried out in the previous sub-section were repeated for a scenario where 'flow of control' construct is introduced in the main application's source codes. Program Code Listing 4 shows highlights of the introduced 'flow of control' construct. The isolated delta obtained were presented in Figure 4.5 and Figure 4.6. A collection of the delta is shown in Figure 4.5 while their distribution in the modified file is depicted in Figure 4.6.

Program Code Listing 4: Extract from ‘Led.C’ file showing the insertion of a ‘Flow of Control’ code construct (Case study 2)

```
toggle_handler(void* request, void* response, uint8_t *buffer,
uint16_t preferred_size, int32_t *offset)
{
  int decide = 0;

if (decide = 1)
 {
       leds_toggle(LEDS_RED);

              PORTEbits.RE0 = !PORTEbits.RE0;
       }
   else
       {
       leds_toggle(LEDS_YELLOW);

              PORTEbits.RE0 = !PORTEbits.RE0;

              }
}
```

```
 SECTION : .text.process_thread_remote_power_switch COUNT : 3

*0->150->6*24030059
*0->150->10*24020059
*0->150->24*24020059
```

Figure 4.5: Delta as reported in the modified File (Case study 2)

```
SECTION NO.: 150 ; SECTION NAME :
.text.process_thread_remote_power_switch ; NAME INDEX.: 142D TYPE :
ProgBits ; LOAD ADDRESS : 9D00F268 ; SIZE : 84

| 0->150->0  | 9D00F268 | 27BDFFE8
| 0->150->1  | 9D00F26C | AFBF0014
| 0->150->2  | 9D00F270 | AFB00010
| 0->150->3  | 9D00F274 | 94820000
| 0->150->4  | 9D00F278 | 10400006
| 0->150->5  | 9D00F27C | 808021
| 0->150->6  | 9D00F280 | 24030059
| 0->150->7  | 9D00F284 | 54430014
| 0->150->8  | 9D00F288 | A4800000
| 0->150->9  | 9D00F28C | B403CB3
| 0->150->10 | 9D00F290 | 24020059
| 0->150->11 | 9D00F294 | F4043E3
| 0->150->12 | 9D00F298 | 0
| 0->150->13 | 9D00F29C | 3C02BF88
| 0->150->14 | 9D00F2A0 | 8C436100
| 0->150->15 | 9D00F2A4 | 7C030004
| 0->150->16 | 9D00F2A8 | AC436100
| 0->150->17 | 9D00F2AC | 3C02BF88
| 0->150->18 | 9D00F2B0 | 8C436110
| 0->150->19 | 9D00F2B4 | 7C030004
| 0->150->20 | 9D00F2B8 | AC436110
| 0->150->21 | 9D00F2BC | 3C04A000
| 0->150->22 | 9D00F2C0 | F4042CF
| 0->150->23 | 9D00F2C4 | 2484257C
| 0->150->24 | 9D00F2C8 | 24020059
| 0->150->25 | 9D00F2CC | A6020000
| 0->150->26 | 9D00F2D0 | B403CB7
| 0->150->27 | 9D00F2D4 | 24020001
| 0->150->28 | 9D00F2D8 | 24020003
| 0->150->29 | 9D00F2DC | 8FBF0014
| 0->150->30 | 9D00F2E0 | 8FB00010
| 0->150->31 | 9D00F2E4 | 3E00008
| 0->150->32 | 9D00F2E8 | 27BD0018
```

Figure 4.6: PDE display delta results obtained from Case study 2

#### 4.1.1.3 Case Study 3: Effecting Changes to 'Function's Name'

In this case, changes were made to the original code by introducing some functions into the application's source code. The deltas obtained were quite large and were unevenly distributed in the program memory map. These changes as reported by the PDE are depicted in Figure 4.7.

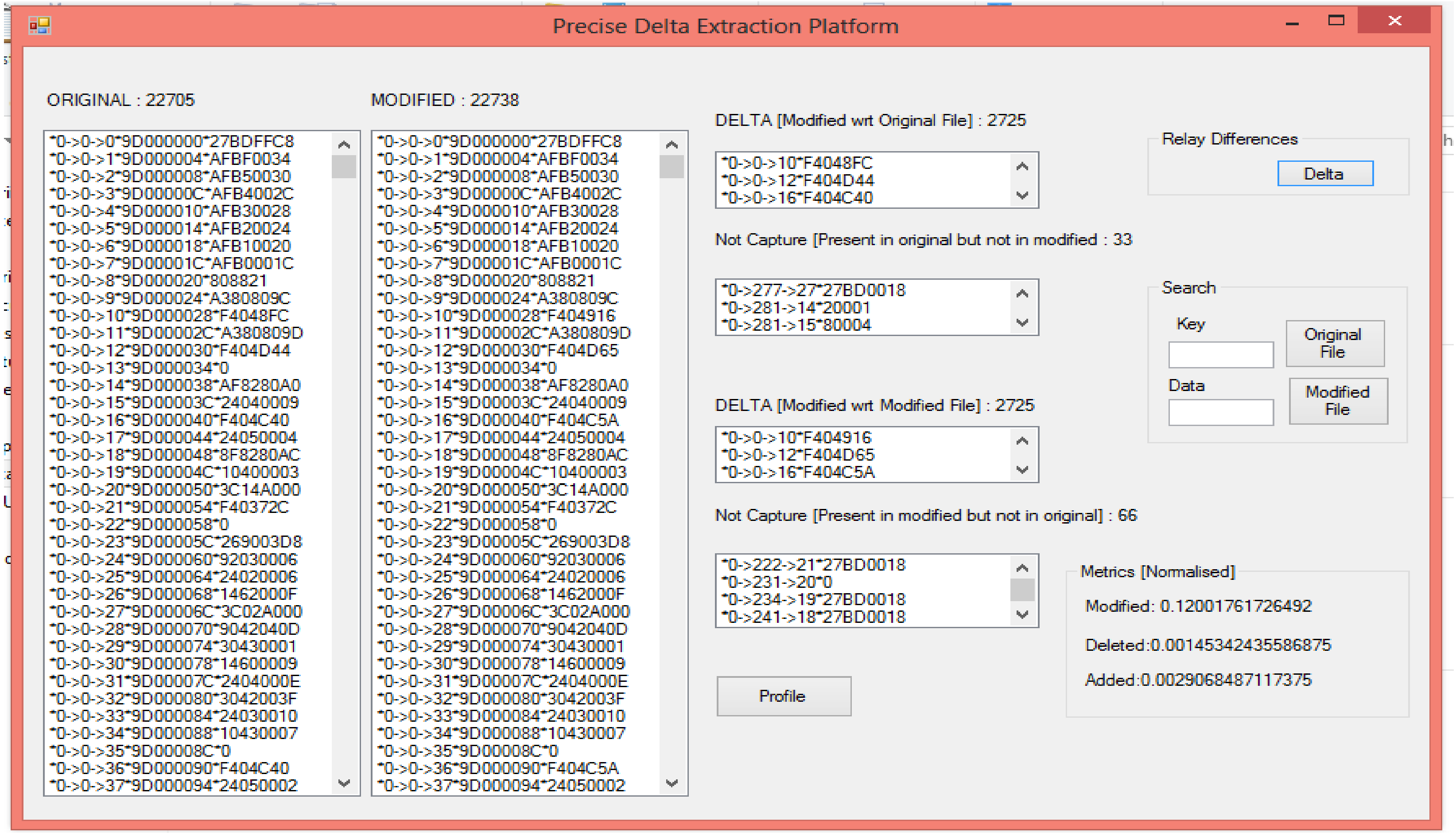


Figure 4.7 : PDE display delta results obtained from Case study 3

### 4.1.2 Summary of the results

A summary of the results obtained in the three case studies earlier presented above is shown in Table 4.2. The results were categorised under the following: the delta(s) size, the physical address range of the delta(s), related ELF segments where the delta resides, delta orientation and the number of segment(s) involved.

Table 4.2: A summary of results obtained for the three case studies

| Case study | Title | Size of changes in byte | Physical Address Range(s) | | ELF segment Name | Orientation of Change in Memory (Delta Orientation) | Number of Segment |
|---|---|---|---|---|---|---|---|
| | | | Start | End | | | |
| **1** | Effecting Changes to 'Constant' Data | 2 | 9D012014 | 9D012014 | .text | Segment confined | 1 |
| **2** | Effecting Changes to 'Flow of Control | 3 | 9D00F280 | 9D00F2C8 | .text | Segment confined | 1 |
| **3** | Effecting Changes to 'Function's Name' | 2725 | 9D000028<br>9FC01280<br>A00025FC<br>BFC00014<br>BFC02FF0 | 9D013884<br>9FC01984<br>A0002784<br>BFC00194<br>BFC02FF0 | .text<br>.vector<br>.data<br>.reset<br>.config_BFC02FF0 | Segments Disjointed | 5 |

## 4.2 Fuzzy Inference Engine

The fuzzy inference engine's performance is critical to achieving the set goals of every context based reconfiguration system for WSN. The details of its design have been extensively discussed in chapter three.  In order to demonstrate the designed fuzzy inference system's application and performance, two simulated scenarios were used.

The first scenario is based on case study one (1) earlier presented in section 4.1.1.2. Here, the delta size of two (2) bytes is defined to belong to the segment-confined membership function (delta-orientation). Varying the battery's energy level over the designated ranges as shown in Figure 4.8 always results in the 'Difference-approach' reconfiguration option being suggested (though of a degree of 0.001, even when the battery's energy level is at a critical level).

The second scenario is based on the case study three. The delta size obtained as indicated in Table 4.2 is 2725, though the size is a single segment range; the delta's orientation is of the disjointed nature. Figures 4.9, Figures 4.10 and Figures 4.11 show the various reconfiguration options selected based on varied battery's energy levels.
Table 4.3 shows the results obtained when the battery's energy level was varied across its selected corresponding membership functions.

Table 4.3: Results obtained for varied battery's energy levels

| SN | Battery state / Degree | Reconfiguration Option / Degree | Best Reconfiguration option/ Degree |
|---|---|---|---|
| **1** | Critical / 0.695 | Suspend reconfiguration / 0.670 | Suspend reconfiguration / 0.670 |
| **2** | Ok / 0.639 | Suspend reconfiguration / 0.168 | Entire-Image-Approach / 0.639 |
| | Fair / 0.168 | Entire-Image-Approach / 0.639 | |
| **3** | Very Ok / 0.758 | Entire-Image-Approach / 0.670 | Entire-Image-Approach / 0.670 |


Figure 4.8: Test demonstration based on case study1

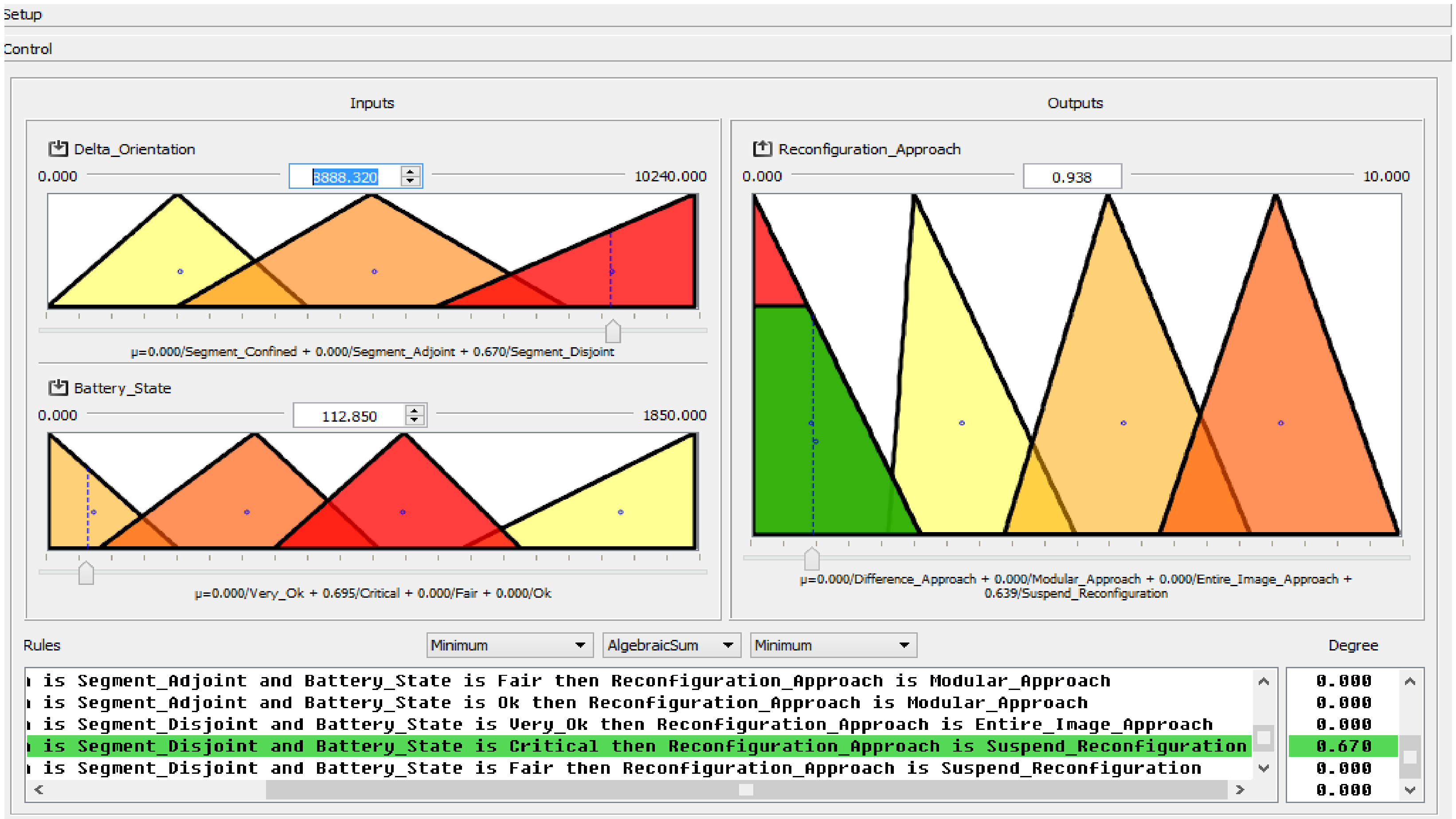


Figure 4.9: Test demonstration based on case study3 for battery state set as critical


Figure 4.10: Test demonstration based on case study 3 for battery state changing from Ok to fair

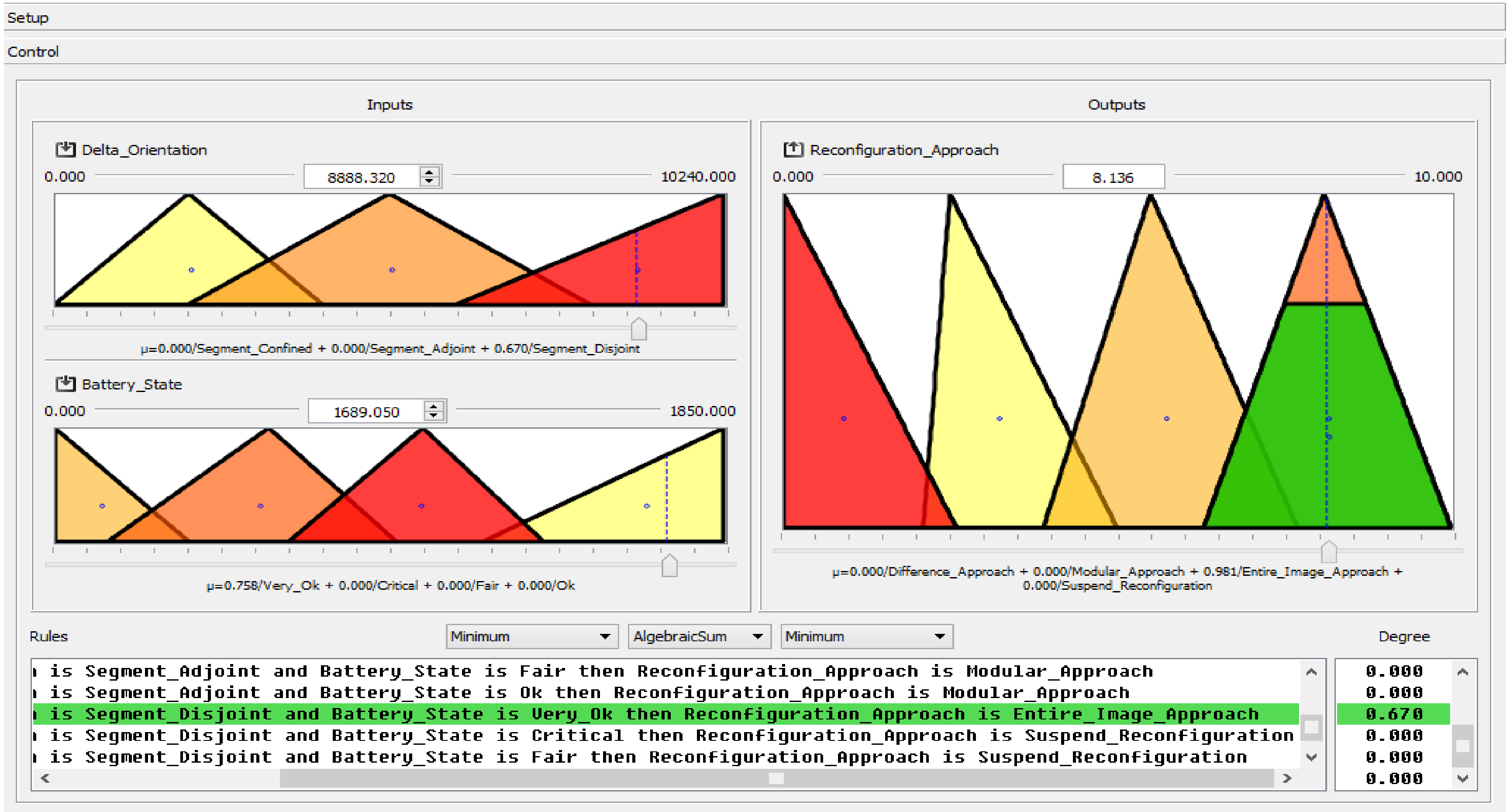


Figure 4.11: Test demonstration based on case study 3 for battery state set to Very Ok

## 4.3 Context-Based Reconfiguration system evaluation

Adopting the Castalia framework, a network of six wireless sensor nodes is setup on the Omnet++ platform (illustrated in Figure 4.12). One of the nodes *SNode [0]* is positioned to serve as the reconfiguration agent. The agent routes both Data and control messages from the base station to the other nodes (*SNode [1]*, *SNode [2]*, *SNode [3]*, *SNode [4]* and *SNode [5]*) in the network. Three of the nodes *SNode [2]*, *SNode [3]* and *SNode [5]* were programed with the context based reconfiguration capabilities and the remaining other two nodes *SNode [1]* and *SNode [4]* take on default reconfiguration paradigm. The set equipped with context based reconfiguration capabilities is tagged as group A while those with default reconfiguration paradigm tagged as group B.

In order to evaluate the benefits of the context-based reconfiguration model, each node in the simulation setup is configured to take on default values of energy consumed per byte and per segment during program memory reprogramming operations (read, erase and write procedures). The intent is to ascertain whether there is a significant difference in the amount of energy consumed by the two set of nodes. The parameters used in configuring each node are listed in Table 4.4. These parameters were used in computing the read, erasure and write energy consumption values in the omnet++ simulation platform. The values were adopted from the literature (Han-Lin, Chia-Lin, and Hung-Wei, 2008; Mathur, Desnoyers, Ganesan, Shenoy, 2006; Mohan, Bunker, Grupp, Gurumurthi, Stan, 2013,) and the datasheet of the testbed's microcontroller (PIC32MX320F128H).

The simulation procedure involves the transmission of a packet of data consisting of delta and control messages from the base station to each of the nodes (*SNode [1], SNode [2], SNode [3], SNode [4]* and *SNode [5]*) via *SNode [0]* as shown in Figure 4.12.

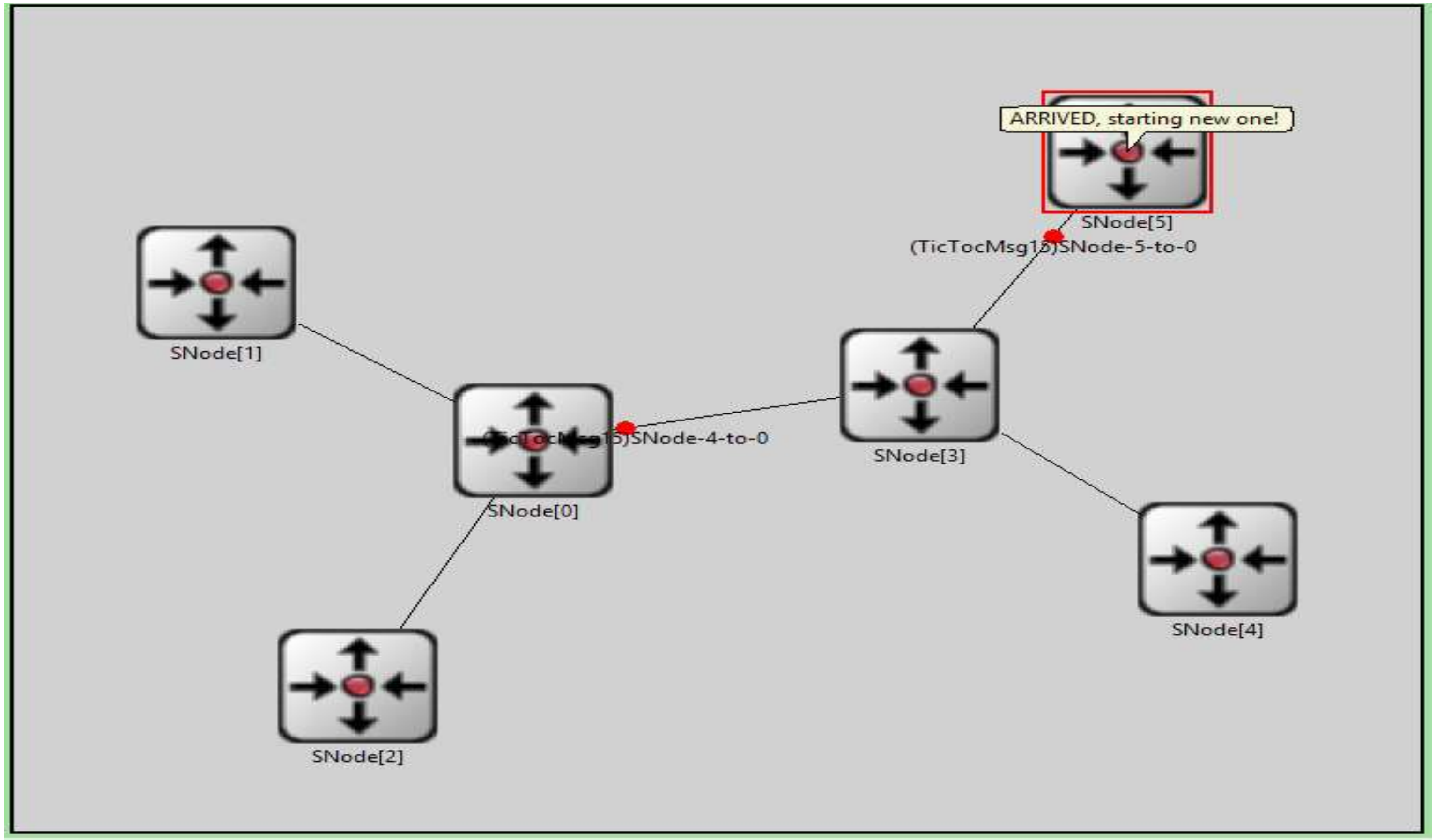


Figure 4.12: A network of six wireless sensor nodes setup on the Omnet++ platform

The sequence chart shown in Figure 4.13 illustrates the pattern of transmission and reception as implemented within the WSN. In addition, the sequence chart presents the history of the simulation carried out. The control message was derived from the output of the Fuzzy logic controller. The integration of the fuzzy logic inference engine into the omnet++ simulation platform was implemented via the use of fuzzilite dataset (extract is provided in Appendix C) generated using the qtfuzzilite tool.

Using a test delta size of 2725, delta orientation of the segment-confined type and battery energy state of 'very ok', the read, erasure, write and the total energy (a summation of read, erasure and write energy values) consumed were obtained and subsequently used to plot the graph shown in Figure 4.14. Similarly, using a test delta size of 2725, delta orientation of the segment-disjoint type and battery energy state of 'very ok', the read, erasure, write and the total energy ( a summation of read, erasure and write energy values) consumed were obtained and subsequently used to plot the graph shown in Figure 4.15.

Table 4.4: Flash Memory Characteristic (Han-Lin, Chia-Lin, and Hung-Wei, 2008)

| **Procedure Scope** | **Energy(μ J)** | | |
|---|---|---|---|
| | **Read** | **Write** | **Erase** |
| Per Byte | 0.004 | 0.009 | 0.047 |
| Segment/Page | 0.0679 | 7.66 | 192.2 |

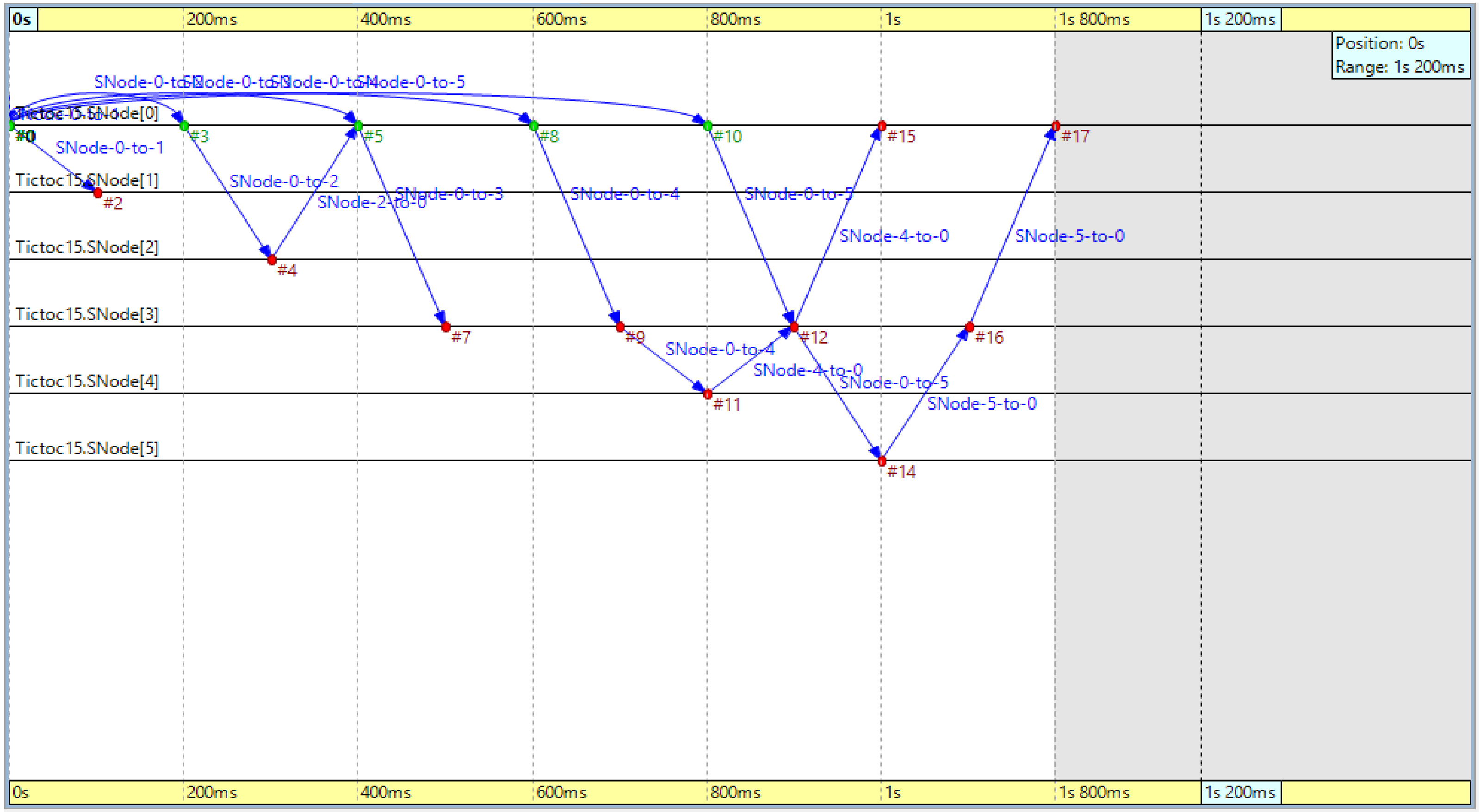


Figure 4.13: Sequence chart showing the history of the simulation carried out

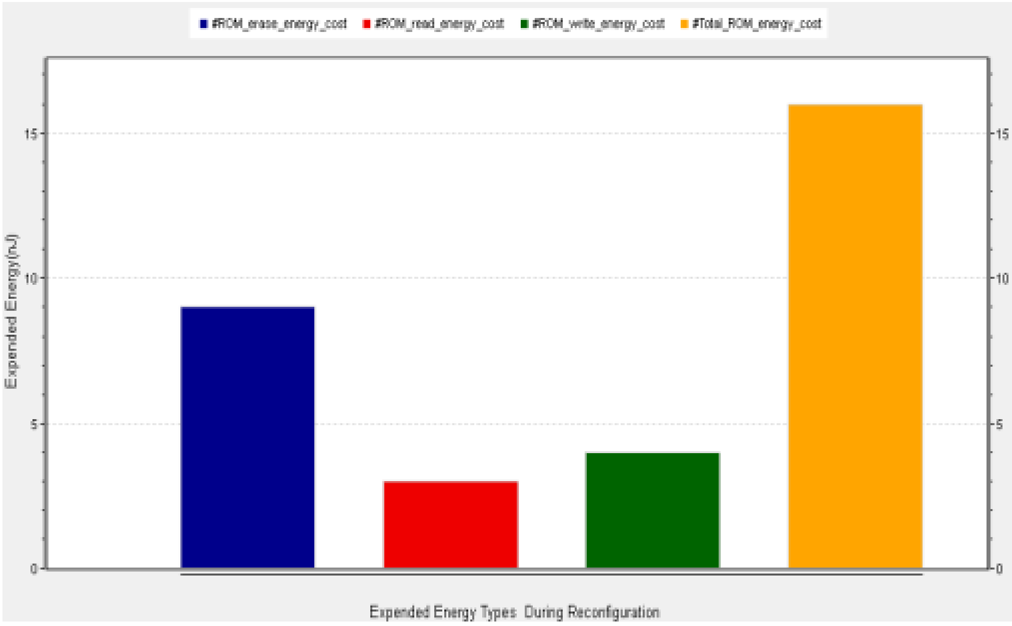


Figure 4.14: Graphical plot made for deltas' size less than program memory's segment Size, having segment-confined orientation type

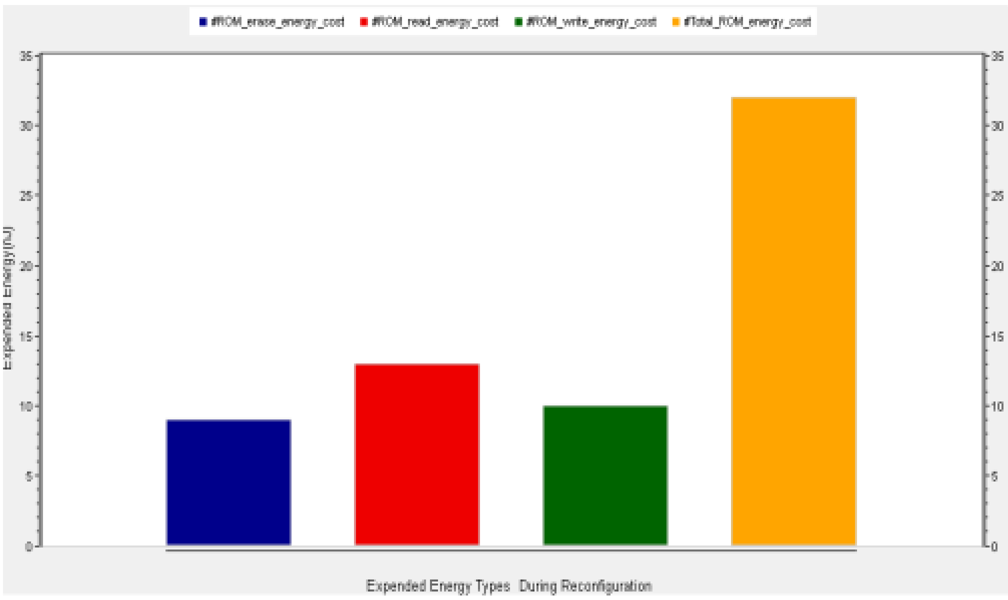


Figure 4.15: Graphical plot made for deltas' size less than the program memory's segment size, having segment-disjoint orientation type

## 4.4 PDE Utilisation Results and Discussion

The PDE isolates delta codes and provides information on the location in memory where appropriate changes are to be made in the new firmware. The information illustrated in Table 4.2 is useful in determining the size of delta involved and nature or characteristic of their distribution in the program memory.

In case study one, it is observed that the size of delta is a single byte, this very small change can mean a lot in real WSN applications. One typical example involves altering the rate at which a sensor samples data in the field or taking an average of the number of samples acquired. These changes in most cases are limited to single byte size or integer size. Using the conventional approach will involve the erasure and rewriting of the entire program memory space or a substantial amount of the memory space if a loadable reprogramming approach is employed.

In case study three, the delta distribution among segments in the flash memory is highly fragmented. These changes spread over five ELF segments, namely: (.text, .vector, .data, .reset, and .config_BFC02FF0). Even though the total number of bytes involved is relatively small (2725) compared to the actual memory size (128KB) of the PIC32MX320F128H microcontroller, the disjointed nature of the delta is best handled by reprogramming the entire available memory space.

The observations inferred from the above case studies were instrumental in devising an inference engine for the fuzzy logic subsystem employed in the context-based reconfiguration system for WSN.

### 4.4.1 PDE Compared to Existing Difference Reconfiguration Algorithms

The limitations attributed to the *difference-based* approach as highlighted in section of section 2.4.5.1 were resolved using the precision delta extraction scheme. The precision delta extraction scheme generate a unified address scheme, which concatenates the segment number, section number and the position of each data contained in the original image and the modified image file separately. The segment and section number are part of the Execution Link File format explained in Appendix A. The unified address scheme gives each set of data contained in the two files unique reference numbers that are similar. Hence, when any of the set of data is missing, its corresponding unified address ceases to exist, though its physical address might still exist, it will definitely point to another data. Similarly, when a set of new data is added, these new data acquire new unified addresses and invariably become easier to isolate.

This approach rules out the need to generate the pair (Checksum, MD5 hash) for each block of the old image and new image for comparison, which subsequently reduces the cost of implementing expensive computations in the base station. Though Panta, Bagchi and Midkiff (2011) tried to justify the use of the host computer in implementing their modified algorithm, issues of degrading performance occasioned by delay in delta dissemination can arise (especially in real time applications). Other variants of the Rysnc algorithm have been proposed and implemented: RDIFF (Milosh, Cuijipers and Lukkien, 2013), VCDIFF (Korn, MacDonald, Mogul and Vo, 2002), and BSDIFF (Percival, 2003). However, since they are derivatives of the original Rysnc algorithm, the lapses highlighted here are very much applicable.

### 4.4.2 Context-Based Reconfiguration system

The graph shown in Figure 4.14 was obtained for group A set of nodes where the delta orientation is of the segment-confined type. In conventional reprogramming procedures, the entire program memory is erased and rewritten all over again with a new image even if the delta (change) is a minute fraction of the entire program memory space. Hence the result obtained in Figure 4.15, also represents what is obtainable for the second group of nodes (group B) for the delta-orientation set as segment-confined. However, when the delta orientation is of the segment-disjoint type and irrespective of what the delta size is, both groups A and B set of nodes adopt the conventional reconfiguration approach. Therefore, the results obtained are similar to that indicated in Figure 4.15.

Comparing the two graphs indicates that 65% of energy expended during the erasure procedure is saved when the context based reconfiguration model is adopted. Similarly, 45% and 69% reduction in energy consumption were obtained for the read and write procedures respectively. The implication is that quite a considerable amount of energy is wasted when very minute deltas with segment-confined orientation are involved.

Additional contextual information are applicable. For example, the signal strength of each sensor node may vary over space and time. These can negatively affect the reconfiguration process especially where retransmission occurs severally due to poor signal reception occasioned by poor weather conditions. In such situation, the norm is to stop reconfiguration completely. However, in a context based reconfiguration approach, if the delta detected is relatively small that it can be handled with much less resources expended, then the reconfiguration process is allowed to take place.

## 4.5 Summary

In this chapter, the results obtained while evaluating the context based WSN reconfiguration system were presented and discussed

The roles of the PDE tool and that of the fuzzy logic controller in implementing the context based reconfiguration model were demonstrated. The fuzzy logic system ensures that reprogramming operations are only allowed when the conditions are right. The condition in this case depends on two contexts:  the nature and location of the delta in program memory and the state of energy available in a wireless sensor node’s battery. WSN reconfiguration related energy cost can be reduced considerably when both application and operational-demand related contexts are taken into consideration.

# CHAPTER FIVE

## 5.0 CONCLUSION AND RECOMMENDATIONS

### 5.1 Conclusion

In this research a software model that dynamically reconfigures wireless sensor network operational functionalities optimally based on evolving application context was developed. In realising the aim and objectives of the research work, a detailed review of existing reconfiguration approaches was conducted. The review findings highlighted the lapses associated with various reconfiguration approaches.

The difference approach appears to be the most efficient reconfiguration paradigm to adopt, especially when the delta values are relatively small compared with the original firmware. However, the minimum size of the program memory that can be erased places some restriction on when such approach should be adopted. Hence, the limitation attributed to FLASH Memory invariably plays down on the advantages of the difference approach. As such in some cases, it is much better to erase the entire flash memory. The way round the problem is the adoption of some form of intelligence that controls or directs the sensor nodes to adopt the most appropriate and less energy consuming reconfiguration approach. This has been demonstrated via the use of Fuzzy logic system to enhance the sensor nodes ability to decide what reconfiguration paradigm to adopt under certain context. In addition, a novel software component that efficiently reprograms flash program memory while taking into consideration the 'segment erasure' constraint has been developed.

In realising the Context-based WSN reconfiguration software system, two main software components were developed. These three components are the Precise Delta Extractor (PDE), Efficient Program memory Re-flashing module and the Fuzzy logic Controller. The first provides information on the degree of changes resulting from the modification of an application as well as relaying the exact changes in bytes form. In addition, it provides fuzzified member set inputs (application context) to the fuzzy logic controller. The second component developed is based on expert knowledge of the energy consumption constraints associated with reprogramming procedures of wireless sensor nodes' program memory. In addition, an algorithm intended to address reprogramming constraints associated with flash program memory was developed.

The PDE Metric tool developed is an improvement over existing similar tools like the Rysnc and its variants. The PDE does not need tuning in order to reduce the overheads associated with Rysnc and its variants. The PDE provides concise physical address and virtual address of deltas. This information is useful for targeting delta locations and allowing reconfiguration procedures to be confined within a single segment of the Flash memory thereby saving enormous amount of energy expended when an entire program memory is reprogrammed.

In order to demonstrate the benefits of the context based reconfiguration model, its performance was evaluated on an Omnet++ simulation platform using pilot data obtained from a testbed. The testbed is composed of Microchips' PIC32MX320F128H microcontroller and MRF24J40MB transceiver. A two context related input variables were used. The delta-orientation information obtained from the ELF profile of the modified code served as the application related context and the Battery energy level state was taken as an operational-demand related context. Inferred expert knowledge on energy consumption pattern during reconfiguration processes was used to develop a robust inference engine for the fuzzy logic controller. The resulting output from the fuzzy logic system controls when and which one of the reconfiguration approaches should be implemented in order to prolong the battery life. In a network of six nodes, two were equipped with the developed model capability and the others were not. The overall energy expended as read, erase and write were obtained from each node for the purpose of comparison. The results obtained show that about 65% of energy expended during the erasure procedure is saved in nodes that adopt the context based reconfiguration model. Similarly, 45% and 69% reduction in energy consumption were obtained for the read and write procedures respectively.

## 5.2 Contribution to Knowledge

i. Developed a much effective delta extraction algorithm and tool for the Contiki operating system adapted to adopt the difference reconfiguration approach.

ii. Demonstrated the use of artificial intelligence (fuzzy logic) in wireless sensor network application to enable it manage its limited available resources efficiently during reprogramming procedures.

iii. The model developed serves as a framework for developing an all-encompassing context base reconfigurable wireless sensor network.

## 5.3 Recommendations

i. Use of other Artificial Intelligence (AI) models like Artificial Neural Networks (ANN) should be explored. The fuzzy logic system is based on human reasoning, and it means modifications will need to be made to the inference engine intermittently as new application and operational context changes evolve.

However, an AI with real-time learning and adaptive capabilities allows the system to update itself.

ii. Adoption and Implementation of multiple contexts variables is encouraged. For example, the influence of Interference, Signal strength on reconfiguration process in real life field situation can be explored.

# REFERENCES


Adomat, J., Furunäs, J., Lindh, L., & Stärner. J. (1996). Real-Time Kernel in Hardware RTU: A step towards deterministic and high performance real-time systems. *In Proceedings of Eighth Euromicro Workshop on Real-Time Systems*, 'Aquila, Italy. 164-168.

Andrews, D., Niehaus, D., & Ashenden, P. (2004). Programming Models for Hybrid FPGA/CPU computational Components , *IEEE Computer*, 2004(1), 118-120.

Alkhawaja, A.R., Ferreira, L.L. & Albano, M. (2012). Message Oriented Middleware with QoS Support for Smart Grids. *Technical Report HURRAY-TR-120709*. Retrieved from: http://www.cister.isep.ipp.pt/docs/message_oriented_middleware_with_qos_support_for_smart_grids/717/view.pdf, Retrieved 27 July 2012

Balani, R., Han, C., Rengaswamy, R. K., Tsigkogiannis, L. & Srivastava, M. (2006). Multi-Level Software Reconfiguration for Sensor Networks. *Proceedings of the 6th ACM & IEEE International conference on Embedded software.* 112-121, Doi: 10.1145/1176887.1176904,

Baldauf, M., Dustdar, S., & Rosenberg, F. (2007). A Survey on Context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing,* 2(4), 263-277.

Bhatti, S., Carlson, J., Dai, H, Deng, J. Rose, J., Sheth,A., Shucker, B., Gruenwald, C., Torgerson, A., and Han, R. (2005). MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. *ACM/Kluwer Mobile Networks & Applications(MONET), Special Issue on Wireless Sensor Networks*, Aug. 2005, 10(4), 563-579.

Barr, M. (1999). *Programming Embedded Systems in C and C++, First Edition*. USA:O' Reilly.

Belanger, L. (2004). Development of a GSM Modem on a DSP/FPGA Architecture Using Simulink and System Generator, *Xcell Journal*, Lyrtech inc. 4(51), 1-18.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., &d Riboni, D.(2010). A Survey of Context Modeling and Reasoning Techniques. *Journal of Pervasive and Mobile Computing*. 6(2), 161-180.

Black, J., Cochran, M. & Highland, T. (2008). A Study of the MD5 Attacks: Insights and Improvements, 3 March 2006. Retrieved 27 July 2008.

Brown, S., Sreenan, C.J. (2006). Updating Software in Wireless Sensor Networks: A Survey. *Technical Report UCC-CS-2006-13-07*, Ireland.

Burleson (1993). The Spring Scheduling Co-Processor: A Scheduling Accelerator, *Proceedings of the International Conference on Computer Design (ICCD)*, 140-144.

Cafaro, G., Gradishar, T. & Guimaraes, H. (2009). A 10Mhz-4Ghz Direct Conversion CMOS Transceiver for SDR Applications. *Proceedings of the SDR 2009 Technical Conference and Product Exposition.*

Carpenter, T., & Barrett, J. (2008). *CWNA Certified Wireless Network Administrator Official Study Guide. Fourth Edition*, USA:McGraw-Hill.

Chen, Y., Chein, T., & Chou, P. (2010). Enix: A Lightweight Dynamic Operating System for Tightly Constrained Wireless Sensor platforms. *In SenSys '10 ( November 3-5, Zurich, Switzland), ACM*.

Chong, C. & Kumar, S. P. (2003). Sensor Networks: Evolution, Opportunities and Chellenges. *Proceedings of the IEEE,* 91(8), 1247-1256.

Cingolani, P. & Alcala-Fdez, J. (2012). jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation. *In Proceedings of the IEEE International Conference on Fuzzy Systems*, 1–8.

Ciampa, M. (2009). *CompTIA Security+ 2008 in depth*. Australia ; United States: Course Technology/Cengage Learning. 290.

Compton, K., & Hauck, S. (2002). Reconfigurable Computing: A survey of Systems and Software. *ACM Computing Surveys*, 34(2), 171-120.

Costa, N., Pereira, A. & Serodio, C. (2007).Virtual machines applied to WSN's: The state-of-the-art and classification. *In Proc. the 2nd International Conference on Systems and Networks Communications (ICSNC 07),* Cap Esterel, French, Riviera, France.

Dong, W., Chen, C., Liu, X.,& Bu, J. (2010). Providing OS Support for Wireless Sensor Networks: Challenges and Approaches, *IEEE Communications Surveys and Tutorials,* 2010, **12**(4), 519-530.

Dong, W., Chen, C., Bu, J. & Huang, C. (2013). Enabling efficient reprogramming through reduction of executable modules in networked embedded systems, *Journal of Ad Hoc Networks*. 11(1). 473-489, doi>10.1016/j.adhoc.2012.07.007.

Dong, W., Liu, Y., Chen, C., Bu, J., Huang, C. & Zhao, Z. (2011). R2: Incremental Reprogramming using Relocatable Code in Networked Embedded Systems. IEEE INFOCOM conference proceedings, 376 - 380

Duffy, C., Utz, R., John, H. & Sreenan, C. (2008). An Experimental Comparison of Event Driven and Multi-Threaded Sensor Node Operating System. *In proceedings of the 5th Annual IEEE International Conference of Pervasive Computing and Communications, (PERCOMW*). 267-271.

Dunkels, A., Gr¨onvall, B. & Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. *Proceedings of the First IEEE Workshop on Embedded Networked Senso*rs, Tampa, Florida, USA.

Dunkels, A., Finne, N., Eriksson, J. & Voigt, T. (2006). Run-time dynamic linking for reprogramming wireless sensor networks, In Proceedings of ACM SenSys,

Engelbrecht, A. (2007) Computational Intelligence: An Introduction, 2nd ed. New York, USA: John Wiley & Sons.

Eronu, E. M., Misra, S. & Aibinu, M. (2013). Reconfiguration Approaches in Wireless Sensor Network: Issues and Challenges. *In proceedings of the Second IEEE International Conference on Emerging & Sustainable Technologies for Power & ICT in a Developing Society (NIGERCON), 2013,* Owerri, Imo state, Nigeria, November 14 - 16, 143 – 152. Doi:10.1109/NIGERCON.2013.6715648. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6715648&isnumber=6715629

Flowers, D. & Yang, Y. (2010). Microchip MiWiTH Wireless Networking Protocol Stack, *Microchip Technology Inc. DS0166B*, 1-18, Retrieved on March 10, 2012 from: http://www.microchip.com/miwi.

Fuentes, L. & Gámez. N. (2011). FamiWare: A Family of Event-Based Middleware for Ambient Intelligence. *Pers. Ubiquitous Comput.* 2011(15). 329–339.

Gámez, N., Cubo, J., Fuentes, L. & Pimentel, E. (2012). Configuring a Context-Aware Middleware for Wireless Sensor Networks. *Sensors (Open Access Journal),* 2012(12). .8544-8570. Doi:10.3390/s120708544.

Gomaa, H. (1993). *Software Design Methods for Concurrent and Real-time Systems, First edition*, USA:Addison-Wesley.

Graziosi, F., Pomante, L. & Pacifico, D. (2008). A Middleware-based approach for heterogeneous wireless sensor networks. *Proceedings of International Conference on Communications(iccom'08*), 52-57.

Gaurav, M., Peter, D., Deepak, G. & Prashant, S. (2006). Ultra-low power data storage for sensor networks, *Proceedings of the 5th international conference on Information processing in sensor networks*, April 19-21, Nashville, Tennessee, USA. Doi:10.1145/1127777.1127833.

Han, C., Kumar, R., Shea, R. & Srivastava, M. (2005). Sensor Network Software Update Management: A Survey. *Department of Electrical Engineering University of California*, Los Angeles.

Han, C., Kumar, R., Shea, R. Kolher, E. & Srivastava, M. (2008). A Dynamic Operating System for Sensor Nodes, *Proceedings of the 3rd International Conference on Mobile Systems Applications and Services, ACM*. 163-176.

Han-Lin, L., Chia-Lin, Y. & Hung-Wei, T. (2008). Energy-Aware Flash Memory Management in Virtual Memory System, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(8), 952-964.

Hadim, S. & Mohamed, N. (2006). Middleware for wireless sensor networks: A survey. In Proc. *the 1st International Conference on Communication System Software and Middleware (Comsware06),* India.

Haykin, S. (1994). Neural networks: A comprehensive foundation. Prentice Hall, 1994.

Heron, J.P., Woods, R., Sezer, S. & Turner, R.H. (2001). Development of a Run-Time Reconfiguration System with Low Reconfiguration Overhead, *Journal of VLSI Signal Processing*, 28, 97-113.

Hill, J., Horton, M., Kling, R. & Krishnamurthy, L. (2004). The Platforms Enabling Wireless Sensor Networks, *Communications of the ACM*, 47(6).

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., & Pister, K. (2000). System architecture directions for networked sensors. *In Proceedings of the 9th Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX),* New York, NY, USA, ACM Press.

Hill, J., Szewczyk, R., Woo, A., Levis, P., Madden, S., Whitehouse, J.P., Polastre, J., Gay, D., Sharp, C., Welsh, M., Brewer, E. & Culler, D. (2005). TinyOS: An Operating System for Sensor Networks. *Ambient Intelligence,* Heidelberg, Netherland: Springer-Velag, 115-148.

Hinkelmann, H., Reinhardt, A., & Glesner, M. (2008). A Methodology for Wireless Sensor Network Prototyping with Sophisticated Debugging Support. *International Symposium on Rapid System Prototyping*.

Hu, P., Ndulska, J. & Robinson, R. (2006). Reconfigurable middleware for sensor based applications. *Proceedings of the 3rd international Middleware doctoral symposium*. New York, NY, USA. Doi:10.1145/1169100.1169105.

Hui, J. W. & Culler, D. (2004). The Dynamic behavior of data dissemination protocol for network programming at scale. *Proceedings of the 2nd International Conference on Embedded networked Sensor systems (SenSys).* 81-94.

Ibrahim, D. (2008). *Advanced PIC Microcontroller projects in C,* New York, USA :Newness.

Indulska, J. & Sutton, P. (2003). Location management in pervasive systems. *CRPITS'03: Proceedings of the Australasian Information Security Workshop*, 143–151.

Jeong, J., Kim, S. & Broad, A. (2008). Network reprogramming, TinyOS documentation, available from : http://www.tinyos.net/tinyos-1-x/doc/xnp.pdf, (10.01.2012)

Jun-Zhao, S. (2010). OS-based Reprogramming Techniques in Wireless Sensor Networks: A Survey. *In proceedings of Ubi-media Computing (U-Media), 3rd IEEE International Conference*. 17-23

Karloff, C., Sastry, N. & Wagner, D. (2004). TinySec: A link Layer Security Architecture for Wireless Sensor Networks, *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems*.

Khan, J., & Vemuri, R. (2005). Energy management in battery powered sensor networks with reconfigurable computing nodes. *In Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '05)*. 543-546.

Krishna, R. P., Bagchi, S., & Khalil, M. I. (2009). Efficient wireless reprogramming through reduced bandwidth usage and opportunistic sleeping, *Ad Hoc Networks*. **7**(1), 42-62.

Krishna, R. P., Bagchi, S. & Midkiff, P. S. (2009). Zephyr: Efficient Increamental Reprogramming of Sensor Nodes using Function call Indirections and Difference Computation, *In Proceedings of the Annual Technical Conference (USENIX)*. 411-424.

Krishnamurthy, L., Adler, R., Buonadonna, P., Chhabra, J., Flanigan, M., Kushalmager, N., Nachman, L., & Yarvis, M. (2005). Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the North Sea, in: *Proceedings of the Third International Conference on Embedded Networked Sensor Systems (Sensys),* San Diego, CA.

Kompis, C. & Sureka, P. (2010). Power Management Technologies to Enable Remote and Wireless Sensing Cyber Security White Paper, Retrieved on May,02 2010 from: *www.libelium.com/libelium.../libelium-ktn-**ower_management**.pdf.*

Korn, D., MacDonald, J., Mogul, J. & Vo, K. (2002). The VCDIFF Generic Differencing and Compression Data Format." RFC 3284 (Proposed Standard).

Kulkarni, K., .Sanyal, S., Al-Qaheri, H. & Sanyal, S. (2009). Dynamic Reconfiguration of Wireless Sensor Networks. *International Journal of Computer Science and Applications*, 6(4), 16-42.

Kulkarni, R. V., Forster, A., & Venayagamoorthy, G. K. (2011). Computational intelligence in wireless sensor networks: A survey. *Communications Surveys & Tutorials, IEEE, 13*(1), 68-96.

Kim, S., Lee, J., Hur, K., Hwang, K., & Eom, D.(2009). Tiny Module–Linking for Energy–Efficient Reprogramming in wireless sensor networks. *IEEE Transactions on Consumer Electronics*. **55**(4), 1914-1920.

Kjær, K.E. (2007). A Survey of Context-aware Middleware. *Proceedings of the 25th conference on IASTED International Multi Conference: Software engineering*, 148-155.

Labroasse, J., (2002). MicroC/OS-II The Real Time Kernel, *Newnes*.

Lalit, S. & Yadav, P. S. (2010). A Comparative Analysis of Wireless Sensor Network Operating Systems. *International Journal and Technical science*, 1(1), 41-47.

Lee, J., Mooney, V., Instrom, K., Daleby, T., Klevin, T. & Lindh, L. (2003). A comparison of the RTU Hardware RTOS with a Hardware/Software RTOS, *Proceedings of Asia and South Pacific Design Automation Conference*, Asia.

Leekwijck, W. V. & Kerre, E. E. (1999). Defuzzification: criteria and classification. *Fuzzy Sets and Systems*. 108(2), 159–178.

Leligou, H. C., Redondo, L., Zahariads, T., Retamosa, D. R., Karkazis, P. Papaefstathiou, I. & Voliotis, S. (2008). Reconfiguration in Wireless Sensor Networks. *ARTEMIS-2008-100032, SMART 2008.*

Levis, P., and Culler, D. (2002). Maté: a tiny virtual machine for sensor networks, In ASPLOX-X: *Proceedings of the 10th International Conference on Architectural support for programming languages and operating systems*. 85-95.

Levis, P., Patel, N., Culler, D. & Shenker, S. (2004). Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. *In proceedings of First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI).*

Linn, Y. (2009). Generation of Bandpass Guassian Noise with Application to Complete Built in Self-Test in Wireless Communications Receiver. *In proceedings of IEEE LATINCOM.*

Luk, M., Mezzour, G., Perri, A. & Gligor, V. (2007). MiniSec: A Secure Sensor Network Communication Architecture. *IPSN'07, April 2007*, Cambridge, Massachusetts, U.S.A.

Marron, P. J., Gauger, M., Lachenmann, A., Minder, D., Saukh, O. & Rothermal, K. (2006). FlexCup: A Flexible and Efficient Code Update Mechanism for Sensor Networks, Springer-Verlag Berlin Hedelberg, 2006(3868), 212-227.

Markowsky, L. & Segee, B. (2011). The octave fuzzy logic toolkit. *In Proceedings of the International Workshop on Open-Source Software for Scientific Computation*, 118–125.

Mallikarjuna, A., Reddy V., Kumar, P., Janakiram, D. & Kumar, A. (2009). Operating Systems for Wireless Sensor Networks: A Survey Technical Report. *International Journal of Sensor Networks (IJSNet).* 5(4),.236-255.ACM.

Macdonald, J. (2011). Xdelta - open-source binary diff.

Mamdani, E. H. & Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1),1–13.

Mathur, G., Desnoyers, P., Ganesan, D. & Shenoy, P. (2006). Ultra-low power data storage for sensor networks, *Proceedings of the 5th international conference on Information processing in sensor networks*, April 19-21, Nashville, Tennessee, USA , doi:10.1145/1127777.1127833

Misra, S. & **Eronu, E.** (2012). Implementing Reconfigurable Wireless Sensor Networks: The Embedded Operating System Approach, Embedded Systems - High Performance Systems, Applications and Projects, ISBN: 978-953-51-0350-9, InTech, Retrieved on October 12, 2012: *http://www.intechopen.com/books/embedded-systems-high-performance-systems-applications-and-projects/implementing-dynamic-reconfigurable-wireless-sensor-networks-*

Milosh, S., Cuijipers, P. J. & Lukkien, J. J. (2013). Efficient reprogramming of wireless sensor networks using incremental updates and data compression. *In international conference of Pervasive Computing and Communications Workshops(PERCOMWorkshops),*584-589 doi:10.1109/PerComW.2013.6529563

Moerschel, G., Dreger, R. & Carpenter, T. (2007). *CWSP Certified Wireless Security Professional Official Study Guide, Second Edition*. New York, USA : McGraw-Hill.

Mohan, V., Bunker, T., Grupp, L., Gurumurthi, S. & Stan, M. R. (2013), Modeling Power Consumption of NAND Flash Memories Using *FlashPower, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 32(7), 1031 – 1043.

Mooney, V. & Blough, D. M. (2002). A Hardware-Software Real-Time Operating System Framework for SOC's, *IEEE design and Test of Computers*, 2002(11). 44-51.

Muralidhar, P. & Rao, C. B. R. (2008). Reconfigurable Wireless sensor network node based on NIOS core. *In processings of the 4th International Conference on Wireless Communication and Sensor Networks(WCSN'08).* 67-72.

Myerson, J. M. (2002). The Complete Book of Middleware. AUERBACH; 1 edition

Omer, M. & Kunz, T. (2011). Operating Systems for Wireless Sensor Networks: A survey. *Sensors (an open access Journal).* 11 (2011), 5900-5930.

Panta, R. K., Bagchi, S. & Midkiff, S. P. (2011), Efficient Incremental Code Update for Sensor Networks**,** *ACM Transactions on Sensor Networks,* 7(4), 30.1-30.32.

Perrig, A., Szewczyk, R., Wen, D., Cullerand, J. & Tygar, D. (2001). SPINS: Security Protocols for Sensor Networks. *Proceedings of 7$^{th}$ Annual International Conference on Mobile Computing and Networks*, Rome, Italy.

Percival, C. (2003). Naive differences of executable code.

Portilla, J., Otero, A., De la Torre, E., Riesgo, T., Stecklina, O.,Peter, S. & Langendorfer, P. (2010). Adaptable Security in Wireless Sensor Networks by using Reconfigurable ECC Hardware Coprocessors. *International Journal of Distributed Sensor Networks*, 2010(740823), 1-12, doi:10.1155/2010/740823

Puder, A., Romer, K. & Pilhofer, F. (2006). *Distributed Systems Architecture: A Middleware Approach*. San francisco, California: Morgan Kaufmann.

Rada-Vilela, J. (2013). Fuzzylite: a fuzzy logic control library in C++". *In Proceedings of the Open Source Developers Conference.* Auckland, New Zealand.

Rada-Vilela, J. (2014). Fuzzylite: a fuzzy logic control library, *URL http://www.fuzzylite.com.*

Ramamurthy, H., Prabhu, B.S. & Gadh, R. (2004). Reconfigurable Wireless Interface for Networking Sensors (ReWINS). *In proceedings of IFIP TC6 , 9th International Conference on Personal Wireless Communication (PWC 2004*).

Ramamurthy, H., Lal, D., Prabhu, B. S., & Gadh, R.(2005). ReWINS: A Distributed Multi-RF Sensor Control Network for Industrial Automation. *IEEE wireless Telecommunication Symposium WTS 2005, Pomona,California.*

Sadiq, A. S., Abu, B. K. & Ghafoor, K. Z. (2010). A Fuzzy Logic Approach for Reducing Handover Latency in Wireless Networks. *Network Protocols and Algorithms*, (2)4, 1 -27, doi: 10.5296/npa.v2i4.527.

Silva, A. R. & Vuran, M. C. (2010). (CPS)^2: Integration of Center Pivot Systems with wireless Underground Sensor Networks for Autonomous Precision Agriculture. *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, 69-88.

Stankovic, J.A. & Ramamritham, K. (1989). The Spring Kernel: A New Paradigm for Hard Realtime Operating Systems, *ACM Operating Systems Review*, 23(3). 54-71.

Steine, M., Ngo, C.V., Oliver, R. S. Geilen, M., Basten, T., Fohler, G. & Decotgnie J. (2011). Proactive Reconfiguration of Wireless Sensor Networks, *Proceedings of the 14$^{th}$ ACM International Conference on Modelling, Anaylsis and Simulation of Wireless and Mobile systems (MSWIM'11),* 31-39.

Sugihara, R., & Gupta, R. K. (2008). Programming Models for Sensors Networks: A Survey, *ACM Transactions on Sensor Networks*. 4(2).

Tanaka, S., Fujita, N., Yanagisawa, Y., Terada, T. & Tsukamoto, M. (2008). Reconfigurable hardware architecture for saving power consumption on a sensor node. *International Conference on Intelligent Sensors, Sensor Networks and Information Processing,* 405-410, doi: 10.1109/ISSNIP.2008.4762022.

Takagi, T. & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control, *IEEE Transactions on Systems, Man and Cybernetics*, 15(1), 116–132.

Tridgell, A. (1999). Efficient algorithms for sorting and synchronization. *PhD thesis, Australian National University*.

Tuttlebee, W. (2002). *Software Defined Radio: Origins, Drivers and International Perspectives*. West Sussex, England: John Wiley & Sons Ltd.

Venayagamoorthy, G. K. (2009). A successful interdisciplinary course on computational intelligence. *IEEE Computational Intelligence Magazine*, 4(1), 14-23.

Walls, C. (1996). RTOS for Microcontroller Applications, *Electronic Engineering*, 68(831), 57-61.

Wilder, J., Uzelac, V., Milenkovic, A. & Jovanov, E. (2007). Wireless Sensor Networks for updating Reconfiguable Logic Design in Real-time". *The University of Alabama in Huntsville, 301 Sparkman Drive, Huntsville, AL 35899,* Retrieved from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.143.2168&rep=rep1&type=pdf

Yang, Y. (2009). Microchip Wireless(Miwi) Application Programming Interface (MiApp). *Microchip Technology Inc. DS01284A*, 1-14.

Yick, J., Mukherjee, B. & Ghosal, D. (2008). Wireless sensor network survey. *Journal of Computer Networks*. 52(2008), 2292-2330.

Xiaoyun, W. & Hongbo, Y. (2005). How to Break MD5 and Other Hash Functions, *Advances in Cryptology – Lecture Notes in Computer Science*, (3494)19–35. Retrieved: 21 December 2009.

Zuberi, K. M. & Shin, K. G. (1996). EMERALDS: A Microkernel for Embedded Real-Time Systems, *Proceedings of RTAS*, 241-249.

# APPENDIX A

## EXECUTION LINK FILE [ELF] STRUCTURE

The ELF standard is intended to streamline software development by providing developers with a set of binary interface definitions that extend across multiple operating environments. This should reduce the number of different interface implementations, thereby reducing the need for recoding and recompiling code.

### File Format

Object files participate in program linking (building a program) and program execution (running a program). For convenience and efficiency, the object file format provides parallel views of a file's contents, reflecting the differing needs of these activities. Figure A.1 shows an object file's organisation.

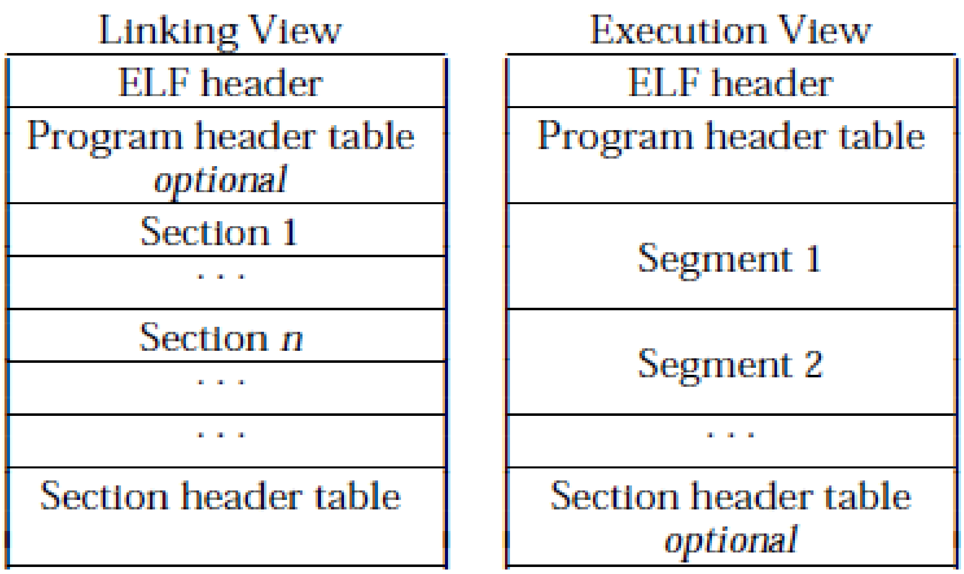


Figure A.1: An object file's organisation.

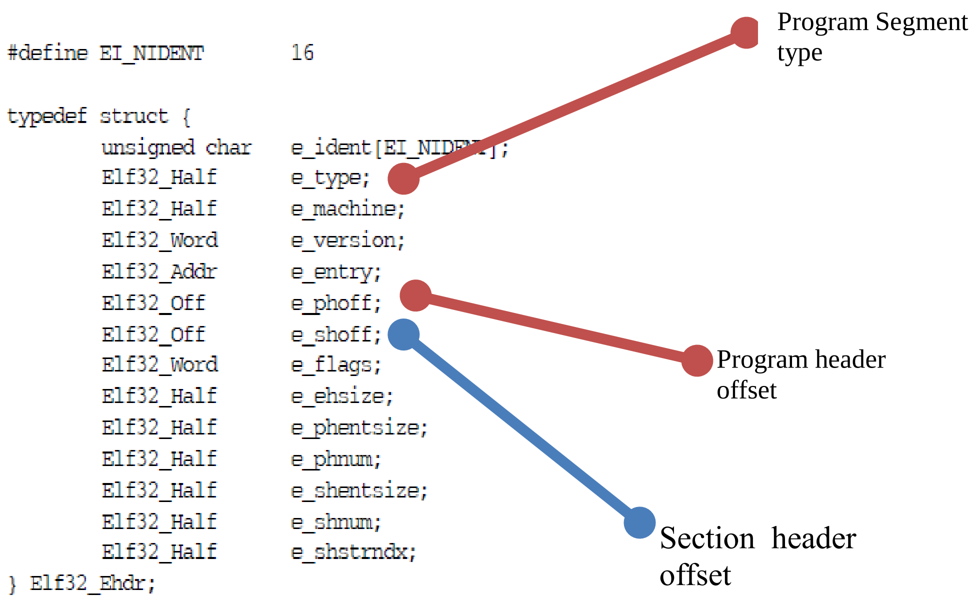


**Figure A.2 : The ELF header organisation**

An ELF header resides at the beginning and holds a ‘‘road map’’ describing the file’s organization. Sections hold the bulk of object file information for the linking view instructions, data, symbol table, relocation information, and the program execution view of the file. Figure A.3 and figure A.4 shows the Program header and the Section header organisation respectively.

**Program Header**

```
typedef struct {
        Elf32_Word      p_type;
        Elf32_Off       p_offset;
        Elf32_Addr      p_vaddr;
        Elf32_Addr      p_paddr;
        Elf32_Word      p_filesz;
        Elf32_Word      p_memsz;
        Elf32_Word      p_flags;
        Elf32_Word      p_align;
} Elf32_Phdr;
```

| Name | Value |
|---|---|
| PT_NULL | 0 |
| PT_LOAD | 1 |
| PT_DYNAMIC | 2 |
| PT_INTERP | 3 |
| PT_NOTE | 4 |
| PT_SHLIB | 5 |
| PT_PHDR | 6 |
| PT_LOPROC | 0x70000000 |
| PT_HIPROC | 0x7fffffff |

| Name | Value |
|---|---|
| SHF_WRITE | 0x1 |
| SHF_ALLOC | 0x2 |
| SHF_EXECINSTR | 0x4 |
| SHF_MASKPROC | 0xf0000000 |

**Figure A.3 : The Program header organisation**

| Name | Value |
|---|---|
| SHF_WRITE | 0x1 |
| SHF_ALLOC | 0x2 |
| SHF_EXECINSTR | 0x4 |
| SHF_MASKPROC | 0xf0000000 |

```
typedef struct {
        Elf32_Word      sh_name;
        Elf32_Word      sh_type;
        Elf32_Word      sh_flags;
        Elf32_Addr      sh_addr;
        Elf32_Off       sh_offset;
        Elf32_Word      sh_size;
        Elf32_Word      sh_link;
        Elf32_Word      sh_info;
        Elf32_Word      sh_addralign;
        Elf32_Word      sh_entsize;
} Elf32_Shdr;
```

**Figure A.4 : The Section header organisation**

# APPENDIX B

## PRECISION DELTA EXTRACTION SOURCE CODE

### PDE CODES

```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace PDE
{
    public partial class pde_platform : Form
    {
        public pde_platform()
        {
            InitializeComponent();
        }
        //
        Dictionary<string, string> gen_elf_old_Dict = new Dictionary<string,
string>();
        Dictionary<string, string> gen_elf_new_Dict = new Dictionary<string,
string>();
        //
        List<extract> pde_para_list_old = new List<extract>();
        List<extract> pde_para_list_new = new List<extract>();
        //
        List<extract> Rpt_Modified = new List<extract>();
        List<extract> Rpt_Addition= new List<extract>();
        List<extract> Rpt_Removed = new List<extract>();


        //

       public double Total_lines_original ;
       public double Total_lines_modified;
       public double Total_lines_deleted;
       public double Total_lines_added;
        //

              //

        private void pde_platform_Load(object sender, EventArgs e)
        {
            int number_original_lines = 0;
            int number_modified_lines = 0;


            var listBox_parameter_list_1 = listBox_parameter_listing_1.Items;
            var listBox_parameter_list_2 = listBox_parameter_listing_2.Items;
```

```
            var  pde_para_old = new pde_parameters();
            var pde_para_new = new pde_parameters();
            //
            var Hex_old = new Hex_File_Content();
            var Hex_new= new Hex_File_Content();
            //

            //
            pde_para_old.obtain_pde_para("temp-sensor-5");
            pde_para_new.obtain_pde_para("temp-sensor-10");

            //



           //
            foreach (var pdex_1 in pde_para_old.Gen_elf_old)
            {
                listBox_parameter_list_1.Add("*" + pdex_1.unified_ptr + "*" +
pdex_1.address + "*" + pdex_1.data);
                pde_para_list_old.Add(new extract() { unified_ptr =
pdex_1.unified_ptr, address = pdex_1.address, data = pdex_1.data ,section_name =
pdex_1.section_name, segment_no = pdex_1.segment_no });
                gen_elf_old_Dict.Add(pdex_1.unified_ptr, pdex_1.data);
                number_original_lines++;

            }
            //
            Total_lines_original = number_original_lines;
            label_original.Text = label_original.Text + " : " +
number_original_lines.ToString();
            //
            foreach (var pdex_2 in pde_para_new.Gen_elf_old)
            {
                listBox_parameter_list_2.Add("*" + pdex_2.unified_ptr + "*" +
pdex_2.address + "*" + pdex_2.data);
                pde_para_list_new.Add(new extract() { unified_ptr =
pdex_2.unified_ptr, address = pdex_2.address, data = pdex_2.data, section_name =
pdex_2.section_name, segment_no = pdex_2.segment_no });
                gen_elf_new_Dict.Add(pdex_2.unified_ptr, pdex_2.data);
                number_modified_lines++;
            }
            //
           // Total_lines_modified = number_modified_lines;
            label_modified.Text = label_modified.Text + " : " +
number_modified_lines.ToString();
            //
        }

        private void button1_Click(object sender, EventArgs e)
        {

            textBox_data.Text = gen_elf_old_Dict[textBox_key.Text];

        }

        private void simple_2_Click(object sender, EventArgs e)
        {
            textBox_data.Text = gen_elf_new_Dict[textBox_key.Text];

        }
```

```
        private void diff_Click(object sender, EventArgs e)

        { //
            FileStream file_modified_report =
File.Create(@"c:\CmdLine\Simple_Modified.txt");
            FileStream file_Addition_report =
File.Create(@"c:\CmdLine\Simple_Addition.txt");
            FileStream file_Removed_report =
File.Create(@"c:\CmdLine\Simple_Removed.txt");
            //
            StreamWriter write_modified = new StreamWriter(file_modified_report);
            StreamWriter write_Addition = new StreamWriter(file_Addition_report);
            StreamWriter write_Removed = new StreamWriter(file_Removed_report);

            //
            int number_modified_content = 0;
            int number_deleted_lines_code = 0;
            //
            int number_modified_content_inv = 0;
            int number_deleted_lines_code_inv = 0;
            //
           var list_parameter_listing_diff =
listBox_parameter_listing_diff.Items;
           var  list_Not_Capture =  listBox_Not_Capture.Items;
            //
           var list_parameter_listing_diff_inv =
listBox_parameter_listing_diff_inv.Items;
           var list_Not_Capture_inv = listBox_Not_Capture_inv.Items;


            //
            foreach (var cont in pde_para_list_old)
            {
                if (gen_elf_new_Dict.ContainsKey(cont.unified_ptr))
                {
                    if (gen_elf_old_Dict[cont.unified_ptr] !=
gen_elf_new_Dict[cont.unified_ptr])
                    {

                        list_parameter_listing_diff.Add("*" + cont.unified_ptr +
"*" + cont.data); // modification detected
                        Rpt_Modified.Add(new extract() { unified_ptr =
cont.unified_ptr, address = cont.address, data = cont.data, section_name =
cont.section_name,segment_no = cont.segment_no });
                        number_modified_content++;
                    }
                }
                else
                {
                    list_Not_Capture.Add("*" + cont.unified_ptr + "*" +
cont.data); // Present in orignal but not in modified
                    Rpt_Removed.Add(new extract() { unified_ptr =
cont.unified_ptr, address = cont.address, data = cont.data, section_name =
cont.section_name, segment_no = cont.segment_no });
                    //
                   // write_Removed.WriteLine(" | " + cont.unified_ptr + " | " +
cont.data);
                    //
                    number_deleted_lines_code++;
                }
            }
```

```
            //
            Total_lines_deleted = number_deleted_lines_code;
            Total_lines_modified = number_modified_content;
            //


            //
            label_modified_cont.Text = label_modified_cont.Text + " : " +
number_modified_content.ToString();
            label_deleted.Text = label_deleted.Text + " : " +
number_deleted_lines_code.ToString();
            //

            //
            foreach (var contx in pde_para_list_new)
            {
                if (gen_elf_old_Dict.ContainsKey(contx.unified_ptr))
                {
                    if (gen_elf_old_Dict[contx.unified_ptr] !=
gen_elf_new_Dict[contx.unified_ptr])
                    {
                        list_parameter_listing_diff_inv.Add("*" +
contx.unified_ptr + "*" + contx.data); // modification detected
                          //
                       // write_modified.WriteLine(" | " + contx.unified_ptr + "
| " + contx.data);
                        // SECTION DISPLAYED

                        number_modified_content_inv++;
                    }
                }
                else
                {
                    list_Not_Capture_inv.Add("*" + contx.unified_ptr + "*" +
contx.data); // Present in modified but not in original
                   // write_Addition.WriteLine(" | " + contx.unified_ptr + " | "
+ contx.data);
                    Rpt_Addition.Add(new extract() { unified_ptr =
contx.unified_ptr, address = contx.address, data = contx.data, section_name =
contx.section_name, segment_no = contx.segment_no });
                    number_deleted_lines_code_inv++;
                }
            }
            // REPORT ON CHANGES TO FILE
            var pde_para_old = new pde_parameters();
            var pde_para_new = new pde_parameters();

            pde_para_old.obtain_pde_para("temp-sensor-5");
            pde_para_new.obtain_pde_para("temp-sensor-10");

            write_modified.WriteLine(" MODIFIED WRT ORIGINAL   ");
            write_modified.WriteLine("             ");
            //
            write_Removed.WriteLine(" REMOVED  WRT ORIGINAL   ");
            write_Removed.WriteLine("             ");
            //
            write_Addition.WriteLine(" ADDITION WRT MODIFIED    ");
            write_Addition.WriteLine("             ");
            //
            for (int secn = 0; secn < pde_para_old.Sections_captured.Count()-1;
secn++)
            {
```

```
                write_Addition.WriteLine(" SECTION : " +
pde_para_old.Sections_captured.ElementAt(secn) + " COUNT : " +
Rpt_Addition.Where(x => x.section_name ==
pde_para_old.Sections_captured.ElementAt(secn)).Count());
                write_Addition.WriteLine("             ");
                //
                write_Removed.WriteLine(" SECTION : " +
pde_para_old.Sections_captured.ElementAt(secn) + " COUNT : " +
Rpt_Removed.Where(x => x.section_name ==
pde_para_old.Sections_captured.ElementAt(secn)).Count());
                write_Removed.WriteLine("             ");
                //
                write_modified.WriteLine(" SECTION : " +
pde_para_old.Sections_captured.ElementAt(secn) + " COUNT : " +
Rpt_Modified.Where(x => x.section_name ==
pde_para_old.Sections_captured.ElementAt(secn)).Count());
                write_modified.WriteLine("             ");

                foreach (var capturedx in Rpt_Modified.Where(x => x.section_name
== pde_para_old.Sections_captured.ElementAt(secn)))
                {
                    //
                    write_modified.WriteLine("*" + capturedx.unified_ptr + "*" +
capturedx.data);
                    //
                }


                foreach (var capturedy in Rpt_Addition.Where(x => x.section_name
== pde_para_old.Sections_captured.ElementAt(secn)))
                {
                    //
                    write_Addition.WriteLine("*" + capturedy.unified_ptr + "*" +
capturedy.data);
                    //
                }

                foreach (var capturedz in Rpt_Removed.Where(x => x.section_name
== pde_para_old.Sections_captured.ElementAt(secn)))
                {
                    //
                    write_Removed.WriteLine("*" + capturedz.unified_ptr + "*" +
capturedz.data);
                    //
                }




            }


            //
            Total_lines_added = number_deleted_lines_code_inv;
            label_modified_cont_inv.Text = label_modified_cont_inv.Text + " : " +
number_modified_content_inv.ToString();
            label_deleted_inv.Text = label_deleted_inv.Text + " : " +
number_deleted_lines_code_inv.ToString();
            //

            //
```

```
            label_mofified_percent.Text = label_mofified_percent.Text +
(Total_lines_modified / Total_lines_original * 100).ToString();
            label_delete_percent.Text= label_delete_percent.Text +
(Total_lines_deleted / Total_lines_original * 100).ToString();
            label_added_percent.Text = label_added_percent.Text +
(Total_lines_added / Total_lines_original * 100).ToString();


            //

            write_modified.Close();
            file_modified_report.Close();
            //
            write_Addition.Close();
            file_Addition_report.Close();
            //
            write_Removed.Close();
            file_Removed_report.Close();
            //
        }

        private void button1_Click_1(object sender, EventArgs e)
        {
           var showform = new PDE.Form_Hex_viewer();
            showform.Show();

        }

        private void button2_Click(object sender, EventArgs e)
        {
            var showform = new PDE.Form1();
            showform.Show();
        }

    }
}
```

# APPENDIX C

| Fuzzilite C++ codes |
| --- |
| **FuzzyLite Dataset** |

## Fuzzilite C++ codes

```
fl::Engine* engine = new fl::Engine;
engine->setName("Reconfig");

fl::InputVariable* inputVariable1 = new fl::InputVariable;
inputVariable1->setEnabled(true);
inputVariable1->setName("Delta_Orientation");
inputVariable1->setRange(0.000, 10240.000);
inputVariable1->addTerm(new fl::Triangle("Segment_Confined", 0.000, 2048.000,
4096.000));
inputVariable1->addTerm(new fl::Triangle("Segment_Adjoint", 2048.000, 4710.400,
8230.000));
inputVariable1->addTerm(new fl::Ramp("Segment_Disjoint", 6451.200, 10240.000));
engine->addInputVariable(inputVariable1);

fl::InputVariable* inputVariable2 = new fl::InputVariable;
inputVariable2->setEnabled(true);
inputVariable2->setName("Battery_State");
inputVariable2->setRange(0.000, 1850.000);
inputVariable2->addTerm(new fl::Ramp("Very_Ok", 1185.000, 1850.000));
inputVariable2->addTerm(new fl::Ramp("Critical", 370.000, 0.000));
inputVariable2->addTerm(new fl::Triangle("Fair", 148.000, 592.000, 943.500));
inputVariable2->addTerm(new fl::Triangle("Ok", 648.000, 1018.000, 1351.000));
engine->addInputVariable(inputVariable2);

fl::OutputVariable* outputVariable = new fl::OutputVariable;
outputVariable->setEnabled(true);
outputVariable->setName("Reconfiguration_Approach");
outputVariable->setRange(0.000, 10.000);
outputVariable->fuzzyOutput()->setAccumulation(new fl::AlgebraicSum);
outputVariable->setDefuzzifier(new fl::Centroid(200));
outputVariable->setDefaultValue(fl::nan);
outputVariable->setLockValidOutput(false);
outputVariable->setLockOutputRange(false);
outputVariable->addTerm(new fl::Triangle("Difference_Approach", 2.100, 2.500,
5.000));
outputVariable->addTerm(new fl::Triangle("Modular_Approach", 3.900, 5.500,
7.700));
outputVariable->addTerm(new fl::Triangle("Entire_Image_Approach", 6.300, 8.100,
10.000));
```

```
outputVariable->addTerm(new fl::Triangle("Suspend_Reconfiguration", 0.000, 0.000,
2.600));
engine->addOutputVariable(outputVariable);

fl::RuleBlock* ruleBlock = new fl::RuleBlock;
ruleBlock->setEnabled(true);
ruleBlock->setName("");
ruleBlock->setConjunction(new fl::Minimum);
ruleBlock->setDisjunction(new fl::AlgebraicSum);
ruleBlock->setActivation(new fl::Minimum);
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Confined and
Battery_State is Very_Ok then Reconfiguration_Approach is Difference_Approach",
engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Confined and
Battery_State is Critical then Reconfiguration_Approach is Suspend_Reconfiguration",
engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Confined and
Battery_State is Fair then Reconfiguration_Approach is Difference_Approach",
engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Confined and
Battery_State is Ok then Reconfiguration_Approach is Difference_Approach", engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Adjoint and
Battery_State is Very_Ok then Reconfiguration_Approach is Modular_Approach",
engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Adjoint and
Battery_State is Critical then Reconfiguration_Approach is Suspend_Reconfiguration",
engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Adjoint and
Battery_State is Fair then Reconfiguration_Approach is Modular_Approach", engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Adjoint and
Battery_State is Ok then Reconfiguration_Approach is Modular_Approach", engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Disjoint and
Battery_State is Very_Ok then Reconfiguration_Approach is Entire_Image_Approach",
engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Disjoint and
Battery_State is Critical then Reconfiguration_Approach is Suspend_Reconfiguration",
engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Disjoint and
Battery_State is Fair then Reconfiguration_Approach is Suspend_Reconfiguration",
engine));
ruleBlock->addRule(fl::Rule::parse("if Delta_Orientation is Segment_Disjoint and
Battery_State is Ok then Reconfiguration_Approach is Entire_Image_Approach",
engine));
engine->addRuleBlock(ruleBlock);
```

## **FuzzyLite Dataset**

| Delta_Orientation | Battery_State | Reconfiguration_Approach |
|---|---|---|
| 0.000 | 1776.000 | nan |
| 0.000 | 1794.500 | nan |
| 0.000 | 1813.000 | nan |
| 0.000 | 1831.500 | nan |
| 0.000 | 1850.000 | nan |
| 102.400 | 0.000 | 1.269 |
| 102.400 | 18.500 | 1.269 |
| 102.400 | 37.000 | 1.269 |
| 102.400 | 55.500 | 1.269 |
| 102.400 | 74.000 | 1.269 |
| 102.400 | 92.500 | 1.269 |
| 102.400 | 111.000 | 1.269 |
| 102.400 | 129.500 | 1.269 |
| 102.400 | 148.000 | 1.269 |
| 102.400 | 166.500 | 2.359 |
| 102.400 | 185.000 | 2.459 |
| 102.400 | 203.500 | 2.459 |
| 102.400 | 222.000 | 2.459 |
| 102.400 | 240.500 | 2.459 |
| 102.400 | 259.000 | 2.459 |
| 102.400 | 277.500 | 2.459 |
| 102.400 | 296.000 | 2.459 |
| 102.400 | 314.500 | 2.459 |
| 102.400 | 333.000 | 2.459 |
| 102.400 | 351.500 | 2.459 |
| 102.400 | 370.000 | 3.520 |
| 102.400 | 388.500 | 3.520 |
| 102.400 | 407.000 | 3.520 |
| 102.400 | 425.500 | 3.520 |
| 102.400 | 444.000 | 3.520 |
| 102.400 | 462.500 | 3.520 |
| 102.400 | 481.000 | 3.520 |
| 102.400 | 499.500 | 3.520 |
| 102.400 | 518.000 | 3.520 |
| 102.400 | 536.500 | 3.520 |
| 102.400 | 555.000 | 3.520 |
| 102.400 | 573.500 | 3.520 |
| 102.400 | 592.000 | 3.520 |
| 102.400 | 610.500 | 3.520 |
| 102.400 | 629.000 | 3.520 |
| 102.400 | 647.500 | 3.520 |
| 102.400 | 666.000 | 3.521 |
| 102.400 | 684.500 | 3.521 |
| 102.400 | 703.000 | 3.521 |
| 102.400 | 721.500 | 3.521 |
| 102.400 | 740.000 | 3.521 |

102.400 758.500 3.521
102.400 777.000 3.521

102.400 795.500 3.521
102.400 814.000 3.521
102.400 832.500 3.521
102.400 851.000 3.521
102.400 869.500 3.521
102.400 888.000 3.521
102.400 906.500 3.521
102.400 925.000 3.521
102.400 943.500 3.520
102.400 962.000 3.520
102.400 980.500 3.520
102.400 999.000 3.520
102.400 1017.500 3.520
102.400 1036.000 3.520
102.400 1054.500 3.520
102.400 1073.000 3.520
102.400 1091.500 3.520
102.400 1110.000 3.520
102.400 1128.500 3.520
102.400 1147.000 3.520
102.400 1165.500 3.520
102.400 1184.000 3.520
102.400 1202.500 3.525
102.400 1221.000 3.521
102.400 1239.500 3.521
102.400 1258.000 3.521
102.400 1276.500 3.521
102.400 1295.000 3.521
102.400 1313.500 3.521
102.400 1332.000 3.521
102.400 1350.500 3.521
102.400 1369.000 3.520
102.400 1387.500 3.520
102.400 1406.000 3.520
102.400 1424.500 3.520
102.400 1443.000 3.520
102.400 1461.500 3.520
102.400 1480.000 3.520
102.400 1498.500 3.520
102.400 1517.000 3.520
102.400 1535.500 3.520
102.400 1554.000 3.520
102.400 1572.500 3.520
102.400 1591.000 3.520
102.400 1609.500 3.520
102.400 1628.000 3.520
102.400 1646.500 3.520
102.400 1665.000 3.520
102.400 1683.500 3.520
102.400 1702.000 3.520

102.400 1720.500 3.520
102.400 1739.000 3.520
102.400 1757.500 3.520
102.400 1776.000 3.520
102.400 1794.500 3.520
102.400 1813.000 3.520
102.400 1831.500 3.520
102.400 1850.000 3.520
204.800 0.000 1.236
204.800 18.500 1.236
204.800 37.000 1.236
204.800 55.500 1.236
204.800 74.000 1.236
204.800 92.500 1.236
204.800 111.000 1.236
204.800 129.500 1.236
204.800 148.000 1.236
204.800 166.500 1.977
204.800 185.000 2.335
204.800 203.500 2.430
204.800 222.000 2.430
204.800 240.500 2.430
204.800 259.000 2.430
204.800 277.500 2.430
204.800 296.000 2.430
204.800 314.500 2.430
204.800 333.000 2.430
204.800 351.500 2.798
204.800 370.000 3.498
204.800 388.500 3.498
204.800 407.000 3.498
204.800 425.500 3.498
204.800 444.000 3.498
204.800 462.500 3.498
204.800 481.000 3.498
204.800 499.500 3.498
204.800 518.000 3.498
204.800 536.500 3.498
204.800 555.000 3.498
204.800 573.500 3.498
204.800 592.000 3.498
204.800 610.500 3.498
204.800 629.000 3.498
204.800 647.500 3.498
204.800 666.000 3.506
204.800 684.500 3.499
204.800 703.000 3.499
204.800 721.500 3.499
204.800 740.000 3.499
204.800 758.500 3.499

204.800 777.000 3.499
204.800 795.500 3.499
204.800 814.000 3.499
204.800 832.500 3.499
204.800 851.000 3.499
204.800 869.500 3.499
204.800 888.000 3.499
204.800 906.500 3.499
204.800 925.000 3.505
204.800 943.500 3.498
204.800 962.000 3.498
204.800 980.500 3.498
204.800 999.000 3.498
204.800 1017.500 3.498
204.800 1036.000 3.498
204.800 1054.500 3.498
204.800 1073.000 3.498
204.800 1091.500 3.498
204.800 1110.000 3.498
204.800 1128.500 3.498
204.800 1147.000 3.498
204.800 1165.500 3.498
204.800 1184.000 3.498
204.800 1202.500 3.506
204.800 1221.000 3.505
204.800 1239.500 3.502
204.800 1258.000 3.499
204.800 1276.500 3.499
204.800 1295.000 3.499
204.800 1313.500 3.499
204.800 1332.000 3.505
204.800 1350.500 3.499
204.800 1369.000 3.498
204.800 1387.500 3.498
204.800 1406.000 3.498
204.800 1424.500 3.498
204.800 1443.000 3.498
204.800 1461.500 3.498
204.800 1480.000 3.498
204.800 1498.500 3.498
204.800 1517.000 3.498
204.800 1535.500 3.498
204.800 1554.000 3.498
204.800 1572.500 3.498
204.800 1591.000 3.498
204.800 1609.500 3.498
204.800 1628.000 3.498
204.800 1646.500 3.498
204.800 1665.000 3.498
204.800 1683.500 3.498

204.800 1702.000 3.498
204.800 1720.500 3.498
204.800 1739.000 3.498
204.800 1757.500 3.498
204.800 1776.000 3.498
204.800 1794.500 3.498
204.800 1813.000 3.498
204.800 1831.500 3.498
204.800 1850.000 3.498
307.200 0.000 1.205
307.200 18.500 1.205
307.200 37.000 1.205
307.200 55.500 1.205
307.200 74.000 1.205
307.200 92.500 1.205
307.200 111.000 1.205
307.200 129.500 1.205
307.200 148.000 1.205
307.200 166.500 1.777
307.200 185.000 2.104
307.200 203.500 2.312
307.200 222.000 2.402
307.200 240.500 2.402
307.200 259.000 2.402
307.200 277.500 2.402
307.200 296.000 2.402
307.200 314.500 2.402
307.200 333.000 2.625
307.200 351.500 2.949
307.200 370.000 3.473
307.200 388.500 3.473
307.200 407.000 3.473
307.200 425.500 3.473
307.200 444.000 3.473
307.200 462.500 3.473
307.200 481.000 3.473
307.200 499.500 3.473
307.200 518.000 3.473
307.200 536.500 3.473
307.200 555.000 3.473
307.200 573.500 3.473
307.200 592.000 3.473
307.200 610.500 3.473
307.200 629.000 3.473
307.200 647.500 3.473
307.200 666.000 3.486
307.200 684.500 3.485
307.200 703.000 3.476
307.200 721.500 3.475
307.200 740.000 3.475

307.200 758.500 3.475
307.200 777.000 3.475
307.200 795.500 3.475
307.200 814.000 3.475
307.200 832.500 3.475
307.200 851.000 3.475
307.200 869.500 3.475
307.200 888.000 3.475
307.200 906.500 3.484
307.200 925.000 3.486
307.200 943.500 3.473
307.200 962.000 3.473
307.200 980.500 3.473
307.200 999.000 3.473
307.200 1017.500 3.473
307.200 1036.000 3.473
307.200 1054.500 3.473
307.200 1073.000 3.473
307.200 1091.500 3.473
307.200 1110.000 3.473
307.200 1128.500 3.473
307.200 1147.000 3.473
307.200 1165.500 3.473
307.200 1184.000 3.473
307.200 1202.500 3.483
307.200 1221.000 3.486
307.200 1239.500 3.486
307.200 1258.000 3.484
307.200 1276.500 3.478
307.200 1295.000 3.475
307.200 1313.500 3.483
307.200 1332.000 3.486
307.200 1350.500 3.474
307.200 1369.000 3.473
307.200 1387.500 3.473
307.200 1406.000 3.473
307.200 1424.500 3.473
307.200 1443.000 3.473
307.200 1461.500 3.473
307.200 1480.000 3.473
307.200 1498.500 3.473
307.200 1517.000 3.473
307.200 1535.500 3.473
307.200 1554.000 3.473
307.200 1572.500 3.473
307.200 1591.000 3.473
307.200 1609.500 3.473
307.200 1628.000 3.473
307.200 1646.500 3.473
307.200 1665.000 3.473

307.200 1683.500 3.473
307.200 1702.000 3.473
307.200 1720.500 3.473
307.200 1739.000 3.473
307.200 1757.500 3.473
307.200 1776.000 3.473
307.200 1794.500 3.473
307.200 1813.000 3.473
307.200 1831.500 3.473
307.200 1850.000 3.473
409.600 0.000 1.175
409.600 18.500 1.175
409.600 37.000 1.175
409.600 55.500 1.175
409.600 74.000 1.175
409.600 92.500 1.175
409.600 111.000 1.175
409.600 129.500 1.175
409.600 148.000 1.175
409.600 166.500 1.648
409.600 185.000 1.945
409.600 203.500 2.146
409.600 222.000 2.289
409.600 240.500 2.374
409.600 259.000 2.374
409.600 277.500 2.374
409.600 296.000 2.374
409.600 314.500 2.534
409.600 333.000 2.742
409.600 351.500 3.026
409.600 370.000 3.448
409.600 388.500 3.448
409.600 407.000 3.448
409.600 425.500 3.448
409.600 444.000 3.448
409.600 462.500 3.448
409.600 481.000 3.448
409.600 499.500 3.448
409.600 518.000 3.448
409.600 536.500 3.448
409.600 555.000 3.448
409.600 573.500 3.448
409.600 592.000 3.448
409.600 610.500 3.448
409.600 629.000 3.448
409.600 647.500 3.448
409.600 666.000 3.464
409.600 684.500 3.467
409.600 703.000 3.462
409.600 721.500 3.452

409.600 740.000 3.452
409.600 758.500 3.452
409.600 777.000 3.452
409.600 795.500 3.452
409.600 814.000 3.452
409.600 832.500 3.452
409.600 851.000 3.452
409.600 869.500 3.452
409.600 888.000 3.460
409.600 906.500 3.467
409.600 925.000 3.464
409.600 943.500 3.448
409.600 962.000 3.448
409.600 980.500 3.448
409.600 999.000 3.448
409.600 1017.500 3.448
409.600 1036.000 3.448
409.600 1054.500 3.448
409.600 1073.000 3.448
409.600 1091.500 3.448
409.600 1110.000 3.448
409.600 1128.500 3.448
409.600 1147.000 3.448
409.600 1165.500 3.448
409.600 1184.000 3.448
409.600 1202.500 3.459
409.600 1221.000 3.464
409.600 1239.500 3.467
409.600 1258.000 3.467
409.600 1276.500 3.463
409.600 1295.000 3.467
409.600 1313.500 3.468
409.600 1332.000 3.464
409.600 1350.500 3.449
409.600 1369.000 3.448
409.600 1387.500 3.448
409.600 1406.000 3.448
409.600 1424.500 3.448
409.600 1443.000 3.448
409.600 1461.500 3.448
409.600 1480.000 3.448
409.600 1498.500 3.448
409.600 1517.000 3.448
409.600 1535.500 3.448
409.600 1554.000 3.448
409.600 1572.500 3.448
409.600 1591.000 3.448
409.600 1609.500 3.448
409.600 1628.000 3.448
409.600 1646.500 3.448

409.600 1665.000 3.448
409.600 1683.500 3.448
409.600 1702.000 3.448
409.600 1720.500 3.448
409.600 1739.000 3.448
409.600 1757.500 3.448
409.600 1776.000 3.448
409.600 1794.500 3.448
409.600 1813.000 3.448
409.600 1831.500 3.448
409.600 1850.000 3.448
512.000 0.000 1.145
512.000 18.500 1.145
512.000 37.000 1.145
512.000 55.500 1.145
512.000 74.000 1.145
512.000 92.500 1.145
512.000 111.000 1.145
512.000 129.500 1.145
512.000 148.000 1.145
512.000 166.500 1.554
512.000 185.000 1.827
512.000 203.500 2.019
512.000 222.000 2.160
512.000 240.500 2.267
512.000 259.000 2.348
512.000 277.500 2.348
512.000 296.000 2.471
512.000 314.500 2.624
512.000 333.000 2.816
512.000 351.500 3.071
512.000 370.000 3.425
512.000 388.500 3.425
512.000 407.000 3.425
512.000 425.500 3.425
512.000 444.000 3.425
512.000 462.500 3.425
512.000 481.000 3.425
512.000 499.500 3.425
512.000 518.000 3.425
512.000 536.500 3.425
512.000 555.000 3.425
512.000 573.500 3.425
512.000 592.000 3.425
512.000 610.500 3.425
512.000 629.000 3.425
512.000 647.500 3.425
512.000 666.000 3.443
512.000 684.500 3.449
512.000 703.000 3.448

512.000 721.500 3.440
512.000 740.000 3.431
512.000 758.500 3.431
512.000 777.000 3.431
512.000 795.500 3.431
512.000 814.000 3.431
512.000 832.500 3.431
512.000 851.000 3.431
512.000 869.500 3.439
512.000 888.000 3.446
512.000 906.500 3.450
512.000 925.000 3.444
512.000 943.500 3.425
512.000 962.000 3.425
512.000 980.500 3.425
512.000 999.000 3.425
512.000 1017.500 3.425
512.000 1036.000 3.425
512.000 1054.500 3.425
512.000 1073.000 3.425
512.000 1091.500 3.425
512.000 1110.000 3.425
512.000 1128.500 3.425
512.000 1147.000 3.425
512.000 1165.500 3.425
512.000 1184.000 3.425
512.000 1202.500 3.437
512.000 1221.000 3.444
512.000 1239.500 3.448
512.000 1258.000 3.450
512.000 1276.500 3.457
512.000 1295.000 3.467
512.000 1313.500 3.468
512.000 1332.000 3.456
512.000 1350.500 3.427
512.000 1369.000 3.425
512.000 1387.500 3.425
512.000 1406.000 3.425
512.000 1424.500 3.425
512.000 1443.000 3.425
512.000 1461.500 3.425
512.000 1480.000 3.425
512.000 1498.500 3.425
512.000 1517.000 3.425
512.000 1535.500 3.425
512.000 1554.000 3.425
512.000 1572.500 3.425
512.000 1591.000 3.425
512.000 1609.500 3.425
512.000 1628.000 3.425

512.000 1646.500 3.425
512.000 1665.000 3.425
512.000 1683.500 3.425
512.000 1702.000 3.425
512.000 1720.500 3.425
512.000 1739.000 3.425
512.000 1757.500 3.425
512.000 1776.000 3.425
512.000 1794.500 3.425
512.000 1813.000 3.425
512.000 1831.500 3.425
512.000 1850.000 3.425
614.400 0.000 1.117
614.400 18.500 1.117
614.400 37.000 1.117
614.400 55.500 1.117
614.400 74.000 1.117
614.400 92.500 1.117
614.400 111.000 1.117
614.400 129.500 1.117
614.400 148.000 1.117
614.400 166.500 1.480
614.400 185.000 1.733
614.400 203.500 1.918
614.400 222.000 2.056
614.400 240.500 2.162
614.400 259.000 2.245
614.400 277.500 2.411
614.400 296.000 2.541
614.400 314.500 2.685
614.400 333.000 2.864
614.400 351.500 3.094
614.400 370.000 3.401
614.400 388.500 3.401
614.400 407.000 3.401
614.400 425.500 3.401
614.400 444.000 3.401
614.400 462.500 3.401
614.400 481.000 3.401
614.400 499.500 3.401
614.400 518.000 3.401
614.400 536.500 3.401
614.400 555.000 3.401
614.400 573.500 3.401
614.400 592.000 3.401
614.400 610.500 3.401
614.400 629.000 3.401
614.400 647.500 3.401
614.400 666.000 3.421
614.400 684.500 3.430

614.400 703.000 3.431
614.400 721.500 3.426
614.400 740.000 3.420
614.400 758.500 3.410
614.400 777.000 3.410
614.400 795.500 3.410
614.400 814.000 3.410
614.400 832.500 3.410
614.400 851.000 3.417
614.400 869.500 3.425
614.400 888.000 3.430
614.400 906.500 3.430
614.400 925.000 3.422
614.400 943.500 3.401
614.400 962.000 3.401
614.400 980.500 3.401
614.400 999.000 3.401
614.400 1017.500 3.401
614.400 1036.000 3.401
614.400 1054.500 3.401
614.400 1073.000 3.401
614.400 1091.500 3.401
614.400 1110.000 3.401
614.400 1128.500 3.401
614.400 1147.000 3.401
614.400 1165.500 3.401
614.400 1184.000 3.401
614.400 1202.500 3.414
614.400 1221.000 3.422
614.400 1239.500 3.428
614.400 1258.000 3.439
614.400 1276.500 3.457
614.400 1295.000 3.467
614.400 1313.500 3.468
614.400 1332.000 3.456
614.400 1350.500 3.427
614.400 1369.000 3.412
614.400 1387.500 3.401
614.400 1406.000 3.401
614.400 1424.500 3.401
614.400 1443.000 3.401
614.400 1461.500 3.401
614.400 1480.000 3.401
614.400 1498.500 3.401
614.400 1517.000 3.401
614.400 1535.500 3.401
614.400 1554.000 3.401
614.400 1572.500 3.401
614.400 1591.000 3.401
614.400 1609.500 3.401

614.400 1628.000 3.401
614.400 1646.500 3.401
614.400 1665.000 3.401
614.400 1683.500 3.401
614.400 1702.000 3.401
614.400 1720.500 3.401
614.400 1739.000 3.401
614.400 1757.500 3.401
614.400 1776.000 3.401
614.400 1794.500 3.401
614.400 1813.000 3.401
614.400 1831.500 3.401
614.400 1850.000 3.401
716.800 0.000 1.089
716.800 18.500 1.089
716.800 37.000 1.089
716.800 55.500 1.089
716.800 74.000 1.089
716.800 92.500 1.089
716.800 111.000 1.089
716.800 129.500 1.089
716.800 148.000 1.089
716.800 166.500 1.419
716.800 185.000 1.657
716.800 203.500 1.834
716.800 222.000 1.969
716.800 240.500 2.074
716.800 259.000 2.245
716.800 277.500 2.411
716.800 296.000 2.576
716.800 314.500 2.729
716.800 333.000 2.897
716.800 351.500 3.107
716.800 370.000 3.379
716.800 388.500 3.379
716.800 407.000 3.379
716.800 425.500 3.379
716.800 444.000 3.379
716.800 462.500 3.379
716.800 481.000 3.379
716.800 499.500 3.379
716.800 518.000 3.379
716.800 536.500 3.379
716.800 555.000 3.379
716.800 573.500 3.379
716.800 592.000 3.379
716.800 610.500 3.379
716.800 629.000 3.379
716.800 647.500 3.379
716.800 666.000 3.400

716.800 684.500 3.411
716.800 703.000 3.415
716.800 721.500 3.412
716.800 740.000 3.408
716.800 758.500 3.400
716.800 777.000 3.392
716.800 795.500 3.392
716.800 814.000 3.392
716.800 832.500 3.397
716.800 851.000 3.406
716.800 869.500 3.412
716.800 888.000 3.414
716.800 906.500 3.412
716.800 925.000 3.401
716.800 943.500 3.379
716.800 962.000 3.379
716.800 980.500 3.379
716.800 999.000 3.379
716.800 1017.500 3.379
716.800 1036.000 3.379
716.800 1054.500 3.379
716.800 1073.000 3.379
716.800 1091.500 3.379
716.800 1110.000 3.379
716.800 1128.500 3.379
716.800 1147.000 3.379
716.800 1165.500 3.379
716.800 1184.000 3.379
716.800 1202.500 3.392
716.800 1221.000 3.402
716.800 1239.500 3.414
716.800 1258.000 3.439
716.800 1276.500 3.457
716.800 1295.000 3.467
716.800 1313.500 3.468
716.800 1332.000 3.456
716.800 1350.500 3.427
716.800 1369.000 3.412
716.800 1387.500 3.399
716.800 1406.000 3.387
716.800 1424.500 3.379
716.800 1443.000 3.379
716.800 1461.500 3.379
716.800 1480.000 3.379
716.800 1498.500 3.379
716.800 1517.000 3.379
716.800 1535.500 3.379
716.800 1554.000 3.379
716.800 1572.500 3.379
716.800 1591.000 3.379

716.800 1609.500 3.379
716.800 1628.000 3.379
716.800 1646.500 3.379
716.800 1665.000 3.379
716.800 1683.500 3.379
716.800 1702.000 3.379
716.800 1720.500 3.379
716.800 1739.000 3.379
716.800 1757.500 3.379
716.800 1776.000 3.379
716.800 1794.500 3.379
716.800 1813.000 3.379
716.800 1831.500 3.379
716.800 1850.000 3.379
819.200 0.000 1.062
819.200 18.500 1.062
819.200 37.000 1.062
819.200 55.500 1.062
819.200 74.000 1.062
819.200 92.500 1.062
819.200 111.000 1.062
819.200 129.500 1.062
819.200 148.000 1.062
819.200 166.500 1.367
819.200 185.000 1.592
819.200 203.500 1.762
819.200 222.000 1.895
819.200 240.500 2.074
819.200 259.000 2.245
819.200 277.500 2.411
819.200 296.000 2.576
819.200 314.500 2.746
819.200 333.000 2.917
819.200 351.500 3.111
819.200 370.000 3.357
819.200 388.500 3.357
819.200 407.000 3.357
819.200 425.500 3.357
819.200 444.000 3.357
819.200 462.500 3.357
819.200 481.000 3.357
819.200 499.500 3.357
819.200 518.000 3.357
819.200 536.500 3.357
819.200 555.000 3.357
819.200 573.500 3.357
819.200 592.000 3.357
819.200 610.500 3.357
819.200 629.000 3.357
819.200 647.500 3.357

| | | |
|---|---|---|
| 819.200 | 666.000 | 3.380 |
| 819.200 | 684.500 | 3.392 |
| 819.200 | 703.000 | 3.398 |
| 819.200 | 721.500 | 3.398 |
| 819.200 | 740.000 | 3.395 |
| 819.200 | 758.500 | 3.389 |
| 819.200 | 777.000 | 3.382 |
| 819.200 | 795.500 | 3.374 |
| 819.200 | 814.000 | 3.379 |
| 819.200 | 832.500 | 3.387 |
| 819.200 | 851.000 | 3.394 |
| 819.200 | 869.500 | 3.397 |
| 819.200 | 888.000 | 3.398 |
| 819.200 | 906.500 | 3.394 |
| 819.200 | 925.000 | 3.381 |
| 819.200 | 943.500 | 3.357 |
| 819.200 | 962.000 | 3.357 |
| 819.200 | 980.500 | 3.357 |
| 819.200 | 999.000 | 3.357 |
| 819.200 | 1017.500 | 3.357 |
| 819.200 | 1036.000 | 3.357 |
| 819.200 | 1054.500 | 3.357 |
| 819.200 | 1073.000 | 3.357 |
| 819.200 | 1091.500 | 3.357 |
| 819.200 | 1110.000 | 3.357 |
| 819.200 | 1128.500 | 3.357 |
| 819.200 | 1147.000 | 3.357 |
| 819.200 | 1165.500 | 3.357 |
| 819.200 | 1184.000 | 3.357 |
| 819.200 | 1202.500 | 3.371 |
| 819.200 | 1221.000 | 3.385 |
| 819.200 | 1239.500 | 3.414 |
| 819.200 | 1258.000 | 3.439 |
| 819.200 | 1276.500 | 3.457 |
| 819.200 | 1295.000 | 3.467 |
| 819.200 | 1313.500 | 3.468 |
| 819.200 | 1332.000 | 3.456 |
| 819.200 | 1350.500 | 3.427 |
| 819.200 | 1369.000 | 3.412 |
| 819.200 | 1387.500 | 3.399 |
| 819.200 | 1406.000 | 3.387 |
| 819.200 | 1424.500 | 3.375 |
| 819.200 | 1443.000 | 3.363 |
| 819.200 | 1461.500 | 3.357 |
| 819.200 | 1480.000 | 3.357 |
| 819.200 | 1498.500 | 3.357 |
| 819.200 | 1517.000 | 3.357 |
| 819.200 | 1535.500 | 3.357 |
| 819.200 | 1554.000 | 3.357 |
| 819.200 | 1572.500 | 3.357 |

819.200 1591.000 3.357
819.200 1609.500 3.357
819.200 1628.000 3.357
819.200 1646.500 3.357
819.200 1665.000 3.357
819.200 1683.500 3.357
819.200 1702.000 3.357
819.200 1720.500 3.357
819.200 1739.000 3.357
819.200 1757.500 3.357
819.200 1776.000 3.357
819.200 1794.500 3.357
819.200 1813.000 3.357
819.200 1831.500 3.357
819.200 1850.000 3.357
921.600 0.000 1.036
921.600 18.500 1.036
921.600 37.000 1.036
921.600 55.500 1.036
921.600 74.000 1.036
921.600 92.500 1.036
921.600 111.000 1.036
921.600 129.500 1.036
921.600 148.000 1.036
921.600 166.500 1.322
921.600 185.000 1.536
921.600 203.500 1.701
921.600 222.000 1.895
921.600 240.500 2.074
921.600 259.000 2.245
921.600 277.500 2.411
921.600 296.000 2.576
921.600 314.500 2.746
921.600 333.000 2.922
921.600 351.500 3.111
921.600 370.000 3.336
921.600 388.500 3.336
921.600 407.000 3.336
921.600 425.500 3.336
921.600 444.000 3.336
921.600 462.500 3.336
921.600 481.000 3.336
921.600 499.500 3.336
921.600 518.000 3.336
921.600 536.500 3.336
921.600 555.000 3.336
921.600 573.500 3.336
921.600 592.000 3.336
921.600 610.500 3.336
921.600 629.000 3.336

921.600 647.500 3.336
921.600 666.000 3.359
921.600 684.500 3.374
921.600 703.000 3.381
921.600 721.500 3.383
921.600 740.000 3.382
921.600 758.500 3.378
921.600 777.000 3.372
921.600 795.500 3.371
921.600 814.000 3.370
921.600 832.500 3.376
921.600 851.000 3.381
921.600 869.500 3.383
921.600 888.000 3.382
921.600 906.500 3.375
921.600 925.000 3.361
921.600 943.500 3.336
921.600 962.000 3.336
921.600 980.500 3.336
921.600 999.000 3.336
921.600 1017.500 3.336
921.600 1036.000 3.336
921.600 1054.500 3.336
921.600 1073.000 3.336
921.600 1091.500 3.336
921.600 1110.000 3.336
921.600 1128.500 3.336
921.600 1147.000 3.336
921.600 1165.500 3.336
921.600 1184.000 3.336
921.600 1202.500 3.352
921.600 1221.000 3.385
921.600 1239.500 3.414
921.600 1258.000 3.439
921.600 1276.500 3.457
921.600 1295.000 3.467
921.600 1313.500 3.468
921.600 1332.000 3.456
921.600 1350.500 3.427
921.600 1369.000 3.412
921.600 1387.500 3.399
921.600 1406.000 3.387
921.600 1424.500 3.375
921.600 1443.000 3.363
921.600 1461.500 3.350
921.600 1480.000 3.339
921.600 1498.500 3.336
921.600 1517.000 3.336
921.600 1535.500 3.336
921.600 1554.000 3.336

921.600 1572.500 3.336
921.600 1591.000 3.336
921.600 1609.500 3.336
921.600 1628.000 3.336
921.600 1646.500 3.336
921.600 1665.000 3.336
921.600 1683.500 3.336
921.600 1702.000 3.336
921.600 1720.500 3.336
921.600 1739.000 3.336
921.600 1757.500 3.336
921.600 1776.000 3.336
921.600 1794.500 3.336
921.600 1813.000 3.336
921.600 1831.500 3.336
921.600 1850.000 3.336
1024.000 0.000 1.011
1024.000 18.500 1.011
1024.000 37.000 1.011
1024.000 55.500 1.011
1024.000 74.000 1.011
1024.000 92.500 1.011
1024.000 111.000 1.011
1024.000 129.500 1.011
1024.000 148.000 1.011
1024.000 166.500 1.282
1024.000 185.000 1.488
1024.000 203.500 1.701
1024.000 222.000 1.895
1024.000 240.500 2.074
1024.000 259.000 2.245
1024.000 277.500 2.411
1024.000 296.000 2.576
1024.000 314.500 2.746
1024.000 333.000 2.922
1024.000 351.500 3.111
1024.000 370.000 3.317
1024.000 388.500 3.317
1024.000 407.000 3.317
1024.000 425.500 3.317
1024.000 444.000 3.317
1024.000 462.500 3.317
1024.000 481.000 3.317
1024.000 499.500 3.317
1024.000 518.000 3.317
1024.000 536.500 3.317
1024.000 555.000 3.317
1024.000 573.500 3.317
1024.000 592.000 3.317
1024.000 610.500 3.317

1024.000 629.000 3.317
1024.000 647.500 3.317
1024.000 666.000 3.341
1024.000 684.500 3.357
1024.000 703.000 3.365
1024.000 721.500 3.369
1024.000 740.000 3.369
1024.000 758.500 3.366
1024.000 777.000 3.368
1024.000 795.500 3.371
1024.000 814.000 3.370
1024.000 832.500 3.365
1024.000 851.000 3.369
1024.000 869.500 3.369
1024.000 888.000 3.366
1024.000 906.500 3.358
1024.000 925.000 3.342
1024.000 943.500 3.317
1024.000 962.000 3.317
1024.000 980.500 3.317
1024.000 999.000 3.317
1024.000 1017.500 3.317
1024.000 1036.000 3.317
1024.000 1054.500 3.317
1024.000 1073.000 3.317
1024.000 1091.500 3.317
1024.000 1110.000 3.317
1024.000 1128.500 3.317
1024.000 1147.000 3.317
1024.000 1165.500 3.317
1024.000 1184.000 3.317
1024.000 1202.500 3.352
1024.000 1221.000 3.385
1024.000 1239.500 3.414
1024.000 1258.000 3.439
1024.000 1276.500 3.457
1024.000 1295.000 3.467
1024.000 1313.500 3.468
1024.000 1332.000 3.456
1024.000 1350.500 3.427
1024.000 1369.000 3.412
1024.000 1387.500 3.399
1024.000 1406.000 3.387
1024.000 1424.500 3.375
1024.000 1443.000 3.363
1024.000 1461.500 3.350
1024.000 1480.000 3.339
1024.000 1498.500 3.328
1024.000 1517.000 3.317
1024.000 1535.500 3.317

1024.000 1554.000 3.317
1024.000 1572.500 3.317
1024.000 1591.000 3.317
1024.000 1609.500 3.317
1024.000 1628.000 3.317
1024.000 1646.500 3.317
1024.000 1665.000 3.317
1024.000 1683.500 3.317
1024.000 1702.000 3.317
1024.000 1720.500 3.317
1024.000 1739.000 3.317
1024.000 1757.500 3.317
1024.000 1776.000 3.317
1024.000 1794.500 3.317
1024.000 1813.000 3.317
1024.000 1831.500 3.317
1024.000 1850.000 3.317

# APPENDIX D

## SAMPLE APPLICATION SOURCE CODE

| |
|---|
| **remotepowerswitch.c** |
| **project-conf.h** |
| **leds.h** |
| **leds.c** |

## Source code: ‘remotepowerswitch.c’

```
/*
 * Remote Power Switch Example for the Seed-Eye Board
 * Copyright (c) 2013, Giovanni Pellerano
 *
/

/**
 * \file   remotepowerswitch.c
 * \brief  Remote Power Switch Example for the Seed-Eye Board
 * \author Giovanni Pellerano <giovanni.pellerano@evilaliv3.org>
 * \date   2013-01-24
 */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "contiki.h"
#include "contiki-net.h"

#include "erbium.h"

#include "dev/leds.h"

#include <p32xxxx.h>

RESOURCE(toggle, METHOD_GET | METHOD_PUT | METHOD_POST, "actuators/powerswitch",
"title=\"Red LED\";rt=\"Control\"");
void
toggle_handler(void* request, void* response, uint8_t *buffer, uint16_t
preferred_size, int32_t *offset)
{
  leds_toggle(LEDS_YELLOW);

  PORTEbits.RE0 = !PORTEbits.RE0;
}
```

```
PROCESS(remote_power_switch, "Remote Power Switch");

AUTOSTART_PROCESSES(&remote_power_switch);

PROCESS_THREAD(remote_power_switch, ev, data)
{
  PROCESS_BEGIN();

  rest_init_engine();

  TRISEbits.TRISE0 = 0;
  PORTEbits.RE0 = 0;

  rest_activate_resource(&resource_toggle);

  while(1) {
    PROCESS_WAIT_EVENT();
  }

  PROCESS_END();
}

/** @} */
```

## Source code: project-conf.h

```
/*
 * Copyright (c) 2010, Swedish Institute of Computer Science.
 * All rights reserved.
 *
*
 *
 */

#ifndef __PROJECT_RPL_WEB_CONF_H__
#define __PROJECT_RPL_WEB_CONF_H__

#define SICSLOWPAN_CONF_FRAG           1

/* Increase rpl-border-router IP-buffer when using 128. */
#ifndef REST_MAX_CHUNK_SIZE
#define REST_MAX_CHUNK_SIZE          64
#endif

/* Multiplies with chunk size, be aware of memory constraints. */
#ifndef COAP_MAX_OPEN_TRANSACTIONS
#define COAP_MAX_OPEN_TRANSACTIONS   2
#endif

/* Must be <= open transaction number. */
#ifndef COAP_MAX_OBSERVERS
#define COAP_MAX_OBSERVERS           COAP_MAX_OPEN_TRANSACTIONS-1
#endif

#endif /* __PROJECT_RPL_WEB_CONF_H__ */
/*
 * Copyright (c) 2005, Swedish Institute of Computer Science
 * All rights reserved.
 *
```

```
*
 */
```

## Source code: 'dev/leds.c'

```
#include "dev/leds.h"
#include "sys/clock.h"
#include "sys/energest.h"

static unsigned char leds, invert;
/*---------------------------------------------------------------------------*/
static void
show_leds(unsigned char changed)
{
  if(changed & LEDS_GREEN) {
    /* Green did change */
    if((invert ^ leds) & LEDS_GREEN) {
      ENERGEST_ON(ENERGEST_TYPE_LED_GREEN);
    } else {
      ENERGEST_OFF(ENERGEST_TYPE_LED_GREEN);
    }
  }
  if(changed & LEDS_YELLOW) {
    if((invert ^ leds) & LEDS_YELLOW) {
      ENERGEST_ON(ENERGEST_TYPE_LED_YELLOW);
    } else {
      ENERGEST_OFF(ENERGEST_TYPE_LED_YELLOW);
    }
  }
  if(changed & LEDS_RED) {
    if((invert ^ leds) & LEDS_RED) {
      ENERGEST_ON(ENERGEST_TYPE_LED_RED);
    } else {
      ENERGEST_OFF(ENERGEST_TYPE_LED_RED);
    }
  }
  leds_arch_set(leds ^ invert);
}
/*---------------------------------------------------------------------------*/
void
leds_init(void)
{
  leds_arch_init();
  leds = invert = 0;
}
/*---------------------------------------------------------------------------*/
void
leds_blink(void)
{
  /* Blink all leds. */
  unsigned char inv;
  inv = ~(leds ^ invert);
  leds_invert(inv);

  clock_delay(400);

  leds_invert(inv);
}
/*---------------------------------------------------------------------------*/
```

```
unsigned char
leds_get(void) {
  return leds_arch_get();
}
/*---------------------------------------------------------------------------*/
void
leds_on(unsigned char ledv)
{
  unsigned char changed;
  changed = (~leds) & ledv;
  leds |= ledv;
  show_leds(changed);
}
/*---------------------------------------------------------------------------*/
void
leds_off(unsigned char ledv)
{
  unsigned char changed;
  changed = leds & ledv;
  leds &= ~ledv;
  show_leds(changed);
}
/*---------------------------------------------------------------------------*/
void
leds_toggle(unsigned char ledv)
{
  leds_invert(ledv);
}
/*---------------------------------------------------------------------------*/
/*   invert the invert register using the leds parameter */
void
leds_invert(unsigned char ledv) {
  invert = invert ^ ledv;
  show_leds(ledv);
}

/*---------------------------------------------------------------------------*/
```

## Source code: ‘dev/leds.h’

```
#ifndef __LEDS_H__
#define __LEDS_H__

/* Allow platform to override LED numbering */
#include "contiki-conf.h"

void leds_init(void);

/**
 * Blink all LEDs.
 */
void leds_blink(void);

#ifndef LEDS_GREEN
#define LEDS_GREEN  1
#endif /* LEDS_GREEN */
#ifndef LEDS_YELLOW
```

```c
#define LEDS_YELLOW  2
  #endif /* LEDS_YELLOW */
  #ifndef LEDS_RED
  #define LEDS_RED  4
  #endif /* LEDS_RED */
  #ifndef LEDS_BLUE
  #define LEDS_BLUE  LEDS_YELLOW
  #endif /* LEDS_BLUE */

  #ifdef LEDS_CONF_ALL
  #define LEDS_ALL    LEDS_CONF_ALL
  #else /* LEDS_CONF_ALL */
  #define LEDS_ALL    7
  #endif /* LEDS_CONF_ALL */

  /**
   * Returns the current status of all leds (respects invert)
   */
  unsigned char leds_get(void);
  void leds_on(unsigned char leds);
  void leds_off(unsigned char leds);
  void leds_toggle(unsigned char leds);
  void leds_invert(unsigned char leds);

  /**
   * Leds implementation
   */
  void leds_arch_init(void);
  unsigned char leds_arch_get(void);
  void leds_arch_set(unsigned char leds);

  #endif /* __LEDS_H__ */
   ---*/
```