WavePro: Clock-less Wave-Propagated Pipeline Compiler for Low-Power and High-Throughput Computation

Yehuda Kra
Faculty of Engineering
Bar-Ilan University
Ramat Gan, Israel
yehuda.kra@biu.ac.il

Tzachi Noy
Faculty of Engineering
Bar-Ilan University
Ramat Gan, Israel
tzachi.noy@biu.ac.il

Adam Teman
Faculty of Engineering
Bar-Ilan University
Ramat Gan, Israel
adam.teman@biu.ac.il

***Abstract*—Clock-less Wave-Propagated Pipelining is a long-known approach to achieve high-throughput without the overhead of costly sampling registers. However, due to many design challenges, which have only increased with technology scaling, this approach has never been widely accepted and has generally been limited to small and very specific demonstrations. This paper addresses this barrier by presenting WavePro, a generic and scalable algorithm, capable of skew balancing any combinatorial logic netlist for the application of wave-pipelining. The algorithm was implemented in the WavePro Compiler automation utility, which interfaces with industry delays extraction and standard timing analysis tools to produce a sign-off quality result. The utility is demonstrated upon a dot-product accelerator in a 65 nm CMOS technology, using a vendor-provided standard cell library and commercial timing analysis tools. By reducing the worst-case output skew by over 70%, the test case example was able to achieve equivalent throughput of an 8-staged sequentially pipelined implementation with power savings of almost 3×.**

***Index Terms*—Wave Propagation, Clock-less Wave Pipeline, High Throughput, Low Power**

I. Introduction

Clocked sequential pipelines are commonly applied to digital designs in order to increase computation throughput by adding intermediate sampling stages, thereby increasing the overall clock rate. Alternatively, clock-less pipelines [1] eliminate the need for sampling stages by structuring the design such that signal transitions of internal nodes occur at a very narrow and deterministic timing window, thereby enabling wave-propagation of the combinatorial evaluation. In this case, the maximum achievable throughput is determined by the worst max-to-min arrival time difference at the circuit output, rather than by the worst propagation delay. Therefore, balancing the signals to reduce the skew between the output arrival times is the key factor in implementing such a design approach. In theory, this should be able to provide a significant speedup as well as power and area savings. However, when considering complex designs that are composed of thousands of gates, the feasibility of existing skew-balancing algorithms and methods, such as suggested in [1]–[3], is questionable. These methods often neglect or roughly approximate secondary effects in idealized delay-graph representations; however, in reality, these effects accumulate into large gaps, which exceed the base modeling assumptions of the algorithms especially in advanced manufacturing nodes.

In this work, we propose a scalable algorithm for skew-balancing any combinatorial logic netlist to support clock-less wave-propagated pipelines (CWPPs). The proposed algorithm iteratively applies small incremental balancing steps to adjust the skew at the output of each gate in the circuit according to actual delays extraction and timing analysis feedback. By applying a self-timed wave launch and capture mechanism, the algorithm is able to tolerate the variation at advanced nodes. The proposed algorithm is implemented in *WavePro Compiler*, an automation utility that interfaces with industry-standard tools to account for all modeled design effects in scaled technologies. The proposed is demonstrated on a dot-product calculation inside a vector multiplication accelerator By using the WavePro Compiler to generate the dot-product unit in a 65 nm CMOS technology, providing a speedup comparable to a 8-stage pipeline with 3× lower power consumption.

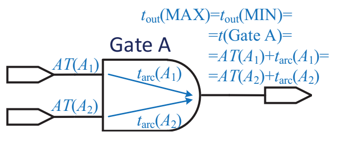
II. Beyond State-of-the-Art

The concept of wave-pipelining was described by Cotten in 1969 [4], observing that for wave-pipelines, the rate at which logic can launched and captured depends on the skew between the longest and shortest path delays rather than the longest path delay through the circuit. Following Cotten, several other groups designed, implemented, and analyzed the concept of wave-pipelining; however, all of the implementations were applying manual design techniques to balance circuits of fixed-delay gates. The seminal work by Wong, et al. [1], was the first to propose an automated algorithm for implementing CWPP suggesting rough delay insertion followed by gates drive strength fine adjustment. The authors conclude that CMOS logic is not well-suited for CWPP, because gate delay depends on the specific input pattern, and therefore, demonstrate their proposed approach with ECL/CML technology gates.

Burleson, et al. [5], provided an extensive review of the work on CWPP that had been published until then and described the sources of delay variation that make the implementation of CWPP challenging and the open problems that were yet to be solved at the time. A later study by Kim and Kim [6] advances the idea of automating CWPP design by first utilizing commercial synthesis tools to generate an initial netlist, and thereafter applying a simplified balancing method to reduce the skew between the outputs. This led to a 20% delay deviation between max and min delays for low-order adders and multipliers. However, their implementation required a custom and limited standard cell library, that leads to a tremendous cost in overall area and power.

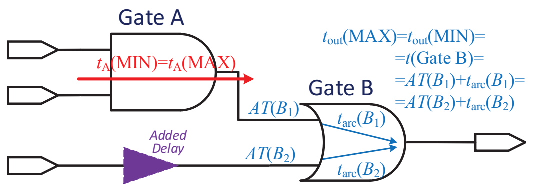
Other than that, very few papers have suggested using CWPP in the recent past, and those that have, have done so on very small and limited circuits. Despite the automation flows proposed in previous studies, they all require special technologies and/or suffer from constraints that limit their scalability. As of now, no solution has been proposed for the integration of CWPP into common ASIC flows.

In this study, we propose a scalable automation algorithm for implementing CWPP on a generic netlist, using CMOS technology, and meeting industry standard sign-off



t_{out}(MAX)=t_{out}(MIN)=
=t(Gate A)=
=$AT(A_1)$+$t_{arc}(A_1)$=
=$AT(A_2)$+$t_{arc}(A_2)$
Gate A
$AT(A_1)$
$AT(A_2)$
$t_{arc}(A_1)$
$t_{arc}(A_2)$

(a) A single gate with equalized input port delays.



(b) An entire circuit with equalized delays.

Fig. 1: Illustration of equalized delays concept.

requirements. We address many of the issues that were either disregarded by previous studies or less relevant in older process technologies. The algorithm is implemented in an automation utility we have developed called *WavePro Compiler* that interfaces with commercial electronic design automation (EDA) tools, applicable for any combinatorial logic block, uses standard CMOS libraries, with the option to improve results by introducing a small number of specialized cells, effectively mitigates process variations and generate a final design that can be verified through conventional ASIC flows.

III. Proposed Skew Balancing Algorithm

In order to achieve maximum speedup, the skew between the arrival times of all paths to the capturing register should be minimal. Considering a circuit with multiple input registers and multiple output registers, albeit lacking any combinatorial loops inside the logic, a very large number of such paths exist. In order to transform the problem into a simple graph, we connect all circuit outputs (inputs of capture registers) to a single virtual output gate. The skew balancing target is to equalize the arrival time (AT) of all paths to the input ports of the virtual output gate. Note that there is no need for a virtual input gate, as all paths originate from a single point – the wave clock signal – which has an AT of zero. This approach has the benefit of also taking the register propagation delay (clk-to-q) into consideration for balancing.

Using an illustrative example, we will now explain the skew balancing concept for a given circuit. Fig. 1(a) illustrates the smallest subset of the circuit – a single gate – including ATs at its input ports and propagation delays through its timing arcs. Neglecting the difference between rising and falling arcs through the gate, we observe that if the paths through all input ports are equal then the minimum and maximum path delays through that gate are equal. This concept is then expanded by taking an additional gate from our circuit ("Gate B" in Fig. 1(b)). If we equalize the path delays through all input ports of Gate B, all paths through the circuit will again have the same latency. By induction, this leads to the observation that in order to equalize the ATs to the virtual output gate, it is sufficient to individually equalize the minimum and maximum ATs to the output port of each intermediate gate. This is the basis for our skew balancing approach, as the concept can be expanded to any combinatorial circuit without loops.

Algorithm 1 Skew Balancing algorithm

```
1: procedure BALANCE STEP($netlist$,$\delta$)
2:     for each $cell$ in $netlist$ do
3:         $t_{\max}(cell) \leftarrow 0$   ▷ initialize max arrival at cell output
4:         $t_{\max}(port) \leftarrow 0$   ▷ initialize max arrival at cell inputs
5:         for each $port$ of $cell$ do
6:             $t_{\max}(port) = AT(port) + t(arc)$
7:             $t_{\max}(cell) = \max(t_{\max}(cell), t_{\max}(port))$
8:         for each $port$ of $cell$ do
9:             $t_{\text{gap}}(port) = t_{\max}(cell) - t_{\max}(port)$
10:            INSERT_DELAY($port, t_{\text{gap}}(port), \delta$)
11: procedure BALANCE CONVERGENCE($netlist, \delta$)
12:     while $skew > target$ do
13:         Perform Timing Analysis
14:         Annotate arrival time for netlist nodes
15:         BALANCE_STEP($netlist, \delta$)
```

Algorithm 1 describes WavePro, the proposed algorithm that slows down faster paths to equalize the skew at every gate. This is achieved by annotating the maximum AT to each gate output port through all its input ports and then adding a delay to the input ports with a earlier AT, such that eventually, all maximum ATs are equal. Note that since all minimum ATs have been delayed, now the maximum and minimum paths are equal, and therefore, we can continue to only refer to the maximum path.

This method of skew balancing is achieved by the Balance Step procedure in Algorithm 1. The algorithm iterates over all the *cells* (logic gates in the netlist), finds the maximum delay to the cell output port of the gate ($t_{\max}(cell)$) and the difference between the maximum delay and the delay through each input port ($t_{\text{gap}}(port)$). A delay is then inserted at each input port, except for the port on the maximum delay path, thereby equalizing the skew at that gate. Note that since delays are added only on the faster paths and the added delay is smaller than $t_{\text{gap}}(port)$, $t_{\max}(cell)$ is not altered.

In an ideal delay model, a single flow iteration may have been sufficient to balance the entire netlist. However, in reality, after a single iteration, the netlist is far from being balanced due to significant side effects arising from the inserted delay cells that effect both the net load and the gate output transitions leading to different actual delays on cell arcs than assumed prior to the iteration. In order to deal with these side effects, we apply an iterative convergence approach, where during each iteration, we fully calculate the timing and skew gaps, but fix only a parameterized percentage of the skew gap per net. These iterations are repeated until the skew gap is negligible. The Balance Convergence procedure in Algorithm 1 describes these iterations, by applying a convergence factor (δ) that represents the partial skew correction amount relative to the calculated skew gap at each step. Our experiments show that a value of $\delta \sim 0.1$ provides a reasonable convergence rate, which roughly means that each iteration fixes approximately 10% of the remaining margin.

IV. Overcoming CWPP Implementation Challenges

In this section, we describe additional details of the WavePro algorithm that deal with challenges that have previously

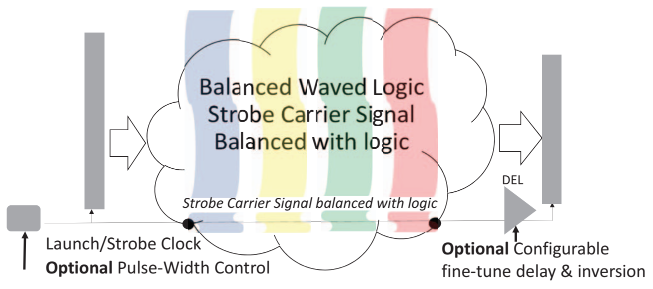


Fig. 2: Self-timed Clocking Approach.

prevented adoption of CWPP into a standard ASIC flow.

A. Delay Application

In addition to delay buffer insertion and discrete gate sizing usable for rough tuning, capacitive or resistive delays can be used during the later convergence algorithm iterations. Capacitive delays can be applied by extending wires as well as adding constant dummy gates to load a net, at the cost of power and area overheads. Resistive delays can be applied by conveying the signal through an always-on pass-gate.

B. Delay Dependence on Input Pattern, Slew, and Load

When presenting the basic assumption for skew balancing in Section III, we noted that we initially ignored the dependence on transition polarity, input slew, and output load. These, of course cannot be disregarded, and are the main reason that previous works required designing special libraries [6] or avoided using CMOS logic altogether [1]. We address these challenges through the iterative process that slowly converges to a solution. By not attempting to fully balance the circuit at each iteration, the algorithm is able to adapt to the actual timing that is extracted, including specific arcs, slew rates, and loads. Furthermore, the WavePro Compiler utility (Section V) interfaces with industry standard tools to extract accurate timing data following each iteration.

C. Dealing with Clock Skew

Since clock skew is a deterministic feature of a timing path, the WavePro algorithm can inherently address it from within the balancing procedure. For the launching registers, the path startpoint is the clock root, and therefore, the clock insertion delay to each register is taken into account in maximum arrival time calculation. For the capture registers, clock skew is just a relative delay between the sampling point at the set of registers, and therefore, this can be extracted from the design and added as an additional delay on the timing arc to the virtual output node.

D. Dealing with Fanout

Delaying a net as part of the balancing algorithm delays all of its fanout ports. Since fanout ports drive different gates, this may add unwanted delay to an already balanced connection, which will need to be re-fixed during the next iteration. To avoid this, an initial signal-splinting phase is applied in order to isolate ports that share the same nets, and therefore, have conflicting timing requirements. Splitting is achieved by buffering such net destinations allowing the delay fixing to take place only on the desired buffered segment. To minimize buffering overhead we first buffer together fanout signals to achieve their common portions that require delays.

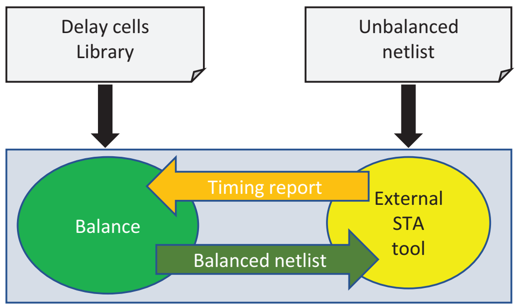


Fig. 3: WavePro Compiler Skew Balancing Flow

E. Process variation tolerance

We propose a self-timed clocking scheme that is adaptive to the operating conditions and to local on-chip variation. Instead of pre-calculating a multiple of clock cycles, after which the data is to be sampled, the capture clock is instead treated as a delay path that is balanced along with the propagated logic wave. Implementation of the proposed self-timing approach is illustrated in Fig. 2. In addition to launching the input data, the launching (or *strobe*) clock is released into the wave-propagation logic and connected to the virtual output node. After balancing, the strobe clock will reach the capture registers at the same point of the maximum path delay through the logic. An additional configurable delay is added to take into account register setup time and enable fine tuning of the sampling point. This also provides a post-silicon remedy to fixing hold violations, which are otherwise considered fatal for conventional sequential design.

F. Sign-off Compliance

Since our implementation is achieved using CMOS standard cell libraries, the methodology for performing timing verification with the proposed algorithm is straightforward. For each process corner, a target clock period is calculated as a full divisor of the delay through the wave circuit. This is defined as the period of the launching clock, and the path through the wave circuit is defined as a multi-cycle path for both setup and hold, which is valid due to the previously described self-timing approach. The self-timed strobe clock enables a clock-data skew validation at each corner, rather than absolute max and min-path calculations. This methodology can be applied within any industry standard timing engine, thereby complying with standard sign-off requirements.

V. The WavePro Compiler

The WavePro algorithm was implemented in a Python based utility, as shown in Fig. 3. A non-balanced netlist is fed in to the utility along with a custom or standard cell library. The netlist is iteratively passed through an (external) timing analysis tool to extract the accurate delay state of the netlist, and the timing information is fed back into the utility. Thereafter, the Balance Step procedure of Algorithm 1 is applied to provide a balance-improved netlist, which is sent back to the timing analysis tool to generate updated timing information. The process is repeated until no skew improvements are achieved. Since the WavePro Compiler interfaces with any commercial timing analysis tool by using standardized formats, the algorithm can be applied during any stage of the design.

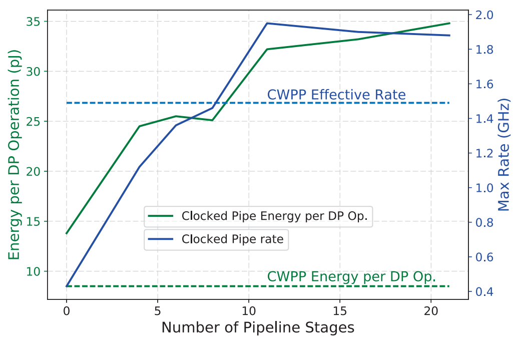


Fig. 4: Energy and throughput per DP operation as a function of re-timed pipe stages, as compared to proposed approach.

TABLE I: Power consumption comparison

Pipeline Architecture	Wave Period[ns]	Giga DP Ops/sec	# Flip-Flops	Seq. Power	Comb. Power	Energy per DP calculation [pJ]
Non-piped Circuit	2.91	0.34	0	0	17.4	**13.8**
Sequential 8-stage Pipe	0.68	1.47	897	13.52	11.62	**25.14**
Sequential 11-stage Pipe	0.51	1.96	1483	21.36	10.81	**32.17**
Balanced Wave	0.67	1.49	0	0	10.1	**8.5**

VI. Test Case Demonstration

A dot-product (DP) accelerator for vector multiplication was implemented for demonstration of the algorithm. The accelerator fetches a pair of 64-bit vectors, each composed of eight 8-bit integer values, and outputs the dot-product result at a rate of a full calculation per wave-period.

The unit was first implemented with a sequential pipe for comparison, using a commercial tool re-timing feature to vary the number of pipeline stages in the resulting implementation. Fig. 4 shows the maximum frequency as a function of the number of inserted pipeline stages (blue line), reaching a peak of 1.95 GHz for an 11-stage pipe. In addition, the figure shows the energy cost per dot-product operation, which is over 32 pJ for the maximum achievable frequency.

The unit was then implemented using the WavePro Compiler, reducing the skew by over 70%, with the final balanced netlist enabling a data launch rate of 1.49 GHz. The dotted lines on Fig. 4 represent the effective rate achieved through CWPP (blue) and the energy cost (green). The cross-over point between the solid blue and dotted blue lines, representing where the sequential pipeline provides similar throughput as the CWPP implementation, occurs for an 8-stage pipeline. The results are summarized in Table I, showing that the CWPP approach provides almost a 3× reduction in power, as compared to the equal throughput 8-stage pipe, which also requires almost 900 additional registers.

We have also carried out logic simulations with back annotated delays, while parametrizing the data launching rate (wave-period) and the strobe delay (output capture time). Since the strobe clock is self-timed, the difference in strobe delay is adjusted by the configurable margin (see Fig. 2). The pass and fail regions are shown in the Shmoo Plot of Fig. 5. While the circuit can be operated anywhere within the green area, best practice would be to adjust the margin to be in the middle of the passing region. Note that this ability to adjust the margin post-silicon is, in fact, a unique method to overcome hold violations.

We have analyzed the effect of local variations by applying

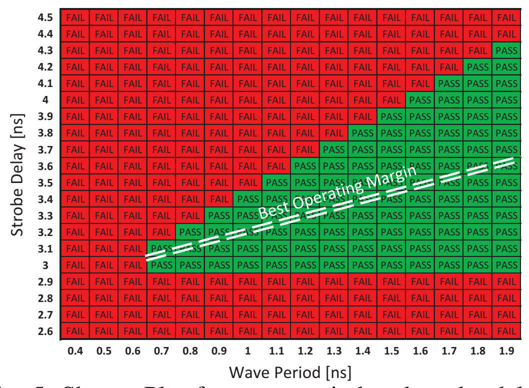


Fig. 5: Shmoo Plot for wave period and strobe delay.

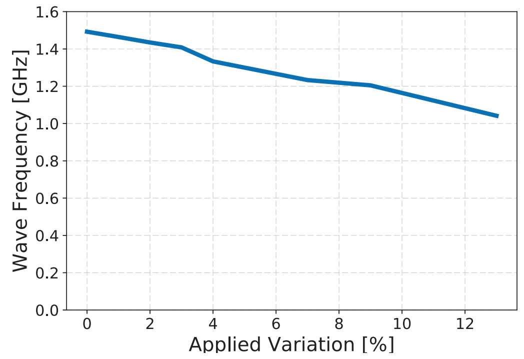


Fig. 6: Process Variation Sensitivity

a random positive or negative derating factor per timing arc in the design and checked the impact on the minimal functioning wave period. Results are captured in Fig. 6, which indicates that for 10% random derating which is commonly applied by industry practice we lose about 25% in performance, but still maintain the 3× power saving range.

VII. Conclusions

In this paper, we proposed WavePro – a utility for implementing clock-less wave-propagated pipelining on any combinatorial logic block that is a candidate for pipelining. The WavePro utility is based on a novel algorithm that iteratively delays the propagation of signals through logic gates in order to minimize the skew at the outputs of the circuit. By using standard cell libraries and interfacing with commercial timing analysis and physical implementation tools, WavePro is compliant with standard ASIC flows and takes into account parasitic effects and variations of modern process technologies. For demonstration, we used WavePro to implement a dot-product accelerator with a 65 nm standard cell library, achieving throughput equivalent to an 8-stage pipeline with a 3× power reduction.

References

[1] D. C. Wong *et al.*, "Designing high-performance digital circuits using wave pipelining," *IEEE TCAD*, 1993.

[2] O. Hauck and S. Huss, "Asynchronous wave pipelines for high throughput datapaths," *IEEE International Conference on Electronics*, 1998.

[3] S. Sethupathy *et al.*, "Logic restructuring for delay balancing in wave-pipelined circuits: an integer programming approach," in *SYNASC*, 2005.

[4] L. Cotten, "Maximum-rate pipeline systems," in *SJCC*, 1969.

[5] W. P. Burleson *et al.*, "Wave-pipelining: a tutorial and research survey," *IEEE TVLSI*, vol. 6, no. 3, pp. 464–474, 1998.

[6] W. Kim and Y. Kim, "Automating wave-pipelined circuit design," in *IEEE design and test of computers, 20(6)*, pp. 51–58, IEEE, 2003.