FPGA-Accelerated 3rd Generation DNA Sequencing

Zhongpan Wu, *Member, IEEE*, Karim Hammad, *Member, IEEE*, Ebrahim Ghafar-Zadeh, *Member, IEEE*, and Sebastian Magierowski, *Member, IEEE*

***Abstract*—DNA measurement machines are undergoing an orders-of-magnitude size and power reduction. As a result, the analysis of genetic molecules is increasingly appropriate for mobile platforms. However, sequencing these measurements (converting to the molecule's A-C-G-T text equivalent) requires intense computing resources, a problem for potential realizations as mobile devices. This paper proposes a step towards addressing this issue, the design and implementation of a low-power real-time FPGA hardware accelerator for the basecalling task of nanopore-based DNA measurements. Key basecalling computations are identified and ported to a custom FPGA which operates in tandem with a CPU across a high-speed serial link and a simple API. A measured speed-up over CPU-only basecalling in excess of 100X is realized with an energy efficiency improvement of three orders of magnitude.**

***Index Terms*—Basecalling, DNA sequencing, FPGA acceleration, HMM, nanopore, PCIe, RIFFA.**

I. Introduction

SEQUENCING is the multi-step process by which DNA is sensed and by which all of its constituent molecules ('bases') are identified in terms of their four possible text symbols: A, C, G, T [1]. For reference, the totality of a human genome consists of about six billion such letters arranged in three billion 'base pairs' (bp) – A paired with T, G paired with C – forming the molecular links between the two complementary strands of the iconic DNA double-helix structure. The ability to extract DNA's textual equivalent is fundamental to a number of the life sciences and is of growing importance in medicine [2], [3]. Recently, significant advances have been achieved in sequencing technologies with implications for the design of associated computational hardware [4], [5], the main concern of this paper.

Among the recent sequencing technology advances is the adoption of nanopore sensors [6], [7]. Nanopores are nanometer-sized holes formed in a crystalline or organic material through which only one DNA strand at-a-time may pass. As it passes, the DNA's sequence of base molecules produce a corresponding electrochemical current signature. With a microelectronic readout system interfaced closely to the sensors, this current is electronically amplified, digitized, and passed to a *basecaller* system as a time-series [8]; from this input the basecaller extracts the DNA's representative text string. An example of such a time-series is shown in Fig. 1 where each level is indicative of a new portion of DNA translocating through the sensor.

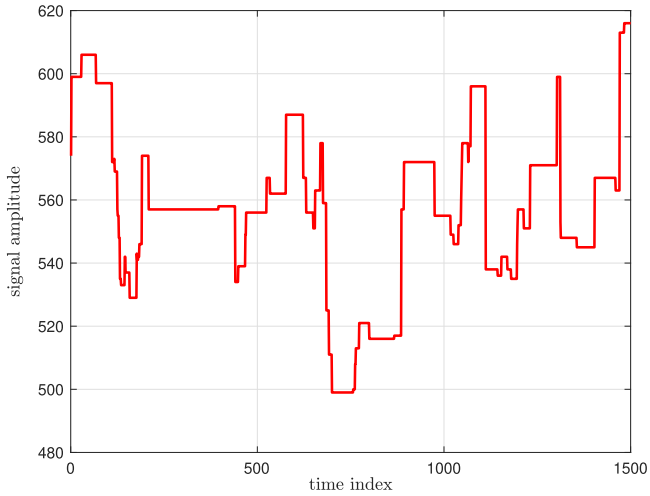

Fig. 1. An example of the time-series signal available from a nanopore sensor.

Nanopores interact with individual DNA molecules at-a-time, effectively making them single-molecule, real-time, DNA sensors, unlike existing methods that inspect aggregates of DNA copies. Such characteristics are a significant departure from incumbent techniques (e.g. 'sequencing-by-synthesis'), warranting nanopores' association with so-called 3rd generation DNA sequencing technologies [9]. Further, nanopore-based sequencing methods offer excellent opportunities for rapid DNA processing in a small device. Their ability to pass DNA at rates exceeding 10^6 bases-per-second (bp/s) per sensor [10] and the possibility of arraying 1000s of nanopores across a thumbnail-sized area [11]–[13] offers the chance for extremely high-throughput DNA processing in mobile platforms.

One impediment to this vision, and the focus of this paper, is the aforementioned basecaller. Due to sensor and measurement system limits, the quality of the time-series from which the basecaller must extract the equivalent text labels (the 'base calls') is relatively poor. As a result, the basecaller must employ a sophisticated detection scheme to achieve a sufficient accuracy,

Manuscript received September 6, 2019; revised October 28, 2019; accepted November 23, 2019. Date of publication December 9, 2019; date of current version February 4, 2020. This work was supported in part by CFIA, in part by CMC Microsystems, in part by the Natural Sciences and Engineering Research Council, and in part by SOSCIP. This paper was recommended by Associate Editor M. Carminati. *(Corresponding author: Karim Hammad.)*

Z. Wu, E. Ghafar-Zadeh, and S. Magierowski are with the Department of Electrical Engineering and Computer Science, York University, Toronto, ON M3J 1P3, Canada (e-mail: zhongpan@eecs.yorku.ca; egz@eecs.yorku.ca; magiero@eecs.yorku.ca).

K. Hammad is with the Department of Electrical Engineering and Computer Science, York University, Toronto, ON M3J 1P3, Canada, and also with the Arab Academy for Science, Technology and Maritime Transport, Cairo, Egypt (e-mail: khammad@eecs.yorku.ca, khammad@aast.edu).

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/TBCAS.2019.2958049

a scheme that needs significant computing resources if the calls are to keep up with the time-series input rate. This compromises the potential of nanopore-based sequencers to serve in mobile platforms. To address this, we propose and demonstrate a basecaller distributed across a CPU/FPGA-accelerator tandem. We show that such a system is capable of increasing the basecaller's throughput by $100\times$ while reducing the power by $20\times$ relative to a CPU-only solution.

Although the application of FPGAs to the acceleration of bioinformatics is not uncommon it has tended to focus on problems at the front-end stage (i.e. dealing with biological samples preparation and DNA-to-signal transduction) and further down the analysis pipeline. For example, contributions have addressed issues such as nanopore signal amplification [14], alignment [15], [16], variant calling [17], and DNA error correction [18]. Thus, we focus in this work on accelerating the basecalling task as it was found to be less studied in the literature in the context of specialized hardware. In particular, this paper strives to bring the basecalling method proposed in [19] into reality by investigating the practical computing challenges of its implementation in emerging mobile DNA sequencing applications. The motivation of this work, in addition to its novelty, pertains to the significant potential for the basecalling accuracy demonstrated by the method in [19] which has reached 98%.

The paper is organized as follows, Section II outlines the manner of the signals available from nanopore sensors; Section III outlines the basecalling detection method considered in this work; Section IV describes the CPU/FPGA acceleration system design while Section V reports on the system's measured performance followed by a concluding discussion in Section VI.

II. Nanopore Signals

A vast number of nanopore sensors for DNA identification have been discussed in the literature. A common trait among them is their limited fidelity, these sensors cannot directly resolve DNA to the level of its individual base constituents. Rather, the electrochemical current signals emerging from nanopores tend to reflect the interaction of groups of k bases (i.e. k-mer) with the pore at-a-time. As a result, rather than outputting four levels, each indicative of a different base, a nanopore signal may express 4^k signal levels.

A specific example of one such sensor, and the reference design for the detection hardware discussed in this paper, is a solid-state nanopore exhibiting a 3-mer sensitivity [19]; the simulated current characteristics of this nanopore are shown in Fig. 2. The plot depicts the average current levels, μ_j, expected for any 3-mer base group – a *state* to which we ascribe the integer label $j \in \{0, 1, \ldots, 4^3-1\}$ – that happens to be interacting with the pore at a given moment, $i \in \mathbb{N}$. Actual measured values (*events*) from the sensor are $x[i] = x_i = \mu_j + n[i]$, where $n[i]$ is a noise term (due to the sensor and ensuing signal conditioning electronics). In Fig. 2 the states are arranged in lexical order from AAA (state, $j = 0$) to TTT (state, $j = 63$).

In Fig. 2 the nanopore's 64 possible outputs are distributed, from $\max(\mu_j)$ to $\min(\mu_j)$, over a current range of about 280 pA. This is equivalent to an average separation of only 4.5 pA between events, a very small headroom considering the potential noise contributions to which such a signal may be subject before entering the basecaller.

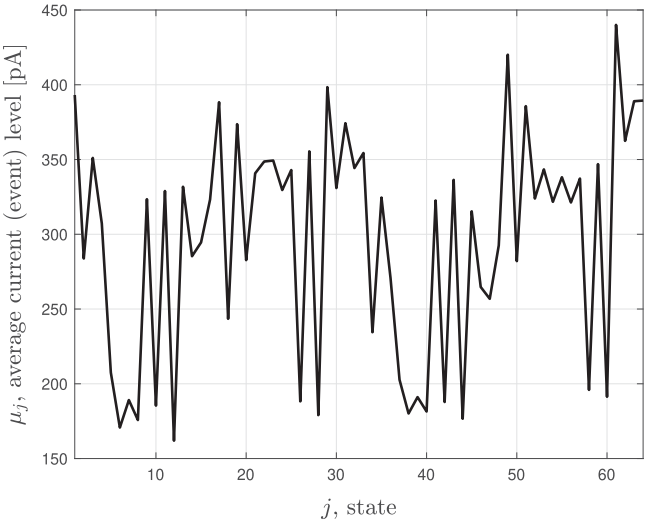

Fig. 2. 3-mer-to-current characteristics of an example nanopore. The 3-mer state indexes ($j \in \{0, 1, \ldots, 63\}$) on the abscissa are arranged in lexical order.

Besides having to deal with a corrupted amplitude, it is also likely that the basecaller will encounter an input signal expressing a degree of temporal jitter. In cases with no jitter, adjacent events in time, x_i and x_{i+1}, represent states corresponding to 3-mers differing by only one base, that is, a simple *step* 'transition'. For example, normally we expect to advance from the observation of say $x_i \sim$ CAT to $x_{i+1} \sim$ ATβ where $\beta \in \{$A, C, G, T$\}$ denotes the new base to enter the pore at $i + 1$ (while the leading/prefix C in present for x_i exits the pore).

But it may be possible, due to a pause in the molecule's translocation through the pore, that exactly the same 3-mer is registered at i and $i + 1$; this is a *stay* transition. It is also possible that at least one base was not registered at all between adjacent events, a miss indicative of a *skip* transition.

Next, we outline a basecalling detection scheme to deal with signals of the characteristics sketched above. This is followed by a discussion of its realization FPGA hardware accelerator form.

III. Basecalling Detection

Above we highlighted the coarse nature, in both amplitude and time, of the nanopore's output signal characteristics. Consequently, the basecaller must employ a form of *sequence detection* [20] to articulate the DNA bases ultimately encoded by the N-event time-series $\boldsymbol{x} = (x_i)_{i=0}^{N-1}$. To achieve this, we focus on methods that employ a hidden Markov model (HMM) of the sensor. Detailed discussions of HMMs for such problems, from a bioinformatics perspective, are given in [21]. Consequently, in the following, we only sketch the essence of the approach (Algorithm 1 provides pseudocode details).

Based on the sensor HMM and on $\boldsymbol{x}$, the basecaller computes $64 \times N$ 'emission' probabilities, $\epsilon_j(x_i)\ \forall\ (j, i)$, associated with the states that the nanopore can possibly exhibit over N event

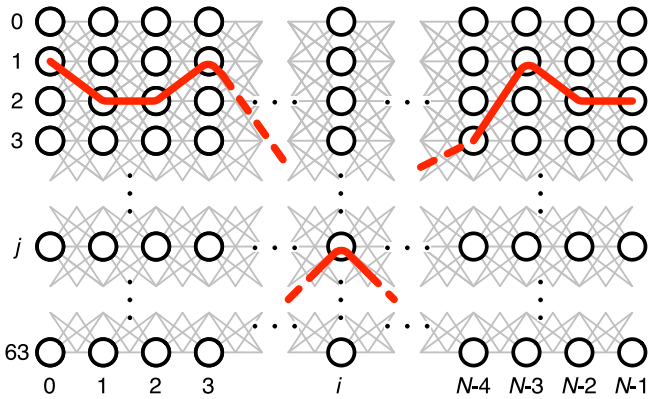

Fig. 3. Trellis representation of the basecaller's computational activity.

Algorithm 1: HMM Basecalling Algorithm.

1: *Initialize (on FPGA):*
2: $\alpha_0(j) \leftarrow \epsilon_j(x_0)\ \forall\ j \in \{0,\ldots,63\}$
3: *Iterate (on FPGA):*
4: **for**: $i \leftarrow 1, N-1$ & $\forall\ j$ **do**
5: $\quad \alpha_i(j) \leftarrow \epsilon_j(x_i) \max_{\nu\in\boldsymbol{\omega}(j)}[\alpha_{i-1}(\nu)\tau(\nu,j)]$
6: $\quad \beta_i(j) \leftarrow \arg\max_{\nu\in\boldsymbol{\omega}(j)}[\alpha_{i-1}(\nu)\tau(\nu,j)]$
7: **end for**
8: *EndState (on FPGA):*
9: $\pi^*_{N-1} \leftarrow \arg\max_j[\alpha_{N-1}(j)]$
10: $\hat{a}_{N-1} \leftarrow \pi^*_{N-1}$ & 3
11: *Traceback (on CPU):*
12 **for** $i \leftarrow N-1$ **to** 1 **do**
13: $\quad \pi^*_{i-1} \leftarrow \beta_i(\pi^*_i)$
14: $\quad \hat{a}_{i-1} \leftarrow \pi^*_{i-1}$ & 3
15: **end for**

observations. These calculations are represented by the trellis diagram in Fig. 3 where the circles denote (in part) the aforementioned state emission probability calculations.

The HMM also specifies the 'transition' probabilities, $\tau(j_{i-1}, j_i)$, (grey lines in Fig. 3) that quantify the probability of nanopore states at $i-1$ evolving to states at i. Generally, any state at i may have 64 such connections within the trellis; as we note below however, in our HMM, only select sets $\boldsymbol{\omega}(j)$, each unique to their corresponding state j, of 21 such transitions are accounted for.

Using an iterative dynamic programming technique (the Viterbi algorithm), the basecaller utilizes its emission and transition values to compute the most likely sequence of states through the trellis, the path $\boldsymbol{\pi}^* = (\pi^*_i)_{i=0}^{N-1}$ (represented by the red line in Fig. 3) and thus the most likely sequence, $\hat{\boldsymbol{a}} = (\hat{a}_i)_{i=0}^{N-1}$, of bases associated with the measured time-series.

The pseudocode listed in Algorithm 1 provides a succinct description of the basecaller's computational process, we highlight its pertinent components now.

The *Iterate* block comprises a significant part of the basecaller and is the focus of our accelerator hardware. This block is as nested for-loop sweeping across all 64 possible state calculations for all N events being measured. Therein, the computation in line 5 combines the emission, $\epsilon_j(x_i)$, and transition, $\tau(\nu, j)$, probabilities (we elaborate on these terms shortly) to update a running measure of each state's *posterior* $\alpha_i(j)$; formally the basecaller's main computation results in estimates of the maximum likelihood for each state after N input observations:

$$\alpha_N(j) = \alpha_0(j) \prod_{i=1}^{N-1} \epsilon_j(x_i)\tau(\nu, j)|_{\nu\in\boldsymbol{\omega}(j)}. \tag{1}$$

In concert (line 6), a *pointer*, $\beta_i(j) \in \{0,\ldots,63\}$, is computed; $\beta_i(j)$ denotes the state at event $i-1$ most likely to have transitioned into state j at event i. $\beta_i(j)$ is drawn, as noted above, from a set $\boldsymbol{\omega}(j)$ of 21 possible predecessors to j (elaboration below).

Keeping track of the most likely preceding steps allows the *Traceback* block to sequentially consult the accumulated pointers and to compute the most likely sequence of states, $\boldsymbol{\pi}^*$, and bases $\hat{\boldsymbol{a}}$ thus completing the basecalling effort.

To complete our sketch, we highlight some details related to the emission and transition terms. The emission characteristics relate the probability that an event x_i may be associated with any of the expected outputs μ_j. This is usually depicted according to the Gaussian distribution as

$$\epsilon_j(x_i) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp[(x_i - \mu_j)^2/2\sigma^2], \tag{2}$$

where σ denotes the standard deviation of the observations around the anticipated levels, a measure of the noise in the system.

The term $\tau(\nu, j)$ denotes the probability that a state, $\nu \in \{0, 1, \ldots, 4^3-1\}$, transitions to a state j. As noted above, for each j, we account for 21 such transitions; one for a transition via a stay mechanism where $\nu = j$; another four that account for four possible step transitions,

$$\nu = 16{\cdot}l + \lfloor j/4 \rfloor,$$

with $l \in \{0, 1, 2, 3\}$; and another 16 that account for 16 possible skip transitions over one base,

$$\nu = 4{\cdot}L + \lfloor j/16 \rfloor,$$

where $L \in \{0,\ldots,15\}$. Thus the set of states preceding j, $\nu \in \boldsymbol{\omega}(j)$, consists of $1 + 4 + 16 = 21 = |\boldsymbol{\omega}(j)|\ \forall\ j$.

IV. Basecalling Accelerator Design

The inner-loop of Algorithm 1 (i.e. lines 5 and 6 iterated over 64 states j) constitutes the basecaller's most compute-intensive basic block. Thus, we propose an FPGA-based computing engine, depicted in Fig. 4, to efficiently accelerate that loop. The calculation of both ϵ_j (implicit in line 5) and $\alpha_i(j)$ for the 64 states, possesses no loop-carried dependencies, and thus, offers a direct path to speed-up via explicit parallel hardware. The recurrence relationship, $\alpha_i \propto \alpha_{i-1}$, imposes a bottleneck that may nonetheless be mitigated with judicious use of pipelining and reduction operations as described below.

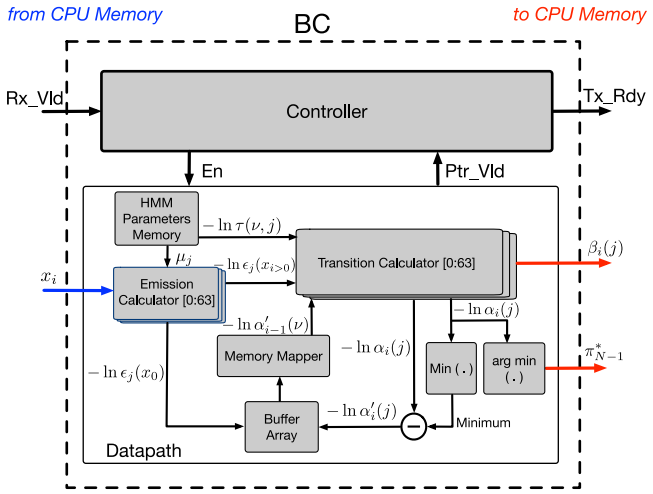

Fig. 4. FPGA architecture of the proposed basecalling acceleration engine.

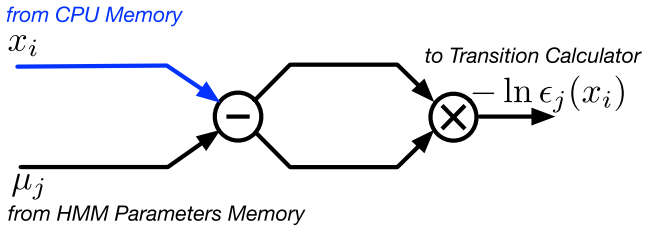

Fig. 5. The arithmetic arrangement of a single slice from the emission calculator.

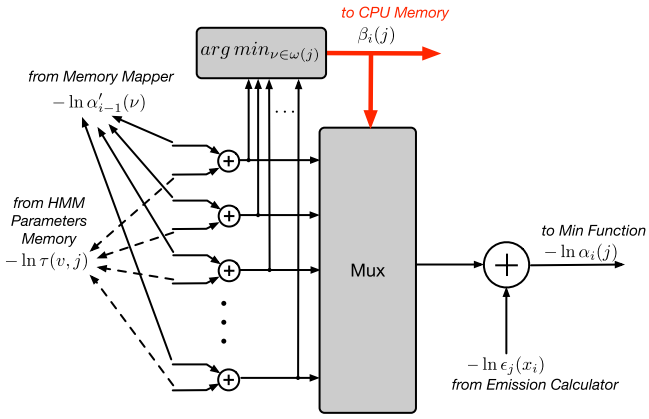

Fig. 6. The architecture of single slice from the Transition Calculator.

A. Basecalling Engine

The block diagram shown in Fig. 4 represents the general architecture of our proposed FPGA basecalling acceleration engine, in essence an implementation of the 64-step inner-loop (lines 5 and 6) of Algorithm 1. The engine receives its input, the event x_i, from the CPU's main memory and sends its output, the event pointers $\beta_i(j)$ and the end-state index π^*_{N-1}, back to the CPU's main memory in a real-time streaming fashion through the PCIe bus. More details about this CPU-FPGA communication interface are provided in the following subsection.

A couple of design choices immediately occur for this system. First, as previously mentioned and also detailed below, to accommodate the level of independence present across the 64 iterations of states j, we employ a fully-unrolled loop-level parallel hardware implementation. Thus, as illustrated in Fig. 4, 64 parallel *emission* and *posterior* calculation blocks ("slices") comprise the datapath and simultaneously compute the 64 pointers $\beta_i(j)$ corresponding to each new event input x_i. These slices carry out the majority of the design's arithmetic.

A second important consideration, directed at improved computational speed and efficiency, is to simplify the arithmetic computations of $\epsilon_j(x_i)$ and $\alpha_i(j)$ themselves. In this vein, following standard practice [22], the basecalling engine computes the negative logarithm of the underlying probabilities rather than the exact probabilities themselves. This requires that the max operation be replaced with a min and, most importantly, it reduces (2) to

$$-\ln\epsilon_j(x_i) = (x_i - \mu_j)^2 \tag{3}$$

where the terms, $\ln(2\pi)$ and $1/2\sigma^2$, have been removed as they serve only as constant offsets to the calculations, without any contribution to the relative probabilities of the states upon which the choice of a final optimal sequence relies. Thus, the FPGA block for a single state emission calculation is comprised of a 2-input subtractor and a 2-input multiplier as portrayed by Fig. 5. It should be noted that the arithmetic logic in Fig. 5, as well as in Fig. 6, is pipelined by placing registers in between consecutive logic stages. For simplicity, the provided figures do not include those pipelining registers and other detailed logic of the actual hardware synthesised by our design tools.

Similarly, the hardware-computed logarithmic posterior and pointer updates become

$$\ln\alpha_i(j) \leftarrow \ln\epsilon_j(x_i) + \min_{\nu\in\boldsymbol{\omega}(j)}[\ln\alpha_{i-1}(\nu) + \ln\tau(\nu,j)] \tag{4}$$

$$\beta_i(j) \leftarrow \arg\min_{\nu\in\boldsymbol{\omega}(j)}[\ln\alpha_{i-1}(\nu) + \ln\tau(\nu,j)]. \tag{5}$$

The function of each building block for the proposed engine architecture in Fig. 4 is as follows.

- The *Controller* block maintains the data flow configuration and timing for all other components of the engine. In particular, it signals enable permissions for each block to indicate the start and end of operation cycles, and receives valid status signals indicating elapsed calculations for event pointers and the end-state pointer. It also manages the engine's external communication with the CPU, relying on valid and ready signals to register the arrival of new input events and to indicate that the engine is done calculating a new set of 64 pointer values, respectively.
- The *Transition Calculator* block is the engine's main processing chain used to compute the 64 posteriors and pointers for each new incoming event x_i using a parallel structure. Each slice of the Transition Calculator handles the computations in (4) and (5) for each new event. Fig. 6 shows the detailed structure of each slice instance which is composed of twenty one 2-input adders, $\arg\min$ function,

multiplexer, and 2-input adder. The twenty one adders are responsible for adding $\ln \alpha_{i-1}(\nu)$ to $\ln \tau(\nu, j)$ as in (4). The $\arg\min$ function and the multiplexer are then used to find the state pointer $\beta_i(j)$ as in (5) and the minimum value as in (4), respectively. More specifically, the $\arg\min$ function is implemented as a 21-input comparator circuit which outputs the index of the minimum value of the 21 input values coming from the adder circuits. The last adder finalizes the addition of $\ln \epsilon_j(x_i)$ to the mux output and calculates $\ln \alpha_i(j)$ as in (4).

- The *HMM Parameters Memory* is a memory block which stores the HMM's transition and emission models' parameters (i.e. $\ln \tau(\nu, j)$ and μ_j, respectively).
- The *Emission Calculator* block calculates the negative logarithm of the 64 emission probabilities $-\ln \epsilon_j(x_i)$ for each new input event x_i according to (3) and Fig. 5. Similar to the Transition Calculator block, the 64 emission calculations are executed in a parallel fashion using 64 identical hardware instances (slices) based on the emission model parameters (i.e. μ_j) stored in the *HMM Parameters Memory* block.
- The *Min* block ensures that the engine's recursive computations across the succession of input events do not experience overflow, a functionality not described in the Algorithm 1 pseudocode. Thus, the Min block finds the minimum of the 64 posteriors calculated by the Transition Calculator block for each event. The minimum value is then subtracted from the event posteriors before storing the result (i.e. $\ln \alpha'_i(j)$) in the buffer array.
- The *arg min* block outputs the end-state index (i.e. π^*_{N-1}) for the last event as in line 9 of Algorithm 1. In particular, the $\arg\min$ block is implemented as a 64-input comparator circuit which outputs the index of the minimum value of the 64 posterior values coming from the Transition Calculator block at the end of processing event x_N.
- The *Buffer Array* block is used to store the normalized posteriors for each event (i.e. $\ln \alpha'_i(j)$) in preparation for the iteration triggered by the arrival of the next event x_i.
- The *Memory Mapper* block is used to allocate the posterior set, $\boldsymbol{\omega}(j)$, designated to each instance of the transition calculator block based on the unique 21 transition states to its state index j. Specifically, the memory mapper is a crossbar switch (with fixed interconnect links) which maps the 21 posterior values to each state slice based on its index *j*. For each state slice instance, the 21-posteriors set is fetched from the 64 normalized posteriors calculated for the previous event and stored in the buffer array.

For the first event ($i = 0$) only the 64 emission values (i.e. $-\ln \epsilon_j(x_o)$) need to be calculated (since all preceding values are assumed to be zero). Thus, the engine's controller puts the Transition Calculator in an idle state for the first input event and only the Emission Calculator block is allowed to initialize the Buffer Array.

For the second and all other subsequent events, the engine operates in the same manner as follows. The HMM Parameters Memory, Emission Calculator and the Memory Mapper provide the Transition Calculator with the logarithm of transition probabilities, the current event's emission probabilities, and the previous event's posterior probabilities. Based on these three inputs, the transition calculator evaluates the current event posteriors and pointers according to (4) and (5), respectively. Only the pointers are sent to the CPU through the PCIe interface (as discussed in the following subsection). Once the engine reaches the last event ($i = N - 1$), the end-state index (i.e. the $\arg\min$ function's output) is sent to the CPU with the pointers (i.e. equivalent to line 9).

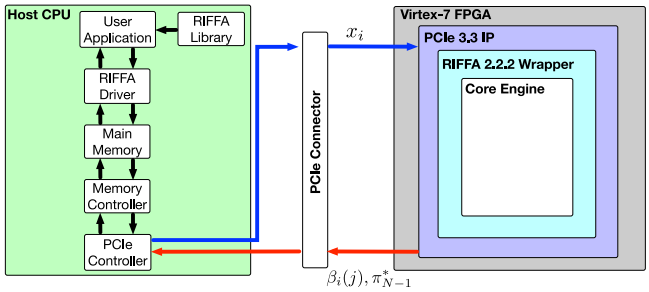

Fig. 7. High-level architecture for the CPU-FPGA hardware interface.

B. CPU-FPGA Communication Interface

In this subsection we describe how the proposed FPGA acceleration engine illustrated in Fig. 4 is linked to and coordinated with a CPU. This interface allows our proposed FPGA accelerator to serve as a rapid real-time co-processor for a CPU basecaller and thus allows it to fit conveniently within a larger DNA sequencing pipeline. To simplify the understanding of this communication framework, Fig. 7 shows the high-level architecture of the CPU-FPGA communication interface. The proposed core acceleration engine (i.e. discussed in the previous subsection) is implemented on a Virtex-7 FPGA device whereas the basecaller's main C-program is running on a host CPU.

On the FPGA (hardware) side, our proposed core engine is wrapped by the Reusable Integration Framework for FPGA Accelerators (RIFFA) [23]. RIFFA is an open-source collection of software libraries and hardware designs to enable real-time streaming between CPUs and FPGAs through PCIe. In particular, RIFFA allows stitching our core engine to a PCIe endpoint (i.e. Xilinx PCIe IP) using so-called Rx and Tx engines which receive and generate PCIe packets, respectively. The RIFFA packet size may be configured by the user to 32-bits, 64-bits or 128-bits.

On the CPU (software) side, the HMM basecaller (i.e. the user application) described in Algorithm 1 is implemented using C. As discussed in the previous subsection, this code offloads the basecaller's computation-intensive loop in lines 5 and 6 of Algorithm 1 to our proposed acceleration engine. This takes place with the aid of a simple RIFFA API that includes send and receive functions that are inserted by a developer directly in a C program to facilitate data exchange between the CPU and hardware-mapped kernels (i.e. `fpga_send()` for CPU transmission to the FPGA, and `fpga_recv()` for CPU reception from the FPGA). The RIFFA driver then allows these API

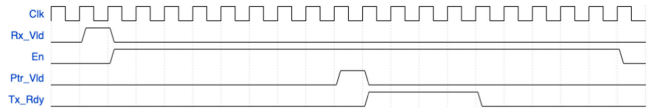

Fig. 8. Simplified timing diagram for RIFFA interface/core engine handshaking protocol.

functions to access the CPU's main memory. The CPU's PCIe controller then allows communicating the CPU's main memory data via the memory controller in the form of PCIe packets. As previously illustrated in Fig. 4, the data sent from the CPU to the FPGA is the sequence of nanopore events x_i whereas the FPGA sends back 64 pointers $\beta_i(j)$ for each event as well as the *end-state index* of the last event (i.e. π^*_{N-1}) to the CPU.

The interaction between the proposed acceleration engine and the RIFFA interface signals can be illustrated with the aid of Fig. 8. This simplified timing diagram provides a high-level description for the processing of a single event (which repeats until the last event). First, the CPU-side basecalling C program, with the aid of the RIFFA driver, packs the N events in a sequence of 128-bit event message packets. Each event is alloted a 32-bit word, thus, each message packet carries four events. When the C program hosted by the CPU invokes the `fpga_send()` function to send a message packet to the FPGA over the PCIe bus, the arrival of the packet is signified by the status signal `Rx_Vld`.

Once the valid signal is received by the basecalling engine's Controller, it enables the appropriate datapath blocks for processing the first event in the received message packet using the `En` signal for 18 clock cycles (i.e. the overall latency of the data path blocks for processing a single event).

As soon as the *arg min* block has the 64 states' pointers calculated, the Transition Calculator block sends the `Ptr_Vld` signal to the Controller. The Controller then arranges the 64 calculated pointers (each allotted 1 byte) into four 128-bit pointer message packets, queues these message packets on the core engine's output data port, and asserts a ready signal, `Tx_Rdy` informing the remainder of the transmission-side IP that data intended for the CPU is ready to be sent. Four cycles of the basecalling engine's clock are needed to send the four pointer messages. The processing of the remaining three input events in the received message packet is handled in the same fashion before receiving the following packet's valid signal.

For the first event the Controller does not signal `Tx_Rdy` since no pointers are calculated for the first event. On the other hand, at the last event the Controller signals `Tx_Rdy` for five clock cycles (instead of four) to send one extra RIFFA packet carrying the end-state index (i.e. π^*_{N-1}) to the CPU.

V. Implementation Results

In this section, the performance of our proposed FPGA-accelerated basecalling engine (Section IV) for Timp's basecaller in [19] is evaluated in terms of speed and power consumption for different accuracy levels. The performance of our proposed basecaller is evaluated comparatively with a traditional CPU-based implementation (i.e. coded in C). The CPU used in our testing platform is the 12-core Intel Xeon E5-2620 v3 clocked at 2.4 GHz with a 32 GB of RAM. The proposed basecalling engine is designed and implemented using Xilinx Vivado HLx v2016.1 tools. Our target FPGA device is a Virtex-7 (i.e. XC7VX485T-2FFG1761 C) which contains 75,900 slices each of which has 4 look-up tables (LUTs) and 8 flip-flops (FFs).

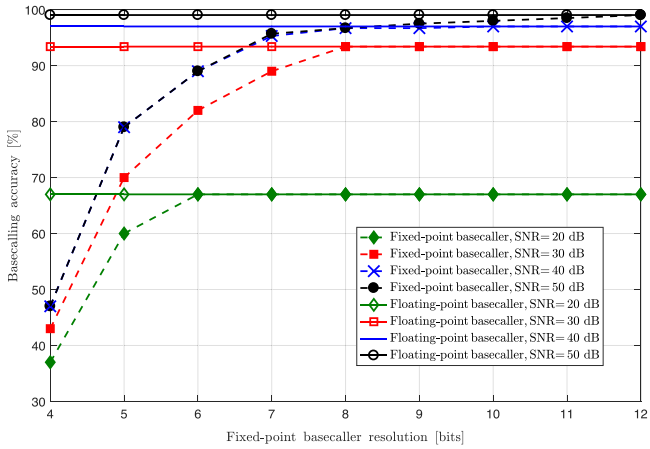

Fig. 9. Fixed-point basecaller accuracy as a function of bit resolution under different input SNR settings.

To guide our FPGA implementation, we built a bit-true MATLAB system-level simulator to evaluate the fixed-point basecaller's prediction accuracy performance at different bit-widths. To account for possible noise effects in the measurement system we examined the basecaller's behaviour at different input signal-to-noise ratios (SNRs) from the sensor channel. In addition to the sensor's additive noise, our HMM basecaller's accuracy depends on its ability to mitigate the stay and skip translocation mechanisms discussed above. Lacking detailed reports on this issue we have assumed a 10% stay probability and a 10% skip probability as exemplary values for our hardware analysis.

The simulation results depicted in Fig. 9 show how the accuracy of the fixed-point basecaller changes with the bit-width at different SNR values for the sensor channel. At each SNR, the accuracy is measured with respect to a floating-point equivalent. As expected, the accuracy for the fixed-point basecaller increases with increasing bit-width until it saturates at the value attained by its floating-point counterpart.

From another perspective, the results show that the bit-width value required to saturate the fixed-point basecaller accuracy increases with increasing SNR (e.g. 6-bits for 20 dB, 8-bits for 30 dB, etc). This could be justified by the dominance of the sensor channel SNR over the bit-width in determining the fixed-point basecaller's accuracy. In particular, in case of a highly corrupted sensor signal (i.e. low SNR) the ability of the fixed-point basecaller to make accurate predictions of the original DNA sequence is very low due to the noise distortion level regardless of its computational resolution (bit-width). That is why, for instance, at 20-dB SNR the accuracy (i.e. 67%) at a resolution of 6-bits is the same as that for 12-bits. On

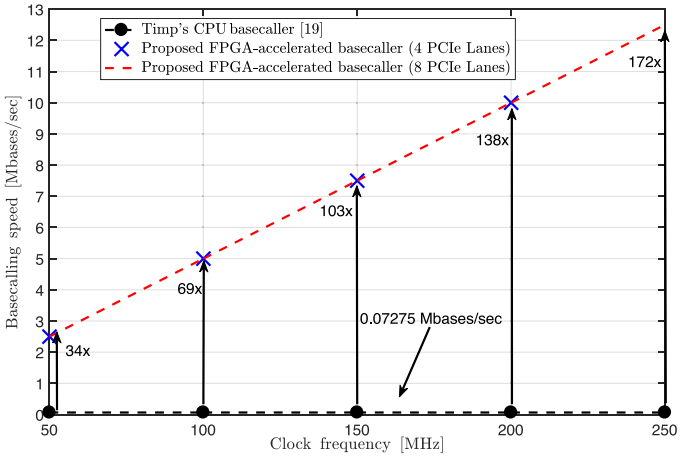

Fig. 10. Measured FPGA-accelerated and CPU-only basecalling speed for Timp's basecaller [19] with the FPGA clock frequency. Numbers denote the relative speed improvement of the FPGA-accelerated design relative to the CPU alone.

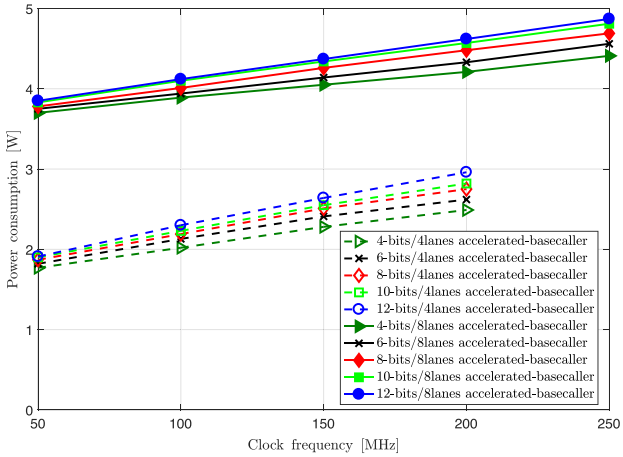

Fig. 11. FPGA-accelerated basecalling power consumption (FPGA only).

the other hand, in case of a low-noise signal (i.e. high SNR signal), the major limiting factor for the fixed-point basecaller's prediction accuracy is its resolution. This could be seen in the 50-dB SNR case at which the basecaller's resolution must be 12-bits to match the floating-point accuracy of 99.04%. It is worth mentioning here that besides the sensor additive noise and the basecaller's resolution, the skip and stay state transitions are still an additional source of discrepancy for the sensor signal and limit the basecaller's accuracy, a deep study of these effects is beyond the paper's scope.

The results presented in Fig. 10 demonstrate a dramatic improvement in the basecalling speed for our proposed FPGA-accelerated basecalling engine compared to a CPU-only implementation thanks to the parallel computing which may be realized in FPGA devices. As expected, the basecalling speed improvement tracks the clock frequency. Our register-transfer language (RTL) model for the basecaller was successfully synthesized and implemented on our target FPGA device at a clock frequency of 250 MHz when using 8 lanes of the Xilinx PCIe 3.3 IP. At this frequency the FPGA-accelerated basecalling speed exceeds the CPU-only implementation by a factor of 172×. In addition, it could be observed that despite the accelerated basecaller attaining similar speeds on both the 4-lane and 8-lane PCIe configurations (i.e. the bandwidth of the 4-lane PCIe channel is accommodating our basecalling engine bit rate), only the 8-lane interface-based basecaller could be implemented at a 250-MHz clock frequency. This pertains to the higher AXI clock frequency (i.e. used to generate our basecalling engine clock) of the 8-lane interface (250 MHz) compared to that for the 4-lane (125 MHz). However, the higher maximum clock frequency attained by the 8-lane interface compared to the 4-lane comes at the expense of the circuit power consumption as will be shown in the following paragraphs.

It should be noted that the 172× speed-up in favor of our proposed accelerator compared to the CPU is due to the parallel computing for three major parts of Algorithm 1. The first part is unrolling the inner loop in lines 5 and 6 for the 64 hidden states of the HMM. The second part is the 21 transition additions in (4) executed for each of the 64 states' calculations. The third part is the normalization subtraction which is executed in parallel on our FPGA accelerator for the 64 posteriors output from the Transition Calculator block (in Fig. 4) at the end of the processing cycle of each input event. Therefore, the aforementioned parallel computations lead to a speed-up factor of 147× (i.e. (21 × 64 + 64)/(2.4 GHz/250 MHz)) for our 250 MHz clocked FPGA compared to the 2.4 GHz CPU. The residual speed-up factor is attributable to other PC computer architecture-related factors (e.g. cache misses, procedure calls, multi-core communications, etc.).

The practical significance of the results reported in Fig. 10, should be considered with respect to the corresponding power consumption and hardware utilization costs. This will better enable the system designer (or even the application layer user) to select the most efficient configuration for the whole system to meet the target application requirements (e.g. energy-efficiency).

We start by considering the measured average operational power consumption of the FPGA accelerator in Fig. 11 as a function of clock frequency. Both the 4 and 8-lane configurations consume on average 8% and 16%, respectively, of the CPU's average "incremental" power, 25.16 W (i.e. the extra power consumed *while* the CPU executes the basecalling algorithm, otherwise the PC's total consumed power is 87.15 W). For further illustration, in Fig. 12, the results of Fig. 10 and Fig. 11 are utilized to demonstrate the energy efficiency (bases computed per joule) of our proposed accelerator compared to the CPU. Fig. 12 confirms that our proposed accelerator is more energy-efficient than the CPU by at least 225× for the 8-lane design at 50 MHz (and 1390× for the 4-bit design with 4 lanes at 200 MHz). The relative energy-efficiency improvement of our proposed accelerator with increasing clock frequency reflects its consistent linear increase in basecalling speed (as shown in Fig. 10) relative to a marginal increase in the corresponding power consumption (as shown in Fig. 11). The power-frequency

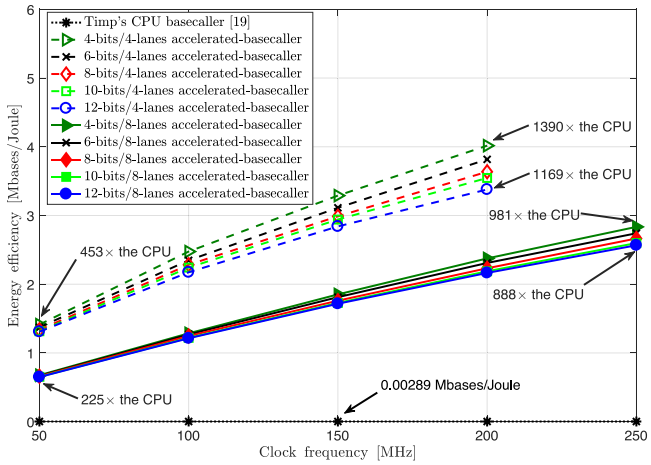

Fig. 12. Basecalling energy-efficiency of the FPGA and CPU, numeric labels denote the relative efficiency improvement of the FPGA.

change ratio demonstrated in Fig. 11 is $\sim 1:3.5$ (rather than the theoretical $1:1$). This pertains to the fact that the power consumption reported in Fig. 11 encapsulates not only that of the core basecalling engine, but also the PCIe and RIFFA IP while the abscissa refers only to the basecalling engine's clock frequency. Thus, increasing the clock frequency only increases the power consumption of the basecalling engine while the power consumed by the communications IP portions remains invariant. Moreover, the power of the basecalling engine occupies an average of 8.25% of the total power consumption budget of our accelerator circuit reported in Fig. 11. Due to these facts, the marginal increase in the power consumption with the clock frequency illustrated in Fig. 11 could be understood.

It should be noted that the PCIe IP and RIFFA IP in our proposed accelerator communicates with a different and independent clock frequency (i.e. AXI interface clock) to the basecalling clock frequency (i.e. used in Figs. 10, 11, 12). Hence, the divergence of the EE lines when increasing the basecalling clock frequency is notable in Fig. 12 especially for the 4-lanes curves for different circuit resolution. This stems from the fact that at low basecalling clock frequencies the dynamic power consumed by our core basecalling engine is still not big enough compared to the bigger constant power (i.e. dependant on the constant AXI clock) consumed by the PCIe and RIFFA logic elements to show a reasonable difference in the total power consumed by the whole circuit. However, at higher basecalling clock frequencies, the dynamic power of our core basecalling engine starts to show more impact as it becomes relatively closer to the constant power component for the PCIe and RIFFA IPs.

In addition to the power consumption, the hardware utilization and the energy density (ED) (i.e. elaborated below) of our proposed basecalling engine on the target Virtex-7 FPGA device is illustrated in Table I. The table shows the resource utilization percentages for the LUTs, FFs and DSP slices with different complexities (i.e. resolution) of our basecalling circuit. As expected, resource usage increases with bit-width. Table I does not show a substantial difference between using 4-PCIe lanes and 8-PCIe lanes. However, 14% of the Gigabit Transceiver (GTX) resource (i.e. not reported in the table) is utilized in

TABLE I
FPGA DEVICE UTILIZATION AND ENERGY DENSITY

	4-PCIe Lanes				8-PCIe Lanes			
	LUT	FF	DSP	ED	LUT	FF	DSP	ED
4-bits	12%	6%	53%	1.47	12%	6%	53%	1.76
6-bits	14%	7%	55%	1.66	15%	7%	55%	1.95
8-bits	17%	8%	55%	1.72	18%	8%	55%	2.0
10-bits	20%	9%	57%	1.87	21%	9%	57%	2.05
12-bits	23%	10%	59%	1.91	24%	11%	59%	2.15

case of the 4-lanes design (i.e. using 4 transceivers out of 28) compared to 29% for the 8-lanes (i.e. using 8 transceivers out of 28). It is worth mentioning that the aforementioned GTX resource utilization for the 4 and 8-lane designs directly justifies the power offset between both designs as noticed in Fig. 11. In addition to the resource utilization, Table I also shows the ED at the maximum clock frequency (i.e. 200 MHz in case of 4-lanes interface and 250 MHz in case of 8-lanes interface) for each design resolution. The basecaller is assumed to decode a long genome with a length of 1 Gbp. The ED metric (i.e. measured in Joules/slice) is another important metric which determines the energy efficiency of the utilized FPGA resources by combining the actual number of the utilized FPGA slices and its corresponding energy consumption. All the ED values reported in Table I are in mJoules/slice. The increase in the ED for both 4 and 8 lanes accelerators with increasing resolution shows consistency with the results in Fig. 12 which conveys that the lower the basecaller's resolution the higher the energy efficiency.

Finally, it is worth mentioning that the utilization of the proposed FPGA accelerator in the HMM basecalling function in this work does not limit its applicability to other HMM-based algorithms such as Pair HMM [17] and Profile HMM [24]. In case of Pair HMM, our accelerator can be efficiently utilized to accelerate the initialization and recursion sections of the algorithm similar to the initialize and iterate blocks, respectively, of our Algorithm 1. On the hand, our accelerator can also speed-up the database search for the Profile HMM algorithm by implementing parallel instances of accelerated Pair HMMs to rapidly identify the sequence family of an arbitrary DNA sequence.

VI. SUMMARY AND CONCLUSION

In this paper, we present an FPGA hardware accelerator for the detection of nanopore signals. In the context of such sensors being used for DNA sequencing, this detection step is commonly referred to as basecalling, the identification of the base make-up of a measured DNA molecule. Although extremely promising as a sensory device for mobile molecular detection, the coarse and jitter-prone nature of the nanopore signal requires a sequence detection algorithm capable of accounting for channel memory while accommodating for null and redundant transitions. Our approach employed a maximum-likelihood scheme to achieve this, with the majority of its computational load mapped onto the FPGA. This effort is intended to demonstrate the potential of specialized computing in this aspect of bioinformatics and

sensing and to potentially suggest means by which low-power implementations suitable for mobile molecular measurement scenarios may be realized.

The nature of the algorithm we employed accommodates a hardware implementation in terms of two synchronized 64-element parallel arrays (one for emission, another for transition calculations) capable of handling all sensor states. Inter-element communication within the device facilitates the sharing of information across the algorithm's iterations. On-FPGA normalization hardware prevents underflow, thus making calculations entirely self-contained and capable of sustaining continuous data inputs. The need for excessive on-FPGA memory resources are avoided by offloading memory-heavy functions to the host CPU. The memory-heavy functions reside in the traceback block of Algorithm 1 which traverse the FPGA-calculated pointers (for all events) in the CPU memory to find the optimal path of the nanopore states which leads to the equivalent DNA bases sequence. As a result, the core basecalling algorithm's resource utilization peaks at 60% (for a Virtex-7 device with $\sim$500 k logic cells and $\sim$3 k DSP slices).

In our implementation, the CPU-accelerator communication interface is facilitated with a PCIe link which allows a streaming data exchange between the CPU and its accelerator in terms of 128-bit information packets; as new measured events are sent to the FPGA, computations are aggregated and returned to the CPU. In this manner, a continuous flow of measurements can be managed. At its peak (clocked at 250-MHz) the accelerator system's measured throughput is equivalent to the equivalent of one human genome in less than 5 minutes, a 172$\times$ rate improvement over the CPU-only basecaller. Perhaps even more importantly, this performance comes at a substantial savings in power, only 16% of the CPU's average power used for the execution of the algorithm alone. As a result, the FPGA-accelerated system achieves a measured energy efficiency (i.e. bases called per Joule) about 1000$\times$ better than the CPU alone.

These results portend very well for the potential of the approach described here. The low memory requirements suggest opportunities for ASIC accelerator implementations and the possibilities of even deeper cost-performance advantages.

REFERENCES

[1] F. Sanger and A. R. Coulson, "A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase," *J. Mol. Biol.*, vol. 94, no. 3, pp. 441–448, May 1975.

[2] E. A. Ashley, "Towards precision medicine," *Nature Rev. Genetics*, vol. 17, no. 9, pp. 507–522, Sep. 2016.

[3] L. Jameson and D. L. Longo, "Precision medicine—Personalized, problematic, and promising," *Obstetr. Gynecol. Surv.*, vol. 70, no. 10, pp. 612–614, Oct. 2015.

[4] Y. Wu, C. Chang, J. Hung, and C. Yang, "A 135-mW fully integrated data processor for next-generation sequencing," *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 6, pp. 1216–1225, Dec. 2017.

[5] Z. Beiki and A. Jahanian, "DENA: A configurable microarchitecture and design flow for biomedical DNA-based logic design," *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 5, pp. 1077–1086, Oct. 2017.

[6] M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The Oxford nanopore MinION: Delivery of nanopore sequencing to the genomics community," *Genome Biol.*, vol. 17, no. 1, 2016, Art. no. 239.

[7] F. D. Guzel and H. Avci, "Fabrication of nanopores in an ultra-thin polyimide membrane for biomolecule sensing," *IEEE Sensors J.*, vol. 18, no. 7, pp. 2641–2646, Apr. 2018.

[8] Z. Wu, K. Hammad, R. Mittmann, S. Magierowski, E. Ghafar-Zadeh, and X. Zhong, "FPGA-based DNA basecalling hardware acceleration," in *Proc. IEEE 61st Int. Midwest Symp. Circuits Syst.*, Aug. 2018, pp. 1098–1101.

[9] H. Lee *et al.*, "Third-generation sequencing and the future of genomics," *bioRxiv*, 2016, doi: 10.1101/048603.

[10] C. Chien, S. Shekar, D. J. Niedzwiecki, K. L. Shepard, and M. Drndic, "Single-stranded DNA translocation recordings through solid-state nanopores on glass chips at 10 MHz measurement bandwidth," *ACS Nano*, vol. 13, no. 9, pp. 10545–10554, Aug. 2019.

[11] P. M. Ashton *et al.*, "MinION nanopore sequencing identifies the position and structure of a bacterial antibiotic resistance island," *Nature Biotechnol.*, vol. 33, no. 3, pp. 296–300, 2015.

[12] M. Jain *et al.*, "Nanopore sequencing and assembly of a human genome with ultra-long reads," *Nature Biotechnol.*, vol. 36, no. 4, pp. 338–345, 2018.

[13] D. V. Verschueren, W. Yang, and C. Dekker, "Lithography-based fabrication of nanopore arrays in freestanding SiN and graphene membranes," *Nanotechnology*, vol. 29, no. 14, Feb. 2018, Art. no. 145302.

[14] C. Hsu, H. Jiang, A. G. Venkatesh, and D. A. Hall, "A hybrid semi-digital transimpedance amplifier with noise cancellation technique for nanopore-based DNA sequencing," *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 5, pp. 652–661, Oct. 2015

[15] H. M. Waidyasooriya and M. Hariyama, "Hardware-acceleration of short-read alignment based on the burrows-wheeler transform," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1358–1372, May 2016.

[16] Y. Li and Y. Lu, "BLASTP-ACC: Parallel architecture and hardware accelerator design for BLAST-based protein sequence alignment," *IEEE Trans. Biomed. Circuits Syst.*, early access, 2019.

[17] D. Sampietro, C. Crippa, L. Di Tucci, E. Del Sozzo, and M. D. Santambrogio, "FPGA-based pairHMM forward algorithm for DNA variant calling," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Architectures Processors*, Jul. 2018, pp. 1–8.

[18] A. Ramachandran, Y. Heo, W. Hwu, J. Ma, and D. Chen, "FPGA accelerated DNA error correction," in *Proc. IEEE Des., Autom. Test Eur. Conf. Exhib.*, 2015, pp. 1371–1376.

[19] W. Timp, J. Comer, and A. Aksimentiev, "DNA base-calling from a nanopore using a Viterbi algorithm," *Biophys. J.*, vol. 102, no. 10, pp. L37–L39, May 2012.

[20] J. R. Barry, E. A. Lee, and D. G. Messerschmitt, *Digital Communication*, vol. 1, 3rd ed. Berlin, Germany: Springer, 2004.

[21] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 1998.

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge Univ. Press, 2007.

[23] M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner, "RIFFA 2.1: A reusable integration framework for FPGA accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 4, pp. 22:1–22:23, 2015.

[24] S. R. Eddy, "Accelerated profile HMM searches," *PLoS Comput. Biol.*, vol. 7, no. 10, Oct. 2011, Art. no. e1002195.

Zhongpan Wu received the B.E. degree in software engineering from Dalain Jiaotong University, Dalian, China, in 2015, and the M.Sc. degree in electrical engineering and computer science in 2019 from York University, Toronto, ON, Canada, where he is currently working toward the Ph.D. degree. He was a Research Assistant from May 2016 to 2017 with York University. His research interests include ASIC and FPGA accelerators design, computer architecture, and machine learning.

Karim Hammad received the B.Sc. and M.Sc. degrees in electronics and communications engineering from the Arab Academy for Science, Technology and Maritime Transport (AASTMT), Cairo, Egypt, in 2005 and 2009, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Western Ontario, London, ON, Canada, in 2016. He is currently an Assistant Professor with the Department of Electronics and Communications Engineering, AASTMT, and a Postdoctoral Visitor with York University, Toronto, ON, Canada. His research interests include wireless networks cross-layer design, physical layer security, and digital circuit design.

Ebrahim Ghafar-Zadeh (M'13) received the B.Sc. degree from the K. N. Toosi University of Technology, Tehran, Iran, in 1994, the M.Sc. degree from the University of Tehran, Tehran, in 1996, and the Ph.D. degree from Ecole Polytechnique, Montreal, QC, Canada, in 2008, all in electrical engineering. He continued his research, as a Postdoctoral Fellow with the Department of Electrical and Computer Engineering, McGill University, Montreal, and with the Department of Bioengineering, University of California, Berkeley, CA, USA. In 2013, he joined the Department of Electrical Engineering and Computer Sciences, York University, Toronto, ON, Canada, where he currently serves as an Associate Professor and the Director of Biologically Inspired Sensors and Actuators (BioSA) Laboratory. He has authored/coauthored more than 125 papers in various BioSA topics in the high-quality journals and international conferences. His research focuses on BioSA for cell and molecular analysis.

Sebastian Magierowski received the Ph.D. degree in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 2004. From 2004 to 2012, he served on the faculty of the Department of Electrical and Computer Engineering, University of Calgary. In 2012, he joined the Department of Electrical Engineering and Computer Science, Lassonde School of Engineering, York University, Toronto, where he is currently an Associate Professor. As part of his industrial experience (Nortel Networks, PMC-Sierra, Protolinx Corporation), he has worked on CMOS device modeling, high-speed mixed-signal IC design, and data networks. His research interests include analog/digital CMOS circuit design, communication systems, biomedical instrumentation, and signal processing for biomolecular sensing and analysis.