

Wireless Sensor Networks for Updating Reprogrammable Logic Designs in Real-time

Joel Wilder, Vladimir Uzelac, Aleksandar Milenković, Emil Jovanov
The University of Alabama in Huntsville, 301 Sparkman Drive, Huntsville, AL 35899
{wilderj | uzelacv | milenka | jovanov}@eng.uah.edu

Key Words: Reconfigurable logic, wireless sensor networks, embedded system software

Abstract — Recent advances in sensors, low-power system-on-a-chip devices, and wireless communications, have prompted a proliferation of wireless sensor networks. These wireless sensor networks promise many new applications, such as health monitoring, traffic surveillance and navigation, habitat monitoring, warehouse inventory and asset tracking, and infrastructure monitoring. Wireless sensor network nodes typically operate under stringent resource constraints, having only limited energy supplies, memory capacity, and processing power. Since the energy required for wireless communication dominates the overall energy requirements, sensor nodes typically employ onboard data processing (including sampling, filtering, feature extraction, compression, and data encryption), forwarding only processed information to the upper layers. With the requirements for advanced integration, intensive onboard processing, and low power consumption, field programmable gate arrays (FPGAs) emerge as a technology of choice that strikes an optimal balance between processing power, energy requirements, and flexibility. Through the power of reconfigurability, wireless sensor network designs containing reprogrammable logic can be upgraded, errors can be fixed, and limited-resource applications can be dynamically reprogrammed in the field. While powerful, this reconfiguration process can be costly in terms of labor and downtime as a large number of nodes must be manually updated by field engineers. Furthermore, devices that have been deployed for strategic military purposes behind enemy lines may be unreachable.

In order to address these issues we propose a REWISE framework (Reconfigurable Wireless Intelligent Sensor Networks) for real-time reconfiguration of the programmable logic on sensor nodes through the wireless sensor network communication infrastructure. We present hardware and software components of the proposed system and give results of the initial testbed evaluation.

1. Introduction

Recent technological advances in integrated circuits, wireless communication, and sensors have enabled a new generation of wireless sensor networks that can be used in a number of military and civil applications. Typically, a large number of miniature and inexpensive sensor nodes is deployed to monitor and control environments without human intervention for a long period of time [1], [2]. A wireless sensor network usually consists of a number of

wireless sensor nodes and a base station. Various communication models can be employed, such as direct, multi-hop, or clustering. With direct communication, each sensor node communicates to the base station. With the multi-hop model, data are routed through individual sensor nodes toward the base station. With clustering, a subset of nodes organizes a cluster with a cluster head elected in such a way as to minimize power requirements for communication.

Each sensor node typically incorporates the following: (a) one or more physical sensors capable of measuring the state of the environment; (b) actuators that affect the state of the environment; (c) a microprocessor that provides on-sensor processing of “raw” data from physical sensors, facilitates communication, and handles control messaging; (d) a radio interface to communicate information with neighboring nodes or a base station and eventually to the external world; (e) a power source, typically a battery. Most sensor networks, regardless of target applications, share common requirements for miniaturization and low-energy operation. However, they vary greatly in scope, complexity, computation and communication requirements, survivability, security needs, and deployment methods. For example, military applications require a diverse body of robust sensors, which are typically very sensitive and precise, capable of self-organization and secure communication with other sensors and a base station. In addition, deployment of surveillance sensor networks is often a very difficult task, as wise decisions must be made regarding sensor placement when target areas are behind enemy lines. This in turn makes it difficult or impossible to change or reconfigure sensors in the field in order to properly respond to changes in the environment. For example, geological, acoustic, and optical properties of target areas can influence the correctness and/or precision of sensor readings, prompting a modification of the existing set of sensor algorithms in order to process signatures according to design.

Since the nodes in a wireless sensor network must operate under severe size, weight, and power consumption constraints, important hardware and software design decisions must be made in order to meet these requirements while fulfilling system goals. One possible option for meeting these objectives is to use an application-specific integrated circuit (ASIC). ASICs are specifically designed for a target application, and can achieve the best

performance, but their lack of flexibility to accommodate design changes and long design-to-fabrication cycles usually prohibit their use in a sensor node. Another option is a general-purpose processor. This type of processor provides the flexibility that the ASIC cannot offer, but at the price of reduced processing speed. A good compromise between hardware inflexibility and software inefficiency can be found in low-power programmable logic [3]. Reprogrammable logic can be utilized to accelerate critical paths and reduce power consumption for a wide range of signal processing algorithms and communication functions that sensor platforms require [4]. Towards this end, reconfigurable sensor platforms offer flexibility and cost-effective customization before deployment and provide for the possibility of run-time reconfiguration. Furthermore, constructing sensor platforms with programmable logic and a customizable front-end may lower costs, as multiple target applications are able to share a common wireless platform.

In this paper, we introduce a framework for remote reconfiguration of programmable logic devices on sensor nodes through wireless sensor networks. The framework is demonstrated on a wireless network testbed that includes a transmit node, a number of relay nodes, and a custom-built reconfigurable node. We describe the hardware and software components used in the REWISE framework, and present initial evaluations of our originally developed testbed.

The proposed framework allows dynamic reconfiguration of sensor nodes based on a variety of situations, from changes in mission goals, needed upgrades, and bug-fixes, to accommodating environmental changes, through initiation from the base. The latter can be an important step towards wireless sensor networks that can autonomously reconfigure based on changes in the environment. The proposed framework can also be used as a “wireless JTAG” offering simultaneous hardware programming of multiple homogenous reconfigurable platforms. This is significant in systems where the hardware is not easily accessible or is expensive to access, resulting in more efficient maintenance cycles and reduced costs.

The remainder of this paper is organized as follows. Section 2 overviews the related work, and Section 3 introduces the case for real-time reconfiguration through wireless sensor networks. Section 4 presents the proposed reconfiguration framework and the system architecture of the planned REWISE testbed. Section 5 describes the hardware components of the wireless sensor nodes used in the REWISE testbed, and Section 6 describes the software modules and initial results of the field tests. Section 7 concludes the paper.

2. Related Work

FPGA devices consist of an array of computational elements known as logic blocks, a set of routing elements, and a set of input/output cells [5]. Their functionality is determined using configuration bits and they can be reconfigured even after deployment. Typically, in prototype

designs, a new design file is passed through a hard-wired connection to an FPGA on a printed circuit board using JTAG communication. In order to take advantage of reprogrammable logic in system designs that have been deployed in the field, different mediums can be used to transfer this configuration data to target devices. Two example designs that provide in-system reconfiguration from remote locations are included here.

The first example describes a method for remotely reconfiguring FPGAs through the Internet [6]. This Internet Reconfigurable Logic (IRL) system includes a design containing an embedded processor (possibly a single board computer running a real-time operating system, with TCP/IP networking connectivity) and an FPGA, implementing some custom hardware function. New designs can be implemented on a host computer, and the reconfiguration payload can then be delivered to the target computer through a TCP/IP network. Upon arriving at the target computer, the processor validates the payload, and if successful, proceeds to reconfigure the target FPGA.

Hulme *et al* describe a configurable fault-tolerant processor (CFTP) that is used for spacecraft onboard processing [7]. CFTP is intended to evaluate, in various orbital regimes, various reconfiguration system-on-chip designs, such as a triple-mode redundancy, fault-tolerant circuit aimed at overcoming single-event upsets. CFTP is a shared platform used by researchers located across the country to perform fault-tolerant computing and reconfiguration experiments. In this example, the communication medium for passing the new FPGA configuration file is through the Internet, to a ground station, where the data is then transmitted to the orbiting satellite. Upon reception at the satellite, the configuration data is passed through to a PC/104 interface to the CFTP, then to an FPGA that is used as a configuration device, which in turn reprograms the target device. This type of test bed allows for FPGA-based hardware/software designs developed on earth to be tested in the space environment.

3. The Case for REWISE

This section of the paper describes an example REWISE system that uses reconfiguration as a design parameter. This system, as shown in Figure 1, is deployed in a military setting, behind enemy lines, where access is inherently limited, and it can be used to monitor types of traffic on local roads, capture vehicle signatures, obtain video or audio data, detect radiation sources, or monitor environmental conditions. Furthermore, signal processing of acquired sensor data is performed in order to provide data points to a base station and/or a command center. These data points can then be used to make strategic decisions for maintaining security in areas that are under surveillance.

The system is made up of three different tiers. Since wireless communication is a fairly expensive operation in terms of power and time, in order to facilitate reconfiguration, this framework uses local hardware/software repositories at each layer. The upper

layers keep track of hardware/software configurations at the lower layers, and a command can be sent either to initiate reconfiguration from a local repository, or if the needed configuration is not present, begin downloading the required hardware/software context.

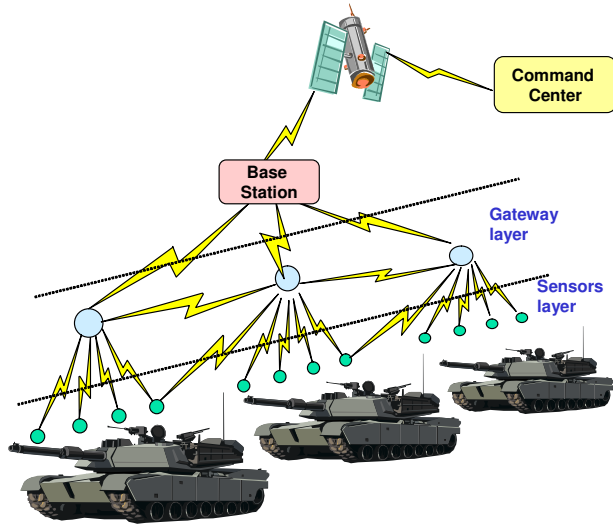


Figure 1. A multi-tier reconfigurable wireless sensor network for surveillance and intelligent signal processing.

At the lowest tier is the sensors layer. A node at this level is comprised of a microprocessor, an FPGA, memory/storage, radio, an energy source, and a type of sensor capable of many different functions, such as measuring magnetic field, radiation, vibration, temperature, humidity, or gathering audio or video data. The number of nodes at the sensors layer could number into the thousands, and the sensor nodes should be so small that they can be embedded in the environment and not easily seen. As such, these nodes will have limited resources, resulting in low power consumption and long life. Since these nodes are resource-limited, the onboard storage can hold different hardware/software configuration data, which can be used to reprogram the FPGA as mission goals change (i.e., alter the function of a node from environmental monitoring to vehicle detection). Each sensor node will gather data, perform basic pre-processing, and then transmit data up to the gateway layer. Figure 2 shows a more detailed explanation of the system.

At the gateway layer, devices have more resources than at the sensors layer. As such, they have greater processing power, higher communication bandwidth, a larger memory/storage capacity, and increased power consumption. These devices can be integrated with an available power grid, but this is not a requirement. A gateway node is comprised of a microprocessor, memory/storage, a radio, and possibly an FPGA, depending on processing needs. Gateway nodes could number into the hundreds, depending on the quantity of sensor nodes, and are responsible for establishing the communication network for the system. A gateway node receives data from

the cluster of sensors they control, synergistically processing this data in order to make intelligent decisions based on mission goals. At this level is a large hardware/software repository that can hold multiple FPGA images. Based on environmental conditions, the current mission, or the results of data processing, a gateway node can initiate reconfiguration at the sensors layer or download new hardware/software contexts to sensor nodes in preparation for near-term needs.

The next level up from the gateway layer is a centralized base station. The base station will have greater resources (increased communication bandwidth, processing power, custom hardware and memory/storage) than nodes at the gateway layer and will serve as a means for providing global interface and control to the system. The base station can be under human-control and may be located at an army base. All data from gateway nodes is gathered and processed at the base station for making decisions regarding mission goals. The base station can be used for initiating reconfiguration at the gateway layer or for downloading new hardware/software contexts to gateway nodes in preparation for future needs.

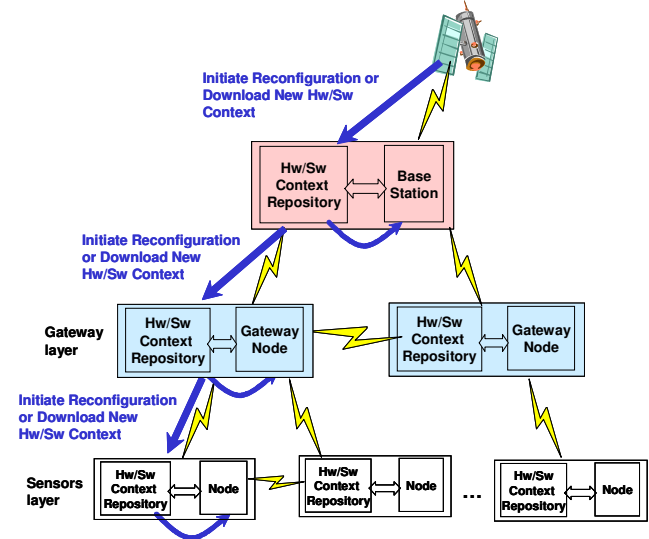


Figure 2. REWISE reconfiguration flow.

Finally, depending on the deployment scenario of the proposed system – perhaps the REWISE system is deployed overseas – a command center can be used to correspond with each base station through satellite communication. This would allow another means for communicating changing mission goals, and new hardware/software designs could be developed in a friendlier environment and then transferred to the system in the field. Furthermore, if the base station needed to be abandoned, then the command center would provide another level of control for initiating reconfigurations or modifying system functionality.

4. System Architecture

Reconfiguration efforts of programmable logic will follow the flow as described in Figure 3. Xilinx software, such as ISE Foundation, is the typical starting point of a design. This design is usually based on a VHDL or Verilog description of a circuit to perform some desired function, which can be improved through simulation and subsequent design iterations. Merged with this existing streamlined, hardware design can be a custom soft or hard processor to enhance functionality. For Xilinx devices, the soft processor is called Microblaze, which is fully customizable with the desired amount of data/instruction block memory, type of memory hierarchy, including cache, an on-chip peripheral bus for controlling communication between external memory controllers, UART peripherals, timers, and interrupt controllers, and a low-latency unidirectional FIFO called a Fast Simplex Link for accommodating communication between other processors or custom hardware. System Generator is a tool that works with Matlab and provides blockset libraries in order to link together Microblaze designs with custom hardware. Once all of these inputs have been integrated to create a project, the ISE tool synthesizes, places-and-routes, and maps the design to a target FPGA. The result of this process is an XSVF file, which is a compressed version of an SVF file. The SVF file is a standard ASCII format syntax specification for describing high-level JTAG bus operations that can be used for programming Xilinx CPLDs, FPGAs, or configuration storage devices.

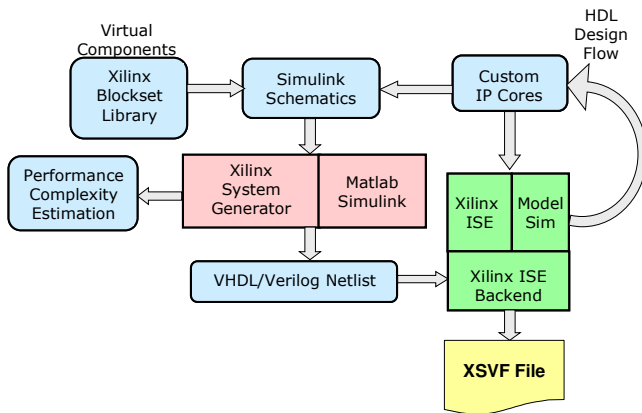


Figure 3. Reconfigurable logic design flow.

With a new design in the form of an XSVF file in place, remote reconfiguration of programmable logic devices through wireless sensor networks is possible. While this method allows for a direct reconfiguration of the SRAM bits inside the FPGA, our method will always reprogram the memory in an EEPROM, which is directly connected to the FPGA and used for the sole purpose of configuring the SRAM configuration bits inside the target device. Thus, if a reboot of the node was ever required, the FPGA would get its configuration from the onboard EEPROM rather than

requiring the XSVF file to be resent from the base station. From this point forward, “reprogramming the FPGA” will refer to the process of reconfiguring the EEPROM, which in turn reconfigures this FPGA. The system architecture of the REWISE testbed to achieve this purpose and demonstrate the concept is shown in Figure 4. It consists of a base station, which is connected to a transmit mote (or node) through a serial link, relay mote(s), and a receive mote, which is connected to the reconfigurable sensor node platform through JTAG. The base station includes a software tool for initiating download of the XSVF file. The XSVF file is streamed serially to the transmit mote, and because of the limited resources of the hardware along the communication path, a flow-control scheme is used in order to direct the data reliably to the reconfigurable platform. The radio transceivers facilitating the wireless communication employ a networking protocol based on IEEE 802.15.4 [8]. Due to the limited range of this networking scheme, relay motes may be needed in order to traverse this path from source to destination. These relay motes will be a part of the wireless sensor network platform, and will typically employ some sensing function in addition to packet forwarding. Finally, a receive mote is connected through JTAG to the reconfigurable platform. The receive mote takes the configuration data and through a software JTAG module provides programming control of the reconfigurable device with the received XSVF file.

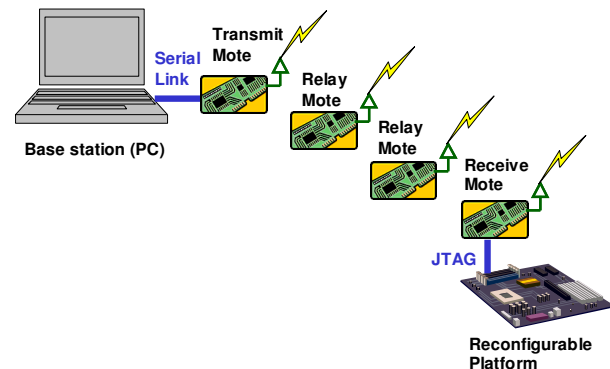


Figure 4. System architecture of the REWISE testbed.

5. Hardware Design

A more detailed description of the hardware modules found in the REWISE testbed is included in this section. First, is the transmit mote, as shown in Figure 5. The transmit mote includes a Texas Instruments MSP430 microprocessor for controlling data transfer from the base station to the downstream receive mote. The MSP430 processor includes a UART peripheral for communication with the base station and a synchronous peripheral interface (SPI) module for handling communication with the Chipcon CC2420 radio transceiver. The CC2420 implements the IEEE 802.15.4 networking protocol and achieves transfer rates of up to 250 kilobits per second. The MSP430 processor uses direct memory access (DMA) transfer in order to efficiently pass data from the UART peripheral into

memory buffers. These buffers constitute a buffering scheme that is used to differentiate between the two tasks of receiving data from the base station and transmitting data downstream through the radio. The MSP430 also handles the flow control for passing data successfully between the base station and the downstream motes. This is necessary because there is not enough onboard storage to allow transfer of the entire XSVF file at once, but it must be divided into smaller blocks.

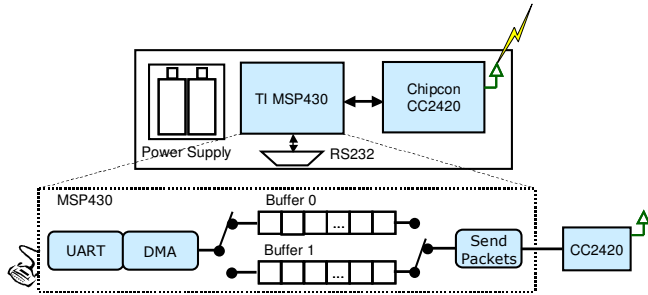


Figure 5. Transmit mote diagram.

As shown in Figure 6, the receive mote is connected to a reconfigurable wireless intelligent platform. This platform is used to both take readings and initiate control over the surrounding environment through sensors and actuators. This is an example REWISE node which is connected to a wireless sensor network through a receive mote. The receive mote also contains a Texas Instruments MSP430 processor and a Chipcon CC2420 radio transceiver for handling communication. The MSP430 handles the flow control responsibilities of the XSVF data stream, receiving the data from upstream nodes, as well as including the JTAG module for providing the configuration mechanism of the reprogrammable device.

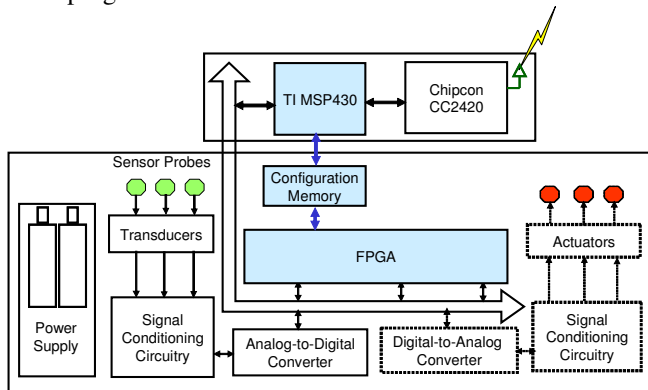


Figure 6. Reconfigurable wireless intelligent platform with receive mote.

6. Software Design

The software design in our REWISE testbed includes the algorithms for the base station, the transmit mote, and the receive mote. We start by describing the data flow that begins at the base station. This is a custom application running on a personal computer (i.e., the base station) and is responsible for initiating the reconfiguration process. This application, as described in Figure 7, takes a configuration

file, which was created by design tools targeting a reprogrammable logic device and is in the compressed XSVF file format, and forwards this data serially on a byte-by-byte basis to the transmit mote. The transmit mote algorithm, as depicted in Figure 8, receives the byte sent from the base station into its MSP430 UART peripheral and subsequently stuffs it into one of two memory buffers (henceforth referred to as the IN/OUT buffers) via a direct-memory access module. This process continues until a kilobyte (kB) of data has been sent.

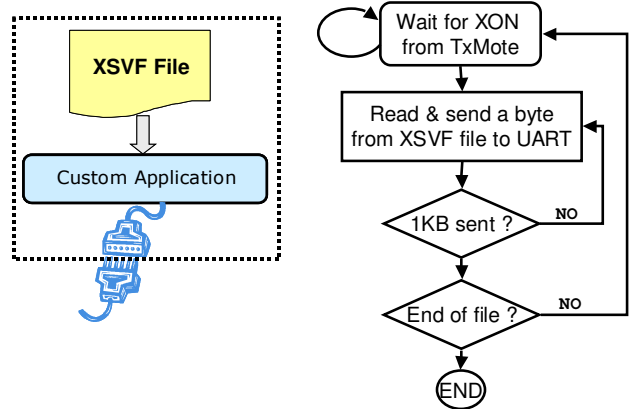


Figure 7. Base station software application for initiating reconfiguration.

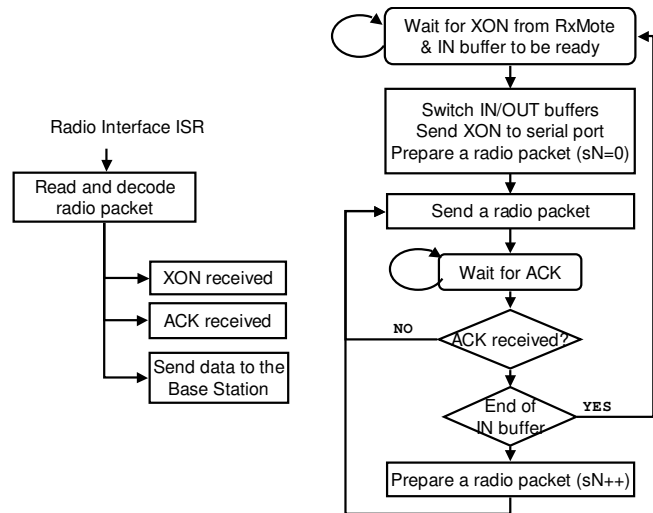


Figure 8. Transmit mote algorithm.

After sending 1 kB of data, the base station application halts and waits to receive a transmit-on character (XON) from the transmit mote. Having received 1 kB of data from the base station, the transmit mote waits to receive an XON character from the receive mote. Upon receiving this response from the receive mote, the transmit mote switches its IN/OUT buffers and sends an XON character to the base station, signifying that there is available buffer space for receiving more data. While the base station begins sending data to the IN buffer in the transmit mote, 64 bytes of data from the OUT buffer are sent to the CC2420 radio

transceiver for wireless transmission. These two tasks can occur simultaneously because the transfer of data into the memory buffer occurs via DMA, which is separate from processor control, allowing the processor to transfer data to the CC2420. The radio sends this packet of data and waits for a given period of time for an acknowledgment from a downstream mote (either a relay mote or a receive mote, as described in the REWISE testbed). If the acknowledgment is not received, then the radio packet is resent. If the acknowledgment is successfully received, the packet sequence number (sN) is incremented and the next 64 bytes from the OUT buffer is packetized and transferred to the radio in preparation for wireless transmission. This process continues on the transmit mote until the full kilobyte of data is sent from the OUT buffer.

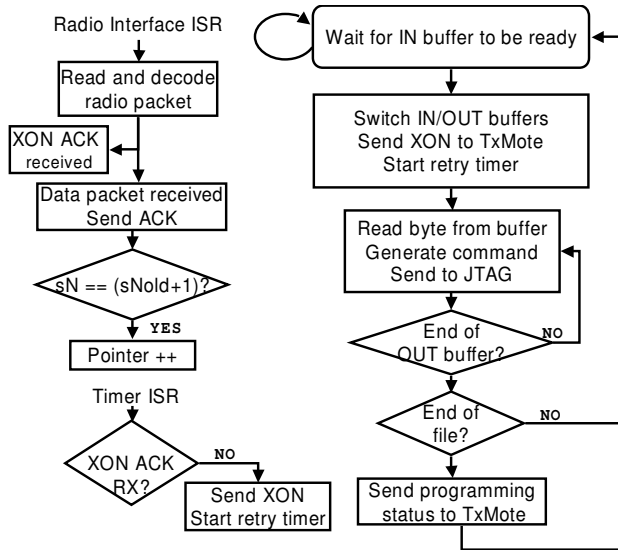


Figure 9. Receive mote software algorithm.

The software application for the receive mote is provided in Figure 9. The receive mote gathers the packets that are sent from the transmit mote, as described in the previous paragraph. Upon receiving a packet into the radio, it checks whether this is a data packet, and if so, sends the acknowledgment for that packet. At this point, the packet is decoded in order to ensure that it contains new data. This is done by checking the sequence number and making sure that this value is the same as the previously received packet's sequence number plus one. If that is the case, then a pointer is incremented, and the data is transferred to one of two memory buffers (henceforth referred to as the IN/OUT buffers). If the new sequence number equals the old sequence number, then the transmit mote did not successfully receive the previous acknowledgment and this packet is discarded, while the acknowledgment is resent. This process will continue until 16 packets have been received, indicating that the IN buffer is now filled with a kilobyte of data. At this point the IN/OUT buffers are switched and an XON message is sent to the transmit mote.

This XON message has the end goal of telling the base station that the next kilobyte of data can be sent.

In order to ensure that the configuration data stream flow is not cut off, the receive mote requires an acknowledgment of this XON message from the transmit mote. This is accomplished through a "retry timer". The retry timer is started as soon as the XON message is sent. Once the timer reaches a certain count, the interrupt service routine is entered and a check is performed to see if the XON message has been acknowledged. If it has not been acknowledged, then the XON message is resent in order to not cutoff the flow of data, and the retry timer is started again.

After the IN/OUT buffers have been switched, data begins to be read from the OUT buffer and sent to the JTAG configuration software module, which then controls the emulated JTAG lines on the MSP430 for the purpose of configuring the reprogrammable device. This JTAG module reads the instructions and arguments from the XSVF file and takes one of three possible paths based on the following instructions: XRUNTEST, XSIR, or XSDR. If the XRUNTEST instruction is encountered, then the next four bytes of data are read, which specify the number of microseconds for which the configuration device stays in an idle state before the next XSIR or XSDR instruction is executed. If an XSIR instruction is encountered, then the JTAG module provides stimulus to the TMS and TCK ports until it arrives in the Shift-IR state. It then reads a byte that specifies the length of the data and the actual data itself, outputting the specified data on the TDI port. Finally, when all the data has been output to the TDI port, the TMS value is changed and successive TCK pulses are output until the idle state is reached again. If an XSDR instruction is encountered, it reads the data specifying the values that are output during the Shift-DR state. The code then toggles TMS and TCK appropriately to transition directly to the Shift-DR state. It then holds the TMS value at 0 in order to stay in the Shift-DR state and the data from the XSVF file is output to the TDI port while storing the data received from the TDO port. After all the data has been output to the TDI port, TMS is set to 1 in order to move to the Exit-1-DR state. Then, the TDO input value is compared to the TDO expected value. If the two values fail to match, an exception-handling procedure is executed. If the TDO input values match the expected values, the code returns to the idle state and waits for the amount of time specified by the XRUNTEST instruction [9].

The JTAG module will operate, as described, on successive bytes from the OUT buffer until the end is reached. Subsequently, the application returns to the state of waiting for the IN buffer to fill, and then the process is repeated until the end of the XSVF file is reached. At this point, the JTAG module provides notification for whether the FPGA was successfully reconfigured. This status is communicated to the transmit mote through the radio, which passes these results to the base station.

With the MSP430 processor on the receive node running at 8 MHz, and the MSP430 processor on the transmit node

running at 6 MHz, and the RS232 port on the base station configured for 115.2 kbits/sec, initial test results based on the REWISE testbed show that it takes 6 msec to transmit and receive acknowledgment of one packet containing 64 bytes worth of data. Thus, it takes 96 msec to transmit 16 packets or 1kB worth of data. This corresponds to a bandwidth of 83 kbits/sec. With a Virtex family FPGA needing an XSVF file containing 675 kB of configuration data, this requires 65 seconds to transmit. Smaller XSVF files are on the order of 45 kB, while larger devices would need around 2.7 MB. FPGAs with a greater degree of capability need more SRAM bits for programming. These values indicate file sizes for complete reconfiguration of the FPGA. However, it is now possible to only perform partial reconfiguration of a device, so this would lower the amount of data needed for transfer, and therefore decrease the overall time required for reconfiguration. Factoring in the JTAG programming time performed on the receive node, which is done in 1kB blocks (set according to the buffer size and available resources in the MSP430 processor) it takes 210 msec for 1kB worth of data transfer. This is the amount of time it takes for the transmit node to send 1kB of data until the XON message is received from the downstream node and the next 1 kB of data is ready to be sent. Thus, the effective bandwidth for transfer and JTAG programming becomes 38 kbits/sec. For an XSVF file containing 675 kB of configuration data, this requires 142 seconds to complete the programming of the downstream reconfigurable node.

7. Conclusions

With recent technological advances in integrated circuits, wireless communication, and sensors a new generation of wireless sensor networks has been conceived that can be used in a number of different applications. Because of the competing requirements for advanced integration, intensive onboard processing, small footprints, and low power consumption found in the hardware making up these wireless sensor networks, FPGAs emerge as a technology of choice that strikes an optimal balance between processing power, energy requirements, and flexibility. Furthermore, leveraging the ability to reprogram FPGAs opens up new options for resource-limited designs, allowing systems to be reconfigured after deployment, which proves especially valuable in terms of handling upgrades, bug-fixes, and modified goals, accommodating environmental changes, and maintaining a large amount of hardware in difficult-to-reach locations.

To achieve these desired goals, we have presented a REWISE framework, made up of reconfigurable wireless intelligent sensor nodes, capable of broadening the functionality and capability of wireless sensor networks.

We have given a test case illustrating the power of a REWISE system and have provided the system architecture of a testbed, upon which our tests were based. Furthermore, the hardware and software components of this REWISE testbed have been described in detail, and initial results have been provided, indicating the feasibility of the design.

Acknowledgment

This work is being supported in part by a National Science Foundation grant IIS-0434156.

REFERENCES:

- [1] D. Culler, D. Estrin, M. Srivastava, "An Overview of Sensor Networks", *IEEE Computer*, pp. 41 – 49, August 2004.
- [2] D. Culler, W. Hong, "Wireless Sensor Networks", *Communications of the ACM*, Vol. 47, No. 6, pp. 30 – 33, June 2004.
- [3] Xilinx, "Spartan-3L Low Power FPGA Family", September 2005.
- [4] J. Rabaey, M. Ammer, J. Silva Jr., D. Patel, S. Roundy, "PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking", *IEEE Computer*, Vol. 33, No. 7, pp. 42 – 48, July 2000.
- [5] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing Surveys*, Vol. 34, No. 2, pp. 171 – 210, June 2002.
- [6] Xilinx, "Architecting Systems for Upgradability with Internet Reconfigurable Logic", June 2001.
- [7] C.A. Hulme, H. H. Loomis, A. A. Ross, and R. Yuan, "Configurable Fault-Tolerant Processor (CFTP) for Spacecraft Onboard Processing," 2004 IEEE Aerospace Conference Proceedings, Vol. 4, pp. 2269 – 2276, March 2004.
- [8] IEEE Std 802.15.4-2006, "Specifications for Low-Rate Wireless Personal Area Networks", <http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf>.
- [9] Xilinx, "Xilinx In-System Programming Using an Embedded Microcontroller", June 2004.