A Domain-Specific Processor Microarchitecture for Energy-Efficient, Dynamic IoT Communication

Shahzad Muzaffar, *Student Member, IEEE*, and Ibrahim M. Elfadel, *Senior Member, IEEE*

***Abstract*— In this paper, we present a domain-specific processor architecture, named pulsed-index communication interface architecture (PICIA), for single-channel IoT communication based on the recently introduced pulsed-signaling protocols, according to which information is encoded as series of pulses representing ON bits. In addition to the traditional aspects of instruction set architecture (ISA) design such as addressing modes, instruction types, instruction formats, registers, interrupts, and external I/O, the ISA includes domain-specific instructions that facilitate bit stream encoding and decoding based on the pulsed-signaling techniques. The domain-specific PICIA microarchitecture employs a set of optimized processing blocks that can be used programmatically to encode and decode the transmitted data in the most economical way. The PICIA allows customizations that support both standard pulsed-signaling techniques and specialized protocols that belong to the same family. The PICIA design further allows an amalgamation of software and hardware that significantly reduces the number of instructions required to implement a given communication interface without impacting the data rates and reliability of the pulsed-signaling protocols. The PICIA processor has been implemented in Verilog HDL and tested using a Xilinx Spartan-6 field-programmable gate array (FPGA). Furthermore, a 65-nm application-specific integrated circuits (ASIC) synthesis of the design confirms the small-footprint and low-power features of PICIA. The consumed power has been evaluated at 31.14 μW with an energy efficiency of less than 10 *pJ/bit*.**

***Index Terms*— Domain-specific architecture (DSA), edge-coded signaling, instruction set architecture (ISA), Internet of Things, low-power communication, microarchitecture, single channel.**

I. Introduction

DOMAIN-specific architectures (DSAs) are computing platforms that are tailored to the characteristics of a well-defined domain. They differ from general-purpose architectures in that they are designed to execute a specific set of tasks extremely well. They also differ from application-specific integrated circuits (ASIC) in that they are programmable to efficiently work on several applications in a given domain. A DSA comes with its domain-specific instruction set, which constitutes the core of its programmability. One important benefit that a DSA has over a general-purpose CPU is that its task-tailored programs have significantly fewer instructions than those of a CPU, and therefore, achieve significant performance gains and power savings due to a much lower instructions-per-program metric.

Another important DSA benefit is that it offers micro-architects with many additional opportunities to improve hardware performance above and beyond instruction-level parallelism or hardware, e.g., multicore, parallelism. These opportunities are due to the domain knowledge that is captured in the domain-specific instruction set. Because of these new hardware opportunities for micro-architects, DSAs have been declared as the “only path left” for a hardware-centric future in computer architecture [1], [2].

The main elements of a successful DSA are the following.

1) *Domain-specific instruction set architecture (ISA):* A new instruction set has to be designed almost from scratch to embody domain knowledge. Currently, popular domains include Artificial Intelligence (AI), bio-informatics, the Internet of Things, and crypto-currencies. Traditional domains include graphics, digital signal processors, and cryptographic processors.
2) *Domain-specific software stack:* A new software stack has to be created in support of the domain-specific ISA. The stack will comprise compilers, debuggers, profilers, software-development kits (SDKs), and application-programming interfaces. A very good example of such a stack is NVIDIA’s CUDA environment for its GPUs.
3) *Domain-specific performance metrics:* Every knowledge domain will introduce its own task-specific performance metrics to evaluate various implementations of a domain-specific ISA. Examples of such new domain-specific metrics include the vertex-shader (VS) duration in graphics and the number of multiply-accumulates (MACs) in artificial neural networks.

Two recent instances of a successful DSA are Google’s Tensor Processing Unit (TPU) [3], [4] and IBM’s AI core chip [5] for the acceleration of machine learning tasks. Each of these accelerators has its own custom instruction set along with a hardware microarchitecture meant to achieve maximum performance. As compared to CPUs and GPUs developed in similar semiconductor technologies, machine learning inference on Google’s TPU is 15–30 times faster and is 30–80 times better in energy efficiency. The IBM AI core has achieved more than 90% sustained utilization across a range of neural network topologies. It is important to stress that these processors are domain-specific in that they can be programed to accelerate

Manuscript received November 29, 2018; revised March 13, 2019; accepted April 2, 2019. Date of publication May 8, 2019; date of current version August 23, 2019. This was supported by the Semiconductor Research Corporation (SRC) under the Abu Dhabi—SRC Center of Excellence on Energy-Efficient Electronic Systems (ACE^{4}S) with customized funding from the Mubadala Development Company, Abu Dhabi, United Arab Emirates, under Contract 2013 HJ2440. *(Corresponding author: Shahzad Muzaffar.)*

The authors are with Khalifa University, Abu Dhabi 127788, UAE (e-mail: shahzad.muzaffar@ku.ac.ae; ibrahim.elfadel@ku.ac.ae).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2911393

not just deep convolutional neural networks as in [6] or [7] but also other machine learning workloads such as multilayer perceptrons (MLP) or long short-term-memory (LSTM) networks.

Incidentally, the TPU and the AI core are meant to accelerate the execution of workloads arising in the context of artificial neural networks and are therefore to be distinguished from brain-inspired, neuromorphic platforms such as [8], [9], and [10], whose computational model is that of the spiking neural activity of the nervous system.

Recent examples of programmable accelerators in application domains other than neural networks or neuromorphic computing include the cryptographic processor reported in [11], the biomedical platform processor [12], and the string matching accelerator [13]. In the area of network communication, it is worth mentioning Cisco's routers where the main CPU (e.g., MPC860 PowerQUICC processor from Motorola/NXP) includes an on-chip Communication Processor Module (CPM) [14]–[16]. The CPM is a RISC microcontroller dedicated to several special-purpose tasks such as signal processing, baud rate generation, and direct memory access (DMA). CPM may, therefore, be considered a domain-specific processor for the network-routing domain.

The major goal of this paper is to introduce the DSA of a novel streaming processor for an application domain that has not been addressed in the open literature, namely, the domain of IoT communication. The presentation of this DSA will be self-contained in that it will include the ISA itself, the minimal microarchitecture needed to implement it, and an analysis of its performance using the thin software stack needed to run programs on the implemented processor. In particular, the microarchitectural opportunities for designing a minimal, area- and power-efficient processor for this important domain, will be highlighted. Metrics such as data rate, energy efficiency, area, and power will be given and compared with those of various hardwired implementations of the IoT signaling techniques under consideration.

In the specific domain of communication protocols, there have been two main approaches for their implementation. In the first approach, the communication engineer programs the entirety of the protocol on a microprocessor and control their selection and parameters through registers. This is a standard practice that is followed for data transfer protocols such as I^{2}C, I^{2}S, serial peripheral interface (SPI), universal asynchronous receiver/transmitter (UART), and controller area network [17]. The second approach is to design an ASIC for the newest generation of the protocol and make it backward compatible with older versions as in the case of USB 2.0 and 3.0 [18]. The latter approach increases silicon area and power consumption and does not provide any customization features. Clearly, a pure software implementation, e.g., assembly language, on a general purpose processor often leads to an inordinate number of instructions to execute, which results in a drastic reduction in achievable data rates. It is one of the major benefits of a DSA to reduce the number of instructions needed to run a specific task while its programmability enables the economic implementation of several variants of a given task.

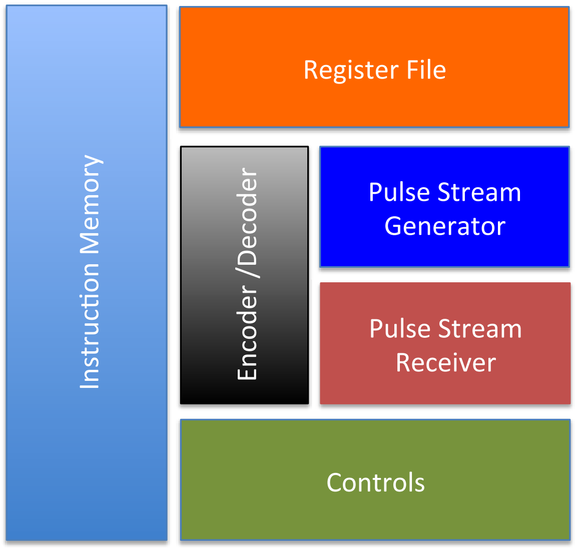

Fig. 1. Main components of the PICIA microarchitecture.

The novel DSA introduced in this paper is dedicated to a family of single-channel, high-data-rate, low-power dynamic communication protocols collectively labeled as Pulsed-Index Communication (PIC). Three different generations of these protocols have been published [19]–[21]. The most important feature of this family of protocols is that they do not require any clock and data recovery (CDR). They are also highly tolerant of clocking differences between transmitter and receiver and are fully adapted to the simple, low-power, area-efficient, and robust communication needs of IoT devices and sensors. In line with the original protocol, the DSA architecture is called "PIC interface architecture" (PICIA). The PICIA processor, whose main components are shown in Fig. 1, possesses the following important properties.

1) Its ISA is a compact RISC-like architecture with 22 instructions, only one of which has a branching condition.
2) It is implemented as a full processor with its own instruction memory, and instruction fetching and decoding.
3) It is a streaming processor with no data memory. Incoming data are streamed directly into the register file.
4) It is a secure processor in that a cryptographic block can be added to the microarchitecture and enabled using additional instruction in the ISA.
5) It has a thin software stack that enables the coding and compiling of ultracompact C programs for implementing the various PIC protocols.
6) It brings power consumption closer to a dedicated ASIC design by enabling the sharing of optimally designed common and dedicated hardware modules and by providing the flexibility to reconfigure information flow between the various modules.
7) It enables the customized implementation not only of the standard PIC protocols but also of any communication protocol that uses the same underlying idea of transmitting information in the form of pulses.

The PICIA processor has been implemented and tested in Verilog HDL over the Xilinx Spartan-6 field-programmable gate array (FPGA) and in ASIC using GlobalFoundries 65-nm process. Among its distinguishing features is a power consumption that is well within the power envelope of a USB-style hardwired protocol, consuming only 31.4μW, which translates into an energy efficiency of less than $10pJ$/bit.

This paper is organized as follows. Section II provides the domain knowledge underlying the novel DSA. In Section III, the RISC-like ISA is reviewed along with its register set, its interrupts, and I/O ports. Section IV is the core of this paper and is devoted to the microarchitectural blocks of the PICIA processor. All hardware design decisions and their relationships with the communication protocols under consideration are explained in this section along with the clocking methodology used to synchronize the operations of the processor. In Section V, two processor implementations are described, FPGA and ASIC. Metrics related to data rate, power, and energy efficiency are quantified and compared with those of hardwired implementations. In particular, the major impact the DSA has on program size reduction is illustrated along with the flexibility it provides to implement several variants of the communication protocols with no significant impact on power consumption or data rate. The paper is concluded in Section VI.

II. Domain Knowledge: Pulsed-Index Signaling

A. Background on Pulsed-Index Communication

Pulsed-signaling techniques are based on the fundamental concept of transmitting binary word attributes rather than modulated bits. The attributes are quantified, coded as pulse counts, and transmitted as streams of pulses. The key to the success of these techniques is the encoding step whose goal is to minimize the pulse count. At the receiver, the decoding is based on pulse counting by detecting the rising edge of each pulse. These techniques have the distinguishing feature that they do not require any CDR, which significantly contributes to their low-power and small foot-print when implemented in hardware. Recently, three techniques based on this concept have been introduced, namely, PIC [19], Pulsed-Decimal Communication (PDC) [20], and (PICplus) with edge-coded signaling (ECS) [21]. With slight differences, these techniques apply an encoding scheme to a data word B to minimize the number of ON bits, and move them to the least-significant-bit (LSB) end of the packet with the goal of lowering the number of pulses required to transmit the data bits. The encoding process includes a segmentation step where the data are broken into N independent segments of size l bits each (i.e., $N = B/l$). To maximize the data rate, they use, on each segment, an encoding combination of bit inversion and/or segment reversion/flipping. For PIC and ECS, this combination is meant to reduce the number of ON bits and decrease their index values. For PDC, the same combination is intended to reduce the number of ON bits and decrease the decimal number represented by each segment. To facilitate decoding, flag pulses representing the type of encoding performed are added to each segment. Unlike PIC, the PDC segment flags

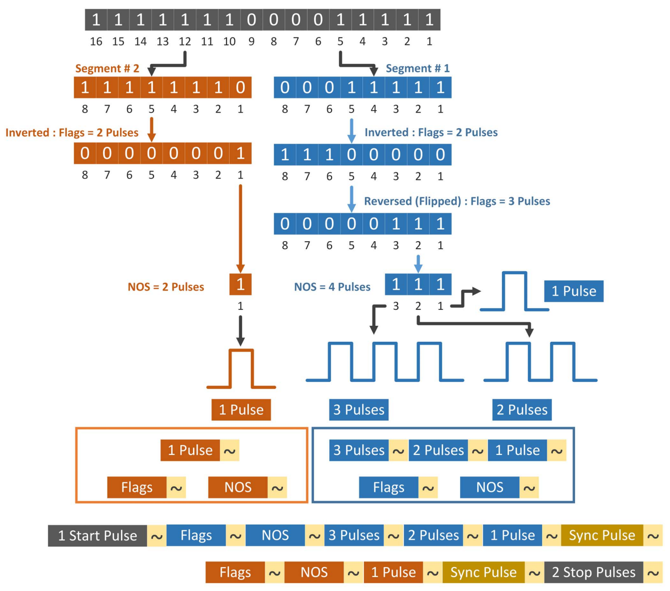

Fig. 2. PIC encoding and packetization example.

of two consecutive segments and the ECS segment flags of four consecutive segments are combined in one data word flag and placed in the header. The PDC further applies a third segmentation step postencoding whose goal is to further reduce the number of pulses per segment and, therefore, increase the data rate.

All the pieces of information including flags, the number of indices, and the indices themselves in the case of PIC and ECS, or the decimal numbers of each segment in the case of PDC, are transmitted in the form of pulse streams. The pulse is characterized by its width which is the number of clock cycles during which it remains high. Within a given packet, segment pulse streams are separated by an intersymbol delay α. To describe the process of PIC data transmission, an example is shown in Fig. 2. A decimal number 65 055 is considered as a 16-bit data word for transmission. The 16-bit data word is divided into two independent segments, each of 8 bits, which reduces the index numbers of most significant bit (MSB), and consequently, the number of pulses to represent the ON bits. Because the number of ON bits in Segment#1 is higher than half of the segment length (5 and 4, respectively), the bits are inverted, and the Flags of Segment#1 are set to 2. This step further reduces the number of ON bits in Segment#1 but the index numbers of the ON bits are located in the MSB part of the segment. The bitwise flip operation is therefore applied to relocate the ON bits to the LSB part, which results in the reduction of the number of pulses. The Flags field of Segment#1 is now updated to 3, signifying that both of the encoding operations are applied. The same steps are applied to Segment#2 except that only the inversion operation is needed. The Flags field of Segment#2 are set to 2, signifying that only the inversion operation is applied. At this point, all the information including the encoded segments, Flags, and the number of ON bit locations (NOSs) are ready to start the transmission. All the pieces of information are transmitted in the form of pulse streams separated by intersymbol delays. In this example, the bit data has a raw number of 106 pulses

TABLE I
COMPARISON OF PIC AND UART

Feature	UART	PIC
Link	Two wires (cabled)	1 channel (cabled or wireless)
Mode	Asynchronous	Asynchronous
Data frame	8 bits	16 bits (or more)
Network	Master-Slave	Master-slave or peer-to-peer
Tolerance	10% of baud rate	20% of clock frequency
Data rate	9600 baud (typical) - 115,200 baud (max)	3.1 - 8.5 Mb/s (Average: 4.1 Mb/s)

that, after segmentation and encoding, have been reduced to just 7 pulses.

The receiver counts the number of pulses for each pulse stream and applies the decoding according to the Flags field in the received packet. The data rate depends on the number of clock cycles required to transmit a packet. Besides the elimination of CDR, the PIC implementation is area-efficient, low-power and highly tolerant of clocking differences between the transmitter and receiver [22]. PIC and ECS have a variable number of symbols per data word, which enables the addition of security layers [23], whereas PDC presents a fixed number of symbols per data word, which improves transmission reliability with respect to packet failures [20]. The pulsewidth is an important parameter that gives the hardware designer an additional knob for managing power consumption [24]. A full IoT network platform based on PIC is described in [25]. The automatic detection and configuration of protocol parameters under various clock frequency settings is discussed in [26].

B. Advantages of PIC Over Other Serial Techniques

PIC has distinct advantages over other serial techniques such as SPI, I2C, and UART. One main advantage is that PIC is a single-channel communication protocol that can be used in both cabled or wireless format. On the other hand, SPI is a cabled protocol with four wires, while I2C and UART are cabled with two wires. Another advantage of PIC is that it can be configured in both master-slave mode and peer-to-peer mode, while the other protocols are restricted to the master–slave mode. The SPI and I2C protocols are synchronous while PIC and UART are asynchronous because there is no clock signal being communicated between transmitter and receiver. When compared specifically with UART, PIC has additional advantages that are summarized in Table I. However, the most important advantage of PIC with respect to UART is that PIC encodes data bit indices as pulse counts and transmits these counts as information to the receiver. With proper data frame segmentation and encoding, the significantly higher data rate can be achieved while avoiding the problems inherent to bit timings and bit pulsewidth. As a result of this encoding, PIC does not require any CDR or duty cycle correction circuitry at the receiver in order to synchronize the receiver sampling clock with the incoming bit stream. PIC can tolerate up to 20% discrepancies between transmitter and receiver clocks and is capable of working with the deformed signal waveform (such as narrow triangular pulses) while supporting a variety of communication media such as electromagnetic, optical, or bio-electrical. Furthermore, PIC's rate of transmitting pulses is limited only by the detection threshold of the CMOS technology used to implement the underlying hardware. The PIC pulsewidth can be as narrow as allowed by the switching times of the pulse generator and the threshold of the pulse detection logic. PIC also allows large flexibility in pulse shaping, with sharp triangular pulses allowed as long as they satisfy switching and peak detection constraints and do not overlap with each other.

III. PULSED-INDEX COMMUNICATION INTERFACE ARCHITECTURE

As is clear from Section II, PIC family members share the same functional and structural features. The functional features include segmentation and encoding while the structural features include using Flags, NOS, and intersymbol delays in the PIC packet. For instance, in PIC the encoding reduces the number of ON bits and their indices in the segments, whereas in PDC, it reduces the decimal number representing sub-segment. In both cases, the net result is a reduced number of pulses. The PIC family members further share the same receiver mechanism, which consists of simply detecting and counting the edges of the incoming pulses. Their packet formats are also quite similar. Although similar in structure and functions, they also leave a significant room for variations on the common themes of segmentation, encoding, representation, packetization, and pulse transmission. Possible variations may include the transmission of both the index numbers and decimal values, which is a form of redundancy to improve transmission reliability. Another form of redundancy is to repeat the transmission automatically with only a single start signal. To support the existing protocols and any possible variations within the PIC family, this paper proposes a domain-specific instruction set, called the PICIA, to bring the various pulsed signaling techniques under one umbrella. PICIA can be used to generate not only the standard protocols with tunable communication parameters (i.e., segment size, intersymbol delay, pulsewidth, etc.) but also it can be used to develop other customized communications techniques that use the same underlying idea of transmitting information in the form of pulses. The PICIA microarchitecture uses a set of fundamental, optimized, hardware building blocks that can be programed to implement various versions of PIC. These building blocks include pulse generators, ON-bit counters, adders, segmenters, encoders, register files, and memories. In this paper, the PICIA processor is considered a self-contained, stand-alone streaming processor. However, it can also be embedded as a coprocessor in a system-on-chip (SoC) along with other CPU's, GPU's, and accelerators. The logic circuit details of the microarchitectural building blocks will be given in Section IV. The rest of this section will be devoted to a review of the ISA itself. For more details on the ISA, the reader is referred to [27].

A. Register Set

To enable protocol customization, registers are needed to enable the configuration of hardware modules, the control of data flow, the buffering of input and output data, and the

TABLE II
PICIA Register Set

	Register	**Type**	**Organization**
1	R_0-R_7	8 bit GP[a]	8-bit Value
2	Ctrl_0	8 bit SP[b]	[0,Mode,3-bit SegNum,3-bit SegSize]
3	Ctrl_1	8 bit SP	8-bit Pulse Width
4	LoadReg	16 bit SP	16-bit Value

[a] General Purpose [b] Special Purpose

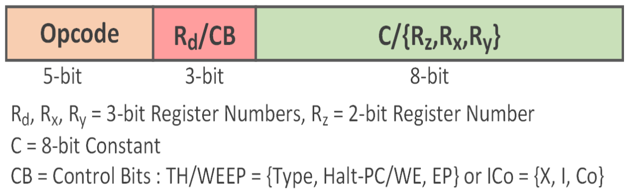
Opcode
Rd/CB
C/{Rz,Rx,Ry}
5-bit
3-bit
8-bit
Rd, Rx, Ry = 3-bit Register Numbers, Rz = 2-bit Register Number
C = 8-bit Constant
CB = Control Bits : TH/WEEP = {Type, Halt-PC/WE, EP} or ICo = {X, I, Co}

Fig. 3. PICIA instruction formats.

tracking of execution status. To accommodate these needs, the PICIA uses three types of registers.

1) General-purpose registers: They form a set of eight 8-bit registers, R_0 through R_7, that are programmer-accessible.
2) Control registers: There are two 8-bit registers, Ctrl_0 and Ctrl_1, which are used to store protocol configuration parameters such as mode of transaction (transmitter or receiver), segment number, segment size, and pulsewidth in terms of a number of clock cycles. These control registers are initially set by the programmer through specific instructions but once set, they become accessible only to the system.
3) IO register: This is the *LoadReg* register, which is a 16-bit, I/O-dedicated register used to read the I/O port, set the I/O port, and store the updated results after an instruction is executed. Like the Control Registers, *LoadReg* is a privileged register and is accessible only to the system.

These register types are summarized in Table II. In the remainder of the paper, the word register will solely refer to a general-purpose register.

B. Instruction Formats and Types

The PICIA instruction formats are shown in Fig. 3 and the assembly language instructions are given in Tables III and IV. The PICIA instructions are all 16-bit long in line with a RISC architecture. The 16-bit length is chosen to accommodate PIC protocols based on 16-bit data segments. Each instruction is divided into three main parts: Opcode, R_d/CB, and $C/\{R_d, R_x, R_y\}$. The 5-bit *Opcode* represents the type of operation. R_d, R_x, and R_y represent 3-bit register numbers. R_z represents a 2-bit register number. C is the constant operand value. The CB field contains the control bits and, depending on the instruction, can be either TH/WEEP or ICo. The TH/WEEP is the combination of three bits representing the *type of operand (T)*, *Halt-PC (H)* or *Write enable (WE)*, and *extra pulse enable (EP)*. The ICo is the combination of I and Co bits that are used to specify the condition for copying the register contents. In view of these two CB combinations, the instructions can be grouped into three types: one that uses TH/WEEP bits (highlighted in gray in Table III), a second that replaces TH/WEEP with ICo bits, and a third that does not care about the CB field (highlighted in dark gray).

1) Type I: The first type of instructions, highlighted in gray, handles one operand at a time and are used in operations such as to read/write the I/O port, set/clear the LoadReg, set various communication protocol parameters, and send/receive pulse streams. These instructions use CB in the second part of the instruction format where CB represents TH/WEEP. T is used to set the type of operand (register or a constant) in an instruction. *H/WE* is used either to halt the PC during the transmission of pulse streams or to enable the store operation of received pulse-count to a specified register. The bit *EP* is used if an extra pulse should be added to the transmitted pulse stream and/or an extra pulse should be removed from the received pulse stream. The last 8-bits-long fragment of instruction is used to indicate a register number or an immediate constant value.

2) Type II: The second type of instructions handles two or three operands simultaneously and are used in operations such as encoding (inversion and reversion with or without condition), combining and splitting encoding flags, and copying register contents or some other information to a specified register conditionally. These instructions use CB in the second part of the instruction format where CB represents ICo. The combinations of I and Co bits are used to select the source of information to be copied. The 3-bit R_x and R_y, and 2-bit R_z *register* fields are used to indicate one of the general purpose registers.

3) Type III: The third group of instructions, highlighted in dark gray, handles two operands at a time and are used in operations such as updating a register with a given constant value and jumping to a specified label in the code depending on the validity of a condition specified by a register. Instead of CB, these instructions use R_d in the second part of the instruction format. The 3-bit *Register* field is used to indicate one of the general purpose registers, and the 8-bits *constant* field is used to provide either a constant value or a label in the code to jump to.

C. Addressing Modes

PICIA does not need any data memory. Therefore, the operands of all the instructions in Table III are either included in the instruction itself or accessed directly through the registers. As a result, PICIA employs only three addressing modes: immediate, register, and autodecrement. In the immediate mode, the source is either a constant or a label while the destination is one of the general-purpose, special-purpose, or program counter (PC) registers. In the register mode, the register contains the value of the operand. The autodecrement mode is used only for a jump operation where

TABLE III
PICIA INSTRUCTION SET

	Instruction	CB	Description	Example
1	RP	-	Load data from Input Pins to data register.	RP
2	WP	-	Output the received data from data register to the Pins.	WP
3	SSS C	TH/WEEP	Set segment size (C = 0,1,2 for 4 bit,8 bit,16 bit).	SSS 1
4	SSN C	TH/WEEP	Select segment number (C = 0,1,2,3).	SSN 2
5	SM C	TH/WEEP	Set Mode (C = 0,1 for Transmitter, Receiver). Setting RX mode clears LoadReg, setting TX loads input into LoadReg.	SM 0
6	SW C	TH/WEEP	Set width of pulse (C = integer specifying cycle count).	SW 2
7	IV R_x,R_y	ICo	Inverse the selected segment. R_x=NOI & R_y=Flags (R_x/R_y= R_0,R_1,…R_7).	IV R0,R1
8	IVC R_x,R_y	ICo	Inverse conditionally the selected segment if encoding condition satisfy (ON bits >Seg. Size/2). R_x=NOI & R_y=Flags (R_x/R_y= R_0,R_1,…R_7).	IVC R0,R1
9	FL R_x,R_y	ICo	Flip selected segment bits. R_x=NOI & R_y=Flags (R_x/R_y= R_0,R_1,…R_7).	FL R0,R1
10	FLC R_x,R_y	ICo	Flip conditionally the selected segment bits if encoding condition satisfy (Seg. >Flip(Seg.)). R_x=NOI & R_y=Flags (R_x/R_y= R_0,R_1,…R_7).	FLC R0,R1
11	IVFL R_x,R_y	ICo	Invert and Flip selected segment bits. R_x=NOI & R_y=Flags (R_x/R_y= R_0,R_1,…R_7).	IVFL R0,R1
12	SP H, EP, C/R_y	TH/WEEP	Send C or R_y number of pulses (R_y = R_0,R_1,…R_7, C = constant). Halt PC if H=1 (H=0,1). Send one extra pulse if EP=1 (EP=0,1).	SP 1,1,4
13	SD H, C/R_y	TH/WEEP	Inter-Symbol delay of C or R_y number of clock cycles (R_y = R_0,R_1,…R_7, C = constant). Halt PC if H=1 (H=0,1).	SD 1,4
14	WR R_d, C	-	Write constant value to a register R_d (R_d= R_0,R_1,…R_7).	WR R0,8
15	SRD C/R_y	TH/WEEP	Set Receiver Inter-Symbol Delay equal to C or R_y number of clock cycles (R_y = R_0,R_1,…R_7, C = constant).	SRD R0 or SRD 4
16	WRI WE, EP, R_y	TH/WEEP	Wait for receiver pulse stream interrupt. PC halts till the interrupt arrives. Remove one extra pulse count if EP=1 (EP=0,1). Enable received pulse count write to register R_y (R_y= R_0,R_1,…R_7) if WE=1 (WE=0,1).	WRI 1,1,R0
17	SDB C	TH/WEEP	Sets the index bits or the data bits in the LoadReg as per the received pulse stream. (C=0,1 for indexing and data respectively).	SDB 1
18	BNZD R_d, label	-	Branch to label and decrement R_d by 1 if the specified register R_d contains non-zero number. (R_d= R_0,R_1,…R_7).	BNZD R0,loop
19	CRC R_x,R_y,I,Co	ICo	Copy register conditionally. R_x= R_y if I=0. R_x= R_y, if I=1 and LoadReg [R_y]=1 and Co=0. R_x=0 otherwise. R_x=Selected Segment, if Co=1. R_y is ignored. (R_x/R_y= R_0,R_1,…R_7). Can be used to clear the register.	CRC R1,R2,1,1
20	CF R_z,R_x,R_y	ICo	Combine Flags. R_z={R_x[1:0], R_y[1:0]}. R_z=R_0,…R_3. R_x/R_y=R_0,…R_7.	CF R0,R1,R2
21	SF R_z,R_x,R_y	ICo	Split Flags. R_x=R_z[3:2], R_y=R_z[1:0]. R_z=R0,…R3. R_x,R_y=R_0,…R_7.	SF R1,R2,R0
22	NOP	-	No operation.	NOP

TABLE IV
PICIA INTERPRETATIONS

Instruction Interpretation	
Control Bit	**Value : Effect**
T (Type R/C)	0 : Register, 1 : Constant
H (Halt-PC)	0 : No Halt, 1 : Halt
WE (Write Enable)	0 : Register Write Disabled, 1 : Enabled
EP (Extra Pulse)	0 : Extra Pulse Disabled, 1 : Enabled
I	0 : No Indexing, 1 : Indexing
Co	0 : Copy Segment Disabled, 1 : Enabled

the branch to a label is taken and a specified register decrements by one if the register contains a nonzero number.

D. External I/O and Interrupts

As highlighted earlier, the PICIA processor can be used as a stand-alone, streaming, core processor or as a co-processor in an SoC. The PICIA I/O ports and the interrupts are designed to accommodate both the stand-alone and SoC configurations, using a simplified interface with three external I/O ports. One of these ports is the 16-bit data I/O port that is used to read from and write back to the external environment. To transmit and receive the packets in the form of pulse streams, a 1-bit *signal* I/O port is used. Another 1-bit *data-ready* port is used to source the generation of I/O interrupts and start the execution of instructions.

To interact with the I/O interface and achieve low-power operation, the PICIA processor supports a workload-based interrupt mechanism consisting of three interrupt signals. The first is the I/O interrupt, which is generated when the data at the I/O port is available. The processor remains in a halt state until the I/O interrupt is received and starts instruction execution from the very start. The second is the transmitter interrupt, which is used to indicate the completion of the transmission of one pulse stream. If halt is enabled, the PICIA processor remains in a halt state until the transmitter interrupt is received, at which time the execution is continued from the point it was halted. The third interrupt is the receiver interrupt, which is generated when the reception of one pulse stream is completed. The PICIA processor remains in a halt state until the receiver interrupt is received, at which time the execution is continued.

E. ISA Discussion

PICIA has a very compact RISC-like instruction set of only 22 instructions that can be functionally put in 4 different categories: 1) configuration: 8 instructions; 2) encoding/decoding: 8 instructions; 3) transmission control: 4 instructions; 4) register/branching: 2 instructions.

It is important to note that this DSA does not contain arithmetic/logic or rotation instructions. Nor does it have a call stack. Its branching instructions are limited to just one. This compact, minimalist approach to the ISA is a hallmark of the current DSA trend. Indeed, the TPU instruction set itself has less than 15 instructions [3] with the interesting twist that they are CISC rather than RISC instructions. The main reason for using CISC-style instructions in the TPU is to enable efficient instruction dispatching from the host memory to the TPU using a PICe bus. In our implementation, the PICIA processor has its own instruction memory and program control.

In terms of execution latency, most of the PICIA instructions execute in one clock cycle, which obviates the need for pipelining to improve data rate. In contrast, and as expected, the average number of execution clock cycles of the TPU CISC-style instructions is between 10 and 20 clock cycles. The TPU has a four-stage execution pipeline [4].

IV. PICIA Microarchitecture

The reader is referred to Fig. 1 for the main blocks of the PICIA processor. Fig. 4 gives a more detailed breakdown of these blocks along with their connectivity, their access to signal and data ports, and their synchronous behavior. This microarchitecture further shows the clock distribution and PC control units (PCCUs), the instruction decoder, the encoding/selection unit, the encoding/selection control, and the interrupt handler. The proposed microarchitecture executes all the instructions listed in Table III. A few of these instructions use only one processing block at a time, while others use multiple blocks simultaneously to generate their outputs. As mentioned in Section III, most of the instructions are executed in one clock cycle only. Instructions such as SP and SD take more than one clock cycle to execute as they need to send or receive a number of pulses. All the micro-architecture blocks of Fig. 4 are explained in the next subsections.

A. Memory Interface

PICIA supports up to 128 KB of instruction memory in a configuration of 64 Kb × 16 with 16-bit address bus. PICIA is a data-streaming processor. Only the instruction memory is needed, and there is no need for data memory. The PC and the relevant controls are used to update the instruction memory address at each clock cycle.

B. Instruction Decoder

The instruction decoder accepts instructions from the instruction memory and decodes them to generate the appropriate control signals. The control signals are then used by several system blocks to take decisions and perform the required tasks. The PICIA decoder, shown in Fig. 5(a), processes the 5-bit opcode field in the 16-bit instruction and generates 27 control signals. The decoder logic works as per Table V where the opcode for each of the instructions is also listed. In Table V, RR is *reverse the roles*, WB is *write back*, WE is *write enable*, RE is *read enable*, SE is *store enable*, LE is *load enable*, and SGE is *stage enable*. The remaining control signals represent a specific instruction. RE and WE enable the read and write operations of a register in the register file. WB enables the write-back operation to update the load register with the results after finishing a particular task. RR is used to reverse the roles of the registers in Type I and Type III instructions. In the CF, SF, and CRC instructions, the register locations can either be used to read from, or write to, a register. For example, in CF, the second operand is used to read a register, but in SF, the second operand is used to store a part of the result. SE and LE are used to read and write the *LoadReg* in the register file. The SGE control signal is a power management signal that is used to shutoff unused modules of the architecture to save power during the execution of an instruction.

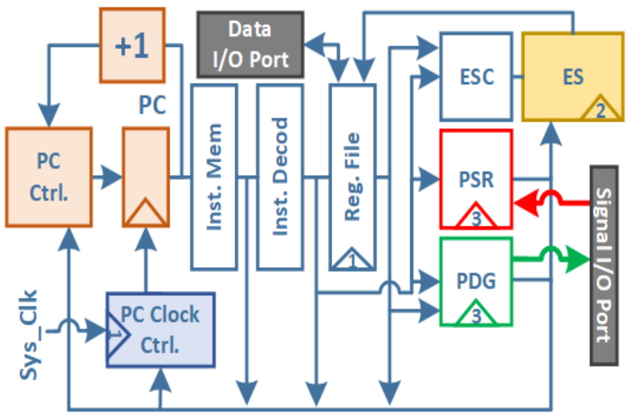

Fig. 4. PICIA microarchitecture block diagram.

C. Register File

The PICIA register file is shown in Fig. 5(b) and consists of all the registers listed in Table II. The PICIA register file supports read and write operations of two registers simultaneously and, therefore, has separate module ports for these. However, the *LoadReg* does not use any of these ports and has separate control signals. The output of the register file is divided into two parts. In the first part, the contents of the registers such as *LoadReg* and the general-purpose registers are set to the output when requested through suitable control signals. In the second part, the contents of the special purpose registers are continuously reflected at the output of the register file so as to guarantee the availability of the communication parameters for system use without the need of issuing an extra request. These parameters include segment size, selected segment number, mode of transmission, and pulsewidth. We will later show how these parameters help in selecting and updating a specific portion of the data word without additional read/write cycles and delays.

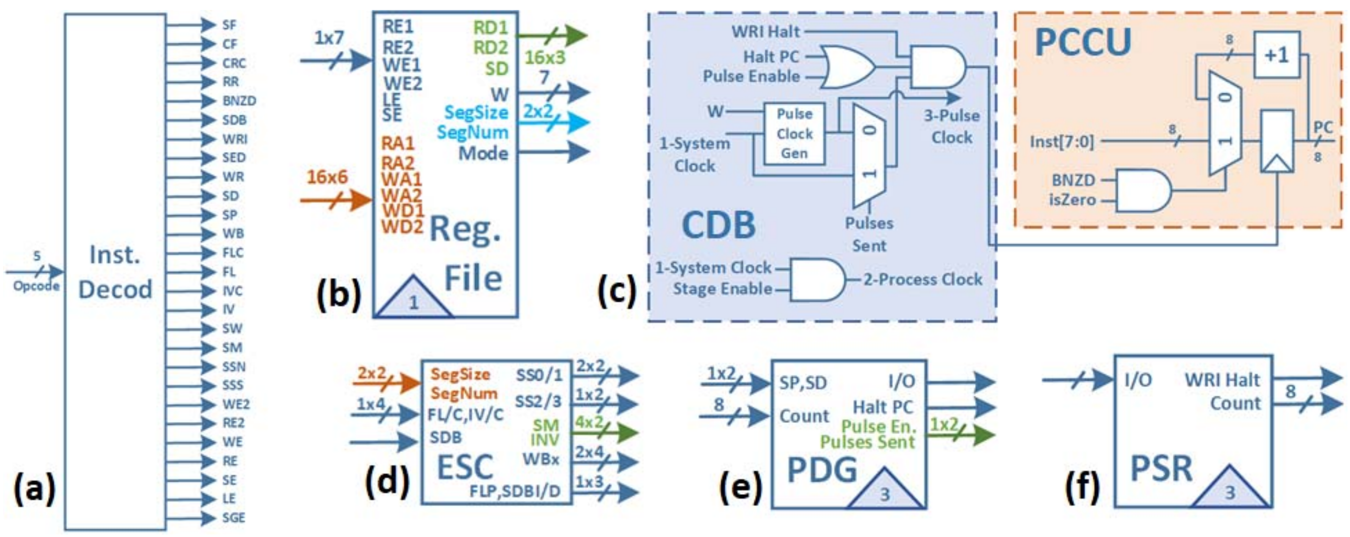

Fig. 5. PICIA microarchitecture hardware blocks. (a) Instruction decoder. (b) Register file. (c) Clock distribution and PC control. (d) ESC. (e) PDG. (f) PSR.

TABLE V
PICIA INSTRUCTIONS DECODER

Instr.	Opcode	Control Signals[a]
NOP	00000	000000000000000000000000000
RP	00001	000000000000000000000000010
WP	00010	000000000000000000000000100
SSS	00011	000000000000000000010000000
SSN	00100	000000000000000000100000000
SM	00101	000000000000000001000000000
SW	00110	000000000000000010000000000
IV	00111	000000001001000100001010011
IVC	01000	000000001001001000001010011
FL	01001	000000001001010000001010011
FLC	01010	000000001001100000001010011
IVFL	01011	000000001001010100001010011
SP	01100	000000000010000000000001000
SD	01101	000000000100000000000001000
WR	01110	000000001000000000000010000
SRD	01111	000000010000000000000000000
WRI	10000	000000100000000000000000000
SDB	10001	000001000001000000000000011
BNZD	10010	000110001000000000000011000
CRC	10011	001100001000000000000011000
CF	10100	010100001000000000000111000
SF	10101	100100001000000000001011000

[a] [SF,CF,CRC,RR,BNZD,SDB,WRI,SRD,WR,SD,SP,WB,FLC, FL,IVC,IV,SW,SM,SSN,SSS,WE2,RE2,WE,RE,SE,LE,SGE]

D. Clock Distribution and PC Control

This module can be divided into two parts: a clock distribution block (CDB) and a PCCU, as shown in Fig. 5(c). The CDB takes care of all the clocks inside the PICIA system. CDB generates two types of subclocks using the main *System Clock* that is represented by "1" in the figure. The *Process Clock* is the gated system clock, and all the blocks execute using this. The process clock is denoted by "2" in the figure, and the gating is enabled if the "Stage Enable" signal is inactive. The second subclock is the *Pulse Clock* and is denoted by "3" in the figure. The pulse clock is used to generate pulse streams and intersymbol delays. The pulse clock is the output of a pulse clock generator that takes a pulsewidth parameter as an input from the register file and generates a clock signal accordingly. Beside clock distribution, the CDB block is also responsible for controlling the halt state activation and deactivation as may be requested. Recall that the halt state is kept active in the reception mode but is activated in the transmission mode only if requested via either the SP or SD instruction.

The PCCU takes care of updating the PC register. In an upcoming clock cycle, PC can either be incremented or updated by a jump address if the jump instruction BNZD is executed. In case of reset or end of instructions (e.g., the transaction is completed), the control keeps the system in halt state unless an interrupt for a new transmission is received at which time the execution restarts from the very first instruction in memory.

E. Encoder and Select Control

The encoder and select control (ESC) block is another specialized decoder that helps in generating the control signals for the most complex unit in the PICIA processor, namely, the encoder and selector (ES). The ES block itself is described in Section IV-F. As for the ESC block, its I/O is shown in Fig. 5(d), with the outputs being control signals for the ES block as per Table VI. The ESC inputs come from two sources. The segment size and segment number parameters come from the register file while the encoding-specific control signals come from the instruction decoder. The generated control signals enable the ES block to identify the requested operation as well as the segment and segment size to which the requested operation should be applied.

F. Encoder and Selector

The ES, shown in Fig. 6, is the core block of PICIA. The ES module selects and processes a particular segment in a given

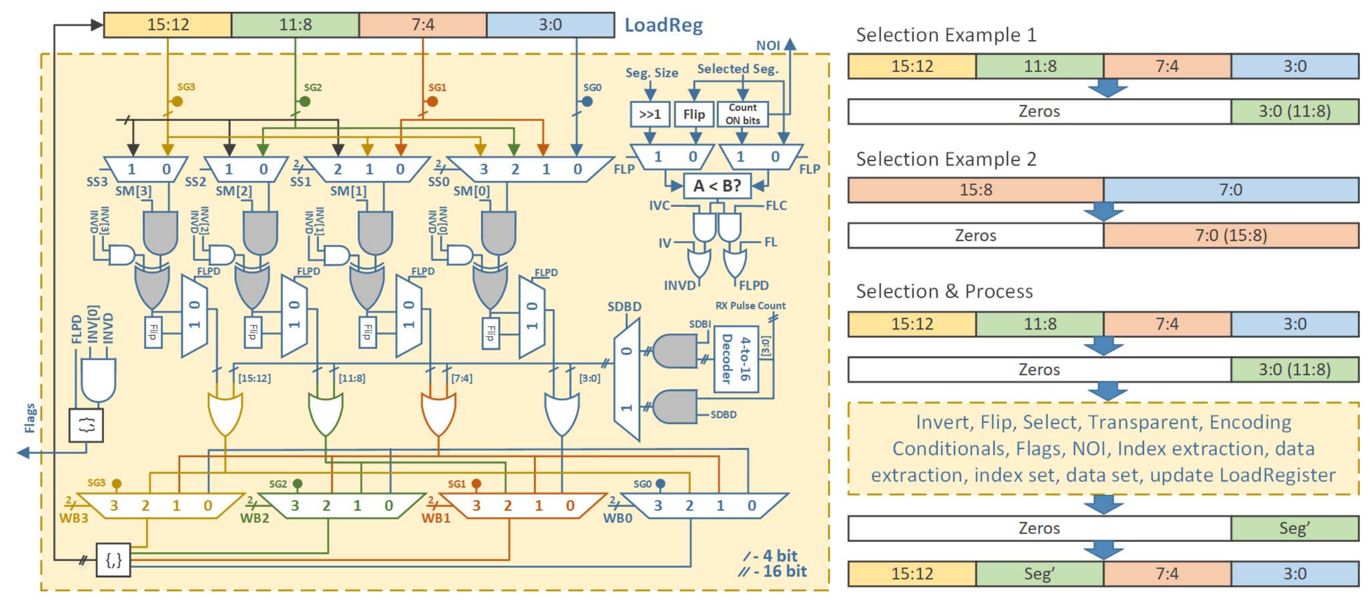

Fig. 6. PICIA microarchitecture hardware blocks: ES.

TABLE VI
DECODER: ESC

Inputs			Outputs								
Seg.Size	**Seg.Num**	**FLP**	**SS0**	**SS1**	**SS2**	**SS3**	**SM**	**WB0**	**WB1**	**WB2**	**WB3**
3-bit	**3-bit**	**1-bit**	**2-bit**	**2-bit**	**1-bit**	**1-bit**	**4-bit**	**2-bit**	**2-bit**	**2-bit**	**2-bit**
0	0	X	0	2	1	1	1	0	3	3	3
	1	X	1	2	1	1	1	3	0	3	3
	2	X	2	2	1	1	1	3	3	0	3
	3	X	3	2	1	1	1	3	3	3	0
1	0	0	0	0	1	1	3	0	1	3	3
		1	0	0	1	1	3	1	0	3	3
	1	0	2	1	1	1	3	3	3	0	1
		1	2	1	1	1	3	3	3	2	0
2	X	0	0	0	0	0	F	0	1	1	2
		1	0	0	0	0	F	2	2	2	0
INV = SM & (IV | IVC), FLP = FL | FLV											

data word as per the size and number specified by the user. It is used for the whole spectrum range of operations of bit processing and packet encapsulation, including

1) conditional and unconditional encodings;
2) generation of packet flags and number of ON bit locations;
3) extraction of the index numbers of ON bits;
4) extraction of the decimal number of a segment;
5) conditional and unconditional decodings;
6) retrieval of packet flags and number of ON bit locations;
7) bit setting according to the index number;
8) segment setting according to the decimal number;
9) conditional register copying.

Operations 1) through 4) are typically executed at transmission while operations 5) through 8) are executed at reception. As such, the ES block is the heart of the PICIA core in the same way the arithmetic logic unit is the heart of a CPU core. Given that the PICIA is a communication core, one of the important design decisions we have to make is whether the ES block should be duplicated for the transmitter and the receiver. To reduce HW resources and power consumption, the ES block is optimized such that it is used for both transmission and reception without any change whatsoever. PIC transmission and reception are designed so that they use the very same modules, including encoding/decoding and segmentation/combination. The main difference between the transmission and reception paths lies in the sources and destinations of the data. Accordingly, the PIC communication process has been divided into three main phases, each with an implementation that is common to transmitter and receiver. The three phases are as follows.

1) *Data Composition:* The source is the I/O port during transmission and the pulse stream receiver (PSR) during the reception. The destination in both cases is *LoadReg* where data is composed for further encoding or decoding process.
2) *Encoding/Decoding:* During the encoding (transmitter) or decoding (receiver), both the source and destination

are the same, namely, *LoadReg*. Segments are picked up from *LoadReg* and are replaced with the updated segments at the end of each encoding/decoding step.

3) *Output:* The source of data for both the transmission and reception is *LoadReg*. The destination is the pulse stream generator for transmission and the I/O port for the reception.

The block selects a segment from the LoadReg as per the set segment number and the segment size, relocates it to the LSB end, processes it as per the issued instruction, moves the updated segment back to its original position, and replaces the segment in the LoadReg with the processed one. Such a selection and place-back operation is shown in Fig. 6 (right), wherein the first example, "Selection Example 1," where the segment size is 4 bits, the second segment is selected and moved to the LSB end with all other bits set to zero. A similar case is shown in the second example, "Selection Example 2," where the segment size of 8 bits and the first segment is selected for transfer to the LSB part. The bottom part of Fig. 6 (right), "Selection & Process," shows the process flow when applied to the first example.

1) ES Segment Processing: The selection and relocation of a segment to the LSB end at the start of the process and the place-back operation to the original location at the end are performed using a set of multiplexers shown in Fig. 6 (left). To perform this task, the control signals *SSx* from ESC are used for segment selection and *WBx* are used for segment place-back. The second layer of AND gates uses the *SM* control signals to zero out all other bits except the selected segment so as to prevent any corruption of the results. The gray-shaded gates form a set of four identical logic functions that are used to AND/XOR all 4 bits of a data segment with the 1-bit control signal. The third layer of AND and XOR gates are used for bit inversion if requested and is controlled by the control signals *invert (INV)* and *inversion decision (INVD)*. The role of *INVD* is to identify whether the requested inversion is conditional or unconditional. The fourth layer is comprised of "Flip" and MUX blocks and is used to perform the segmentwise flipping operation if requested and is controlled by the control signal *flip decision (FLPD)*. The same *FLPD* is used to decide if the flipping is conditional or unconditional. The *FLPD* and *INVD* are generated using circuitry that is shown at the top right corner of Fig. 6 (left). If the operation is to flip, the selected segment and its flipped version are compared to generate *FLPD* accordingly. If the operation is to *INV*, the number of ON bits in the segment is compared with half of the segment size and *INVD* is generated accordingly. As its name indicates, the *Count-*ON*-Bits* block counts the ON bits in the selected segment, and the shift-right block divides the segment size by two. The flags are generated by a very simple AND and concatenate operations as shown at the bottom left side in Fig. 6. The inversion and flip operations are used in segment encoding and decoding.

2) ES in Reciever Mode: The process of reception is executed in two steps. In the first step, the received data is stored appropriately in LoadReg. To perform this step, the fifth layer in the ES block, which is comprised of a decoder, AND gates, OR gates, and a MUX, is used to either set a segment bit at index number specified by the received *RX Pulse Count*, or to translate this count into a decimal number and set the segment bits accordingly. For decimal number extraction, *set data bits - data (SDBD)* is used to pass the count through MUX, which is ORed with the selected segment bits. For setting a bit at an index, both set data bits - indices and SDBD are used to pass, through MUX, only the ON bit that is generated by the decoder, which is then ORed with the selected segment. Recall that LoadReg gets cleared when the reception mode is selected and, therefore, the decimal number or the decoded bit at the output of MUX is ORed with zeros of the selected segment. This guarantees that the received data is not garbled and is stored successfully. During the first step of the reception mode, the middle layers of encoding go transparent and do not affect the selected segments. All the segments and their bits are updated iteratively in LoadReg. In the second step of the reception mode, the updated LoadReg is used to decode as was mentioned in the previous paragraph.

3) ES in Transmitter Mode: In this mode, the extraction of the index numbers of ON bits or the segment decimal numbers from the data loaded in LoadReg is performed using the instruction CRC whose functionality is described in Table III. In executing CRC, the middle encoding layers and the received data-extraction layer go transparent. The selected segment is directly copied as a decimal number to a specified register that could be used later to transmit the pulses. To extract the index numbers of the ON bits, a program loop is used in which, for each iteration, a bit in the selected segment at the index specified by the loop index number is checked. If the bit is ON, the iteration number is copied to the specified register. Otherwise, 0 is copied. This register could be used later to transmit the pulses. It must be noted that the place-back operation of LoadReg gets disabled during the extraction operations in the transmitter mode so as to protect the data. With such transmission mode, the PICIA processor can transmit decimal numbers, ON-bit index numbers, or a combination of both to form a legal packet according to the specification of the pulsing-index protocol.

G. Pulse and Delay Generator

The PDG block transmits a pulse stream or an intersymbol delay. The module interface diagram is shown in Fig. 5(e). When SP or SD control signal at the input is set active by the instruction decoder, the module collects the count information from its input port and generates a pulse stream that is comprised of a number of pulses equal to the count. This count at the input is set by the instruction SP either through a provided immediate constant or by reading a specified register content, as described in the instruction set section. The pulse clock generated by the CDB is used to transmit the pulse stream and hence is routed to the PDG block through CDB. The width of the pulses is the same as the pulse clock and, hence, is already set by the CDB. Toward this end, the output signal *Pulses Sent* of PDG helps CDB to route the pulse clock when the SP instruction is issued and reroutes to system clock when the transmission ends. If specified in SP instruction, the signal *Halt PC* is used to halt the system until the transmission is complete. At the completion of transmission, instruction fetch is normally continued. The execution mechanism to send intersymbol delay using SD is the same as that of SP except that the signal *Pulse Enable* is inactive. *Pulse Enable* gates the pulse clock for a number of pulses specified by the count, and as a result, a delay is generated at the I/O port. The I/O port of the block is directly connected to the output port of the PICIA processor.

H. Pulse Stream Receiver

The PSR block receives one pulse stream at a time. The block interface diagram is shown in Fig. 5(f). The PICIA input is connected to the I/O port of PSR. To receive a pulse stream, the edges of the incoming pulses are counted that ends with the detection of an intersymbol delay as described in the review section. To detect such a delay, an intersymbol delay duration in terms of a number of pulses is necessary to know. This delay duration information is set by the execution of SRD instruction that updates the internal delay comparison register *RxDelay* of PSR. The WRI instruction, wait for receiver interrupt, activates this block and puts the system in halt state through CDB using the block's output signal *WRI Halt*. PSR keeps counting the stream pulses until the delay is detected. The delay is detected by counting the edges of local pulse clock when the input signal is low and comparing the count continuously with the RxDelay register. At the detection of a delay, PSR brings the system back to the normal state by generating WRI interrupt and updates the output port *Count* of the PSR block with the count of stream pulses. This count is then stored in the specified register if the option to write enable (WE) is set in the instruction. Otherwise, the count is discarded. The WE helps in receiving all the pulse streams in a particular communication packet but storing the count of desired streams only. The examples of such pulse streams include the start, stop, and synchronization patterns.

I. Interrupt Handler

The three types of interrupts (I/O, transmitter, and receiver) and their operations are described earlier in the ISA section. These three interrupts are taken care of using a small finite-state machine that in collaboration with CDB, PSR, and PDG blocks helps to halt and resume the system operation. The PICIA interrupt handler is very simple and does not use any interrupt vector or priority queue since only one interrupt can occur at a time, and such an interrupt can be handled using elementary logic.

J. Microarchitecture Discussion

As pointed out in the introduction, the major benefit of a domain-specific instruction set is that it provides hardware designers with additional opportunities for optimizing the micro-architecture to maximize performance. In the case of PICIA, this optimization is clear in the specialized blocks for encoding and selection (the ES block), for pulse generation (the PDG block) and for pulse reception (the PSR block). The ES block is the core of the PICIA microarchitecture as it is the one responsible for all processing needed to either encode or decode packets. All the instructions from the encoding/decoding category are handled through the ES block. Similarly, the PDG and PSR blocks are specialized hardware modules that attend to the fundamental nature of the communication protocols under consideration, namely, the sending and receiving of pulse streams. They can be considered hardware shortcuts for the execution of instructions belonging to the transmission control and configuration categories.

In a similar vein, the *Matrix Multiply Unit* is the heart of Google's TPU accelerator and is the main execution block of the *MatrixMultiply* domain-specific CISC instruction [4]. As mentioned earlier, the TPU acts as a coprocessor and has no instruction memory of its own. Rather the TPU CISC instructions are dispatched by the host processor to the TPU through the Peripheral Component Interconnect express (PCIe) bus. On the other hand, PICIA is implemented as a full, single-cycle, streaming processor with its own instruction storage, fetching and decoding. Although larger word lengths can conceivably be chosen for the PICIA data bus, the 16-bit word embodies domain knowledge related to the maximization of the transmission data rate of the pulsed-signaling protocols. The reader is referred to [19]– [21] for the algorithmic and experimental details on PIC data-rate maximization.

The I/O ports of the PICIA microarchitecture that are explained in Subsection III-D and illustrated in Fig. 4, represent the minimal number required for a functional processor. They are a data port, a signaling port, and received data ready interrupt port. This interface is sufficient for stand-alone operation, embedded operation, or SoC integration. However, other interface designs are conceivable, including the following.

1) Support for an external interrupt to start data transmission.
2) Support for data retransmission without recoding or resegmentation.
3) Support for internal interrupt mechanisms for data transmission and reception.
4) Support of serial and/or parallel data port options along with a SerDes module implementation.
5) Support of resegmentation and reencoding of *LoadReg*.

Finally, note that the PICIA opcode leaves room for ten more instructions, and of course, its ISA can be extended to cover other functionalities such as information security. A PIC cryptographic block [23] may be added to the microarchitecture. Such a block would be enabled and initialized with its own special instructions. A similar ISA extension approach has been recently used to secure the RISC-V processor [28].

V. Experimental Verification and Results

Verilog HDL is used to code a fully functional processor based on the proposed DSA and PICIA microarchitecture. A full testbed is set up using the Xilinx Spartan-6 FPGA platform. The prototype platform is used to verify PICIA functionality and performance. Extensive simulations and real-time hardware verification are performed to confirm the results. A clock rate of 25 MHz is used for the PICIA testbed. The verification methodology requires that the PICIA transmitter sends 16-bit data words starting at 0 with an increment of 1 at each transmission. The PICIA receiver resends the

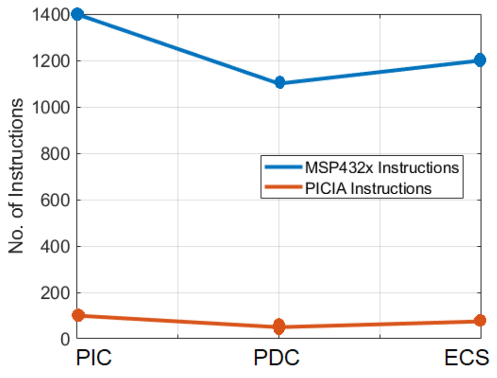

Fig. 7. PIC family implementation. PICIA versus MSP432X.

TABLE VII
RESULTS

Software Implementation Comparison			
	PICIA	**MSP432X**	**% Change**
No. Of Instructions[a]	50-100	1100-1400	92.8-95.4[b]
Data Rate (Mbps)[a]	≈4.1-7.1	≈0.041-0.071	99[c]

Hardware Synthesis Comparison			
	PICIA	**Stand-alone**	**% Change**
Power (μW)	≈31.14	≈19-26.6	14.6-38.9[c]
Avg. E$_b$ (pJ/bit)	≈4.3-7.6	≈2.7-6.5	14.5-37.2[c]
Area (gate count)	≈4700	≈2100-2400	48.9-55.3[c]
	PICIA	**Combined**	
Power (μW)	≈31.14	≈30[d]	3.6[c]
Avg. E$_b$ (pJ/bit)	≈4.3-7.6	≈4.2-7.3	2.3-3.9[c]
Area (gate count)	≈4700	≈6600	28.8[b]

[a]Average [b]Increase [c]Decrease [d]Control Logic + One active PIC protocol

same data back. The returned and original 16-bit data words are compared to check for any bit errors during the round-trip.

In another experiment, the software aspects of the two implementations are compared. In one implementation, the PIC protocols are programed on a low-power RISC processor, namely, TI's MSP432X. In another implementation, the same PIC protocols are programed using the PICIA assembly language and run on the PICIA processor. Both implementations use a 25-MHz clock. The number of instructions required to implement these techniques using MSP432X is approximately 1100 to 1400 on average, whereas PICIA needs only 50 to 100 instructions. The reduction is by a factor of 13–28. The data rates of the MSP432X implementation have suffered as a result of clock cycles being "wasted" on computation rather than used for communication. The data rate reduction was by a factor of 100. On the other hand, the PICIA data rates are maintained close to a hardwired implementation of the PIC protocols. This is a direct result of the PICIA optimized hardware and in line with the DSA philosophy of having the processor do one task extremely well. The software implementation comparison is shown in Table VII and Fig. 7.

Furthermore, the PICIA processor has been synthesized using GLOBALFOUNDRIES 65-nm technology and estimated to consume around 31.14μW with a gate count of about 4700 gates. The power estimate has been obtained from the Synopsys design compiler power report. This estimate includes both dynamic power and leakage power. Furthermore, the dynamic power includes both standard-cell switching power and interconnect switching power. A hardware solution combining the three pulsed-index family members (PIC, PDC, and ECS) has approximately 6600, which exceeds the PICIA core gate count by more than 40%. Given that two of the three hardwired protocols are gated on the combined chip to save power, the PICIA core will be comparable in terms of power and ahead in terms of area. The hardware synthesis comparisons are shown in Table VII. Note that the core logic voltage of the PIC DSA is 1.1V which is the default VDD of GlobalFoundaries 65-nm CMOS technology. The I/O voltage of the PICIA processor is 3.3 V, which, in our experiments, is the I/O voltage value provided on the Xilinx Spartan-6 FPGA board.

The PICIA power consumption remains well within the power budget of a fully hardwired implementation of a stand-alone pulsed-signaling protocol. Other advantages of the DSA are reduced gate count, preserved data rate, and reduced programming effort. Shorter programs are less likely to have bugs than long programs, and as a result, the PIC development cycle using PICIA is likely to be much shorter. The PICIA solution offers a fully programmable communication interface that is specifically geared to the realization of pulsed-transmission techniques. Such capability can, of course, be used to implement not only PIC, PDC, and ECS but also any custom nonstandard protocol without the need for any change in hardware. Table VII shows a comparison between the PICIA codes of the PIC protocols and their full hardware counterparts.

The maximum clock frequency of the PICIA processor depends on the clock frequency limits of the hardware implementing PIC. In our experiments, we have used the Spartan-6 FPGA kit that allows a maximum clock frequency of 100 MHz. However, migrating to a high-end FPGA such as Virtex-7, the maximum speed would increase to more than 500 MHz. Similarly, for an ASIC implementation, the underlying hardware technology defines the maximum clock frequency. In our experiments, we have synthesized the PICIA processor for a maximum clock frequency of 100 MHz.

Processor performance is usually analyzed using the CPU performance equation for a given workload [29]

$$t = IC \times CPI \times T \tag{1}$$

where t is workload run time on the processor, IC is the workload's instruction count, CPI is the average number of clock cycles per instruction, and T is the clock time period. Any of the three parameters can be tuned to improve CPU performance. In an IoT network, the majority of the devices use low-end processors with few tens of megahertz of the clock frequency and, therefore, the parameter T cannot be decreased beyond a certain limit. On the other hand, the reduction in the total number of instructions IC can bring a remarkable increase in performance. This is one of the major benefits of a DSA. Under the assumption that the MSP432X processor has CPI = 1 and uses the same 25-MHz clock as PICIA, then MSP432X would need 52–56 μs to execute

TABLE VIII
TIME TO FINISH WORKLOAD

	General Purpose	PICIA DSA	Speedup
PIC	≈164 Sec	0.5243 Sec	312.8
PDC	≈ 90 Sec	0.2862 Sec	314.5
ECS	≈103 Sec	0.3277 Sec	314.3

1100–1400 program instructions while the PICIA needs only 2–4 μs to execute the same task using its 50–100 instructions. Since PICIA is a communication core and the PIC data rate is dynamic, the more appropriate performance comparison is to use the following equation:

$$t_{WL} = C \times \text{WorkLoad} \times T_P \tag{2}$$

where t_{WL} is the run time of the workload, C is the average number of clock cycles needed to transmit 16-bit data words, T_P is the time period of these pulses, and WorkLoad is $2 \times 2^{16} = 131072$ 16-bit data words sent on a round trip along the PIC communication link. The C for PIC, PDC, and ECS are 100, 55, and 63, respectively. The T_P for MSP432X and PICIA are 12.5 μs and 40 ns, respectively. This is because MSP432X needs more instructions than PICIA to generate PIC pulses while the PICIA microarchitecture has a specialized block dedicated to pulse generation that is activated with a single instruction. The t_{WL} results are shown in Table VIII. It is clear that the proposed DSA for pulsed-index signaling outperforms the general-purpose CPU while remaining within the low-power budget of a hardwired protocol. As the testing workload is made of all possible combinations of 16-bit words, all other workloads will be a subset of the one we have used. Therefore, we estimate that the speedup numbers of Table VIII will remain qualitatively the same under any testing workload.

VI. CONCLUSION

The PICIA is a DSA for single-channel, low-power, high data rate, dynamic, and robust communication based on pulsed-index signaling protocols. Its RISC-style ISA is designed to facilitate the efficient coding of user-defined programs that are specific to such communication interfaces. The PICIA micro-architecture is comprised of optimized processing blocks in support of the economical execution of such programs. The PICIA supports both standard and customized pulsed-signaling protocols and enables the amalgamation of software and hardware to significantly reduce the number of instructions required to execute a given task while preserving the data rate and reliability of a bare-silicon design. The PICIA processor has been synthesized in GlobalFoundries 65-nm technology and has been found to consume only 31.14 μW, which translates into less than 10 pJ per transmitted bit. The PICIA microarchitecture has been evaluated and compared with bare-silicon designs using a domain-specific metric based on the communication workload. The ISA can be extended to include support for a cryptographic block [23] that can be readily added to the PICIA microarchitecture. One important direction for future research is to explore the usage of PICIA as a lightweight, communication core either in an IoT communication hub or in a heterogeneous computing and communication environment.

REFERENCES

[1] J. Hennessy and D. Paterson. (2018). A new golden age for computer architecture: domain-specific hadrware/software co-design, enhanced security, open instruction sets, and agile chip development. Turing Lecturea at ISCA. [Online]. Available: https://www.youtube.com/watch?v=3LVeEjsn8Ts

[2] D. Patterson, "50 years of computer architecture: From the mainframe CPU to the domain-specific TPU and the open RISC-V instruction set," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, Feb. 2018, pp. 27–31.

[3] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th ACM/IEEE Annu. Int. Symp. Comput. Archit. (ISCA)*, Toronto, ON, Canada, Jun. 2017, pp. 1–12.

[4] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *Commun. ACM*, vol. 61, no. 9, pp. 50–59, Aug. 2018.

[5] B. Fleischer and *et al.*, "A scalable multi- TeraOPS deep learning processor core for AI trainina and inference," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, Jun. 2018, pp. 35–36.

[6] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[7] S. Li *et al.*, "Heterogeneous systems with reconfigurable neuromorphic computing accelerators," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 125–128.

[8] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[9] J. Sawada *et al.*, "TrueNorth ecosystem for brain-inspired computing: Scalable systems, software, and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2016, pp. 130–141.

[10] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[11] L. Han, J. Han, X. Zeng, R. Lu, and J. Zhao, "A programmable security processor for cryptography algorithms," in *Proc. 9th Int. Conf. Solid-State Integr.-Circuit Technol.*, Beijing, China, Oct. 2008, pp. 2144–2147.

[12] A. Kulkarni *et al.*, "An energy-efficient programmable manycore accelerator for personalized biomedical applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 1, pp. 96–109, Jan. 2018.

[13] J. Van Lunteren, C. Hagleitner, T. Heil, G. Biran, U. Shvadron, and K. Atasu, "Designing a programmable wire-speed regular-expression matching accelerator," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Vancouver, BC, Canada, Dec. 2012, pp. 461–472.

[14] linux-mips.org. (2012). *Cisco Systems Routers*. [Online]. Available: https://www.linux-mips.org/wiki/Cisco

[15] Freescale Semiconductors. (2004). *MPC860 PowerQUICC Family User's Manual*. [Online]. Available: https://www.nxp.com/docs/en/reference-manual/MPC860UM.pdf

[16] S. Andrade-Morelli, E. Ruiz-Sánchez, S. Sendra, and J. Lloret, "Router power consumption analysis: Towards green communications," in *Green Communication and Networking*. Berlin, Germany: Springer, 2013.

[17] C. dos Reis Filho, E. da Silva, E. de L. Azevedo, J. A. P. Seminario, and L. Dibb, "Monolithic data Circuit-Terminating Unit (DCU) for a one-wire vehicle network," in *Proc. 24th Eur. Solid-State Circuits Conf. (ESSCIRC)*, Hague, The Netherlands, Sep. 1998, pp. 228–231.

[18] R. T. N. V. S. Chappa, B. R. Jammu, M. Adimulam, and M. Ayi, "VLSI implementation of LTSSM," in *Proc. Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA)*, Coimbatore, India, Apr. 2017, pp. 129–134.

[19] S. Muzaffar, J. Yoo, A. Shabra, and I. A. M. Elfadel, "A pulsed-index technique for single-channel, low-power, dynamic signaling," in *Proc. Design, Autom. Test Eur. (DATE)*, Grenoble, France, Mar. 2015, pp. 1485–1490.

[20] S. Muzaffar and I. M. Elfadel, "A pulsed decimal technique for single-channel, dynamic signaling for IoT applications," in *Proc. 25th IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Abu Dhabi, UAE, Oct. 2017, pp. 1–6.

[21] S. Muzaffar and I. M. Elfadel, "Dynamic edge-coded signaling for low-power IoT communication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, to be published.

[22] S. Muzaffar and I. M. Elfadel, "Timing and robustness analysis of pulsed-index protocols for single-channel IoT communications," in *Proc. 23rd IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Daejeon, South Korea, Oct. 2015, pp. 225–230.

[23] S. Muzaffar, O. T. Waheed, Z. Aung, and I. A. M. Elfadel, "Single-clock-cycle, multilayer encryption algorithm for single-channel IoT communications," in *Proc. IEEE Conf. Dependable Secure Comput. (DSC)*, Taipei, Taiwan, Aug. 2017, pp. 153–158.

[24] S. Muzaffar and I. A. M. Elfadel, "Power management of pulsed-index communication protocols," in *Proc. 33rd IEEE Int. Conf. Comput. Design (ICCD)*, New York, NY, USA, Oct. 2015, pp. 375–378.

[25] S. Muzaffar and I. A. M. Elfadel, "A versatile hardware platform for the development and characterization of IoT sensor networks," in *Proc. 59th IEEE Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Abu Dhabi, UAE, Oct. 2016, pp. 1–4.

[26] S. Muzaffar, N. Saeed, and I. M. Elfadel, "Automatic protocol configuration in single-channel low-power dynamic signaling for IoT devices," in *Proc. 24th IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Tallinn, Estonia, Sep. 2016, pp. 1–6.

[27] S. Muzaffar and I. A. M. Elfadel, "An instruction set architecture for low-power, dynamic IoT communication," in *Proc. 26th IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Verona, Italy, Oct. 2018, pp. 37–42.

[28] A. Menon, S. Murugan, C. Rebeiro, N. Gala, and K. Veezhinathan, "Shakti-T: A RISC-V processor with light weight security extensions," in *Proc. Hardware Architectural Support Secur. Privacy*, New York, NY, USA, 2017, pp. 2:1–2:8.

[29] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann, 2011.

Shahzad Muzaffar (S'16) received the B.S. degree in telecommunication engineering from the Electrical Engineering Department, National University of Computer and Emerging Sciences, Lahore, Pakistan, in 2008, and the M.Sc. degree in microsystems engineering and the Ph.D. degree in interdisciplinary engineering from Khalifa University, Masdar Campus, Abu Dhabi, United Arab Emirates, in collaboration with Massachusetts Institute of Technology, Cambridge, MA, USA, in 2015 and 2018, respectively.

From 2011 to 2013, he was a Senior Development Engineer at the ESD Division, Mentor Graphics, Lahore. From 2009 to 2011, he was a Design Engineer (Team Lead) at the Center for Excellence in FPGA/ASIC Research (CEFAR), National University of Sciences and Technology, Islamabad, Pakistan. He is currently a Post-Doctoral Fellow at the Electrical and Computer Engineering (ECE) Department, Khalifa University. Since 2013, he has been with the Masdar Institute, Abu Dhabi, where he has been a Graduate Research Assistant with the SRC Center of Excellence on Energy-Efficient Electronics Systems (ACE^{4}S) and the System Design Lead of one of two ACE^{4}S technology demonstrators. He has authored more than ten refereed publications and book chapters. He holds five pending U.S. patents.

Dr. Muzaffar was a recipient of the Best M.Sc. Thesis Award at the Masdar Institute from 2014 to 2015 and the Third Best-Paper Prize at the 2017 Cyber Security Awareness Week, New York University, Abu Dhabi Campus.

Ibrahim (Abe) M. Elfadel (S'88–M'91–SM'02) received the Ph.D. degree from MIT in 1993.

Since 2014, he has been the Program Manager of TwinLab MEMS, Abu Dhabi, United Arab Emirates, a joint collaboration with GlobalFoundries, Singapore, and the Singapore Institute of Microelectronics, Singapore, where he was involved in microelectromechanical systems. From 2013 to 2018, he was the Founding Co-Director of the Abu Dhabi Center of Excellence on Energy-Efficient Electronic Systems (ACE^{4}S), Abu Dhabi. From 2012 to 2015, he was the Founding Co-Director of Mubadala's TwinLab 3DSC, Abu Dhabi, a joint research center on 3-D integrated circuits with the Technical University of Dresden, Dresden, Germany. From 2013 to 2016, he was the Head at the Center for Microsystems (iMicro), Masdar Institute, Abu Dhabi. From 1996 to 2010, he was with the corporate CAD organizations at IBM Research, Yorktown Heights, NY, USA, and the IBM Systems and Technology Group, Yorktown Heights, NY, USA, where he was involved in the research, development, and deployment of CAD tools and methodologies for IBM's high-end microprocessors. He is currently a Professor of electrical and computer engineering at Khalifa University, Abu Dhabi, UAE. He is the Inventor or Co-Inventor of 50 issued U.S. patents with several more pending. His current research interests include IoT platform prototyping, energy-efficient edge and cloud computing, IoT communications, power and thermal management of multicore processors, low-power, embedded digital-signal processing, 3-D integration, and CAD for VLSI, MEMS, and silicon photonics.

Dr. Elfadel also served on the Technical Program Committees for several leading conferences, including DAC, ICCAD, ASPDAC, DATE, ICCD, ICECS, and MWSCAS. He was a recipient of the six Invention Achievement Awards, one Outstanding Technical Achievement Award, one Research Division Award, all from IBM, for his contributions in VLSI CAD, the SRC Board of Director Special Award (with Prof. Mohammed Ismail) for pioneering semiconductor research in Abu Dhabi. He was a co-recipient of the D. O. Pederson Best Paper Award from the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN FOR INTEGRATED CIRCUITS AND SYSTEMS in 2014. He was the General Co-Chair of the IFIP/IEEE 25th International Conference on Very Large Scale Integration (VLSI-SoC 2017), Abu Dhabi. He served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN from 2009 to 2013. He is currently serving as Associate Editor for the IEEE TRANSACTIONS ON VLSI SYSTEMS and on the Editorial Board of the *Microelectronics Journal* (Elsevier). He is the Lead Co-Editor of three books *3-D Stacked Chips: From Emerging Processes to Heterogeneous Systems* (Springer, 2016), *The IoT Physical Layer: Design and Implementation* (Springer, 2019), and *Machine Learning in VLSI CAS* (Springer, 2019).