From Arbitrary Functions to Space-Time Implementations

Georgios Tzimpragos*, Nestan Tsiskaridze*, Kylie Huch*, Advait Madhavan†‡, and Timothy Sherwood*

* University of California, Santa Barbara, CA 93106
† National Institute of Standards and Technology, Gaithersburg, MD 20899
‡ University of Maryland, College Park, MD 20742

***Abstract*—Processing in the temporal domain brings the promise of orders of magnitude performance improvements for certain classes of applications when compared with conventional binary implementations. However, the lack of a general design methodology poses a significant barrier to realizing the true potential of this new foundation. To help overcome these challenges, we present an automated (and formally verified) tool capable of checking arbitrary functions against the properties of space-time algebra, simplifying their expression, and automatically generating hardware implementation in race logic.**

I. Introduction

The end of Dennard's scaling and the slowdown of Moore's law marks the beginning of a new "heterogeneous" era in computing, where new materials, device structures, and computing architectures are being continually introduced in efforts to deliver the anticipated growth. To successfully navigate this era of innovation though, it is necessary to (a) distinguish between new paradigms for computation and new technological advancements and (b) understand how the one can motivate or even unlock the true potential of the other [5].

Along these lines, in our prior work we introduce the idea of race logic [3], [7]. The approach lives at the boundary of analog and digital worlds where fully digital signals (all wires carry only a "high" or "low" signal) are used to encode continuous rather than binary variables. Events are represented by low to high edges and computation emerges through the purposeful interaction of edges and their relative delays. Only a single "wire" is required per variable (because the time it takes for an event to appear on that wire is what encodes the value) and the operators forming its foundation are MIN, MAX, DELAY, and INHIBIT, rather than AND, OR, and NOT. Even when off-the-shelf digital CMOS components are used, the encoding can result in orders of magnitude performance improvements for certain classes of applications. There might be even more potential improvement when novel devices are used in this new way.

In contrast with conventional digital computing, which relies on a binary system of ones and zeros, processing in race logic happens in the temporal domain and it is governed by the rules and properties of space-time algebra [6]. As with any information representation change, some computations become easier to perform while others become more difficult — but we now know it to be efficient in several important cases. For example, A. Madhavan et al. [3] use temporal relationships to implement Needleman and Wunsch's popular DNA sequence algorithm. M. H. Najafi et al. [4] demonstrate a low-cost bitonic sorting network circuit using temporal processing. G. Tzimpragos et al. [7] apply race logic to accelerate ensembles of decision trees, while J.E. Smith [6] explores the relationship between temporal codes and spiking neural networks. However, all of these designs have been developed in an ad-hoc manner and tools that help us check, optimize, and efficiently implement applications in this new logic are still missing.

Here we explore a more systematic way of developing and checking space-time systems. We describe a formalization of the known properties of space-time algebra, present an automated checker capable of verifying whether an arbitrary function is implementable in the "temporal" domain or not that works even high dimensional cases, and then, if the given function passes the checks, we simplify its expression and return a synthesizable RTL design.

II. Space-Time Algebra Properties

Functions defined over space-time algebra take temporal events as inputs, produce temporal events as outputs, and the relationships between input and output events must be consistent with the flow of time. Under this formulation, any space-time function must satisfy the properties of *invariance* and *non-prescience* [6]. A more formal definition of these properties in discretized time follows.

Let $N_0^\infty = \{0, \mathbb{N}^+, \infty\}$, where $\mathbb{N}^+$ is the set of positive natural numbers [1] and ∞ represents an event that never occurs; we assume that ∞ is always greater than any number in $\mathbb{N}^+$. In the case of a *bounded* domain, we assume that any number beyond the upper bound is indistinguishable from ∞. Let n be a positive integer and $X = \{x_1, \ldots, x_n\}$ be a set of variables over N_0^∞. Let f be an n-ary function defined as a mapping from X^n to N_0^∞, i.e. $f : X^n \to N_0^\infty$. Let $\sigma : X \to N_0^\infty$ be an *assignment* function over X, such that $\sigma(x_j) = i_j$ for $x_j \in X$, $i_j \in N_0^\infty$, and $j \in [1, n]$.

[1]Each element of N_0^∞ represents a specific time at which an event occurs.

Given a function f, we define by $f\sigma$ an *evaluation* of f after replacing all variables x_j in f by $\sigma(x_j)$, i.e. $f\sigma = f(i_1, \ldots, i_n)$.

Invariance. Invariance captures the property of a function that, if one is to shift all of the inputs in time by a constant amount, the output will shift by that same amount. More formally, if $c \in \mathbb{N}^+$ and σ_c is an assignment function such that $\sigma_c(x_j) = \sigma(x_j) + c$ for all $x_j \in X$, then $f\sigma_c = f\sigma + c$; i.e. $f(i_1 + c, \ldots, i_n + c) = f(i_1, \ldots, i_n) + c$.

An important ramification of invariance is that, no output event will ever occur before at least one input, associated with it, is observed. We formalize this as follows: $\forall\sigma\exists x_j : x_j \in X, \sigma(x_j) \leq f\sigma$.

Non-prescience. The property of non-prescience denotes that a function cannot use any "future" information for the determination of what output to return. To formally define it, we first introduce the set $X_{f,\sigma} = \{x_j | x_j \in X, \sigma(x_j) > f\sigma\}$ to denote a subset of all variables in X that carry "future" information under the current assignment. The non-prescience property is formalized as follows[2]: if σ' is an assignment function, such that

$$\sigma'(x_j) = \begin{cases} \sigma(x_j), & x_j \in X - X_{(f,\sigma)} \\ i'_j, & i'_j \in (f\sigma, \infty], x_j \in X_{(f,\sigma)} \end{cases},$$

then $f\sigma' = f\sigma$.

III. Checking for Space-Time Violations

Any space-time function can be expressed with a *function table*. However, not every function table represents a valid temporal function; it may violate one of the properties that govern the space-time algebra. To address this issue, we develop an automated checker capable of verifying whether an arbitrary function is implementable in race logic or not.

Function table. Given a set of variables $X = \{x_1, \ldots, x_n\}$ we define a function table $\mathbb{T}_f$ as an n-dimensional table where each dimension represents a variable in X and is indexed in the range of N_0^∞. A position $i_1, \ldots, i_n$ in the table is indexed by an assignment σ, where $\sigma(x_1) = i_1, \ldots, \sigma(x_n) = i_n$. An entry in the table at the position $i_1, \ldots, i_n$ is denoted as $\mathbb{T}_f[i_1, \ldots, i_n]$. If the function f is not specified for a particular input, the corresponding table entry is marked as "N/A", otherwise the entry stores the value $f\sigma$, i.e. either $\mathbb{T}_f[i_1, \ldots, i_n] =$ "N/A" or $\mathbb{T}_f[i_1, \ldots, i_n] = f\sigma$. Intuitively, the indices of $\mathbb{T}_f$ are inputs and the entries in $\mathbb{T}_f$ are outputs. In Panel (a) of Figure 1, we illustrate a two-dimensional function table.

[2]We assume that all computations are performed instantly thus events at time t can be used to trigger other events at time t

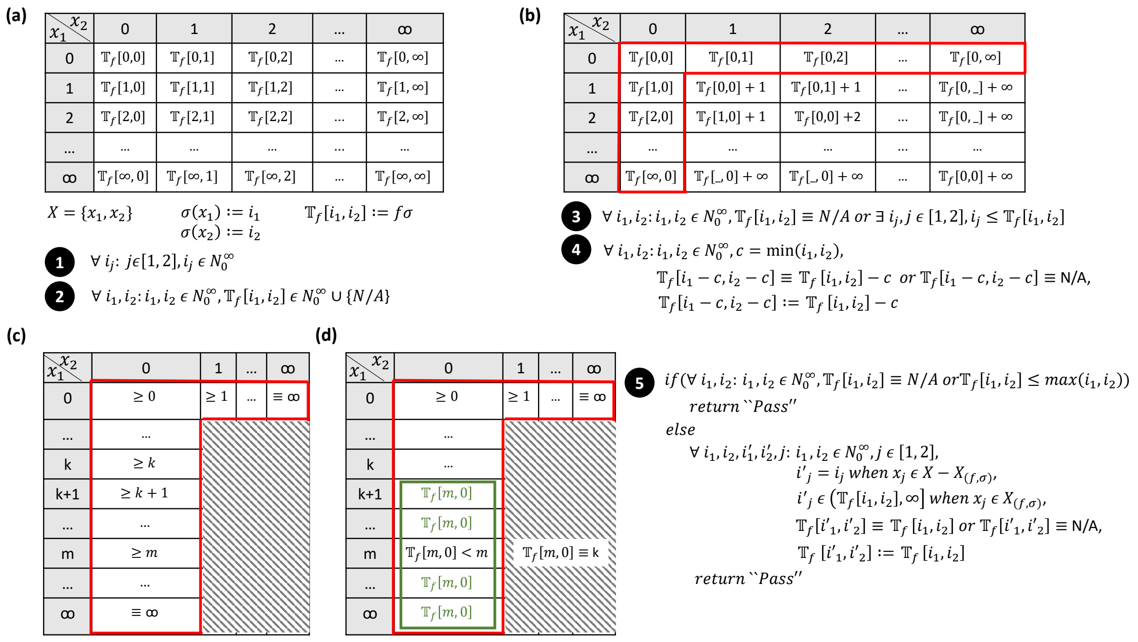
Function Table Checker. Given an arbitrary function table $\mathbb{T}_f$, possibly not fully specified, the checker operates through a chain of verification steps to verify whether the function table is implementable in race logic.

Steps 1-3. The first three steps, shown in Figure 1(a-b), are sanity checks and verify (a) that the indices and entries in the given table are of appropriate type and (b) that for every defined entry of $\mathbb{T}_f$ at least one index (input) is observed before this entry (output) occurs. These checks, while not strictly necessary, catch easy errors quickly before more exhaustive checking is employed.

Figure 1(b) also illustrates how one can use invariance to relate entries diagonally across the table. If *Step 3* returns *TRUE*, then all entries can be shifted to the table's "surface"; column 0 and row 0 of the function table will then determine fully all the remaining entries. Given this property of a "temporal" function table, our checker should assure that there are no *conflicting* entries — the situation where multiple normalized entries map to the same slot, but with different values, during the normalization phase. Step 4 completes this final check by normalizing the function table in this way and checking whether conflicts occur: $\forall i_1, \ldots, i_n : i_1, \ldots, i_n \in N_0^\infty, c = min(i_1, \ldots, i_n)$, $\mathbb{T}_f[i_1 - c, \ldots, i_n - c] \equiv \mathbb{T}_f[i_1, \ldots, i_n] - c$ or $\mathbb{T}_f[i_1 - c, \ldots, i_n - c] \equiv$ N/A, $\mathbb{T}_f[i_1 - c, \ldots, i_n - c] := \mathbb{T}_f[i_1, \ldots, i_n] - c$.

The property of non-prescience is the most interesting to check and *Step 5* consists of two "branches". More specifically, the first branch, shown in Figure 1(c), verifies whether there is any table entry that uses "future" information. For example, the case where $x_1 = 0$, $x_2 = 4$, and $\mathbb{T}_f[x_1, x_2] = 5$ *passes* the check as $\mathbb{T}_f[x_1, x_2] \geq max(x_1, x_2)$, while the case where $x_1 = 0$, $x_2 = 4$, and $\mathbb{T}_f[x_1, x_2] = 1$ does not as $1 < 4$. If no such entry is found the checker returns *"Pass"* and it is guaranteed that it is implementable in race logic. Otherwise, ($X_{(f,\sigma)} \neq \emptyset$) and the checker provides the user an option to *"extrapolate"*. In other words, based on the given table entries the checker is able to infer all other entries related to any case that violates the non-prescience property. For example, in the case where $x_1 = 0$, $x_2 = 4$, and $\mathbb{T}_f[x_1, x_2] = 2$, if the extrapolation option is enabled, the checker will return: $\forall x_1, x_2 : x_1 \in [0], x_2 \in [3] \cup [5, \infty], \mathbb{T}_f[x_1, x_2] = 2$. More formally:

if ($\forall i_1, \ldots, i_n : i_1, \ldots, i_n \in N_0^\infty, \mathbb{T}_f[i_1, \ldots, i_n] \equiv$N/A or $\mathbb{T}_f[i_1, \ldots, i_n] \geq max(i_1, \ldots, i_n)$): {**return** *"Pass!"*}
else: {$\forall i_1, \ldots, i_n, i'_1, \ldots, i'_n, j :$
$i_1, \ldots, i_n \in N_0^\infty, j \in [1, n], X_{(f,\sigma)} \neq \emptyset$,
$i'_j = i_j$ when $x_j \in X - X_{(f,\sigma)}$,
$i'_j \in (\mathbb{T}_f[i_1, \ldots, i_n], \infty]$ when $x_j \in X_{(f,\sigma)}$,
$\mathbb{T}_f[i'_1, \ldots, i'_n] \equiv \mathbb{T}_f[i_1, \ldots, i_n]$ or $\mathbb{T}_f[i'_1, \ldots, i'_n] \equiv$N/A,
$\mathbb{T}_f[i'_1, \ldots, i'_n] := \mathbb{T}_f[i_1, \ldots, i_n]$}
return *"Pass!"*



(a)

$x_1 \backslash x_2$	0	1	2	...	∞
0	$\mathbb{T}_f[0,0]$	$\mathbb{T}_f[0,1]$	$\mathbb{T}_f[0,2]$	...	$\mathbb{T}_f[0,\infty]$
1	$\mathbb{T}_f[1,0]$	$\mathbb{T}_f[1,1]$	$\mathbb{T}_f[1,2]$	...	$\mathbb{T}_f[1,\infty]$
2	$\mathbb{T}_f[2,0]$	$\mathbb{T}_f[2,1]$	$\mathbb{T}_f[2,2]$	...	$\mathbb{T}_f[2,\infty]$
...	...	...	...	...	...
∞	$\mathbb{T}_f[\infty,0]$	$\mathbb{T}_f[\infty,1]$	$\mathbb{T}_f[\infty,2]$	...	$\mathbb{T}_f[\infty,\infty]$

$X = \{x_1, x_2\}$ $\sigma(x_1) := i_1$ $\sigma(x_2) := i_2$ $\mathbb{T}_f[i_1, i_2] := f\sigma$

1 $\forall\, i_j\colon j \epsilon [1,2], i_j \,\epsilon\, N_0^\infty$

2 $\forall\, i_1, i_2\colon i_1, i_2 \,\epsilon\, N_0^\infty, \mathbb{T}_f[i_1, i_2] \,\epsilon\, N_0^\infty \cup \{N/A\}$

(b)

$x_1 \backslash x_2$	0	1	2	...	∞
0	$\mathbb{T}_f[0,0]$	$\mathbb{T}_f[0,1]$	$\mathbb{T}_f[0,2]$	...	$\mathbb{T}_f[0,\infty]$
1	$\mathbb{T}_f[1,0]$	$\mathbb{T}_f[0,0]+1$	$\mathbb{T}_f[0,1]+1$	...	$\mathbb{T}_f[0,\_]+\infty$
2	$\mathbb{T}_f[2,0]$	$\mathbb{T}_f[1,0]+1$	$\mathbb{T}_f[0,0]+2$	...	$\mathbb{T}_f[0,\_]+\infty$
...	...	...	...	...	...
∞	$\mathbb{T}_f[\infty,0]$	$\mathbb{T}_f[\_,0]+\infty$	$\mathbb{T}_f[\_,0]+\infty$	...	$\mathbb{T}_f[0,0]+\infty$

3 $\forall\, i_1, i_2\colon i_1, i_2 \,\epsilon\, N_0^\infty, \mathbb{T}_f[i_1, i_2] \equiv N/A \text{ or } \exists\, i_j, j \,\epsilon\, [1,2], i_j \leq \mathbb{T}_f[i_1, i_2]$

4 $\forall\, i_1, i_2\colon i_1, i_2 \,\epsilon\, N_0^\infty, c = \min(i_1, i_2),$
$\mathbb{T}_f[i_1 - c, i_2 - c] \equiv \mathbb{T}_f[i_1, i_2] - c \text{ or } \mathbb{T}_f[i_1 - c, i_2 - c] \equiv$ N/A,
$\mathbb{T}_f[i_1 - c, i_2 - c] := \mathbb{T}_f[i_1, i_2] - c$

(c)

$x_1 \backslash x_2$	0	1	...	∞
0	≥ 0	≥ 1	...	$\equiv \infty$
...	...			
k	$\geq k$			
k+1	$\geq k+1$			
...	...			
m	$\geq m$			
...	...			
∞	$\equiv \infty$			

(d)

$x_1 \backslash x_2$	0	1	...	∞
0	≥ 0	≥ 1	...	$\equiv \infty$
...	...			
k	...			
k+1	$\mathbb{T}_f[m,0]$			
...	$\mathbb{T}_f[m,0]$			
m	$\mathbb{T}_f[m,0] < m$	$\mathbb{T}_f[m,0] \equiv$ k		
...	$\mathbb{T}_f[m,0]$			
∞	$\mathbb{T}_f[m,0]$			

5
```
if (∀ $i_1, i_2\colon i_1, i_2 \,\epsilon\, N_0^\infty, \mathbb{T}_f[i_1, i_2] \equiv N/A \text{ or } \mathbb{T}_f[i_1, i_2] \leq max(i_1, i_2)$)
    return ``Pass''
else
    $\forall\, i_1, i_2, i_1', i_2', j\colon i_1, i_2 \,\epsilon\, N_0^\infty, j \,\epsilon\, [1,2],$
        $i'_j = i_j$ when $x_j \,\epsilon\, X - X_{(f,\sigma)}$,
        $i'_j \,\epsilon\, (\mathbb{T}_f[i_1, i_2], \infty]$ when $x_j \,\epsilon\, X_{(f,\sigma)}$,
        $\mathbb{T}_f[i'_1, i'_2] \equiv \mathbb{T}_f[i_1, i_2]$ or $\mathbb{T}_f[i'_1, i'_2] \equiv$ N/A,
        $\mathbb{T}_f[i'_1, i'_2] := \mathbb{T}_f[i_1, i_2]$
    return ``Pass''
```

Fig. 1. Panel (a) Any function f can be expressed with a function table $\mathbb{T}_f$. *Steps* 1 and 2 check the domain of table indices and entries. Panel (b) shows how all entries across the diagonals in a ”temporal” function table are related. *Steps* 3 and 4 check that the invariance property holds and that there are no conflicting entries. The property of non-prescience is checked in *Step* 5 and it is shown in Panel (c). If this check fails, the tool provides the user with the option to extrapolate and infer "new" table entries using the property of non-prescience as shown in Panel (d). To return "Pass" no conflicting entries should be identified.

As can be also seen in Figure 1(d), the checker uses the property of non-prescience to define sets of table entries that are indistinguishable. During the extrapolation phase, "new" table entries are generated, and thus it is required to check again for possible conflicts. If all entries of the set $X_{(f,\sigma)}$ have been processed and no collisions are identified, then the checker returns *"Pass"* and it is guaranteed that the generated function table is implementable in race logic.

Theorem 1: (Termination) The checker always terminates for finite function tables. ■

Theorem 2: (Correctness) Given an input function table $\mathbb{T}_f$, if the checker returns "Pass" — $\mathbb{T}_f$ is implementable in race logic. Otherwise, $\mathbb{T}_f$ is not implementable in race logic.

Proof: (sketch) By the definition of space-time algebra if an arbitrary function f has the invariance and non-prescience properties, it is implementable in race logic. We show, that if the checker succeeds in all steps 1-5, $\mathbb{T}_f$ has these properties. *Steps* 1 and 2 verify that $\mathbb{T}_f$ is a syntactically correct function table. *Steps* 3 and 4 fail if and only if the invariance property does not hold. Finally, it is easy to see, that *Step* 5 succeeds if and only if there exists no table entry that uses any "future" information. The step of "extrapolation" is not part of the compatibility check itself, but if enabled it comes with a mechanism for identifying conflicting entries, as *Step* 4 does too, which guarantees the implementability of $\mathbb{T}_f$ in race logic. ■

It should be noted that the properties of invariance and non-prescience are complementary. The order we perform the related checks can significantly affect the execution time of the extrapolation step as its complexity grows exponentially with the number of input variables; using the invariance property first leads to significant performance improvements as rather than the whole table the checker can process only its "surface" entries without any information loss.

IV. From Function Table to Hardware

While the checks above ensure we have function theoretically implementable as race logic, it does not find a good implementation.

In his recent paper [6], J. E. Smith presents a "general" way of implementing space-time functions from their respective function table; each entry of the function table corresponds to a *minterm* and the outputs of all minterms are combined by a MIN function. Our tool extends this idea and provides an automated way to generate efficient race logic designs. More specifically, once the checker returns *"Pass"* and the function table is normalized, the tool generates a list including all minterms and then collapses it returning a reduced set. This set is then processed by our hardware generator that returns synthesizable RTL code [2]. To deliver more efficient designs, the hardware generator pushes all delays to inputs in order to increase the reusability of the "costly" clocked components.

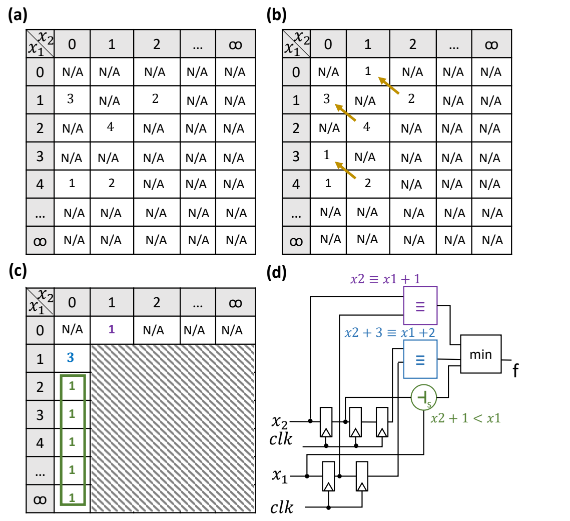


Fig. 2. Panels (a)-(c) show the checks and transformation that our tool performs to an example function table. Panel (d) illustrates its race logic implementation after simplification. To increase the reusability of the "costly" clocked components all delays are pushed to the inputs.

For better understanding, Figure 2 provides a complete step-by-step overview of our tool for a valid example case. More specifically, Panel (a) illustrates a syntactically correct function table. Panels (b) and (c) show how the invariance and non-prescience properties are used to normalize it and extrapolate, respectively. Our checker verifies at east step that no conflicts between various entries exist, creates a full list of the function's minterms and then collapses it for a more compact and concise representation. Finally, its race logic implementation with off-the-shelf CMOS components is given in Panel (d).

V. Future Directions

Race logic uses a temporal number representations, with only a single "wire" and at most one bit-flip required per variable, to achieve highly efficient implementations for certain classes of applications. We present first steps towards a more robust set of tools for design under this new encoding including both an automated checker and logic synthesizer capable of transforming arbitrary function tables into normalized temporal tables (if possible), simplifying their expressions, and generating synthesizable RTL.

Looking forward there are many more advances needed and opportunities for many to contribute:

1) Even our formalization is a simplification of a more natural computational temporal logic that is defined over reals (continuous time) instead of natural numbers (discrete time), i.e. over $R_0^\infty = \{0, \mathbb{R}^+, \infty\}$, where $\mathbb{R}^+$ is the set of positive real numbers. It is unclear how the expressive power changes with the assumptions made.

2) Our checker could be extended to handle unbounded parametric function tables, where table indices and entries are specified mathematically. Even MIN and MAX require infinite sized function table to enumerate and yet can clearly be captured with a single gate. Direct symbolic checking may be a way past this problem.

3) The checking algorithm as it stands now scales poorly with the size of the table (which itself grows exponentially with the number of inputs) and its optimization is even less well understood. Perhaps something akin to a BDD or AIG exists for this logic system that represents a good trade-off between computational complexity and expressiveness.

4) None of these designs exist in vacuum. Thus, the cost of translating between domains (analog, binary, race, bit-serial, one-hot, etc.) must be firmly established and considered in our design process.

Finally, looking beyond tool support, we see new opportunities to apply these concepts to try and fully exploit the computational nature of new materials and devices. As pointed out by Ceze, Hill, and Wenish [1], we need to (1) develop more efficient encodings to make better use of current devices and technology and (2) use new materials that can provide more efficient switching, denser arrangements, and unique computing models. Under temporal logic, computation happens through the interaction of events in time. As new devices offer completely new interactions, we need new tools to help us understand how their computation potential can be fully realized.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 1763699, 1740352, 1563935, and a gift from Cisco Systems.

References

[1] L. Ceze, M. D. Hill, and T. F. Wenisch. Arch2030: A vision of computer architecture research over the next 15 years. *CoRR*, 2016.

[2] J. Clow, G. Tzimpragos, D. Dangwal, S. Guo, J. McMahan, and T. Sherwood. A pythonic approach for rapid hardware prototyping and instrumentation. In *27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[3] A. Madhavan, T. Sherwood, and D. Strukov. Race logic: A hardware acceleration for dynamic programming algorithms. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA, 2014.

[4] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan. Low-cost sorting network circuits using unary processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2018.

[5] J. M. Shalf and R. Leland. Computing beyond moore's law. *Computer*, 2015.

[6] J. E. Smith. Space-time algebra: A model for neocortical computation. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA, 2018.

[7] G. Tzimpragos, A. Madhavan, D. Vasudevan, D. Strukov, and T. Sherwood. Boosted race trees for low energy classification. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2019.