EURASIP Journal on Embedded Systems
a SpringerOpen Journal

RESEARCH Open Access

FPGA based wireless sensor node with customizable event-driven architecture

Junsong Liao, Brajendra K Singh, Mohammed AS Khalid and Kemal E Tepe*

Abstract

This article presents the design and implementation of modular customizable event-driven architecture with parallel execution capability for the first time with wireless sensor nodes using stand alone FPGA. This customizable event-driven architecture is based on modular generic event dispatchers and autonomous event handlers, which will help WSN application developers to quickly develop their applications by adding the required number of event dispatchers and event handlers as per the need of a WSN application. This architecture can handle multiple events in parallel, including high priority ones. Additionally, it provides non-preemptive operation which removes the timing uncertainty and overhead involved with interrupt-driven processor-based sensor node implementation, which is required in real-time wireless sensor networks (WSNs). Thus, higher computation power of FPGAs combined with the non-preemptive modular event-driven architecture with parallel execution capability enables a variety of new WSN applications and facilitates rapid prototyping of WSN applications. In this article, the performance of FPGA-based sensor device is compared with general purpose processor-based implementations of sensor devices. Results show that our FPGA-based implementation provides significant improvement in system efficiency measured in terms of clock cycle counts required for typical sensor network tasks such as packet transmission, relay and reception.

Keywords: Wireless sensor nodes, FPGA, ZigBee, Sensor networking

Introduction

A wide variety of wireless sensor network (WSN) applications have been emerging that requires innovation in sensor devices. Most of the current sensor devices employ general purpose processor-based embedded system such as motes developed by the University California at Berkeley [1]. Those devices consist of a general purpose processor (also referred as commodity microcontroller), wireless transceiver and sensors subsystem. The general purpose processor needs event-based operating systems (OS) tailored for sensor network applications such as TinyOS [2] used in Berkeley motes. The available software tools and debugging platform offer easy development of WSN protocols and applications in these devices. That is why these motes are popular in academic research as well as in commercial applications.

General purpose processors are efficient in sequential execution of instructions and are not primarily designed to exploit the inherent parallelism available in event-driven sensor applications. Furthermore, these processors requires a software based implementation of WSN protocol stack and application, which is always slower than their fully hardware based counterparts. Processor-based sensor nodes are interrupt-driven systems, which brings uncertainty about the processing time of an event as it can be interrupted by other events any time. Additionally, interrupt-driven system have overhead of storing all the states and variables of a running event in order to handle the other event that interrupted the processor. After processing the new event, the processor has to load the states and variables of the interrupted event [3]. Therefore, interrupt-driven systems are not suitable for real-time WSN applications, where strict deadlines and delay guarantees are necessary.

An alternative approach of WSN protocol and application development using the general purpose processor is to use virtual run-time environment instead of event-based OS such as TinyOS. The virtual run-time environment facilitates application development in higher level programming languages such as Java and C# as in IBM's

*Correspondence: ktepe@uwindsor.ca
Department of Electrical and Computer Engineering, University of Windsor, Windsor, Ontario, N9B 3P4, Canada

Moterunner [4]. Though virtual run-time environment offers rapid application development, it is slower and requires more hardware resources than the operating system based solutions such as TinyOS to accomplish the same amount of tasks.

In order to alleviate above mentioned problems with processor-based senor nodes and virtual run-time environments, a sensor device using a non-preemptive modular customizable event-driven processing architecture is proposed and described in this article. The design goal of the proposed sensor node is to provide real-time computation and communication capabilities to a wireless sensor node. Thus, the proposed architecture is implemented fully in hardware using a field programmable gate array (FPGA). In this architecture, an event dispatcher assigns an event to an event handler and allows it to complete the event. Removing preemption gives the ability to define the timeliness in our architecture and removes the need of storing states and variables of ongoing events and loading them back, which significantly reduces the memory usage and time needed for transitions. Apart from this, the proposed event-driven architecture is modular; therefore, a WSN application designer can take as many generic event dispatchers and autonomous event handlers as needed by a WSN application. This facilitates the rapid prototyping of WSN application. This type of modularization is not the inherent feature in interrupt-driven systems. The hardware implementation of the proposed design shows the significant improvement in the sensor device performance in terms of execution cycle count as compared to processor-based sensor nodes. The improvement in cycle count enables the FPGA based sensor device to run at slower system clock while maintaining the same performance as that of a processor-based system.

The proposed FPGA based implementation offers hardware flexibility and speed as compared to the software flexibility offered by the processor-based sensor node implementations. With this hardware flexibility, a WSN application developer can rapidly implement a sensor device for any complex WSN application for which commercial off-the-shelf sensor devices may not be an optimum choice. Additionally, if needed, then the FPGA based sensor device design can be used to develop economical system-on-chip (SOC) or application specific integrated circuit (ASIC) for large scale production.

The main contributions of this article are as follows.

1. A fully hardware based non-preemptive modular customizable event-driven architecture is proposed for a wireless sensor device. In this architecture, various autonomous functional modules such as event dispatchers and event handlers can be added or removed easily for rapid WSN application development. This architecture can be particularly useful for real-time WSN applications.
2. The proposed architecture is implemented on an FPGA to utilize its speed, computation power, and hardware flexibility. Usage of FPGA further enables computationally extensive WSN applications related to multimedia, image processing, and security.

These benefits come at a cost. In order to utilize the proposed architecture, WSN developers have to design various types of autonomous event handlers for different subtasks of their specific WSN applications. They then can use as many copies of an event handler as needed. In the case of processor-based design, the developers are still required to write code for various subtasks, but they may not need to divide their tasks to various autonomous subtasks. Additionally, FPGA based sensor nodes seem to have higher energy consumption than processors based sensor node implementations. However, there are ways to mitigate the power consumption issue in FPGA based sensor nodes, which are discussed in Section Implementation details and test results.

The rest of the article is organized as follows: Section Related study presents a brief background of typical sensor node architectures and related research. Section System architecture and design describes the system architecture and design. Section System components describes the main components used in the implementation of the sensor node. Section Implementation details and test results presents implementation details and experimental results related to the performance of the sensor node. A discussion on the power consumption issue is also provided in this section. Finally, Section Conclusions concludes the article.

Related study

WSNs have a range of applications with vastly varying requirements and characteristics [5]. For example, active sensors, such as sonar, require much more computational power for signal processing than passive sensors, such as smoke detector. Sensors for applications that support mobility need more processing power for complex network protocols and algorithms than their fixed relatives. That is why a single hardware platform is not sufficient to support such wide range of possible applications [6]. In order to cater this range of WSN applications, various sensor node implementations are available both in the academia and industry. Berkeley Mica [7], UCLA Medusa [8], and MIT μAMP [9] are some examples of dedicated embedded sensor device architecture. These embedded sensor devices have a general purpose processor on which the event-driven operating system, such as TinyOS [2] can be installed. The general purpose processor has to run the operating system (OS) routines and process interrupt

handlers, which increase the execution cycles required per task. For example, a sequence of tasks of sampling sensor output, averaging the sampled data, and displaying the result in Berkeley's Mica Mote needs overhead of 781 cycles for interrupt handling and scheduling out of total 1118 execution cycles to complete the tasks, which translates into 70% overhead [10]. There have been efforts to design dedicated sensor network processors such as sensor network asynchronous processor (SNAP) [11] and second-generation sensor processor [12]. These processors have optimized instruction sets and improved architectures for better task scheduling for sensor network applications. However, OS related overhead still exists in these dedicated processors based sensor nodes. In addition to this, efforts have been made to put additional hardware modules along with processors to accommodate event-driven nature of WSN applications. For example, an event processor is proposed to work in conjunction with the general purpose processor in [13]. This event processor is designed to efficiently perform most of the basic functions of the processor in an efficient way. For example, it provides better repetitive interrupt handling and accelerates tasks such as message preparation and routing. As a result, this event processor-based sensor node is able to reduce 40% to 96% of execution cycle counts to process regular WSN application tasks [13] as compared to TinyOS based Mica2 sensor nodes. Though, the processor is proposed to be used minimally in this system, overhead and timing uncertainty related to interrupt handling still exists. Similar to this study, our sensor device implementation also tries to harness the event-driven nature of WSN applications. However, our implementation completely eliminates the need of a general purpose processor by directly implementing the proposed event-driven architecture into the dedicated hardware (an FPGA).

FPGAs are emerging as a viable option in implementation of WSN nodes. An extensive survey on the suitability and challenges of using FPGA for various WSN applications are provided in [14]. It is pointed out in this survey that FPGAs are particularly useful for computationally demanding WSN applications. For example, Xilinx Spartan-3 FPGA based WSN nodes are used to implement complex functions such as fuzzy logic based link cost calculation for routing decision, image compression, running Gaussian average for image background subtraction, and symmetric cryptography algorithm in [15-18], respectively. All above FPGA implementations are used to implement some specific type of WSN application to harness higher computational power capabilities of FPGAs. These tasks were either very time consuming or not possible due to limitation of resources in processor-based sensor nodes. The other category of WSN applications that use FPGA based sensor nodes are multimedia data transmission [19,20], sensor data security related to encryption algorithms [21,22], environmental monitoring [23-25]. As a continuation of this effort, the proposed FPGA based sensor node can enable a new class of applications such as real-time WSN applications mentioned in [26-30]. In addition to this, our implementation will also enable WSN applications that need both higher computational power and real-time capability such as with smart grid gateway node [31].

In the literature, WSN nodes are implemented in four modes, (1) standalone FPGA based sensor nodes without processor core inside, (2) standalone FPGA based sensor nodes with processor core inside, (3) sensor nodes based on combination of onboard processor and FPGA, and (4) standalone processor-based sensor nodes. Each WSN application has its own requirements and one of the above sensor node implementation can offer a meaningful solution. It is up to the WSN developer to make this decision based on the application's requirement. For example, there are cases where sensor nodes with both onboard FPGA and processor, as in [32,33], have advantages such as utilizing high computational capability of FPGA combined with software programmability of processor. To get the advantage of both computational capability and software programmability while using only FPGA, there are various FPGA based wireless sensor device implementations proposed in the literature that use processor core inside the FPGA [34-37]. As mentioned earlier, standalone FPGA based sensor nodes without a processor core inside [15-18] are customized for various types of WSN applications, and these customized designs cannot be used for other WSN application. In this article, a generic modular event-driven architecture is implemented on standalone FPGA based sensor node without processor core which can be applied to any WSN application. As it avoids interrupt related overhead and timing uncertainties, it is particularly suitable to applications that require real-time processing and communication capabilities.

The following section, provides details of the proposed FPGA-based sensor node design.

System architecture and design

The block diagram of the proposed hardware based sensor node implementation is shown in Figure 1. There are five main function units in this implementation and these are the event-driven systems, except the parsing and classification unit. Further details of these functional units are given in Section System components. The event-driven architecture of our sensor system is presented in the following three sections.

Event-driven architecture

The general architecture of the event-driven system is shown in Figure 2. There are event handlers and event dispatchers in the architecture. The event dispatcher is

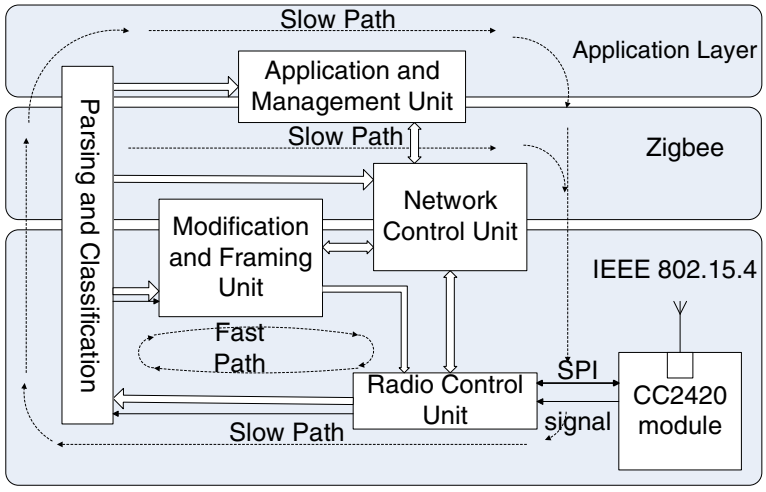

Figure 1 Block diagram FPGA based sensor node with wireless interface.

used to detect the occurrence of an event and activate the corresponding event handler to perform the tasks related to this event. An event could be internal, such as a timeout, or external, such as detection of a packet in the radio module's buffer. Event handlers are specific processing hardware subsystems activated by an event dispatcher in response to a specific event or by an event handler in the case that a new event is triggered by that event handler. Event handlers consist of event processing unit, local registers and shared registers. The event processing unit contains customized state machine and data path for processing a particular event. Local registers are used for temporary storage of intermediate data during event processing. It could only be accessed by the event handler. The shared registers are used to store event information received from the event dispatcher. Both event handler and event dispatcher are able to access to the shared registers. A control signal is used to trigger the event handler to change its state from idle to active. The state signals indicate the state of event handlers to event dispatchers. The data bus is used to transfer event information from event dispatcher to event handler or from one event handler to another event handler. The output ports of the system are driven by the event handlers that are responsible for communication between functional units.

An event dispatcher is a state machine designed to monitor the input ports of hardware blocks, receive event information, decode event information and activate corresponding event handler. Figure 3 illustrates a simplified version of actual state machine of an event dispatcher. The event dispatcher stays in the idle state after resetting or power ON until an event happens. When an event happens, the event dispatcher receives preliminary event

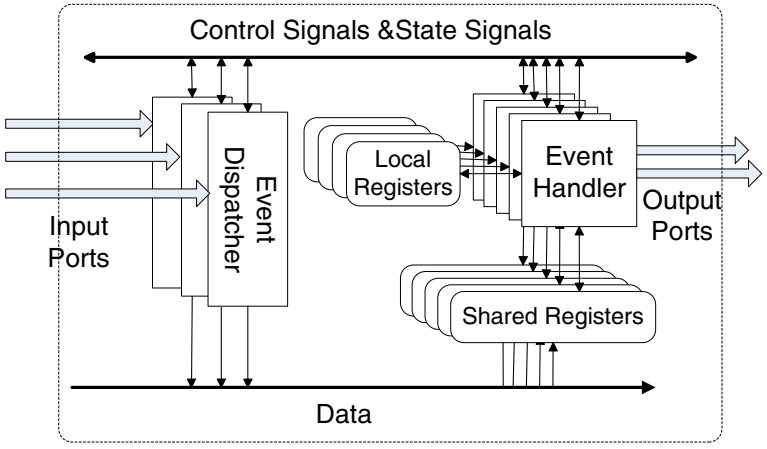

Figure 2 Event-driven architecture.

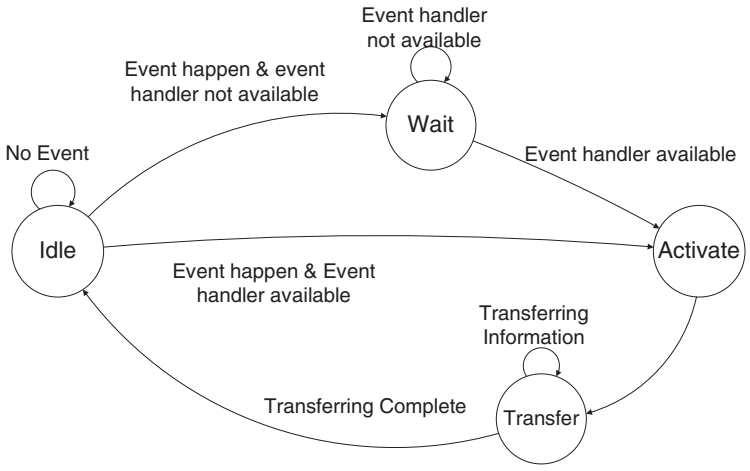

Figure 3 Event dispatcher state machine.

information from the input port and determines the type of event to find out which event handler needs to be activated. If the corresponding event handler is idle, the event dispatcher activates the event handler by signaling the event handler and changing its state from idle to JobAdd state. Then the event dispatcher receives the complete information about the event from the input port and sends it to the shared registers of activated event handler. After the information transaction is complete, the event dispatcher signals the event handler to leave JobAdd state and begins processing the task. Finally, event dispatcher returns to idle state and continues to monitor the input interface. If the event handler is not in idle state when an event dispatcher detects an event, the event dispatcher will hold the communication on the input port and change its state to the wait state until the event handler is idle. When event handler is idle, the event dispatcher continues the activation procedure as mentioned above.

Each event handler is a small processing unit with customizable control logic and data path to perform a specific task. Figure 4 illustrates a simplified version of state machine of an actual event handler. There are three states in all event handlers' state machine: (1) Idle, (2) JobAdd, and (3) JobStart. All the event handlers have the same functionality in any given state; and the change of state of a event handler from idle to JobAdd and from JobAdd to JobStart is solely controlled by the event dispatcher or another event handler. An event handler stays in Idle state after reset or power ON. In Idle state, the event handler is free and ready to accept a task. The event handler is signaled to change from Idle to JobAdd state after the event dispatcher detects an event. The JobAdd state shows that the event handler is engaged in receiving the event information. When in this state, the event handler waits for the completion of the transfer of necessary information for task processing from event dispatcher to its shared registers. When the information transfer is complete, the event dispatcher signals the event handler to change its state from JobAdd to JobStart. In that state, the event handler begins to process the specific task. After completing the task, the event handler will return to Idle state and wait for the next event to happen. The event handlers do not provide preemption operation and there is no priority to access a particular event handler. An event is processed on first-come-first-serve basis. When an event handler is busy, event dispatcher has to wait until it is free. The status bit of an event handler is used to ensure the non-preemptive operation of event handlers.

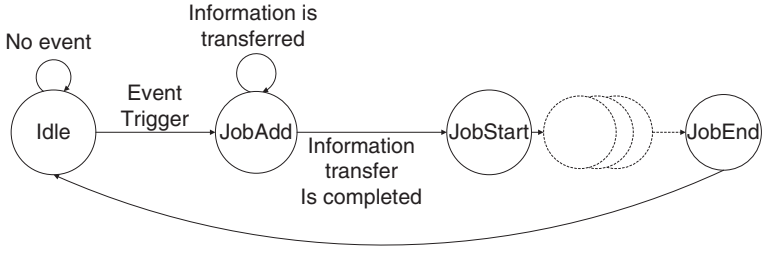

Figure 4 Event handler state machine.

The status of an event handler is stored in a flip-flop. A '0' bit in the flip-flop indicates free status, whereas, '1' bit indicates busy status. Thus, when event dispatcher has input event, it checks the status of the event handler. If it is busy, then event dispatcher returns to wait state and does not assign any task until the event handler is free. Once event handler finishes its job, it switches its status to free; additionally, it informs the status change to event dispatcher as well so that the dispatcher can assign the next task to the event handler if it still has events in its queue to be processed. Not allowing preemption is not restricting the usability of the proposed architecture in event-driven WSN applications. For urgent or high priority events, some event dispatcher and event handler unit can be reserved, which will not be used for other normal events. Thus, a WSN application designer can choose the number of event dispatcher and event handler as per the WSN application demand and can reserve some of them for high priority applications. It is an alternate way of having the similar functionality as the preemption operation.

As described above, the event dispatcher is responsible for the event detection and the event handler is responsible for event processing. Therefore, the event detection and event processing phases could overlap. Also, because event handlers are able to work independently, the event processing phases of different events can overlap, which makes it possible to process the events in parallel. These are the key features that make our event-driven architecture as customizable and scalable. The proposed sensor node architecture locates all computation resources locally as needed by the event handler to process a particular event. Therefore, event handlers do not need to signal an interrupt to request the resources for task processing. As a result, there is no interrupt overhead in our architecture.

Hardware acceleration

The most frequent activities of a sensor device are sending, receiving and relaying packets. These activities involve operations directly on packets, such as parsing, classification and framing. Those operations can be accelerated by hardware in network processor using parallel processing [38,39]. As illustrated in Figure 1, the proposed system adopts the architecture similar to network processor to accelerate packet operations. There are two information paths in the proposed system: (1) fast path and (2) slow path. The functional units connected with the fast processing path deal with operations that are directly performed on packets, such as header modification, parsing, classification, and packet framing. These fast processing paths are customized for handling IEEE 802.15.4 frames and ZigBee packets. The slow processing path deals with operations that relate to network management and upper layer applications, such as network setup, route searching and routing table updates. Usually, these functions are implemented by the central processing units in a processor-based sensor node. However, in our system, a set of specific event handlers are used to perform these tasks.

Parallel distributed computation

Each event handler and event dispatcher has its own data path and control hardware inside their respective functional units. As described earlier, the event handlers work in parallel. The parallel architecture in the proposed system increases the efficiency of message processing and the throughput of the system. The computation capability is distributed among event handlers enabling the designed system to respond to an event whenever it occurs. As shown by the test results in Section Implementation details and test results, the parallel distributed computation allows the system to respond faster to an application's needs.

System components

As shown in Figure 1, the proposed system consists of five functional units. These are parsing and classification unit, framing unit, network control unit, application management unit, and radio control unit. These functional units are connected to each other through dedicated links. The functions of these units are described as follows.

Parsing and classification unit

The parsing and classification unit parses the received packets into information fields. The parsed information is sent to a corresponding module for further processing based on the packet type. For example, the MAC command is sent to network control unit and the application data is sent to application and management unit.

Framing unit

The framing unit is responsible for packing the data into a ZigBee frame before the packet is sent to other ZigBee nodes. It also requests the next-hop address from network control unit for relaying the packet received from parsing and classification unit. Some fields in these packets, such as MAC source address and MAC destination address need to be modified before being sent to ZigBee network.

Network control unit

The network control unit is responsible for ZigBee network management. It provides network functions, such as network discovery, network start, network join, neighbor table maintenance, and routing. It also provides the operational parameters and commands to CC2420 transceiver.

Application management unit

The application management unit is responsible for the application control and user interface. In order to test our system, we developed an application in this unit to read data from a counter, send the data over the network, and display the received data.

Radio control unit

The radio control unit provides an interface with the CC2420 transceiver module. It performs three operations: get-packet, send-packet and transceiver-configuration. Get-packet operation gets packets from CC2420's buffer and sends the packets to parsing and classification unit. Send-packet operation receives packets from other module, transfer the packets to CC2420 TX buffer and issue send command. Transceiver-configuration operation is used to configure CC2420 module and control the operation of the transceiver.

CC2420 unit

The low power, IEEE 802.15.4 [40] and ZigBee compatible, RF chip CC2420 [41] is used as a wireless transceiver. The chip provides features such as error detection and data buffering for the packets that need to be transmitted and received.

Implementation details and test results

Implementation details

Two sensor nodes were implemented using Celoxica's RC10 FPGA development boards [42] and CC2420 evaluation boards (CC2420EB) [41] from Chipcon. The FPGA on Celoxica's RC10 development boards is Xilinx Spartan3 XC3S1500-FG320-4, which contains 29,952 logic cells. Each logic cell has a 4 input look-up table (LUT) and a D flip-flop. The system capacity of the FPGA is equivalent to 1.5 million gates. The CC2420EB contains CC2420 chip and peripherals (i.e., antenna, receiver and transmitter filters) that support the normal operation of the chip. The integrated system is shown in Figure 5.

To compare the circuit sizes of hardware based implementation and the processor-based sensor node implementation, a Xilinx's soft core CPU, MicroBlaze, is also implemented in FPGA as a reference design of processor-based sensor node. MicroBlaze is a 32-bit soft processor with RISC architecture. The processor-based implementation includes a 32 Kbyte RAM, which is sufficient for both ZigBee protocol stack and the application program.

The hardware based sensor node implementation is developed in VHDL. This VHDL design and MicroBlaze CPU IP core are synthesized using Xilinx ISE7.0 [43] and simulated using Mentor's ModelSim simulator [44]. The synthesis details are given in Table 1. As illustrated in Table 1, the hardware based implementation of our event-driven architecture and a WSN packet generation and routing application uses 52% of the total FPGA logical capacity, and IP core of MicroBlaze processor uses 13%. Since the FPGA logic capacity usage is directly related to its circuit size, it is fair to say that the circuit size of the hardware-based sensor node implementation is about four times larger than that of the MicroBlaze processor-based software implementation. It is a trade-off between performance and circuit size. The proposed event-driven architecture is not loaded into the MicroBlaze processor as our event-driven architecture is not processor compatible. IP core of MicroBlaze processor is loaded in our FPGA just to know how much FPGA resources it takes as compared to our fully hardware based FPGA implementation of sensor nodes.

Test results

In this article, a unique design of a non-preemptive modular customizable event-driven architecture is proposed for a standalone FPGA based sensor node. Therefore, the standard benchmark tests are not available to test our implementation. Thus, we tried to find standard application that can run on the proposed sensor node. As one of the prime objectives of WSN is to transmit and receive data packets, a standard ZigBee transceiver application is implemented to test the data packet processing and routing functionality of the proposed FPGA based sensor node. In general, every sensor node in a WSN has to have routing capability. Thus, implementing routing within the FPGA based sensor node is important. However, event handlers can be designed to provide any kind of sensor data processing and packet handling tasks in addition to routing functionality. Thus, the functional testing of routing capability of a sensor node is carried out. However, a full-fledged event-driven architecture for the FPGA based sensor node is designed and implemented using VHDL in this research rather than just FPGA based router. As transceiver application is implemented on a sensor node, measuring packet error rate (PER) and transmission range is the common practice to validate a transceiver. Thus, PER and transmission range are measured experimentally as part of functional verification of the HDL implementation. For performance analysis, the execution cycle count and the minimum clock frequency requirement to achieve the maximum possible throughput are recorded. Execution cycle count and minimum clock frequency requirement are measured using ModelSim simulator.

The transceiver test application mimics data monitoring scenario of a real-world sensor application, in which the system periodically collects sampled data and transmits packets to the destination node. The destination node receives packets, processes them and displays the sampled value. Functional test cases covered in this application include ZigBee network searching, network setup,

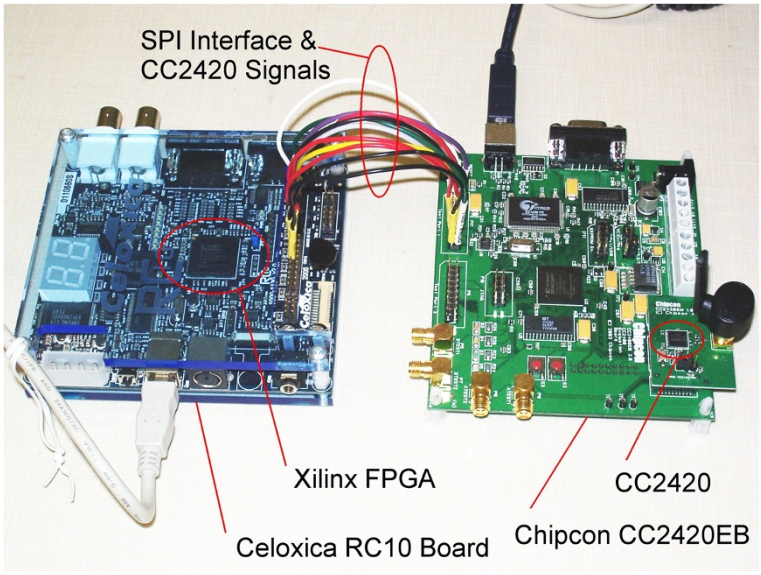

Figure 5 FPGA based ZigBee sensor device.

network association, data sending, data receiving etc. The test results of transmission range and PER are shown in Figure 6. From the experiments, it is observed that our FPGA based sensor node is able to provide reliable communication up to 30 m at indoor environment and 55 m at outdoor environment with PER 2%. After these distances, the system still functions with higher PER (about 20%) up to 55 m at indoor environment and 85 m at outdoor environment, respectively. During these tests, FPGA board operated at 1.2 MHz clock frequency. Radio transmitter output power is 0 dBm and antenna gain is 4.4 dBi. Our PER results are consistent PER results obtained from the MicaZ motes based testbed developed for ZigBee performance measurement in our lab [45].

To evaluate the increase in speed provided by our hardware-based sensor device implementation, execution cycle counts are compared with the typical sensor devices found in the literature, namely general purpose processor-based solution [7], sensor processor-based solution [11], and event processor-based solution [13].

Execution cycle counts required for the regular sensor network tasks are used as a performance metric in order to identify the gain that is achieved with event-driven architecture based on FPGA design. The execution clock is used in the literature as one of the standard ways [11,13] to compare the efficiency of any implementation. From that, how fast a task can be executed can be derived by using the clock frequency. The need for fewer execution cycles means that system can either run for shorter time at the same clock frequency or run for longer time at lower clock frequency (while still ensuring the task completion on time).

Table 1 Logic usage comparison between hardware based implementation and MicroBlaze CPU

Resource type	Resource available	Hardware based design		MicroBlaze CPU based design	
		Used	Percent	Used	Percent
Slices	13312	6888	52%	1665	13%
Slice flip flops	26624	3549	13%	1248	5%
4 input LUTs	26624	12967	49%	2247	8%
bonded IOBs	221	36	16%	81	37%
GCLKs	8	2	25%	3	38%
BRAMs	32	0	0%	16	50%
DCM_ADVs	4	1	25%	2	50%

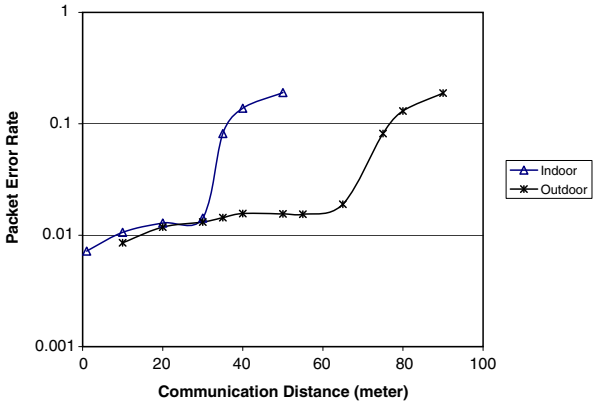

Figure 6 Transmission distance versus packet error rate.

Completing a given task in shorter time or at lower clock frequency can reduce dynamic power consumption; however, the power consumption comparison is not made in this article. Here, it is worth mentioning that operating frequency of contemporary FPGAs for sensor nodes is 10 to 20 times more than the contemporary processor (i.e., microcontroller) for sensor nodes. Processors that are proposed for WSN operates at lower frequencies (such as ATMEL Atmega128 used in MICA2 motes operates at around maximum frequency of 16 MHz [43]), whereas the operating frequencies of Spartan-3 series FPGA can be from 20 to 200 MHz depending upon speed grade and the built-in functions of FPGA [43]. Newer generation FPGAs such as low power Artrix-7 series from Xilinx runs from 200 to 500 MHz; whereas the newer processors for sensor nodes such as Atmel's ATmega1281 microcontroller in IRIS motes still runs at 16 MHz [46]. Thus, because of disparity in maximum operating frequencies of contemporary FPGAs and microcontrollers for sensor nodes, the timing analysis based on clock frequency alone would make it difficult to identify the gain achieved with the proposed hardware implementation compared to processor-based software implementations for a given width of the datapath.

The regular sensor network tasks used to measure the cycle counts in the evaluation are described as follows.

1. Packet transmission: The application management unit periodically collects samples and sends these values to the network control unit, which decides the destination address for the packet and looks up next-hop address. All the packets are sent to framing unit to pack into a MAC frame. The frame is then transferred to radio control unit and transmitted.
2. Packet reception: The radio control unit fetches the MAC frame from CC2420 buffer when the packet is ready, and sends the frame to parsing and classification unit. If the destination of the packet is the current sensor node, then the content is parsed from the frame and sent to its network control unit or application management unit.
3. Packet relay: After reception of a packet, if the packet is not for the current sensor node, the system will modify the relevant fields of the frame, and send it to the next-hop device or to the destination node.

The execution cycle counts for the tasks explained above is provided in Table 2 for different sensor node implementations. From Table 2, our sensor node implementation is about 11 times faster than Mica2 nodes in packet transmission task and about 3 to 4 times faster in packet reception task and packet relay task. Our fully hardware based event-driven architecture has comparable cycle count performance in packet transmission and 30% less cycle count for packet reception as compared to the event processor-based architecture (implemented on general purpose processor). Thus, implementation of network tasks directly into hardware reduces the execution cycle count by 3 to 10 times as compared to a typical processor-based sensor device. Such improvements are possible due to the absence of an operating system and interrupt handling overhead as well as the utilization of parallel processing in our implementation. For example, in packet relay task, framing unit will start a route lookup event as soon as it receives the destination address of the relay packet. The framing event handler and route lookup event handler are able to run in parallel. The packet relay performance of our system is the best among all the systems evaluated in this article.

It is worth mentioning here that if we had operated Xilinx Spartan3 FPGA to its maximum possible operating frequency of 20 to 200 MHz, we would have got more increase in speed. Additionally, if we take high end FPGAs for sensor nodes such as low power Xilinx Artix-7 series, then certainly increase in speed would be better.

Figure 7 shows the minimum clock frequency requirement of our FPGA based implementation to maintain 250 kbit/s system throughput (the maximum possible data rate through CC2420 based wireless interface) for typical sensor node tasks described earlier in this section. The packet transmission graph shown in the figure includes the tasks of event sampling, packet generation and packet transmission, therefore, the minimum clock frequency requirement for transmission task is high as compared to relay and reception task. The performance of the proposed event-driven architecture is quite stable. Once the system clock frequency is increased to the level to achieve the maximum possible system throughput (250 kbit/s), the system throughput remains constant irrespective of increase in system clock frequency, frame size, or the incoming or outgoing packet load conditions.

Table 2 The execution cycle counts for regular sensor network tasks for different sensor node implementations

Task	Mica2 node (8-bit)	SNAP node (16-bit)	Event processor-based node (8-bit)	FPGA based node (8-bit)
Packet transmission	1532	331	127	137
Packet reception	234	258	136	71
Packet relay	429	418	165	115

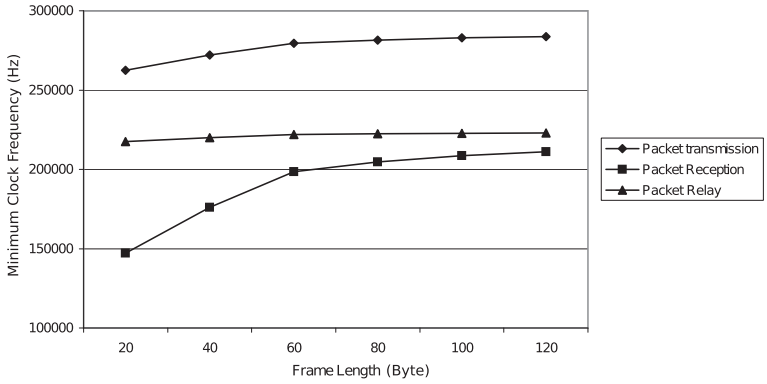

Figure 7 Minimum clock frequency requirement to achieve maximum system throughput.

Discussion on power consumption

In this research, the reason of using FPGAs for sensor nodes was not the minimization of energy consumption but improving timing efficiency to meet certain types of WSN application's demanding real-time operations and computation power. Some of these applications are multimedia data transmission and smart grid gateway nodes.

If low power consumption is required simultaneously with real-time and computational power, emerging low power FPGAs can be used to extend the lifetime of batteries on the WSN nodes. For example, IGLOO series from Actel consumes $2\,\mu$W in ultra low-power mode. Thus, low power FPGAs based sensor nodes can compete with processor-based sensor nodes. Even our implementation with relatively older generation Xilinx Spartan3 XC3S1500-FG320-4 FPGA with its static power consumption of 41 mW provides comparable power consumption performance to that of commonly used processor-based implementations. For example, Mica2 mote with ATMega128 microcontroller operating at 7.4 MHz with CC1000 radio module consumes total active power of 89 mW, and similarly Telos-B mote with TiMSP430 microcontroller operating at 8 MHz with CC2420 radio module consumes total active power of 32 mW [14].

Another power mitigation technique is to implement the design in ASICs. The experimental study conducted in [47] with 90 nm CMOS FPGA and ASIC demonstrated that on average, an FPGA consumes 14 times more dynamic power than the ASIC implementation. Porting our implementation to both low power FPGAs and ASIC can be done with minimum effort as the code is written in VHDL.

We also would like to point out that high computational power provided by FGPA based sensor nodes allows better in-node data processing, which can minimize the amount of data that is to be forwarded/sent/received by a sensor node instead of forwarding/sending raw sensor data. Thus, FPGA's higher processing capability can be used to achieve higher level of data aggregation and compression, which can minimize the amount of transmission for a given amount of information.

Conclusions

In this study, a fully hardware based sensor node is implemented using general purpose FPGA and ZigBee radio interface. This design eliminates timing problems associated with interrupt-driven processor-based WSN nodes and overhead associated with operating system's interrupt handling. The implementation can be tailored to meet strict deadlines, hence provide real-time operation. The hardware implementation can reduce the number of execution cycles required to complete a task by at least 30% when compared to processor-based implementations for the same datapath width. Although processor-based sensor node implementations offer flexible architecture for researchers, this study demonstrated that a hardware based sensor node implementation can provide performance improvement in terms of fewer execution cycle count for applications such as multi-media, data compression, and real-time transmission in WSNs. The proposed architecture provides an FPGA based viable WSN node architecture.

Competing interests

The authors declare that they have no competing interests.

Acknowledgments

This study was supported in part by the NSERC Discovery Grant program.

Received: 20 September 2012 Accepted: 27 January 2013
Published: 19 April 2013

References

1. J Hill, R Szewczyk, A Woo, S Hollar, D Culler, K Pister, in *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*. System architecture directions for networked sensors (ACM Press, New York, 2000), pp. 93–104
2. TinyOS. http://www.tinyos.net
3. J McGivern, *Interrupt Driven PC System Design*. (Annabooks, San Diego, 1998)
4. IBM Moterunner. http://www.zurich.ibm.com/moterunner/
5. F Zhao, L Guibas, *Wireless sensor networks: An information processing approach*. (Morgan Kaufmann Publishers Inc., San Francisco, 2004)

6. K Romer, F Mattern, The design space of wireless sensor networks. IEEE Wirel. Commun. **11**(6), 54–61 (2004)
7. JL Hill, DE Culler, Mica: a wireless platform for deeply embedded networks. IEEE Micro. **22**(6), 12–24 (2002)
8. A Savvides, MB Srivastava, in *ICCD '02: Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02)*. A distributed computation platform for wireless embedded sensing (IEEE Computer Society, Washington, 2002), p. 220
9. A Chandrakasan, R Min, M Bhardwaj, S Cho, A Wang, in *Proceedings of the 28th European Solid-State Circuits Conference (ESSCIRC)*. Power aware wireless microsensor systems (Florence, Italy, 2002), pp. 47–54
10. L Nazhandali, B Zhai, J Olson, A Reeves, M Minuth, R Helfand, S Pant, T Austin, D Blaauw, in *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*. Energy optimization of subthreshold-voltage sensor network processors (IEEE Computer Society, Washington, 2005), pp. 197–207
11. V Ekanayake, I Clinton Kelly, R Manohar, in *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*. An ultra low-power processor for sensor networks (ACM Press, New York, 2004), pp. 27–36
12. L Nazhandali, M Minuth, B Zhai, J Olson, T Austin, D Blaauw, in *CASES'05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*. A second-generation sensor network processor with application-driven memory optimizations and out-of-order execution (ACM Press, New York, 2005), pp. 249–256
13. M Hempstead, N Tripathi, P Mauro, GY Wei, D Brooks, in *ISCA'05: Proceedings of the 32nd annual international symposium on Computer Architecture*. An ultra low power system architecture for sensor network applications (IEEE Computer Society, Madison, 2005), pp. 208–219
14. A de la Piedra, A Braeken, A Touhafi, Sensor systems based on FPGAs and their applications: a survey. Sensors. **12**(9), 12235–12264 (2012)
15. T Haider, M Yusuf, in *9th International Multitopic Conference, IEEE INMIC*. FPGA based fuzzy link cost processor for energy-aware routing in wireless sensor networks—design and implementation (IEEE Computer Society, Karachi, 2005), pp. 1–6
16. Y Wang, A Bermak, F Boussaid, in *2010 2nd Asia Symposium on Quality Electronic Design (ASQED)*. FPGA implementation of compressive sampling for sensor network applications, Penang, Malaysia, 2010), pp. 5–8
17. DM Pham, S Aziz, in *2011 Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. FPGA architecture for object extraction in wireless multimedia sensor network (Adelaide, Australia, 2011), pp. 294–299
18. E Eryilmaz, I Erturk, S Atmaca, in *International Conference on Application of Information and Communication Technologies*. Implementation of Skipjack cryptology algorithm for WSNs using FPGA (Baku, Azerbaijan, 2009), pp. 1–5
19. M Kaddachi, A Soudani, I Nouira, V Lecuire, K Torki, in *2010 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. Efficient hardware solution for low power and adaptive image-compression in WSN (Athens, Greece, 2010), pp. 583–586
20. CH Zhiyong, LY Pan, Z Zeng, MH Meng, in *IEEE International Conference on Automation and Logistics*. A novel FPGA-based wireless vision sensor node (Shenyang, China, 2009), pp. 841–846
21. P Hamalainen, M Hannikainen, T Hamalainen, in *48th Midwest Symposium on Circuits and Systems*. Efficient hardware implementation of security processing for IEEE 802.15.4 wireless networks. vol. 1 (Cincinnati, Ohio, USA, 2005), pp. 484–487
22. O Song, J Kim, in *7th IEEE Consumer Communications and Networking Conference (CCNC)*. An efficient design of security accelerator for IEEE 802.15.4 wireless senor networks (Las Vegas, Nevada, USA, 2010), pp. 1–5
23. R Garcia, A Gordon-Ross, A George, in *17th IEEE Symposium on Field Programmable Custom Computing Machines*. Exploiting partially reconfigurable FPGAs for situation-based reconfiguration in wireless sensor networks (Napa, California, USA, 2009), pp. 243–246
24. H Liu, N Bergmann, in *2010 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. An FPGA softcore based implementation of a bird call recognition system for sensor networks (Edinburgh, Scotland, UK, 2010), pp. 1–6
25. P Muralidhar, C Rao, in *Fourth International Conference on Wireless Communication and Sensor Networks*. Reconfigurable wireless sensor network node based on Nios core (IIIT, Allahabad, 2008), pp. 67–72
26. R Leon, V Vittal, G Manimaran, Application of sensor network for secure electric energy infrastructure. IEEE Trans. Power Delivery. **22**(2), 1021–1028 (2007)
27. Y Yang, D Divan, R Harley, T Habetler, in *IEEE Power Engineering Society General Meeting*. Power line sensornet - a new concept for power grid monitoring (Montreal, Quebec, Canada, 2006), p. 8
28. S Gumbo, H Muyingi, in *2008 Third International Conference on Broadband Communications, Information Technology Biomedical Applications*. Performance investigation of wireless sensor network for long distance overhead power lines; Mica2 motes, a case study, Pretoria, Gauteng, South Africa, 2008), pp. 443–450
29. I Stoianov, L Nachman, S Madden, T Tokmouline, in *Proceedings of the 6th international conference on Information processing in sensor networks*. PIPENET: a wireless sensor network for pipeline monitoring (ACM, New York, 2007), pp. 264–273
30. H Wang, Y Hou, Y Xin, in *International Conference on Measuring Technology and Mechatronics Automation*. Plant running management based on wireless sensor network. vol. 1 (Zhangjiajie, Hunan, China, 2009), pp. 138–141
31. X Fang, S Misra, G Xue, D Yang, Smart grid—the new and improved power grid: a survey. IEEE Commun. Surv. Tutor. **14**(4), 944–980 (2012)
32. O Berder, O Sentieys, in *Workshop on Ultra-Low Power Sensor Networks (WUPS), co-located with Int. Conf. on Architecture of Computing Systems (ARCS 2010)*. PowWow: power optimized hardware/software framework for wireless motes (Hannover, Allemagne, 2010), pp. 229–233
33. X Zhang, H Heys, C Li, in *2011 6th International ICST Conference on Communications and Networking in China (CHINACOM)*. FPGA implementation of two involutional block ciphers targeted to wireless sensor networks (Harbin, China, 2011), pp. 232–236
34. Y Sun, L Li, H Luo, in *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*. Design of FPGA-based multimedia node for WSN (Wuhan, China, 2011), pp. 1–5
35. G Chalivendra, R Srinivasan, N Murthy, in *International Conference on Electronic Design*. FPGA based re-configurable wireless sensor network protocol (Penang, Malaysia, 2008), pp. 1–4
36. J Wei, L Wang, F Wu, Y Chen, L Ju, in *IEEE Youth Conference on Information, Computing and Telecommunication*. Design and implementation of wireless sensor node based on open core (Beijing, China, 2009), pp. 102–105
37. S Lu, X Huang, L Cui, Z Zhao, D Li, Design and implementation of an ASIC-based sensor device for WSN applications. IEEE Trans. Consumer Electron. **55**(4), 1959–1967 (2009)
38. P Crowley, *Network Processor Design: Issues and Practices*. (Academic Press Inc., Orlando, 2002)
39. PC Lekkas, P Lekkas, *Network Processors: Architectures, Protocols and Platforms*. (McGraw-Hill Inc., New York, 2003)
40. LAN/MAN Standards Committee of the IEEE Computer Society: IEEE Standard 802.15.4, *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs) (2003). http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf*
41. Texas Instruments: CC2420 Data Sheet. v1.4 (2006). http://www.ti.com/lprf
42. Celoxica: RC10 Manual. http://www.celoxica.com
43. Xilinx. http://www.xilinx.com
44. Mentor Graphics Inc. http://www.model.com
45. PR Casey, KE Tepe, N Kar, Design and implementation of a testbed for IEEE 802.15.4 (Zigbee) performance measurements. EURASIP J. Wirel. Commun. Netw. **2010**, 23:1–23:2 (2010)
46. Atmel. http://www.atmel.com
47. I Kuon, J Rose, Measuring the gap between FPGAs and ASICs. IEEE Trans. Computer-Aided Design Integrat. Circ. Syst. **26**(2), 203–215 (2007)

doi:10.1186/1687-3963-2013-5
Cite this article as: Liao *et al.*: **FPGA based wireless sensor node with customizable event-driven architecture.** *EURASIP Journal on Embedded Systems* 2013 **2013**:5.