Performance of Parallel Prefix Circuit Transition Localization of Pulsed Waveforms

Yuanwei Fang
Unversity of Chicago
Chicago, Illinois

Andrew A. Chien
Unversity of Chicago and
Argonne National Laboratory
Chicago, Illinois

Andrew Lehane
Keysight Laboratories
Keysight Technologies, Inc.
Edinburgh, Scotland

Lee Barford
Keysight Laboratories
Keysight Technologies, Inc.
Reno, Nevada

***Abstract*—An early step in measuring jitter in communication signals is locating the transitions, the points in time when the waveform changes between logic levels. Transition localization can be the most time-consuming step in jitter measurement because it is the last step where every sample must be processed. We transform the localization FSM (finite state machine) into a Cayley table of a semigroup equivalent to the FSM, in the sense that FSM execution is equivalent to taking products over the semigroup, enabling highly parallel localization yeilding high, gapless throughput. We describe a novel, parallel hardware architecture for executing these semigroup products. We evaluate the practical potential of this approach using two exemplar FSM's, studying the throughput and hardware resource consumption of FPGA (field-programmable gate array) of this parallel architecture, varying two parameters: one that controls FSM size and another that controls the peak hardware parallelism. We use a state of the art FPGA as the technology model, reporting resulting sample rate, power, and resource consumption for a range of designs. These results show that for the simplest FSMs, samples can be examined for transitions at rates as high as 40 gigasamples/second (GSa/s) can be achieved, but that sample rate decreases rapidly for increased p. Also, the explosion in resource requirements with increase p limits data parallelism to below 1024 samples. Likewise, power consumption can be a significant limit for large FSMs.**

I. Introduction

Specifications of digital communications systems designs continue to evolve toward requiring higher data rates at lower bit error rates (BERs). As bit errors are often due to timing jitter, whether random or deterministic, the measurement and analysis of jitter continues to be fundamental to the design, test, commissioning, and maintenance of digital communications equipment. As data rates increase, faster jitter measurements are desired. This goal is complicated by that fact that as BER decreases, events resulting in bit errors are necessarily increasingly rare. Thus, waveforms (or a set of waveforms) containing more communicated bits must be measured in order (1) to capture and identify communications error events [1] so that they may be analyzed and the design improved and (2) extrapolate with accuracy the BER from a limited number of measurements [2]. It is also desirable to identify other errors, in addition to jitter, that arise from design or manufacturing quality issues in the transmitter or receiver, or from unexpected behaviors in the channel between them.

Particularly in wired digital communications, the physical layer operates by sending and receiving pulsed waveforms that encode one or more bits by transitioning between predetermined voltage levels. An early step in measuring jitter in such communication schemes is therefore locating the transition instants [3], the points in time when the waveform changes from logic level to logic level. Transition localization can be the most time-consuming step in jitter measurement because every sample must be processed. The important jitter parameters, such as the jitter histogram and standard deviation, can be computed from the transition instants alone [4] and there are much fewer of these than samples.

We consider a symbol-based transition localization approach that encodes a signal as a series of discrete symbols, enabling flexible implementation in software, FPGA (field-programmable gate array), and custom hardware; flexible approaches are essential to accommodate the large and growing number of different physical layer standards. For example, multicore processors [5] and graphics processors (GPUs) [6] have been explored, and a recent efforts described the parallel approach [7], [8] we explore here. That method transforms small FSMs (finite state machine) into transition semigroups, enabling parallel symbol processing.

We evaluate the practical potential of the semi-group approach using two exemplar and varying two parameters. One parameter controls the minimum width of a pulse that is to be considered a valid signal and not a glitches or spikes in the input signal. Increasing this parameter results in larger FSMs requiring more memory and computational resources for transition localization. The second parameter scales the amount of hardware resources used, allowing exploration of tradeoffs between acheivable sample rate and hardware resources. We use an Altera Stratix V, a 28nm FPGA, as the technology. Our study systematically explores the design space, reporting achievable sample rate as a function of parallelism, power, and resource consumption.

Contributions of this paper include:

- Description of a general, flexible architecture for high-throughput transition localization.
- Empirical studies that show the Cayley table approach can be used to achieve $>$ 40 Giga-samples/second (GSa/s) sample rate with an FPGA operating clock of $<$200Mhz.
- Evidence that achievable sample rate decreases rapidly as pattern-length increases, dropping to $<$ 15 GSa/s for $p = 6$, with resource requirements (RAM blocks) that

grow faster than quadratically in symbol parallelism thus limiting scaling.
- Characterizing the significant power requirements of the parallel Cayley table approach, with logic power for highest performance systems ranging from 10 to 20 W.

The remainder of the paper is structured as follows. First, the FSM transition localization approach and parallel execution is described in Section II. We discuss architecture of a family of FPGA implementations in Section III. Empirically evaluation results are reported in Section IV, followed by discussions that put our results in context.

II. Approach

We employ a discretized symbol-based approach to locate transitions [3]. Firstly, we determine voltage thresholds for the low and high logic states and use them to label samples with corresponding symbolic logic states. For example a two logic state waveform samples labeled Low, High, or Medium (in between) would be denoted L, H, and M. These symbols are passed to a finite state machine that accepts when a transition is located, identifying the transition locations only when a low logic level or high logic level has been fully established.

IEEE 181 [3] defines a parameter p which controls when a pulse is wide enough to be considered a valid pulse and not a spike or glitch. More precisely, p is the number of consecutive samples after a transition from a low to high (or a high to a low) logic state during which the signal must remain in the new logic state. For example, when $p = 2$ an upward transition requires two L's, any number of M's, and at least two H's. A downward transition must have two H's followed by any number of M's, then at least two L's.

An example FSM, for $p = 2$, is given in Figure 1. State 0 is the initial state. States 1 (resp. 6) indicates that one L (resp. H) has been input but no sequence of $p = 2$ L's or to H's has yet been seen. State 7 is reached when $p = 2$ consecutive samples are L's; the waveform is now in the low logic state. If from there, some number of M's are input followed by $p = 2$ H's, state 9 is reached. State 9 is an accepting state, as $p = 2$ L's, some number of M's, then $p = 2$ H's have been found. Thus, reaching state 9 means that an up-going transition has been located in the waveform. Similarly, state 4 is reached whenever $p = 2$ H's, some number of M's, then $p = 2$ L's are found. That is, reaching state 4 means that a down-going transition has been located in the waveform.

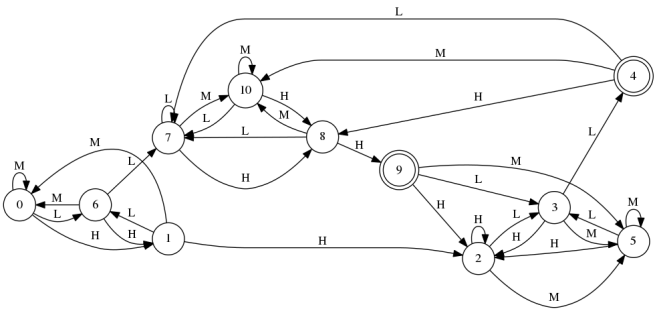


Fig. 1: The transition localization FSM for $p = 2$.

The reader will note that initial states 0, 1 and 6 used only at the start, to establish whether the waveform is initially in the low or high logic level. Once the initial logic level is determined, these states will not be re-entered. Such "startup" states are an increasing fraction of the FSM for increasing values of p. For example, see Figure 2, the machine that finds transitions with $p = 6$ has 40% of its states are such startup states. Here, the startup states are states 0 through 4 and 14 through 18, or 40% of the machine's states.

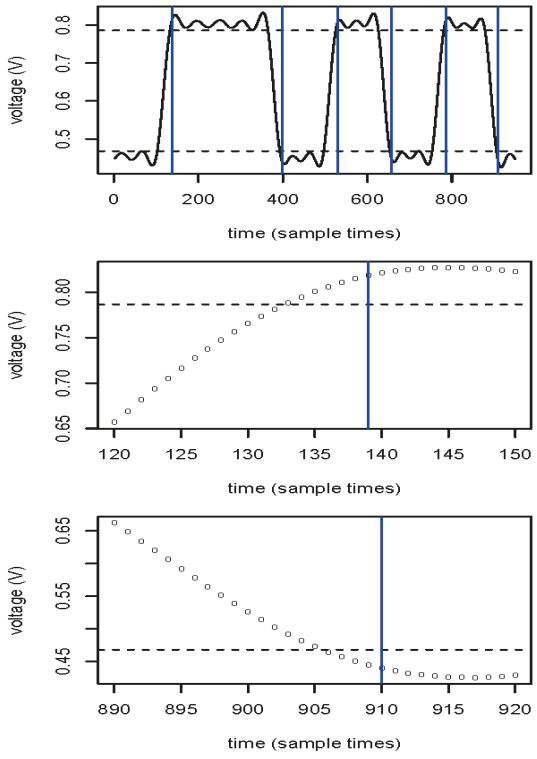


Fig. 3: Example of transition localization using an FSM with $p = 6$. Horizontal dashed lines show the low and high logic level thresholds. Vertical blue lines show the samples where the FSM is in the accepting state. Bottom two graphs are zoom-ins into the top graph showing that transitions are found after 6 samples at the new logic level.

The natural way to implement FSMs, either in hardware or software, is to read one symbol (here, L, M, or H), update the current state, and repeat until the input is exhausted. Performance is limited to one input sample per clock cycle (hardware) or loop iteration (software). We use the parallel FSM execution method of [7] and [8], transforming the FSM into a mathematical structure with an associative operator, that is, a semigroup. Then we exploit efficient data and pipeline parallel methods for repeated application of that operation. The methods have different names ("parallel scan" or "prefix sum" for software, "parallel prefix circuit") for hardware—but both names represent the same ordering of parallel execution of the same operations [9] and the same opportunities for data and pipeline parallelism.

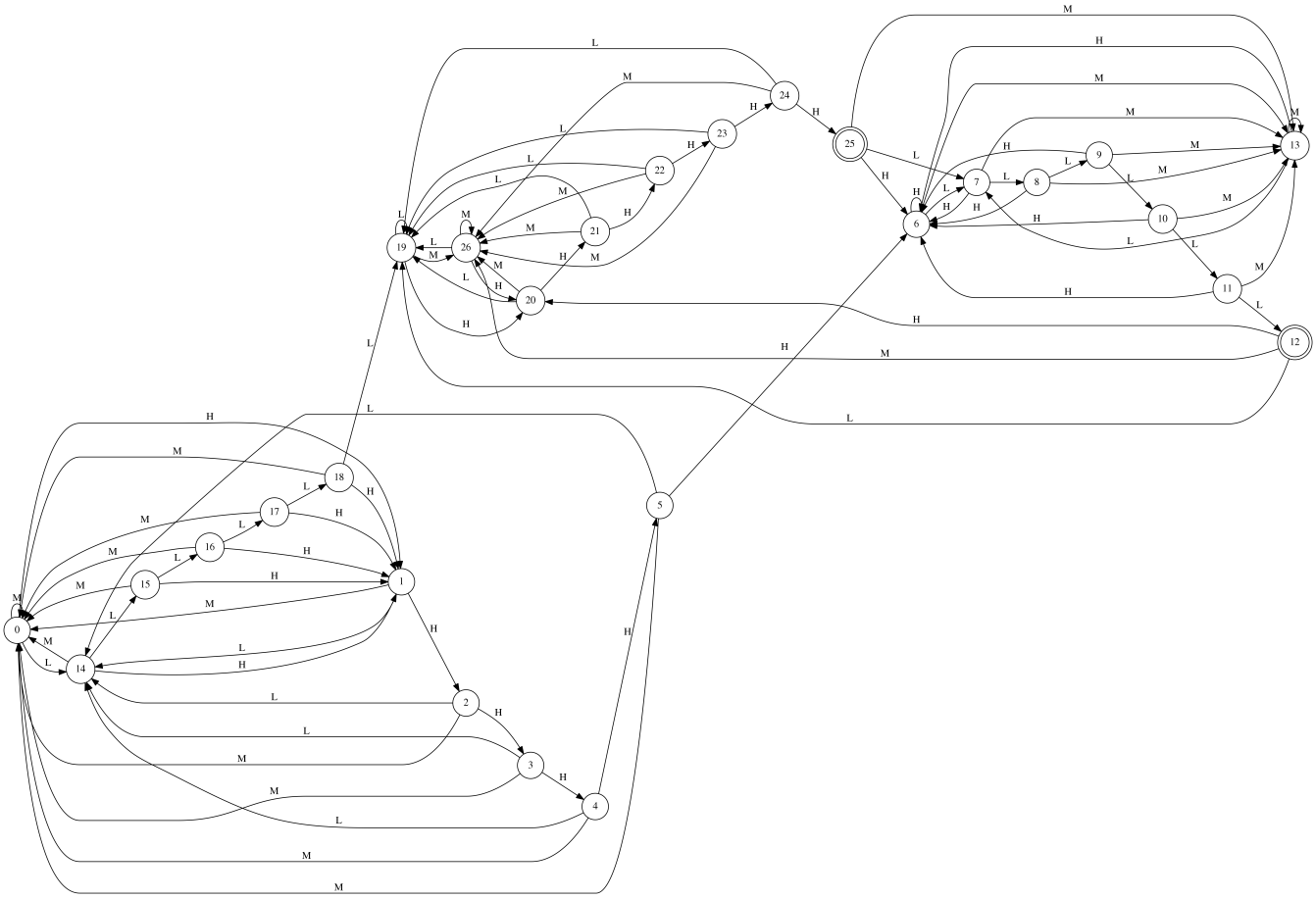


Fig. 2: The transition localization FSM for $p = 6$. State 0 (initial state) and State 12 or 15 (transition recognition)

TABLE I: Semi-group size and memory requirements

p	2	4	6	7	13
N	25	81	169	225	729
Memory (bits)	3125	45,927	228,488	405,000	5,314,410

We transform the FSM [7], [8] into:

- A set of natural numbers, $S = \{0, ..., N-1\}$, and
- An $N \times N$ table containing numbers from S that gives an associative binary operation $\oplus$ from S to S,
- A mapping f : FSM symbols $\rightarrow S$, and
- A value $a \in S$.

with the property that if $s_0, s_2, s_3, \ldots, s_i, \ldots$ are the symbols input to the FSM, then the FSM will be in accepting state after reading symbol s_i iff $f(s_0) \oplus f(s_1) \oplus f(s_2) \oplus \ldots f(s_i) > a$. S under $\oplus$ is sometimes called the transition semigroup of the FSM [10]. N is called the order of the semigroup. The parallel scan/parallel prefix circuit construction computes the cumulative sum $f(s_0) \oplus f(s_1) \oplus f(s_2) \oplus \ldots f(s_M)$ with the following properties:

- All k inputs are read in parallel,
- All k cumulative sums $f(s_0)$, $f(s_0) \oplus f(s_1)$, $f(s_0) \oplus f(s_1) \oplus f(s_2)$, ..., $f(s_0) \oplus f(s_1) \oplus f(s_2) \oplus \ldots f(s_k)$ are produced in parallel,
- Requires $2k \oplus$ operations, and
- The critical path is $2 * \log_2 k$ operations.

The degree of parallelism is k, the number of symbols processed at a time. However, this parallelism is not without cost. In general, the transformation of an FSM produces a significantly larger semigroup of order N. Because it must be customized to each FSM, the semigroup is implemented as a table in memory that grows faster than N^2. Table I gives several example the semigroup orders N and memory sizes (in bits) $N^2\lceil \log_2 N \rceil$ for typical values of p. We consider the practical consequences of this growth and the resulting overall performance in Section IV.

III. A CANONICAL ARCHITECTURE FOR PARALLEL SEMIGROUP EXECUTION OF FSMS

A. Semigroup in Prefix Sum Tree

It is attractive to speed transition localizaiton using the semigroup approach using custom hardware. First we consider the parallel computation structure required by the semigroup approach.

Suppose it is desired to perform the parallel prefix operation on k input symbols at a time. That is, a semigroup parallel prefix block B takes k input symbols (x_0,x_1,...,x_{k-1}) and generates k output symbols (x_0,$x_0 \oplus x_1$,...,$x_0 \oplus x_1 \oplus \ldots \oplus x_{k-1}$). Our hardware's ciruit topology for block B is exactly the same

as a classical prefix sum tree except each node becomes a single-port memory bank storing the whole semigroup (see Figure 4). Compared to the serialized structure with $O(k)$ processing time, our tree organization has the potential to achieve $O(\log k)$ processing time. For example, Figure 4 with 8 inputs ($X_0 ... X_7$) and 8 outputs ($Y_0 ... Y_7$) shows one critical path of length 4 (in red). If there are resource constraints, our design will reuse memory banks, potentially increasing the critical path length. For example, an 8-input prefix sum tree employing four memory banks (that is, four Cayley table lookups can occur per cycle) has a critical path of four, but with two memory banks (two lookups per cycle) it has a critical path of six.

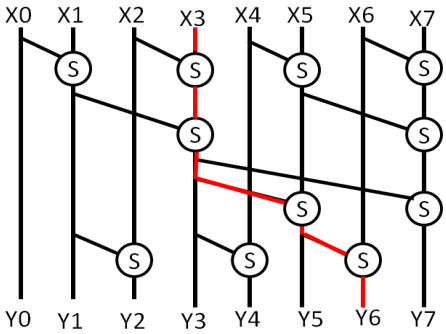


Fig. 4: A Prefix sum tree: Eight inputs, critical path of four.

B. Overall Architecture

To assess performance and power efficiency we consider a single design that processes k symbols in parallel. This design organizes k inputs and m memory banks to compute a prefix sum tree. We use a greedy algorithm to schedule operations, achieving good load balancing across memory units, and maintaining minimum critical path. Our algorithm also ensures minimum fan-in and fan-out. Figure 5 shows the architecture of an 8-input with 4 available memory units block for simple-class FSM with p=2. Because memory units are limited, they must be reused to achieve the correct logical scheduling for an 8-input prefix sum tree (Figure 4) with a critical path of four operations. With fewer than $k/2$ memory units, the critical path may become longer. Mapping the logical schedule on the physical memory units requires different routing of outputs at each cycle. Note: in a real system, k-input tree would process $k-1$ inputs at a time; we ignore this fencepost counting detail to simplify explanation and increase result clarity. This canonical design is used for broader system exploration in Section IV.

Each memory unit in Figure 5 implements a Cayley table, but that table can be used at various points in a parallel prefix computation. To achieve this, we connect each memory to both a selector and mux that enables results at each level of the tree to be routed to the appropriate memory units. Thus each memory unit consists of 2 encoders, 2 multiplexors, 1 address ALU and 1 memory bank as shown in Figure 6. Reducing m not only lengthens the schedule due to competition for

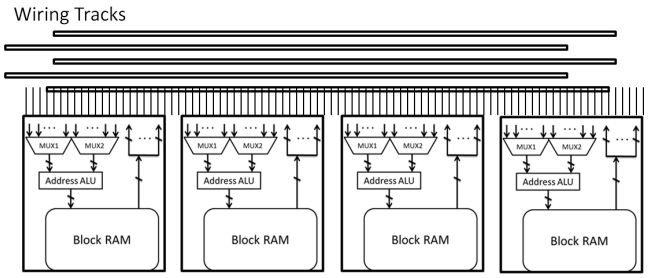


Fig. 5: Example of the architecture with $k = 8$, $m = 4$.

memory units, but it also increases the size of multiplexors and encoders thus lengthening the clock cycle. All memories sizes are integral powers of 2, enabling fast address calculation using efficient bit shifts.

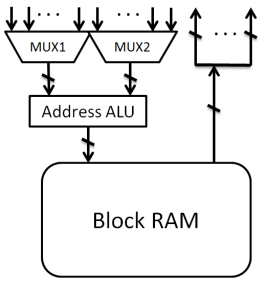


Fig. 6: Each Memory Unit has many inputs, and sends its result to many outputs.

IV. Modeling Results

To explore the design space, we constructed a Verilog code generator that produces our architecture implementations for a range of k and m. We synthesized the designs using Altera's QuartusII 15.0.0, targeting the Stratix V device 5SGTMC7K2F40C1 that includes 2560 20Kbit BRAMs. To be conservative, we use the Slow 85C corner and set all I/O pins to be virtual except for the clock. Clock rate, resource usage, and power estimates are generated using Altera's tools. Our results do not include input/output power.

We evaluate the two classes of FSMs while varying p. One class, called here the simple class, are the FSMs described in [7], [8]. We call the FSMs described in Figures 1 and 2 the complex class. Our primary metric is sample rate (Gigasamples/second, GSa/s), converted from clock rate. We report GSa/s as a function of p and power (milliwatts). The most usual range of p values in practice is between 2 and about 7, which is enough to suppress accepting glitches as pulses at reasonable numbers of samples per minimum signalling pulse width. We report results for $p = 13$ in order to give some idea of how well the proposed architecture would scale when used to implement larger FSMs that might arise from other signal feature recongition tasks.

Figure 7a presents the estimated, maximum sample rate and power consumption for simple FSMs with p=2. Scaling

k from 8 to 8192, increases the achievable sample rates from 0.47 to over 37.8 GSa/s at 7.5 W and 42.5 GSa/s at 15.9 W. Higher values of k make much greater sample rates possible due to decreasing memory bank contention up to $m \approx k/8$, and then increasing only incrementally beyond that point. Best performance per watt is achieved at the knee in the curve. Clock speeds increase steadily with m as the complexity of the wiring and multiplexors decreases. There is little compensating penalty of increased wire length in an FPGA. Power consumption is dominated by memory reads for Cayley table operations and grows generally with m. Figure 7b and 7c present sample rates for larger values of p, including 6, 7, and 13. These plots reflect that greater pattern complexity slows clock rates by as much as 2.5x, which combined with resource limits holds performance to 13.2 GSa/s at 18.7 W and 3.3 GSa/s at 11 W for p=6/7 and 13 respectively.

Figure 8 shows limitations due to increased resource requirements. All 2560 block RAMs available on the FPGA are consumed by even modest k for higher values of p. For $p = 6, 7$, the parallel approach requires dramatically more resources preventing the design from reaching the knee of ideal performance for $k = 4096$. For $p = 13$ these effects are even more pronounced thus limiting full exploitation of parallelism to below $k = 1024$.

For complex class FSMs (see Figure 9,10), similar trends emerge, with sample rates exceeding 25 GSa/s achievable for $p = 2$, but significant limits on full exploitation of parallelism for $p = 6, 7, 13$. The generally greater hardware resource requirements for complex FSMs further limits parallelism.

V. Related Work

We first compare our results to other published work on the basis of sample rate and sample rate/watt.

For the presented FPGA approach, the simple and complex FSM with $p = 2$ achieves a sample rate of 42.5 GSa/s and 25 GSa/s respectively. Prior work on graphics processing units (GPU) employing the same semigroup table sizes as the simple and complex FSMs with $p = 2$ has been reported [8] to have peak throughputs of 0.8 GSa/s and 0.7 GSa/s respectively on an NVIDIA Geforce GTX 780. A speculative parallel-prefix based implementation of FSMs on multi-core processors achieve rates of 3 GSa/s for 8 bit ASCII characters on an 8-core, 16-thread 2.67Ghz Xeon X5650 [11].

For $p = 2$, our FPGA approach achieves a power efficiency of 3.5 GSa/s/W (gigasamples per second per watt) and 3.7 GSa/s/W for the simple and complex FSMs respectively. The GPU implementation [8] at 100W achieves power efficiency of of 0.008 GSa/s/W and 0.007 GSa/s/W respectively. While no published numbers exist, assuming ≈50W for the multicore workstation would correspond to 0.06 GSa/s/W.

A radically different approach, the Unified Automata Processor (UAP) [12]–[14] extends the basic processor instruction set with features that accelerate a wide range of automata processing, enabling high-level software to access extreme finite-automata performance seamlessly and with no overhead. Using a 28nm ASIC design, we estimate the UAP could achieve a linespeed of 9.1 Gbits/s (4.55 GSa/s, at the 2 bits per sample) for the simple and complex FSMs at an on-chip power of 7.9 mW [15], and a power efficiency of 575 GSa/s/W.

VI. Summary

The bottleneck for jitter and BER measurement of long-duration pulsed waveforms is often determining the temporal locations of the events of interest, the transition locations. This is because those locations must be derived from the raw sampled data, further processing can then focus on the smaller dataset of locations—the amount of data that must be processed in the later processing steps is much reduced.

We have described a novel architecture based on parallel prefix circuit execution of cumulative sums which is equivalent to running a finite state machine based search for those events of interest, the transition locations. Our exploration of the design space for FPGA-based implementations, and comparison to prior work, yields both promising performance and important insights into the design's trade-offs and limits. The throughput power efficiency of our FPGA approach is superior to both GPUs and multicore implementations.

References

[1] D. Murray, "Using RF recording techniques to resolve interference problems," in *Proc. IEEE AUTOTEST Conference*, 2013, pp. 1–6.

[2] W. Maichen, *Digital Timing Measurement: From Scopes and Probes to Timing and Jitter*. Springer, 2006, ch. 9.3.

[3] "IEEE standard for transitions, pulses, and related waveforms," *IEEE Std 181-2011 (Revision of IEEE Std 181-2003)*, pp. 1–71, 6 2011.

[4] T. Loken, L. Barford, and F. Harris, "Massively parallel jitter measurement from deep memory digital waveforms," in *Instrumentation and Measurement Technology Conference (I2MTC), 2013 IEEE International*. IEEE, 2013, pp. 1744–1749.

[5] L. Barford, "Speeding localization of pulsed signal transitions using multicore processors," *Instrumentation and Measurement, IEEE Transactions on*, vol. 60, no. 5, pp. 1588–1593, 2011.

[6] V. Khambadkar, L. Barford, and F. Harris, "Massively parallel localization of pulsed signal transitions using a GPU," in *Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International*. IEEE, 2012, pp. 2173–2177.

[7] L. Barford, "Parallelizing small finite state machines, with application to pulsed signal analysis," in *Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, 2012, pp. 1957–1962.

[8] L. Barford, Y. Liu, and S. S. Bhattacharyya, "Data flow algorithms for processors with vector extensions: handling actors with internal state," *Journal of Signal Processing Systems*, accepted for publication 2015.

[9] G. E. Blelloch, "Scans as primitive parallel operations," *Computers, IEEE Transactions on*, vol. 38, no. 11, pp. 1526–1538, 1989.

[10] J.-E. Pin, "Syntactic semigroups," in *Handbook of language theory*, G. Rozenberg and A. Salomaa, Eds., 1997, ch. 10.

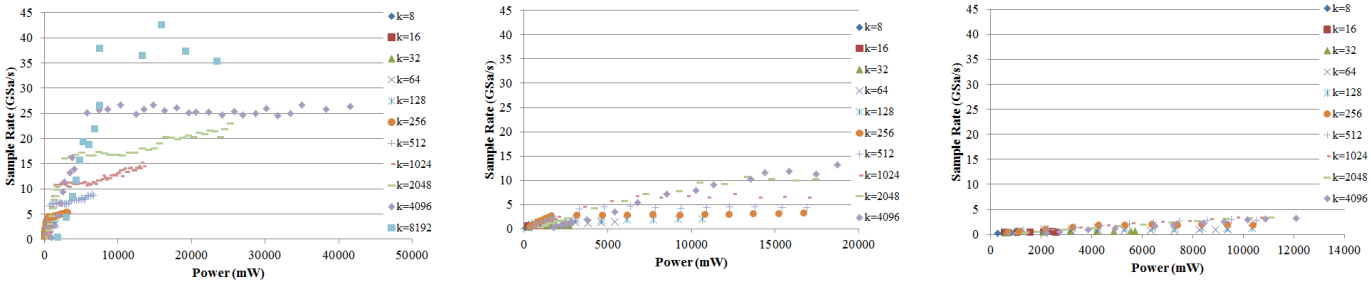
[11] T. Mytkowicz, M. Musuvathi, and W. Schulte, "Data-parallel finite-state machines," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14, 2014, pp. 529–542.

[12] A. Chien, A. Snavely, and M. Gahagan, "10x10: A general-purpose architectural approach to heterogeneity and energy efficiency," in *The Third Workshop on Emerging Parallel Architctures at the International Conference on Computational Science*, 2011.

[13] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, no. 5, pp. 67–77, May 2011.

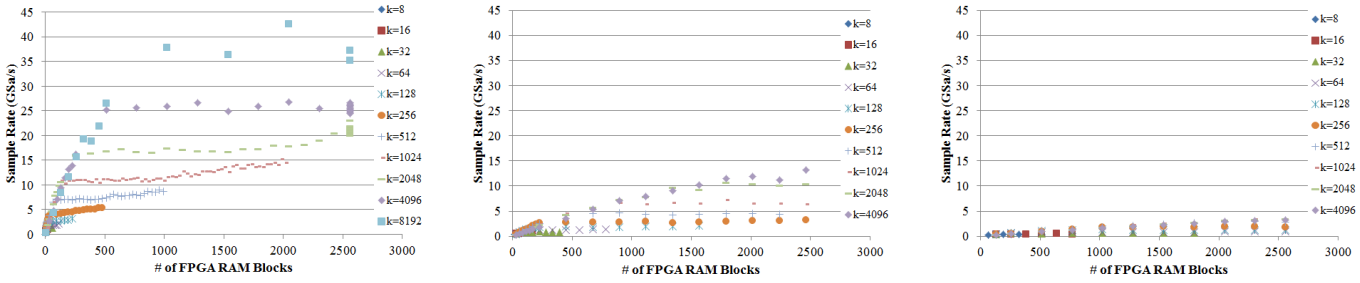
[14] D. Vasudevan and A. A. Chien, "The bit-nibble-byte microengine (bnb) for efficient computing on short data," in *Proc. of the 25th Great Lakes Symposium on VLSI*, ser. GLSVLSI, 2015, pp. 103–106.

[15] Y. Fang, T. T. Hoang, M. Becchi, and A. A. Chien, "Fast support for unstructured data processing: the unified automata processor," in *Proc. 48th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2015.



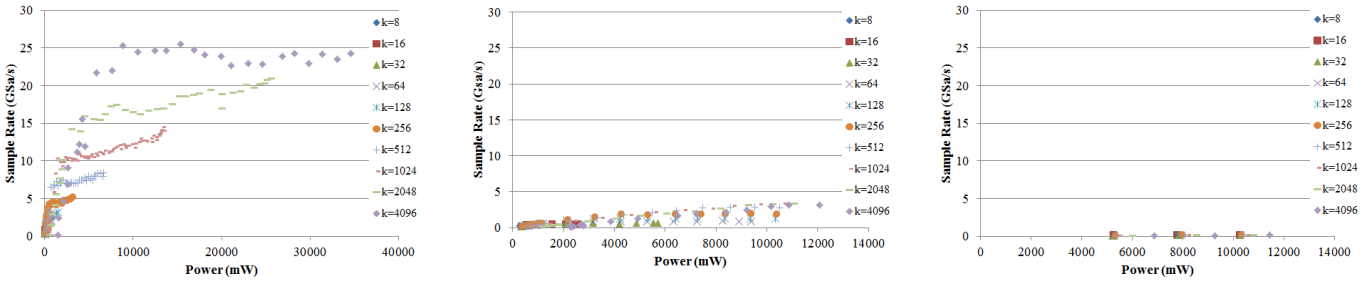
(a) p = 2 (b) p = 6 and p = 7 (c) p = 13

Fig. 7: Simple Class FSMs, Symbol Rate vs. Power



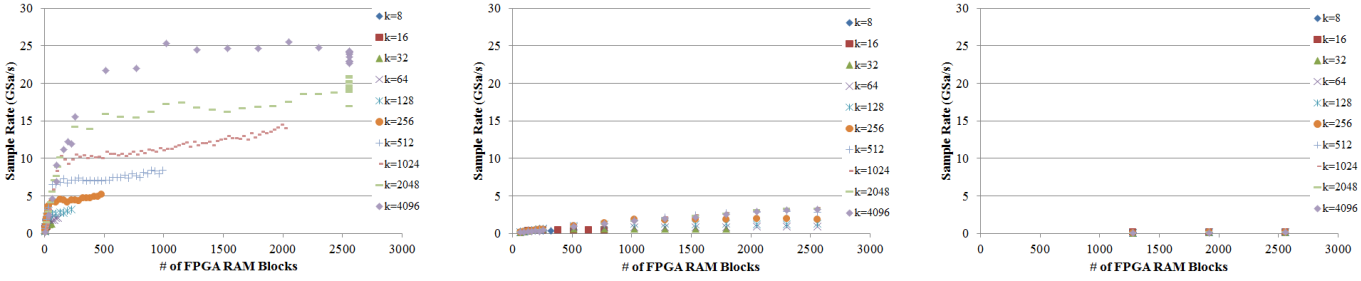
(a) p = 2 (b) p = 6 and p = 7 (c) p=13

Fig. 8: Simple Class FSMs, Symbol Rate vs. # of RAM Blocks



(a) p = 2 (b) p = 6 and p = 7 (c) p = 13

Fig. 9: Complex Class FSMs, Symbol Rate vs. Power



(a) p = 2 (b) p = 6 and p =7 (c) p = 13

Fig. 10: Complex Class FSMs, Symbol Rate vs. # of RAM Blocks