Microprocessors and Microsystems 73 (2020) 102966

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications

Wei-Pau Kiat[a], Kai-Ming Mok[a,1], Wai-Kong Lee[a,1,*], Hock-Guan Goh[a,1], Ramachandra Achar[b,1]

[a] Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman, Kampar, Perak, Malaysia
[b] Department of Electronics, Carleton University, Ottawa, K1S 5B6, Canada

ARTICLE INFO

Article history:
Received 26 October 2018
Revised 27 August 2019
Accepted 21 December 2019
Available online 23 December 2019

Keywords:
Field Programmable Gate Array
Internet-of-Things
Low power
Partial reconfiguration
Multi-cycle execution (ME)
Pipeline execution (PE)
RISC
Sensor node (SN)

ABSTRACT

Low power consumption and high computational performance are two important processor design goals for IoT applications. Achieving both design goals in one processor architecture is challenging due to their conflicting requirements. This paper introduces a reconfigurable micro-architectural level technique that allows a Reduced Instruction Set Computing (RISC) processor to support IoT applications with different performance and energy trade-off requirements. The processor can be reconfigured into either multi-cycle execution mode (low computational speed with low dynamic power consumption) or pipeline execution mode (high computational speed at the expense of high dynamic power), based on dynamic workload characteristics in IoT applications. Switching between modes is accomplished by exploiting the partial reconfiguration (PR) feature offered by the recent advancements in modern FPGAs. A RISC processor was designed based on the proposed micro-architectural level technique and implemented on FPGA as IoT sensor node. Experimental results demonstrate that the proposed technique with reconfigurable micro-architecture is able to significantly reduce the dynamic energy consumption, compared to conventional multi-cycle and pipeline only micro-architectures, while allowing better performance-energy trade-off in IoT applications.

1. Introduction

Internet-of-Things (IoT) technology has been advancing by leaps and bounds in the past decade, with various useful applications rapidly emerging to improve our daily life [1,2]. One of the key technologies supporting the emergence of IoT is the Wireless Sensor Network (WSN), wherein the sensor node (SN) is traditionally designed based on low-power ASIC microcontroller. Typical deployment of IoT application (e.g., smart agriculture) requires tens to hundreds of SNs, which are distributed across a designated region. Since these SNs usually depend on battery power, designing them with low power consumption is becoming very crucial. Within each SN, the SN processor is responsible for executing the firmware tasks, such as, sensor data collection and processing, wireless data transmission and sleep-wake cycle control (power management), etc.

Although ASIC microcontrollers are widely used to build SNs due to their smaller size and comparatively lower power consumption, there are many situations where they may not be the best design choice for IoT applications. Some of these special situations are outlined below, which motivate us to design an FPGA based reconfigurable IoT SN.

1. In practice, the workload characteristics of IoT applications vary dynamically [3]. The processing of data occurs at different rates for different firmware tasks. For example, reading data from temperature sensors is trivial and it can be processed at a lower rate; in contrast, encryption of data is computationally demanding, which needs to be executed at a higher rate to ensure timely response to the gateway/server request. To overcome such dynamic requirements, current common practice is to employ an ASIC microcontroller that matches the highest performance requirement of a particular IoT application, which often ends up sacrificing an opportunity to further reducing the energy consumption (e.g. through dynamically adjusting the processing rate based on actual workload requirements).
2. Each IoT application has specific requirements on the computational and I/O peripheral interface capability. For example,

* Corresponding author.
E-mail addresses: weipau0525@1utar.my (W.-P. Kiat), mokkm@utar.edu.my (K.-M. Mok), wklee@utar.edu.my (W.-K. Lee), gohhg@utar.edu.my (H.-G. Goh), achar@doe.carleton.ca (R. Achar).
[1] Member, IEEE.

https://doi.org/10.1016/j.micpro.2019.102966
0141-9331/

SNs that capture video signal [4] or accelerometer [5] require high computational capability to keep up with the processing rate, while measurements from slow changing parameters (e.g. temperature, humidity and atmospheric pressure) do not need high computational capability. An environmental monitoring station [6] collecting multiple data, such as gas concentration, soil humidity, lighting conditions, wind speed sensors, etc. may require multiple I/Os. On the other hand, biomedical device [7] may require only a few I/Os. Companies that employ IoT technologies for such various applications may need to use different ASIC microcontrollers (together with their specific tool-chain) for each specific application. This incurs heavy overhead during development (e.g., learning new toolchain and develop new firmware), production (e.g., maintaining different production lines for the microcontrollers used) and human resource management (e.g, software/hardware bugs may be specific to certain microcontroller/toolchain, thus requiring engineers with experience in that particular microcontroller).

3. Another limitation attributed to an ASIC microcontroller is the associated inflexibility. It is IP protected, neither configurable nor modifiable, making it difficult for the user to further optimize the firmware and hardware at the micro-architecture level (software-hardware co-design) to achieve targeted program execution time or low power. Moreover, reliance on vendors may lead to the risk of obsolete support (hardware or toolchain), resulting in additional overhead for re-starting a new development cycle.
4. Also, the choice of board level communication protocols (e.g., UART, SPI and I^2C) may affect the energy efficiency of SNs as pointed out by Mikhaylov and Tervonen [8]. Hence, flexibility to select suitable communication protocols is important, but not supported by ASIC microcontroller.

The above circumstances warrant the customizability of a processor so as to design SNs that are optimized to cater to a wide variety of IoT application requirements. This leads us to consider developing an IoT processor based on FPGA with reconfigurable hardware design for rapid customization. It is to be noted that an FPGA device is not as cost effective as its ASIC counterpart in terms of large volume production. Also, the ASIC counterpart consumes lesser energy compared to FPGA [20]. However, FPGA based systems can be advantageous for small to medium scale products that require rapid prototyping and reconfiguration to handle the variety of circumstances that are outlined above.

Energy efficiency is one of the main concern in FPGA based IoT SN. Existing FPGA-based techniques to save power include Dynamic Voltage and Frequency Scaling (DVFS) [9], power gating [10] and clock gating [11]. Despite the effectiveness of these techniques, all prior works utilizes only single type of core with fixed micro-architecture, which have untapped opportunity for further energy reduction. In particular, a fixed micro-architecture with high computational capability can waste battery energy unnecessarily when processing tasks with low computational speed requirement. On the other hand, using a micro-architecture with low computational capability can save energy, but may not be able to handle the tasks with high computational speed requirements on time. When dealing with IoT applications that have widely varying dynamic workload characteristics, the disadvantage of using fixed micro-architecture is obvious: we lost the opportunity to further reduce dynamic energy consumption.

This paper presents a novel and generic design of an FPGA based SN with optimized processor micro-architecture for reducing the dynamic power consumption by SN. The proposed FPGA based reconfigurable processor is capable of switching between multi-cycle execution micro-architecture (ME; for computationally less intensive tasks that require low dynamic power) and pipeline execution micro-architecture (PE; for computationally intensive tasks that require more dynamic power) to optimize the energy usage with flexible performance-energy trade-off.

2. Background

In this section, a brief review of the prior work as relevant to the proposed work on reconfigurable FPGA based design for reduction of dynamic power consumption is given.

A customized FPGA implementation for vibration monitoring system with additional AES and 1024-bit FFT block was presented by Bengherbia et al. [12]. More examples of FPGA based SN can be found in the survey conducted by Piedra et al. [13]. Compared to an ASIC microcontroller, FPGA has shorter development time and highly customizable system resources, including the number of I/Os and communication protocols (SPI/UART/I^2C channels). This makes FPGAs attractive for implementing IoT SNs, in particular, customizing the system resources to meet a wide range of performance and functional requirements, which is also the main reason for FPGA adoption in prior arts.

Conventional power reduction technique using DVFS on Xilinx Virtex 5 was implemented by Nunez-Yanez [9]. This adaptive voltage scaling technique for reducing both static and dynamic power consumption, reported the following results: a minimum voltage (0.62V) after scaling with a maximum power reduction of 86% for maximum working frequency of 40 MHz. Hosseinabady and Nunez-Yanez [10] showed that power gating is able to reduce the power consumption up to 96%. In their implementation, a hard-core processor (Cortex A9) was used to power-off the FPGA chip when it is idle, with a minimum timing overhead (the time for turn-off, turn-on and reconfigure the programmable logic) of 42.58 ms. Major drawback in this technique is the requirement of an extra hard-core processor to serve as the watchdog which itself consumed power.

Multi-cycle [15] and pipeline [16] execution are two different approaches in a processor design. In the past, both approaches have been developed and studied as independent separate cores. Natheswaran and Athisha [15] show that multi-cycle execution requires lesser design area and dynamic power as compared to the pipeline execution for a processor designed for SN applications. On the other hand, pipeline micro-architecture can be further optimized through static pipelining technique to reduce the power consumption, which was presented by Finlayson et al. [17]. Jain et al. [14] proposed an adaptable pipeline micro-architecture to allow micro-architecture-driven voltage scaling to reduce energy consumption. The micro-architecture was adapted to the most energy efficient throughput target or supply voltage requirement to achieve optimal power control. However, the authors demonstrated this technique to a specific radix-4 FFT processor; the applicability of this technique to general purpose processors is still unknown.

To address the above issues, in this paper, a generic method is proposed to reduce dynamic energy consumption by flexibly switching between multi-cycle and pipeline execution, based on the dynamic workload characteristics. Detailed development of the proposed method is discussed in the next section.

3. Development of the proposed FPGA based IoT sensor node

In this section, details of the proposed framework for reducing dynamic energy consumption by appropriately switching between ME and PE modes, is discussed. For this purpose, the partial reconfiguration (PR) scheme that is offered in modern FPGA is exploited. With the appropriate use of this feature, execution can be switched between ME or PE mode, in the same core. For example, when executing firmware tasks with low computational requirement, the processor can be configured to multi-cycle execution to

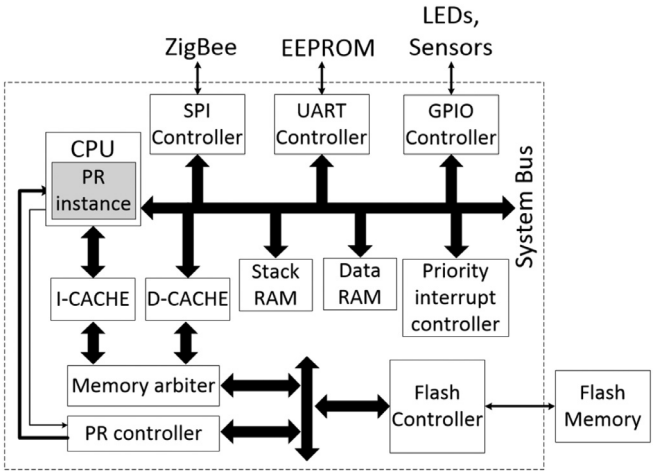

Fig. 1. High Level View of FPGA Based Sensor Node.

Table 1
Supported instruction set.

Arithmetic	Logical	Data transfer	Program control	System
add	*and*	*mfhi*	*jr*	*syscall*
addu	*or*	*mflo*	*jalr*	*eret*
sub	*xor*	*mtc0*	*beq*	***tma****
subu	*nor*	*mfc0*	*bne*	
mult	*andi*	*lui*	*blez*	
multu	*ori*	*lw*	*bgtz*	
sll	*xori*	*lwl*	*j*	
srl		*lwr*	*jal*	
sra		*lh*		
slt		*lhu*		
sltu		*lb*		
addi		*lbu*		
addiu		*sw*		
slti		*swl*		
sltiu		*swr*		
		sh		
		sb		

* ***tma*** is a newly created instruction used for PR in our processor.

conserve dynamic energy. In contrast, the processor can be configured to pipeline execution to ensure that the complicated firmware tasks (e.g., encryption) are completed quickly. The reconfiguration, occurring between the tasks, is determined by the computational speed requirement of a task and controlled from the user program using a newly created assembly instruction, Toggle Microarchitecture (*tma*).

3.1. Description of the system configuration

A block diagram of the proposed FPGA based sensor node is given in Fig. 1. It consists of communication module (ZigBee), sensors, flash memory, processor and its peripherals. Sensor data can be sampled through ADC (analog sensor) or GPIO (digital sensor) by either polling or interrupt method, then transmitted wirelessly through ZigBee module (via SPI communication). The incoming data (run-time or dynamic data) is stored in the Data RAM.

For validation purposes, a 5-stage 32-bit MIPS Instruction Set Architecture (ISA) compatible processor was developed. It supports up to 50 instructions, covering arithmetic, logical, data transfer, program control, and system instruction classes, which are detailed in Table 1.

Referring to Fig. 1, Stack and Data RAMs are implemented using FPGA block RAM (BRAM). The external flash memory is used to store the user program code, static data and the FPGAâs full and partial configuration bitstreams. The *PR instance* is an abstraction of the targeted hardware for PR. The firmware is programmed into the flash memory through the flash programming tool. When instruction cache (icache) miss occurs, the processor will stall and fetch a block of instruction words (8 words) from the flash memory into the icache, and then continues its execution. The GPIO, SPI and UART I/O modules are connected to the processor via Wishbone bus interface. The *load* and *store* instructions are used to check and configure the I/O module (UART, SPI, GPIO) registers, respectively.

3.2. Proposed multi-cycle execution based micro-architecture (ME)

The proposed multi-cycle micro-architecture is presented in Fig. 2. The components shaded in gray are grouped into the PR region. Note that in Fig. 2, the control signals, *Ctrl*, from the Control FSM are the control signals to each stage of the datapath respectively. The multi-cycle micro-architecture executes an instruction in several clock cycles, which varies between 3–5 clock cycles

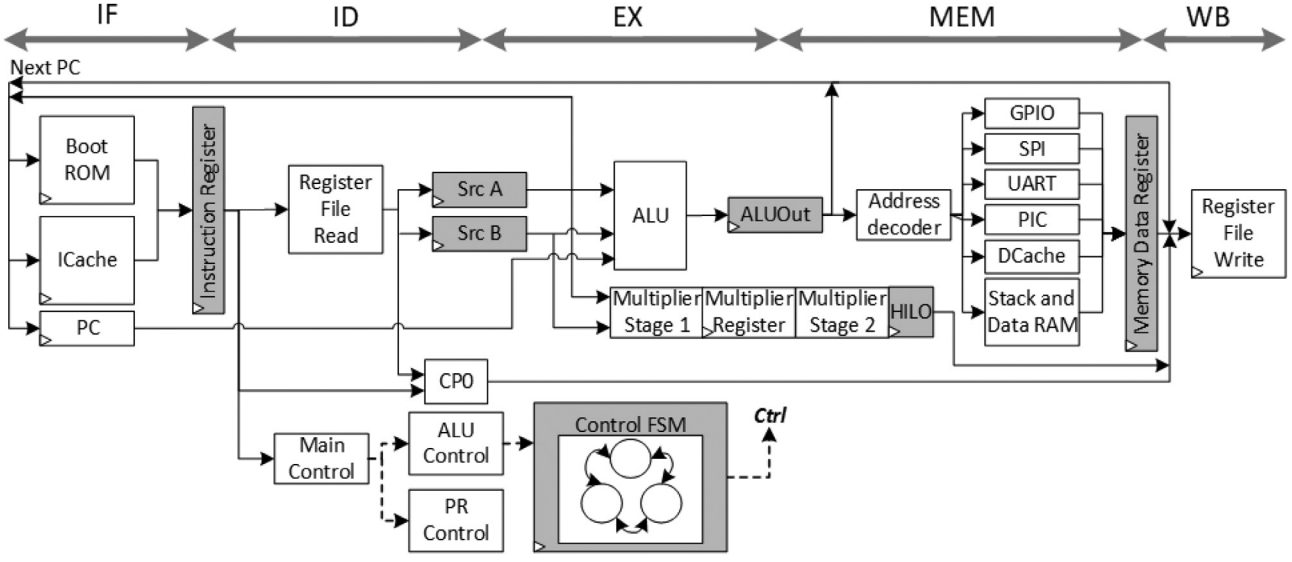

Fig. 2. Proposed multi-cycle micro-architecture (functional view with components). The gray color blocks are grouped into the PR region.

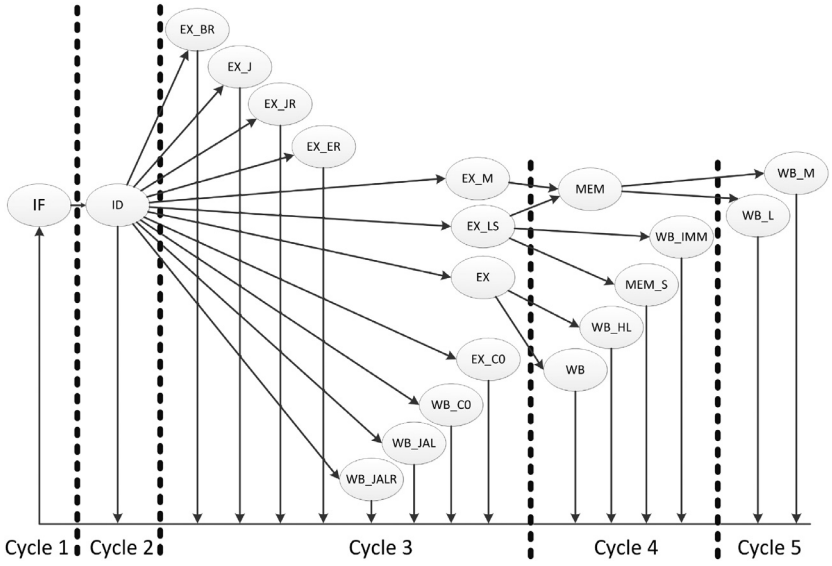

Fig. 3. Proposed FSM state diagram for Multi-cycle control-path unit.

(excluding special instruction, *tma*). Each instruction in the multi-cycle execution must complete it's execution before the subsequent instruction starts, so as not to allow any overlapping execution. Due to this reason, no extra circuitry would be needed for data hazard handling.

In the proposed multi-cycle micro-architecture, only one stage is active at every clock cycle. This characteristic allows high reusability of the CPU hardware components and lesser signals transition, which reduces the design area and dynamic power consumption. In this paper, the multi-cycle micro-architecture developed mainly consists of five hardware stages (IF, ID, EX, MEM and WB). To handle the varying instruction cycle (from 3 to 5 clock cycles), a Moore model Finite State Machine (FSM) control path unit is developed, and is shown in Fig. 3.

Table 2 describes the function of each state in this FSM. For example, an *add* instruction requires 4 instruction cycles, i.e., IF, ID, EX and WB. At the first clock cycle (IF stage), the *add* instruction is fetched from the I-CACHE. At the second clock cycle (ID stage), the add instruction is decoded and register operands are fetched from the Register File. ALU performs add operation on the operands fetched at the third cycle (EX stage). The results of the ALU are written back to the Register File at the next clock cycle (MEM stage).

Table 2
Definition of the States in Multi-cycle control-path unit FSM.

State Name	Definition
IF	Instruction fetch from instruction memory
ID	Instruction Decode and Register File Read
EX_BR	Determine branch taken or untaken. Copy branch target address calculated to the PC register if branch taken.
EX_J	*j* instruction detected. Copy jump address to PC register
EX_JR	*jr* instruction detected. Copy *$rs* register value to PC register
EX_ER	*eret* instruction detected. Copy the exception return address, *$epc* to PC register
EX_C0	mtc0 instruction detected. Move a data from Register File to CP0 register
WB_C0	*mfc0* instruction detected. Copy a data from CP0 register to Register File
WB_JAL	*jal* instruction detected. Copy PC register to *$ra* register in the Register File and copy jump address to PC register
WB_JALR	*jal* instruction detected. Copy PC register to *$ra* register in the Register File and copy *$rs* register data to PC register
EX_M	*jal* instruction detected. Activate multiplier
EX_LS	I-type instruction detected. Further decode whether load, store or immediate instruction is issued
EX	R-type instruction detected. Activate ALU.
MEM	Load data from data memory (D-CACHE, RAM or I/Os register) or perform as dummy cycle for multiplication operation
WB_IMM	Write back immediate instruction result to Register File
MEM_S	Store data to data memory (D-CACHE, RAM or I/Os register)
WB_HL	Write back HI or LO register to Register File
WB	Write back R-type instruction result to Register File
WB_M	Reserved for write back multiplication result to HI or LO register
WB_L	Write back data memory (D-CACHE, RAM or I/Os register) data to Register File

3.3. Proposed pipeline execution based micro-architecture (PE)

Fig. 4 shows the pipeline micro-architecture of our IoT processor. The components shaded in gray are grouped into the PR region. Note that, in Fig. 4, the control signals, *Ctrl*, from the Main Control, ALU control, Forwarding control and Interlock control are the control signals to each stage of the datapath respectively. At the Instruction Fetch (IF) stage, an instruction is fetched from the I-CACHE and registered at the IF/ID pipeline registers. If the cache miss occurs during the IF stage, the I-CACHE will send a signal to stall the processor execution. The execution continues when the respective instruction is successfully copied from the flash memory to the I-CACHE.

At the Instruction Decode (ID) stage, the instruction previously registered in IF/ID pipeline registers is decoded by the Main Control block and the Arithmetic Logic Control block. Output signals from both hardware components are registered to the ID/EX pipeline register and also passed to the remaining hardware components in the ID stage (i.e., Register File, Forwarding Control block, CP0 block, Branch Predictor block and Interlock block). At the same time, IF stage continues to fetch the consecutive instruction from the I-CACHE.

At the Execution (EX) stage, ALU block covers all operations of the supported instructions except multiplication. Multiplier block starts the multiplication operation at the EX stage and requires two clock cycles (EX and MEM stage) to perform a multiplication operation on two 32-bit operands.

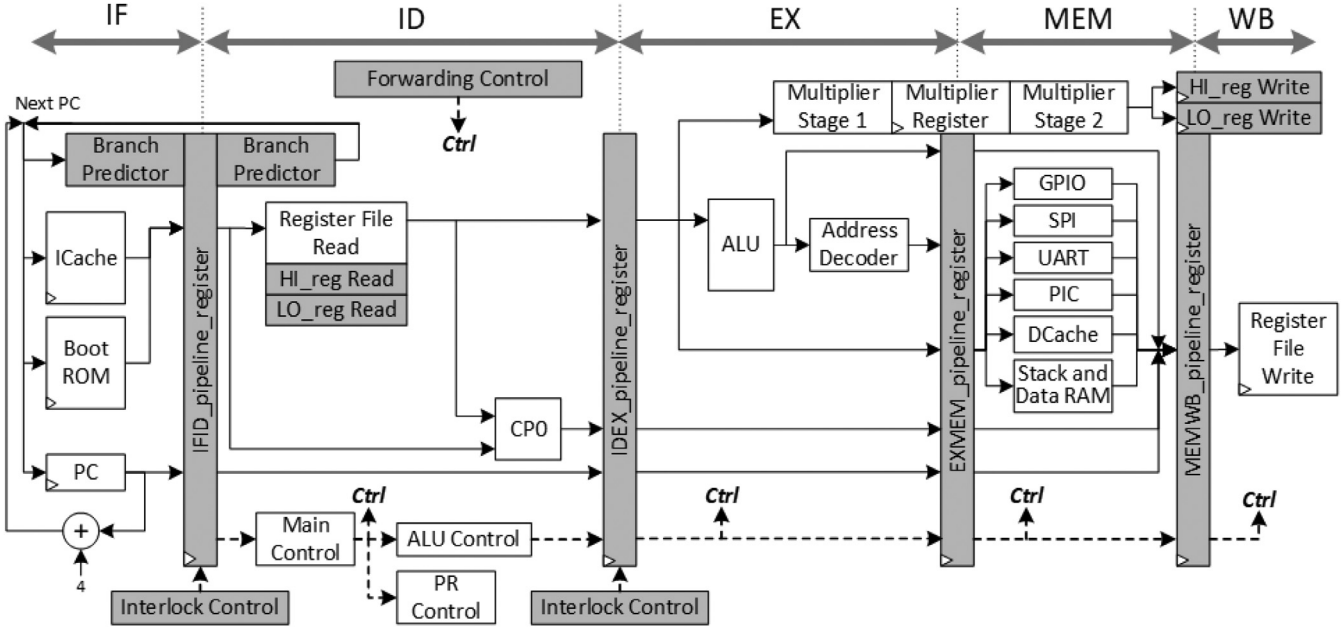

Fig. 4. Proposed Pipeline micro-architecture (functional view with components). The gray color blocks are grouped into the PR region.

At the Memory (MEM) stage, only load and store instructions are permitted to perform the operation, while other instructions would bypass this stage. Load or store is used to access the memory components at the MEM stage (i.e. D-CACHE, RAM and I/Os registers). At the Write Back (WB) stage, the result is updated to the Register File at the positive clock edge.

A Branch Predictor block is used instead of the branch delay slots. The block is meant to resolve the control hazard problem [18] in PE. Thus, enhances the program execution time. However, in ME mode, the block is removed since control hazard is non-existence in ME. If branch delay (NOP instruction) slot is used instead of a Branch Predictor, every conditional and unconditional branch/jump instructions will need a branch delay slot to resolve the control hazard in PE. This has a negative impact whereby the program code will grow larger. On one hand, it takes up more memory space. On the other, the ME has to bear the consequence of executing a larger program code than necessary.

Table 3 gives the specifications for both the multi-cycle and pipeline micro-architectures.

3.4. Proposed reconfigurable micro-architecture

The proposed reconfigurable approach categorizes the hardware parts of the processor core (CPU) into two groups: static and reconfigurable regions. The static region consists of the hardware portion that does not change regardless of the pipeline or the multi-cycle execution. The reconfigurable region (referred to as the Partial Reconfiguration instance or *PR instance*) consists of hardware portion that can be reconfigured to either the pipeline or multi-cycle structure depending on dynamic workloads. As can be seen from the shaded region in Fig. 5, the *PR instance* consists of selected reconfigurable components that are solely from the Central Processing Unit (CPU). *PR instance* is controlled by the PR Controller, details of which are given below.

PR Controller: The PR activity is triggered from the user program using the *tma* instruction. The currently configured processor will pass the control to the proposed PR controller (corresponds to the PR control block in Figs. 2 and 4).

Referring to Fig. 6, the PR controller is responsible for the following tasks:

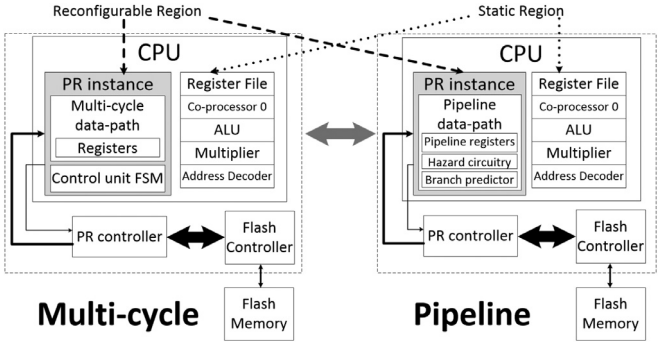

Fig. 5. Selected reconfigurable components from CPU.

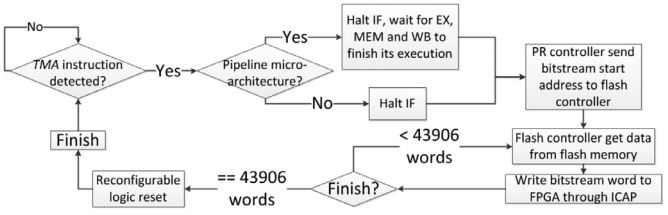

Fig. 6. Proposed PR process flow.

1. Activate PR process when a *tma* instruction is detected
2. Hold Program Counter (PC) register value when the PR is in progress
3. Identify which micro-architecture (pipeline or multi-cycle) should be reconfigured to:
   a) If the current micro-architecture is in pipeline micro-architecture, the execution of the instruction following *tma* instruction, which is already in IF stage, will be halt by the PR controller. At the same time, the instructions in EX, MEM and WB stages will be executed until completion. And then reconfigure the processor to multi-cycle micro-architecture.
   b) If the current micro-architecture is in multi-cycle micro-architecture, in the next clock cycle when the *tma* instruction reaches the EX stage, the PR controller will hold

Table 3
Reconfigurable and static zones of the proposed micro-architecture.

		Multi-cycle	Pipeline
Cycle per instruction		3 - 5	1
Branch predictor		-	64 entries 4 ways associative
Composition of reconfigurable region (*PR instance*)		Data-path unit, Control unit Finite State Machine	Data-path unit, branch predictor, pipeline registers, hazard circuitry
Common features	Memory	4kBytes boot ROM, 128kBytes user access flash, 8kBytes RAM (Data & Stack), 1kBytes i-cache, 128Bytes d-cache, 512Bytes Memory Mapped I/O Register	
	Communication interface	UART, SPI, 32 GPIO pins	
	PR controller resources	51 LUTs, 40 FFs	
Bitstream start address		0x00A0_0000	0x00A8_0000
Bitstream size		1,404,992 bits / 43,906 words	
FPGA board		Nexys 4 DDR (XC7A100T)	
FPGA Resources (Overall)	LUT (63400)	11.71% (7421)	13.04% (8266)
	LUTRAM (19000)	0.67% (127)	1.66% (315)
	FF (126800)	0.41% (5223)	0.45% (5643)
	BRAM (135)	2.59% (3.50)	2.59% (3.50)
	IO (210)	21.90% (46)	21.90% (46)
	BUFG (32)	3.13% (1)	3.13% (1)

the IF stage to prevent the execution of a new instruction. And then reconfigure the processor to pipeline micro-architecture.
4. Fetch the partial bitstream from the flash memory
5. Write the partial bitstream to FPGA through Internal Configuration Access Port (ICAP).
6. Perform reset on PR region

As shown in Fig. 6, the PR controller obtains the partial bitstream which was earlier stored in the flash memory (which is placed on the development board). Next, each 32-bit word of bitstream is written to the FPGA through ICAP. The process finishes when the last bitstream word is written into the FPGA, following a soft reset on the reconfiguration region as well as to fetching the instruction following the *tma* instruction in the user program code.

The PR bitstream size is determined by the area allocated by the designer for reconfiguration region (PR instance). Large reconfiguration area results in large PR bitstream size, which in turn incurs longer PR time.

4. Experimental results and discussions

In this section, the details of the experimental setup, collection of data and validating results are presented.

4.1. Experimental setup

The proposed FPGA based SN is implemented on the Nexys 4 DDR board with Xilinx Artix-7 XC7A100T FPGA, 4860 Kbits BRAM, on-chip ADC and 16 MB serial flash memory. The proposed reconfigurable processor is synthesized on the FPGA chip for measurement on its energy consumption and timing performance. To carry out the experiment, a simple firmware was written based on typical tasks for IoT sensor node, details of which are given below:

1. *Data collection*: Get *N* bytes of data from sensor.
2. *Data processing*: Encrypt *N* bytes of data received using AES-128.
3. *Data transmission*: Send the encrypted data through SPI.

The *N* bytes data is first collected from a sensor, where *N* ranges from 128 to 1024. This is to evaluate the dynamic energy reduction of the proposed reconfigurable processor under different data sizes. Next, the *N* bytes data is being encrypted with AES-128 algorithm, and finally transmitted out via SPI (which is connected to a wireless ZigBee module). These three tasks are executed repeatedly until the energy source (battery) drains. In practical IoT applications, the size of *N* can be larger than 1024 bytes, so that the sensor nodes do not have to transfer wireless data frequently, which leads to lower energy consumption. In fact, for sensor network that utilizes multi-hop communication [21], the sensor nodes take turn to transmit data. In such cases, the sensor nodes usually collect large amount of sensor data and then wait for their turn to transmit; our experiments follow exactly this setting. On the other hand, the data processing can also involve other operations like averaging and compression, etc., but we only implement the encryption as proof of concept in this paper.

The data collection and processing (encryption) involves large number of computational operations (involving arithmetic and logic operations in AES-128), while data transmission only involves some memory transfer and I/O communication (SPI port) operations.

We further group these three firmware tasks into two categories:

1. *Computationally intensive tasks:* data collection and data processing.
2. *Computationally less intensive tasks:* data transmission.

For computationally intensive tasks, the pipeline micro-architecture can be used to complete the tasks faster, thus reducing the dynamic energy consumption. On the other hand, for computationally less intensive tasks (which do not involve many arithmetic and logical operations) multi-cycle micro-architecture can be used due to its low dynamic power consumption. To optimize the energy efficiently for IoT sensor nodes, we configure the micro-architecture to pipeline mode when executing computationally intensive tasks, and reconfigure it to multi-cycle mode for computationally less intensive tasks.

4.2. Experimental results

To validate the proposed technique, experiments were carried out for data size ranges from $N = 128$ to 1024 bytes at 20MHz and 40MHz per IoT task. The descriptions of each experiment is provided as below:

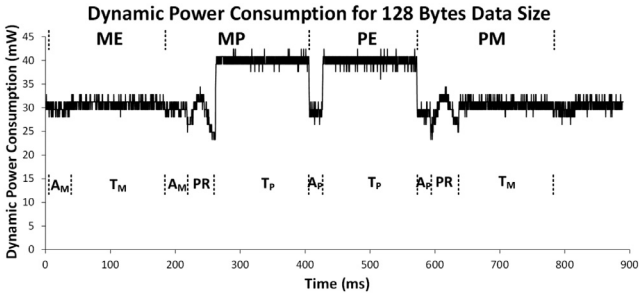

Fig. 7. Dynamic power consumption measurement for 128 Bytes experiment running at 20 MHz.

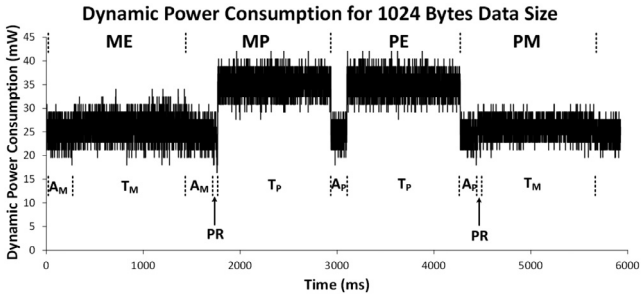

Fig. 8. Dynamic power consumption measurement for 1024 Bytes experiment running at 20 MHz.

a) **PE:** Both computationally intensive as well as less-intensive tasks are executed in the pipeline mode, no PR takes place.
b) **ME:** Both computationally intensive as well as less-intensive tasks are executed in the multi-cycle mode, no PR takes place.
c) **PM:** This represents the proposed reconfigurable mode where computationally intensive tasks are executed in the pipeline mode, followed by PR, then computationally less-intensive tasks are executed in the multi-cycle mode.
d) **MP:** This represents the opposite configuration of **PM**, where this configuration unnecessarily reduces the energy efficiency.

ME and **PE** represents the common practice in designing IoT sensor nodes by using single micro-architecture for design goals of either low power (**ME**) or for fast performance (**PE**), respectively. **PM** represents the proposed technique to reconfigure the micro-architecture based on different dynamic workload characteristics to further optimize the energy efficiency while achieving acceptable performance. For each experiment, current consumption of the FPGA device is measured for calculating the dynamic energy consumption.

The dynamic power consumption for different micro-architectural configuration using data sizes of N = 128, 256, 512 and 1024 bytes at 20 MHz and 40 MHz were measured. Two samples, N = 128 and N = 1024 running at 20 MHz are shown in Figs. 7 and 8. Since the focus of this paper is to reduce the dynamic energy consumption, measurement related to the dynamic parts were taken. Static energy consumption by the CPU and ZigBee modules (which remain constant independent of the modes) are not considered.

- A_M: Data collection & processing using multi-cycle microarchitecture
- T_M: Data transmission using multi-cycle microarchitecture
- A_P: Data collection & processing using pipeline microarchitecture
- T_P: Data transmission using pipeline microarchitecture

Fig. 9 and 10 show the program execution time and energy consumption for each micro-architecture configuration, while Figs. 11 and 12 show the energy-delay product, which is a common way to demonstrate the energy and timing performance trade-off [22].

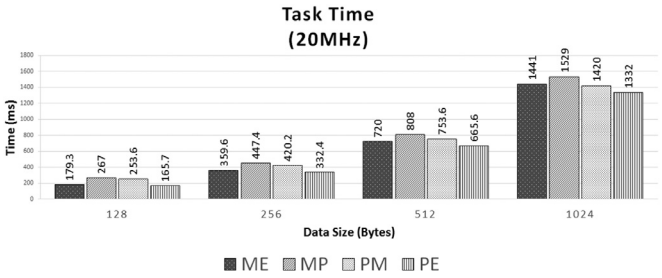

Fig. 9. Program execution time for different micro-architectural configurations with varying data sizes.

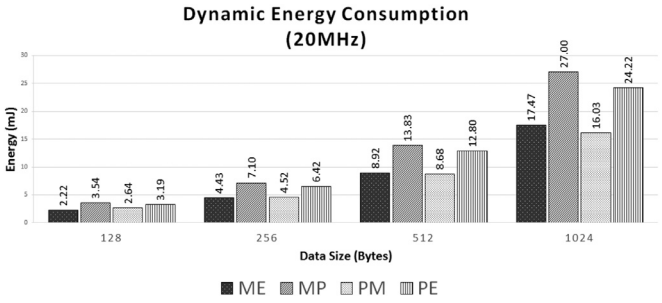

Fig. 10. Dynamic energy consumption for different micro-architectural configurations with varying data sizes.

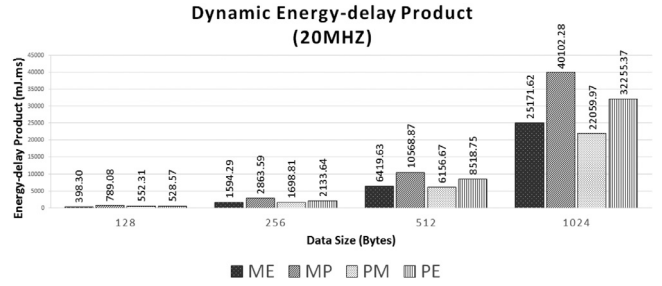

Fig. 11. Dynamic energy-delay product for different micro-architectural configurations with varying data sizes at 20 MHz.

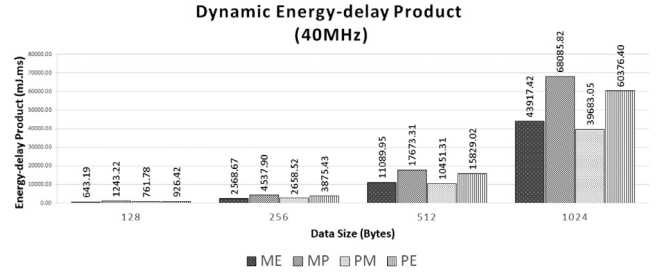

Fig. 12. Dynamic energy-delay product for different micro-architectural configurations with varying data sizes at 40 MHz.

From Fig. 9, it is evident that **PE** requires the least time to complete all tasks, while **ME** takes longer time to execute. For all cases where $N > 512$, the proposed reconfigurable scheme **PM** is faster than **ME** but slower than **PE**. Considering the case for 1024 bytes ($N = 1024$), **PM** is 1.46% faster than **ME** while it is 6.61% slower than **PE** per IoT task.

However, as can be noted in Fig. 10, although **PE** provides superior timing performance, it consumes significantly more energy to complete all the tasks compared to **ME**. This is because, fast performance and low energy consumption are two contradicting design

Table 4
Static energy consumption versus Dynamic energy consumption in 20 MHz and 40 MHz system frequency.

	static energy		dynamic energy		total energy
	(mJ)	%	(mJ)	%	(mJ)
20 MHz	72.50	81.89	16.03	18.11	88.53
40 MHz	67.07	68.27	31.17	31.73	98.24

Remark: Experiment conducted using **PM** with 1024 bytes.

Table 5
FPGA resources used by PR Controller versus Multi-cycle and Pipeline micro-architectures.

FPGA Resources	PR controller	Multi-cycle	Pipeline
LUT	51	7421	8266
LUTRAM	0	127	315
Register	40	5223	5643
BRAM	0	3.50	3.50
IO	0	46	46
BUFG	0	1	1

goals that are hard to achieve utilizing a single micro-architecture (**ME** or **PE**). On the other hand, proposed **PM** is the most energy efficient option among all the three micro-architectural configurations while providing superior timing performance, except for the case of $N = 128$. For example, considering the case for 1024 bytes ($N = 1024$), **PM** is 8.22% and 33.80% more energy efficient compared to **ME** and **PE** respectively per IoT task. Hence, we can conclude that the proposed technique can achieve better energy efficiency when the data size (N) increases. This is because although **ME** is better optimized for "low power" mode; it takes much longer time to complete computationally intensive tasks, and hence ends up consuming more energy as the size N is sufficiently large. In contrast, **PM** executes only computationally intensive tasks using **PE** (to reduce total time spent), then switches to **ME** for computationally less-intensive tasks (to conserve energy). Hence, the PM provides better energy efficiency as N increases. This finding is important as many wireless sensor networks actually employ multi-hop techniques [21] in practical field deployment, wherein the sensor nodes collect large amount of data and take turn (based on the designed protocol) to transmit it. Under such scenario, the data size N can be much larger than 1024 bytes, which highlights the advantages of using the proposed technique.

Considering the trade-off between performance and energy consumption, energy-delay product is obtained for these four micro-architectural configurations at 20 MHz system frequency; the result is shown in Fig. 11. As can be seen, **PM** has the least energy-delay product; it is 12.3% lesser than **ME**. This also implies that **PM** is the most optimized option when taking into account both energy and timing performance. On the contrary, **PE** and **ME** are only optimized for either timing performance or energy efficiency. Similar conclusion can also be drawn when the system frequency increases to 40MHz (refer to Fig. 12).

4.3. Total energy consumption: 20 MHz vs 40 MHz

Our focus in this paper is to reduce the dynamic energy consumption at a reasonable program execution time (IoT task time), through a good trade-off between the fixed and PR CPU micro-architectures. Comparison of the total energy consumption at 20MHz and 40MHz system frequency is shown in Table 4. At lower system frequency (20MHz), the static energy consumption (81.89%) is dominant against dynamic energy consumption (18.11%). However, when the system frequency increased to 40MHz, the static energy becomes less dominant (reduced from 81.89% to 68.27%), while the percentage of the dynamic energy consumption has increased from 18.11% to 31.73%. As the system frequency increases, the static energy component become less dominant. This is due to the fact that dynamic power is proportional to the switching frequency [24], which can be described in Eq. (1):

$$P_{dyn} \propto \alpha C V_{dd}^2 f \tag{1}$$

where α represents the switching activity, C is the capacitance, V_dd is the supply voltage, and f is the operating frequency.

The dynamic energy is dependent on the FPGA technology (the load capacitance, C_L and voltage source, V_{dd}) and mainly on the switching activities. By assuming the use of similar FPGA technology, the dynamic energy is then directly affected by the micro-architectural design. The static energy, on the other hand, depends on the choice of the FPGA technology [23] and the FPGA size (amount of resources provided). Take the FPGA resource usage from Table 3 as an example; the total resource usage for the multi-cycle structure is around 6.11%. If a much smaller FPGA is used, this can significantly reduce the static energy component (shifting the dominance energy component towards the dynamic energy), leaving us to deal with only the dynamic energy component through micro-architectural design. In summary, as the system frequency increases, with our proposed technique couple with a low-power FPGA with the smallest possible size, the total energy consumption can further be reduced.

4.4. Overhead of partial reconfiguration

When PR takes place, it introduces an additional 44ms of overhead (at 20 MHz system clock) for any data length, which should be taken into consideration when developing firmware with time-critical tasks. Besides that, every time the reconfiguration (PR) takes place, it also consumes 1 mJ additional energy. This implies that there is a lower bound condition when applying the proposed technique to any existing computer architecture, as switching between two micro-architecture too frequently may not yield the expected result. In our experimental setup, N has to be at least 256 bytes in dynamic energy-delay product as shown in Fig. 11, in order to achieve satisfactory result in energy-delay product.

The PR timing overhead can be reduced by: 1) increasing the clock frequency up to 100 MHz (based on ICAP requirement); 2) buffering the partial bitstream in an FPGA Block RAM to reduce the bitstream read time; 3) using a DMA controller when copying the partial bitstream during PR. All these techniques are at the expense of more energy consumption and resources [19].

Apart from the above, an additional component, the PR controller, is required which consumed the resources as shown in Table 5. Comparatively to the resources used by the multi-cycle and pipeline structures, the PR controller only used a small fraction of the resources. For LUT usage, only 0.69% and 0.62% as compare to the multi-cycle and pipeline LUT resources respectively. For register usage, only 0.76% and 0.71% as compare to the multi-cycle and pipeline register resources respectively.

4.5. Comparison with existing work

RISPP [25] and i-core [26] are two related work that explore the reconfigurable instruction set in processor. Similar to our work, RISPP [25] and i-core [26] also uses the partial reconfiguration feature of the reconfigurable fabric. The intention is to reconfigure the accelerators in a sequence specific to an application, use together with either a hardcore CPU (e.g. ARM on eFPGA, softcore on FPGA), to speed up the computation through the increase of instruction parallelism execution. The reconfiguration of the accelerators depends on the extraction of the Special Instructions

Table 6
Comparison with existing Sensor Node.

Sensor Node	Energy Consumption	Hardware Platform
ATmega128 [28]	57 µJ	Microcontroller
ATmega128 [28]	0.456 mJ (scaled)	Microcontroller
This work	89.53 mJ	Artix-7 FPGA (XC7A100T)
This work	11.19 mJ (scaled)	Artix-7 FPGA (XC7A12T)

(SI). In particular, RISPP proposed the concept of "instruction rotation" to optimize the resource sharing in run-time to achieve through run-time adaptation. The focus of these work are on achieving good performance while using less hardware resources (through pre-arranged reconfiguration of the accelerators) as compare to Application Specific Instruction Processor (ASIP). The hard core CPU part is just another embedded processor or microcontroller/microprocessor which implies fixed CPU micro-architecture.

Different from these two prior work, our focus is on reconfiguration of the micro-architecture in a processor with adaptation to the IoT task. This allows us to further reduce the energy consumption without sacrificing the speed performance, eventually achieving good energy-delay product compared to using only fixed micro-architecture.

Comparison with existing off-the-shelf sensor nodes are presented in Table 6. MicaZ [27] is a commonly used sensor node for IoT application, which is designed based on ATmega128 microcontroller. From the measurement conducted by Panait and Dragomir [28], the energy consumption of ATmega128 is 57 µJ when it is encrypting 128 bytes data using software AES. Since our total energy measurement is conducted based on 1024 bytes encryption, we scale the result from Panait and Dragomir [28] by a factor of 8; it consumes 0.456 mJ, which is far less compare to 89.53 mJ in our experiment. Even though we can still reduce the total energy measurement by using a smaller FPGA (XC7A12T), the estimated energy (11.19 mJ, scaled by the amount of logic units) consumption is still 24 $\times$ higher compared to ATmega128. This comparison is not completely fair, since the measurement in ATmega128 does not include data sampling and IO communication. However, it can still give an idea about the energy consumption gap between our FPGA solution and state-of-the-art sensor node. Nevertheless, FPGA based sensor node is still advantageous in some IoT applications, as it is flexibly configurable, in the expense of higher energy consumption. Application specific hardware module (e.g. deep learning [29] and image processing) can also be added into FPGA to support the processor for timely processing, which potentially open up more possibility in IoT applications.

5. Conclusion

In this paper, a novel technique is presented to reduce the dynamic energy consumption of IoT sensor node based on reconfigurable micro-architectural FPGA level design. A proof of concept prototype, wherein with the PR feature offered by FPGAs, a multi-cycle and a 5-stage pipeline micro-architecture are designed to run intermittently in a processor core to achieve better performance-power trade-off. The experimental result for $N = 1024$ shows that the proposed technique can achieve the least energy-delay product, which is 12.36% and 31.61% lesser compared to multi-cycle and pipeline micro-architecture, respectively. For sensor nodes that process larger data sizes ($N > 1024$), the energy-delay product can be further reduced. Other combinations, for example, multi-cycle, 5-stage and 8-stage pipeline execution can also be used for more refined performance-power trade-off.

The proposed technique is not platform specific, it can be applied to other SRAM based FPGA devices that support partial reconfiguration feature. Other useful techniques for lowering power consumption, such as DVFS, clock gating and power gating can be applied on top of the proposed technique to further reduce the total power consumption, which is also the future direction that we aim to pursue.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research work is fully supported by UTAR Research Fund (UTARRF), Malaysia under grant IPSR/RMC/UTARRF/2015-C2/G01. The hardware and software tools are provided by Xilinx under Xilinx University Program (XUP).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.micpro.2019.102966.

References

[1] K.A.A. Rabaiei, S. Harous, Internet of things: applications and challenges, in: 12th International Conference on Innovations in Information Technology (IIT), Al-Ain, 2016.
[2] H. Ma, L. Liu, A. Zhou, D. Zhao, 'On networking of internet of things: explorations and challenges, IEEE Internet Things J. 3 (4) (2016) 441–452.
[3] M. Hempstead, M.J. Lyons, D. Brooks, G.-Y. Wei, Survey of hardware systems for wireless sensor networks, J. Low Power Electron. 4 (1) (2008) 11–20.
[4] N. Cao, S.B. Nasir, S. Sen, A. Raychowdhury, Self-optimizing iot wireless video sensor node with in-situ data analytics and context-driven energy-aware real–time adaptation, IEEE Trans. Circuits Syst. I 64 (9) (2017) 2470–2480.
[5] F. Civerchia, S. Bocchino, C. Salvadori, E. Rossi, L. Maggiani, M. Petracca, Industrial internet of things monitoring solution for advanced predictive maintenance applications, J. Ind. Inf. Integr. 7 (2017) 4–12.
[6] M.T. Lazarescu, Design of a WSN platform for long-term environmental monitoring for iot applications, IEEE J. Emerg. Sel. Top.Circuits Syst. 3 (1) (2013) 45–54.
[7] M. Yasin, T. Tekeste, H. Saleh, B. Mohammad, O. Sinanoglu, M. Ismail, Ultra-low power, secure iot platform for predicting cardiovascular diseases, IEEE Trans. Circuits Syst. I 64 (9) (2017) 2624–2627.
[8] K. Mikhaylov, J. Tervonen, Evaluation of power efficiency for digital serial interfaces of microcontrollers, in: 5th International Conference on New Technologies, Mobility and Security (NTMS), May, 2012.
[9] J.L. Nunez-Yanez, Adaptive voltage scaling with in-situ detectors in commercial FPGAs, IEEE Trans. Comput. 64 (1) (2015) 45–53.
[10] M. Hosseinabady, J.L. Nunez-Yanez, Run-time power gating in hybrid ARM-FPGA devices, in: 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, Sept., 2014.
[11] Y. Zhang, J. Roivainen, A. Mammela, Clock-gating in FPGAs: a novel and comparative evaluation, in: 9th EUROMICRO Conference on Digital System Design (DSD'06), Dubrovnik, Sept., 2006.
[12] Bengherbia, M.O. Zmirli, A. Toubal, A. Guessoum, FPGA-based wireless sensor nodes for vibration monitoring system and fault diagnosis, Measurement 101 (2017) 81–92.
[13] A. de la Piedra, A. Braeken, A. Touhafi, Sensor systems based on FPGAs and their applications: a survey, Sensors 12 (12) (2012) 12235–12264.
[14] S. Jain, L. Lin, M. Alioto, Dynamically adaptable pipeline for energy-efficient microarchitectures under wide voltage scaling, IEEE Journal of Solid-State Circuits (2017) 1–10. In press.
[15] S. Natheswaran, G. Athisha, Remote reconfigurable wireless sensor node design for Wireless Sensor Network, in: International Conference on Communication and Signal Processing, Melmaruvathur, 2014.
[16] A. Iwasaki, Y. Shibata, K. Oguri, R. Harasawa, An energy-efficient FPGA-based soft-core processor with a configurable word size ECC arithmetic accelerator, IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XVIII), Yokohama, Apr., 2015.

[17] I. Finlayson, G.R. Uh, D.B. Whalley, G. Tyson, An overview of static pipelining, IEEE Comput. Archit. Lett. 11 (1) (2012) 17–20.
[18] W.-P. Kiat, K.-M. Mok, W.-K. Lee, H.-G. Goh, I. Andonovic, A comprehensive analysis on data hazard for RISC32 5-stage pipeline processor, in: 31st International Conference on Advanced Information Networking and Applications Workshops (AINA), Taipei, Taiwan, 2017.
[19] L. Pezzarossa, M. Schoeberl, J. Sparsø, A controller for dynamic partial reconfiguration in FPGA-based real-time systems, IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC), Toronto, ON, May, 2017.
[20] A. Tajalli, Y. Leblebici, Design trade-offs in ultra-low-power digital nanoscale CMOS, IEEE Trans. Circuits Syst. I Regul. Pap. 58 (9) (2011) 2189–2200.
[21] S.Y. Liew, C.K. Tan, M.L. Gan, H.G. Goh, A fast, adaptive, and energy-efficient data collection protocol in multi-channel-multi-path wireless sensor networks, IEEE Comput. Intell. Mag. 13 (1) (2018) 30–40.
[22] K. Neshatpour, W. Burleson, A. Khajeh, H. Homayoun, Enhancing power, performance, and energy efficiency in chip multiprocessors exploiting inverse thermal dependence, IEEE Trans. Very Large Scale Integr. VLSI Syst. 26 (4) (2018) 778–791.
[23] T. Gomes, S. Pinto, T. Gomes, A. Tavares, J. Cabral, Towards an FPGA-based edge device for the internet of things, in: IEEE 20th Conference on Emerging Technologies& Factory Automation (ETFA), 2015, pp. 1–4.
[24] Z. Lin, S. Sinha, W. Zhang, 'An ensemble learning approach for in-situ monitoring of FPGA dynamic power, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 38 (9) (2019) 1661–1675.
[25] L. Bauer, M. Shafique, S. Kramer, J. Henkel, RISPP: rotating instruction set processing platform, in: 44th ACM/IEEE Design Automation Conference, 2007.
[26] J. Henkel, L. Bauer, M. Hubner, A. Grudnitsky, i-core: A run-time adaptive processor for embedded multi-core systems, in: International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), July, 2011.
[27] A. Ouadjaout, N. Lasla, M. Bagaa, M. Doudou, C. Zizoua, M.A. Kafi, A. Derhab, D. Djenouri, N. Badache, DZ50: Energy-efficient wireless sensor mote platform for low data rate applications, Procedia Comput. Sci. 37 (2014) 189–195.
[28] C. Panait, D. Dragomir, Measuring the performance and energy consumptionof AES in wireless sensor networks, in: Proceedings of the Federated Conference onComputer Science and Information Systems, 2015, pp. 1261–1266.
[29] W. Zhang, Y. Zhang, L. Xu, J. Zhou, Y. Liu, M. Gu, X. Liu, S. Yang, An intelligent traffic load prediction-based adaptive channel assignment algorithm in SDN-IoT: a deep learning approach, IEEE Access 7 (2019) 32754–32765.

Wei-Pau Kiat received the Bachelor of Information Technology in Computer Engineering and Master of Science in Computer Science from Universiti Tunku Abdul Rahman, Malaysia in 2015 and 2019 respectively. He is currently an SoC Design Engineer in Intel Malaysia. His research interests are in the areas of computer arithmetic and architecture, embedded system design, image processing and robotic.

Kai-Ming Mok received the Diploma in Technology (Electronic Engineering) from College Tunku Abdul Rahman in 1997. He later obtained M.Sc Mechatronics from De Montfort University in 1998 and M.Sc Electronics from Queen's University Belfast in 2001. He is currently a Senior Lecturer in the Department of Computer and Communication Technology, Faculty of Information and Communication Technology, University Tunku Abdul Rahman, Malaysia. His research interests include digital systems design, computer architecture and micro-architecture design, and FPGA.

Wai-Kong Lee received the B.Eng. in Electronics and M.Sc. degree from Multimedia University in 2006 and 2009 respectively. He had obtained the Ph.D. degree in Engineering from University Tunku Abdul Rahman, Malaysia in 2018. He was also visiting scholar to Carleton University, Canada (2017), Feng Chia University, Taiwan (2016 and 2018) and OTH Regensburg, Germany (2015 and 2018). He had served as reviewer for several international journals, such as IEEE Transactions on Dependable and Secure Computing (2016 and 2017) and Computer and Electrical Engineering (2017). His research interests are in the areas of cryptography, numerical algorithms, GPU computing, Internet of Things (IoT) and energy harvesting.

Hock-Guan Goh received his Bachelor of Information Technology majoring in Multimedia from University of Malaya in 2003, Master of Science in Information Technology from Multimedia University in 2006, and Doctor of Philosophy in Electronic and Electrical Engineering from University of Strathclyde in 2014. He is currently working as an Assistant Professsor in Faculty of Information and Communication Technology (FICT) Perak Campus, Universiti of Tunku Abdul Rahman (UTAR), Malaysia. His research interest includes Cognitive Wireless Sensor-Actor Networks, Agriculture/Environmental Monitoring System, Internet of Things, and Data Analysis/Processing.

Ramachandra Achar (S'95-M'00-SM'04-FM'13) received the B. Eng. degree in electronics engineering from Bangalore University, India in 1990, M. Eng. degree in microelectronics from Birla Institute of Technology and Science, Pilani, India in 1992 and the Ph.D. degree in Electrical Engineering from Carleton University in 1998. Dr. Achar currently is a professor in the department of electronics engineering at Carleton University. Prior to joining Carleton university faculty (2000), he served in various capacities in leading research labs, including T. J. Watson Research Center, IBM, New York (1995), Larsen and Toubro Engineers Ltd., Mysore (1992), Central Electronics Engineering Research Institute, Pilani, India (1992) and Indian Institute of Science, Bangalore, India (1990). His research interests include signal/power integrity analysis, circuit simulation, parallel and numerical algorithms, EMC/EMI analysis, microwave/RF algorithm and mixed-domain analysis. Dr. Achar has published over 200 peer-reviewed articles in international transactions/conferences, six multimedia books on signal integrity and five chapters in different books. Dr. Achar received several prestigious awards, including Carleton university research achievement awards (2010 & 2004), NSERC (Natural Science and Engineering Research Council) doctoral medal (2000), University Medal for the outstanding doctoral work (1998), Strategic Microelectronics Corporation (SMC) Award (1997) and Canadian Microelectronics Corporation (CMC) Award (1996). He was also a co-recipient of the IEEE best transactions paper award for T-AdvP (2007) and T-CPMT (2013). His students have won numerous best student paper awards in international forums. Dr. Achar currently serves as the Chair of the Distinguished Lecturer (DL) Program of the EMC society and DL for the Electron Devices Society. He also previously served as the DL for the CAS society (2011, 12) and EMC Society (2015, 16). He currently serves on the executive/steering/technical-program committees of several leading IEEE international conferences, including EPEPS, EDAPS, SPI, HPCPS, SIPI and in the technical committees, EDMS (TC-12 of CPMT), CAD (MTT-1) and SIPI (TC-10 of EMCS). Dr. Achar previously served as a guest editor of IEEE Transactions on CPMT, for two special issues on "Variability Analysis"g and "3D-ICs/Interconnects"g (2015) and as the General Chair for HPCPS (2012–2017), as General Co-Chair of NEMO (2015), SIPI (2016) and EPEPS (2010, 2011) and as an International Guest Faculty on the invitation of the Dept. of Information Technology of Govt. of India, under the SMDP-II program (2012). He is a founding faculty member of the Canada-India Center of Excellence, chair of the joint chapters of CAS/EDS/SSC societies of the IEEE Ottawa Section, and is a consultant for several leading industries focused on high-frequency circuits, systems and tools. Dr. Achar is a practicing Professional Engineer of Ontario and is a Fellow of IEEE and a Fellow of Engineers Institute of Canada.