Low-Voltage Low-Overhead Asynchronous Logic

Akshay Sridharan, Carl Sechen and Roozbeh Jafari
Department of Electrical Engineering, The University of Texas at Dallas
Richardson, TX, USA
akshay.sridharan@utdallas.edu, carl.sechen@utdallas.edu, rjafari@utdallas.edu

***Abstract*— A new delay-bounded asynchronous logic technique aimed at maximizing reliability at very low voltages is proposed. Compared to previous asynchronous logic approaches, the area and nominal delay overheads are small. Conventional standard cell libraries and conventional logic synthesis tools are used. The bounding delay elements used by the asynchronous controller feature programmable delays that are initially set based on static timing analysis. However, the delay elements are updated on-the-fly during actual operation of the circuit, resulting in strong resiliency even at low voltages and with extreme variations. Several benchmark circuits were implemented with the new asynchronous design flow using the 45nm TI process. Monte Carlo analysis demonstrates the expected resiliency. Compared to the equivalent synchronous circuits, the asynchronous versions have area overheads averaging 40%, although much smaller for large circuits. Nominal delay overheads average about 10%.**

Keywords*— *Asynchronous logic, de-synchronization, low voltage

I. INTRODUCTION

Of particular interest today are wearable computers that could rely at least in part on energy scavenging. In this environment, energy consumption must be minimized. In such applications, the required throughput (delay) is largely well established. In such a case, we have demonstrated that the energy efficiency is optimized when VDD is lowered as much as possible to just meet the throughput requirements. In other words, due to the quadratic nature of dynamic energy and the fixed throughput requirement, using a higher VDD to quickly finish the computation and then turning off the supply by means of sleep transistors to reduce leakage consumes more energy overall than lowering VDD to the point where there is just enough time to finish the computation.

The minimum possible power supply voltage for the standard cells (and transistors) to function properly is typically in the 150-200 mV range. For wearable computers, power supply voltages in this minimum range are sufficient to meet the throughput requirements. Several recent investigations have demonstrated the throughput requirements of, for instance, movement monitoring applications using wearable computers which range between a few hundred to a few thousand instructions per second [1], [2].

Conventional synchronous logic seems difficult to apply in such an environment. First, for voltages in the sub-threshold regime, delays are exponentially related to both threshold voltages and the power supply voltage. For example, for just 100mV of variation in Vt for a typical circuit path operating at 200mV, we have performed Montecarlo analysis for a 45nm process that shows a delay variation range of almost 10X above the nominal. The substantial extra guardband needed for a synchronous logic implementation would appreciably increase energy consumption, without any guarantee that a particular guardband would be sufficient.

Second, power supply variations may be large for such wearable computers. In order to keep the supply voltage in a somewhat narrow range, a power hungry and large voltage regulator would be required, in which the regulator also needs large external passive components. The use of such a regulator is not possible since miniaturization and a single-chip solution is the ultimate objective for the wearable computers. In fact, a minimalistic voltage regulator amendable to a single-chip solution may result in power supply voltages that have appreciable variation, as high as 20%. A synchronous implementation of the wearable computers would require a yet much larger degree of guardband to accommodate such a large power supply variation. Again, there would not be any assurance that a particular guardband would be enough in light of unpredictable energy availability due to scavenging, combined with the minimalistic voltage regulator.

The need to achieve robustness in light of extreme PVT variations faced by the energy-scavenging wearable computers, and still achieve very low power operation, all while avoiding the huge guardbands faced by synchronous logic, opens a new door for deployment of an effective asynchronous logic style. The new asynchronous logic approach avoids the need for a significant guardband and thus obtains close to the most performance possible from sub-threshold operation. However, it must be noted that the asynchronous logic methodology must utilize conventional EDA tool flows and libraries, as there is little interest from the commercial world in departing from these.

This work presents a new bounded-delay clockless logic methodology that utilizes conventional EDA tool flows and libraries. A bounded-delay datapath approach to asynchronous logic design is employed where each stage is not a single gate or a single logic level as in many previous approaches, but rather entire logic blocks as generated by commercial (synchronous) synthesis tools. This allows two disadvantages of many previous asynchronous logic approaches to be over-come: namely, high area overhead

National Science Foundation and STARnet.

and high delay overhead. The challenges with bounded-delay reliability are over-come by using an on-chip assessment of the adequacy of the initially set bounded delays (via static timing analysis) for each logic block both during a test mode and during actual operation.

II. RELATED WORK

There are still drawbacks that have slowed the popularity of asynchronous logic [3]-[7]. A key obstacle for many approaches is the lack of widely available CAD tools, such as synthesis tools, and cell libraries to support asynchronous design, since many prior approaches require customized cells.

A good review of prior work in asynchronous logic circuits has recently appeared [8]. Null Convention Logic [3] has large area overhead due to its gate-level handshaking approach and is extremely slow compared to state-of-the-art synthesized synchronous logic. A number of very high throughput fine-grain pipelining techniques based on asynchronous logic were developed [9]- [12]. However, our application does not require such ultra-high throughput. Special synthesis tools and cell libraries are generally needed to implement these circuits. There has been prior work on asynchronous logic capable of subthreshold operation [13]-[17].

Our work has some similarity to the "desynchronization" approach developed by Cortadella and Andrikos [18], [19]. While this approach captures a few of the elements we are seeking, namely to utilize conventional EDA flows while keeping area and delay overhead low. None of the prior work considers reliability to the degree addressed here.

III. PROPOSED ASYNCHRONOUS PROTOCOL

The new approach to asynchronous logic design starts with standard Synopsys or Cadence synthesis, essentially pretending that the design will be synchronous, and using a conventional standard cell library. The logic stages are identified by the following flip-flop levelization algorithm:

Algorithm FF_Levelization

1. Flip-flops (FFs) that receive an input through a combinational path from the primary inputs (PIs) are labeled 'level 1'.
2. All flip-flops that have a path from level 1 FFs but do not have combinational paths from the PIs are labeled 'level 2'.
3. For i starting at 2 until all FFs are labeled:
   - All FFs reachable from level i FFs are labeled 'level i+1'
   - If a FF to be labeled i+1 already has a label (less than or equal to i), note this as a feedback loop.
   - All combinational paths between level i FFs and level i+1 FFs are labeled 'level i+1 logic'.
   - If there exists a combinational path from level i to level x (where x less than or equal to i) then the combinational path is labeled as 'level x feedback logic'.

After the levelization is complete, the flip-flops at a given level (*e.g.*, level x) will receive inputs from combinational paths emanating from one or more other levels. Fig. 1a shows a typical example where the FFs at level x receive inputs emanating from FF levels i, j, and k. The depiction in Fig. 1a is termed a generalized asynchronous logic stage. Exactly one programmable bounding delay element and one asynchronous handshaking controller is used for such a logic stage. The delay of the

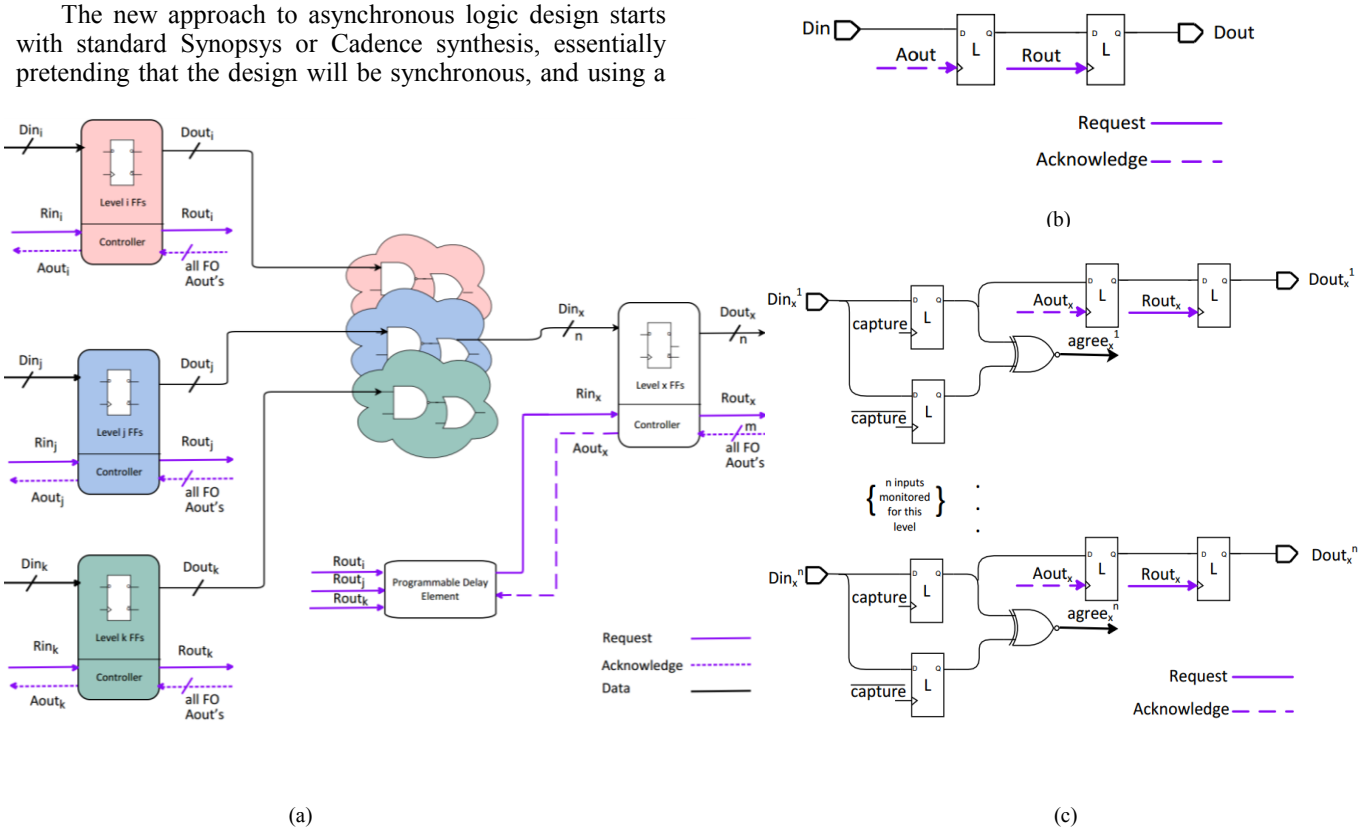

Fig. 1. (a) Example of our generalized asynchronous logic stage; (b) Non-monitored path; (c) Monitored path with dual mode check

bounding delay element must represent the longest possible delay from any of the FFs at levels *i, j,* and *k* to the FFs at level *x*, and in the new approach significant effort is made to assure the delay bound is adequate despite the possibility of very significant process, voltage and temperature (PVT) variations and sub-threshold delay variations.

After a stage, such as that shown in Fig. 1a, has received all of its requests ($Rout_i$, $Rout_j$, and $Rout_k$ in Fig. 1a), then *REQ* (for the overall stage request) goes high to initiate (in effect) propagation through the programmable delay. The programmable delay is generated by counting rising edges arising from a local oscillator (again, one per FF level), as shown in Fig. 2b. When a *request* (or *Rout*) is received from all of levels *i, j* and *k* in Fig. 1a, the local oscillator in Fig. 2b is enabled (by signal *EN*). The programmable delay counter in Fig. 2b then counts the number of rising edges among signals *capture* and *capture-bar*. When that count matches the desired programmed count for level *x*, a request is input to the level *x* controller (Rin_x in Fig. 1a). Note that the flip-flops in Figs. 2a and 2b that conceptually always have a high *D* input are used for edge detection.

In general, each FF receiving actual data at a level (such as level *x* in Fig. 1a) is converted into its respective two latches, similar to the desynchronization method [18], [19], as illustrated in Fig. 1b. However, outputs that could possibly be critical for a stage are outfitted with additional so-called dual monitoring circuitry as shown in Fig. 1c. We run static timing on each logic stage to determine which outputs in each stage could possibly ever yield the worst-delay path (given extreme variations). Only those outputs are then equipped with the dual-mode monitoring hardware, saving overhead. The dual-latch capturing process has some similarity to the (single bit) Razor flip-flop developd at the University of Michigan [20], [21], but extends the concept.

As shown in Figs. 1b and 1c, each data line being input into level *x* (if it could possibly be critical under any type of variations) has its data captured by two latches, controlled by *capture* and *capture-bar*, whose rising edges are roughly 3 gate delays apart. As shown in Fig. 2a, if all of the monitored data inputs into level *x* latch the same data on *capture* and *capture-bar* (indicated by all of $agree_x^1$ through $agree_x^n$ being high) and if the request input (Rin_x) was received, an acknowledge-out ($Aout_x$) is output from level *x*. This indicates that all inputs to level *x* have settled to their final values and that new outputs from levels *i, j*, and *k* can start to be produced. To enable that and to avoid hold problems, upon receipt of a high $Aout_x$, the data input to level *x* is captured by the latches controlled by $Aout_x$ in Figs. 1b and 1c. Then, after acknowledges are received from all of the FF levels fanning out from FF level *x* ($FO\_Aout_x^1$ through $FO\_Aout_x^m$ in Fig. 2a), level *x* can begin processing new data outputs ($Dout_x$ in Fig. 1a). This commences with the issuance of $Rout_x$ as shown in Fig. 2a and Fig. 3.

As mentioned previously, after a stage has received all of its requests ($Rout_i$, $Rout_j$, and $Rout_k$ in Fig. 1a), *REQ* goes high. This high *REQ* resets the edge detectors for the next round of receiving incoming requests. *REQ* also resets the edge detector for the acknowledge signal that will go high after the programmable delay and after all data line captures agree. This protocol allows this asynchronous logic approach to properly implement a shift register, as well as any type of circuit generated by synchronous synthesis.

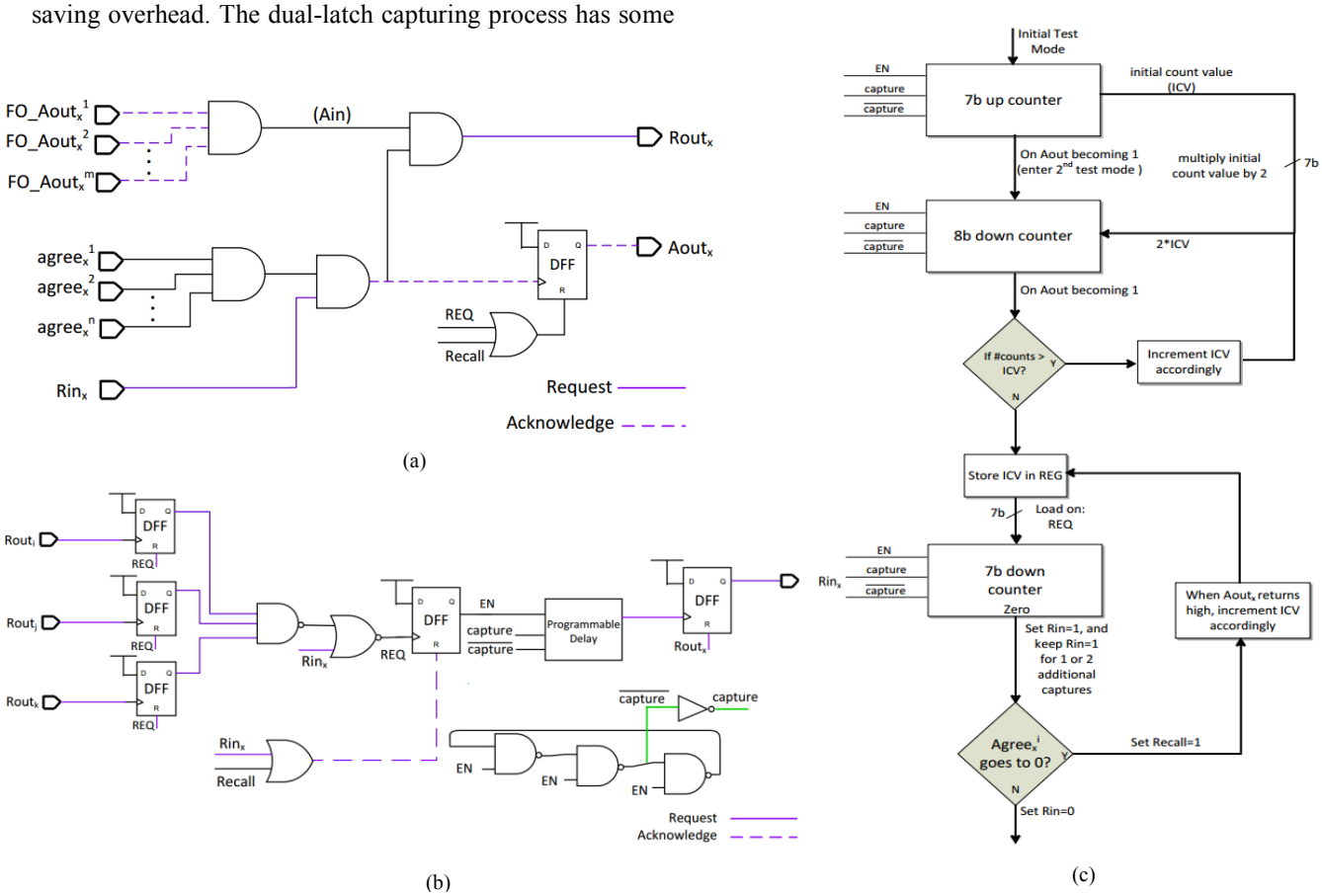

Fig. 2. (a) Controller; (b) Programmable Delay Element (PDE); (c) Delay bounding flow.

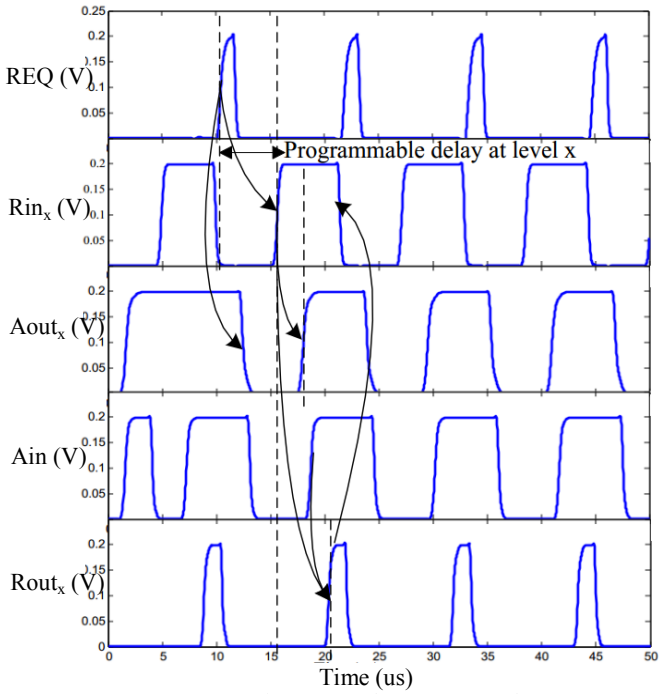

Fig. 3. Asynchronous Protocol.

IV. PROGRAMMABLE DELAY AND RESILIENCE

Perhaps the most critical aspect of the new asynchronous logic approach is the degree to which the proper (programmable) delay bound is ascertained for each generalized asynchronous logic stage. The objective is to implement the programmable delay using a down counter, which counts the number of rising edges among both *capture* and *capture-bar*, and when the count value reaches zero, issue a request into the next FF level (such as level *x* in Fig. 1a). However, the key challenge is to find the correct initial value stored in the down counter so that when it reaches zero it is for sure the case that all data inputs to FF level *x* have correctly settled to their final values.

The first attempt at a proper initial value to be placed in the down counter is set based on static timing analysis (using nominal conditions) to determine the input vector pair (among levels *i, j* and *k* in Fig. 1a to level *x*) that excites the longest delay path for that stage. On power up, this path is excited by a corresponding vector pair that is stored in hardware. We then tune the bounding delay down counter to match this delay, since we know when the correct output arrives on the critical path. While this may not be the longest path post fabrication, or may not be the longest path due to PVT, aging or sub-threshold related variations, it will be sufficiently close that the several subsequent bounding delay updating processes will successfully adapt the bounding delay to the actual worst-case delay during on-the-fly testing as well as during actual operation.

The first attempt at determining the initial count value (ICV) is shown in the first box in Fig. 2c. An up-counter is used to count rising edges of *capture* and *capture-bar* until the correct nominal worst-case output value appears. This is done for each asynchronous logic stage in parallel.

After completion of the above first step in delay bounding, the asynchronous design is prepared for on-the-fly testing using actual primary input data vectors. In order to be sure that the delay bounds are more than adequate, the ICVs determined in the first step are doubled. Although other powers of two are easily supported, we have found that doubling is quite adequate in practice, given that the local oscillator (Fig. 2b) is embedded within the logic stage in the layout so that it experiences a very similar voltage and temperature conditions. While doubling the ICVs slows down operation (throughput) of the system in this so-called second test mode, it can be observed how far down the (now) 8b down counter counts before all of $agree_x^1$ through $agree_x^n$ become high (using Fig. 1a and Fig. 2a as an example). As indicated in the second box in Fig. 2c, should the 8b down counter initialized with 2*ICV count down more than half way, that is, should it count more than ICV rising edges of *capture* and *capture-bar* before *Aout* ($Aout_x$ for the example of Fig. 1a) is issued high, then the ICV needs to be incremented by the number of excess rising edges counted (which is equal to ICV minus the 8b down counter value at the time *Aout* is issued high).

After running the asynchronous design in this second test mode for an arbitrary amount of time, the ICVs for each stage are stored in a register (REG, as shown in Fig. 2c) used for normal operation. During normal operation, any time that REQ (request) is raised, the register REG for a given stage is loaded into the 7b down counter.

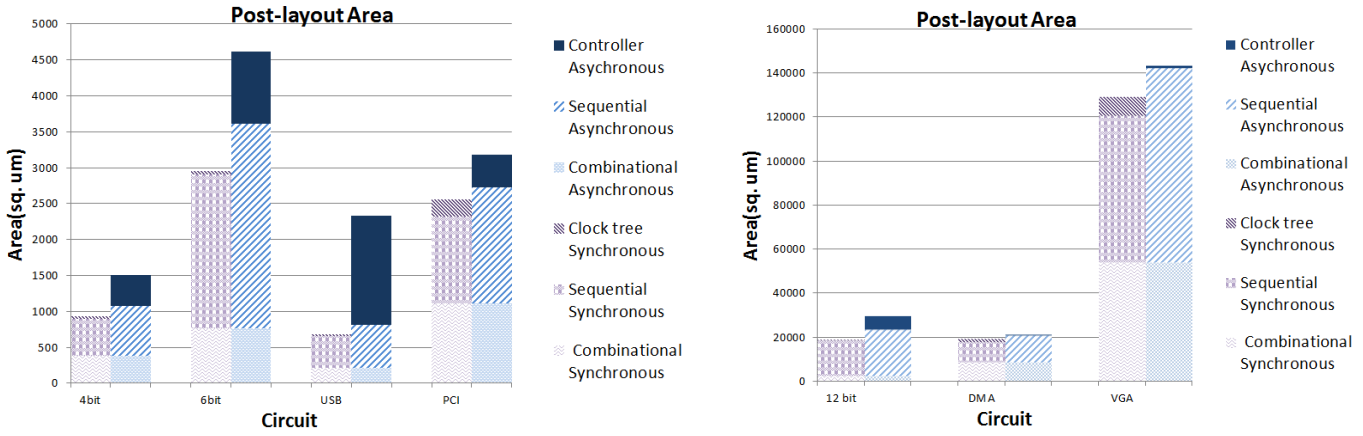

Fig. 4. Post-layout area of the DTW and ISPD12 benchmarks. The portions of the area due to combinational, flip-flops (sequential) and clock tree circuitry for the synchronous circuits are shown, as are the areas due to the controllers, flip-flops and combinational circuitry for the asynchronous circuits.

TABLE I. SUMAMRY OF DTW BLOCKS AND ISPD12 BENCHMARKS

Property	Circuit						
	DTW Blocks			ISPD 2012 Benchmarks			
	DTW4	DTW6	DTW12	USB	PCI	DMA	VGA
Levels	10	28	192	10	3	2	6
Synchronous Cell Count	457	1145	5382	360	1606	13649	74885
Post-layout Area Overhead (%)	63	53	58	106	21	4.4	13
Throughput Delay Overhead (%)	9.8	7.7	7.8	12.6	17.4	6.7	2.5

However, further dynamic resiliency optimization is applied, as shown by the last part of Fig. 2c. After a request to the next stage is issued (*e.g.*, $Rout_x$ in Fig. 2a), there is time to do 1 or 2 additional "verification" captures after that, before new data is actually released into the current stage (*i.e.*, before having to worry about 'hold time' issues). If an inconsistency in the data-capturing latch pairs (*e.g.*, in Fig. 1b one of the $agree_x^i$ signals goes low) occurs within the period of 1 or 2 additional captures, $Aout_x$ is returned to 0 and the request $Rout_x$ is recalled (returned to zero). This is implemented by resetting the flip-flop in Fig. 2a that produces $Aout_x$, and also resetting the flip-flop that generates the enable EN in Fig. 2b for the FF levels that received $Rout_x$. Note that $Rout_x$ is automatically reset to 0 by the circuitry in Fig. 2a (the falling $agree_x^i$ signal).

After some number of additional captures, when all of the $agree_x^i$ signals match high again in Fig. 2a, $Aout_x$ is re-raised, as is $Rout_x$. The ICV is incremented by the number of additonal captures required so that this problem will not re-occur. The raising of $Rout_x$ also results in resetting the Rin_x signal (the output of the programmable delay in Fig. 2b).

V. EXPERIMENTAL RESULTS

Verilog designs for 4b, 6b and 12b dynamic time warping (DTW) signal processing blocks, respectively, DTW4, DTW6 and DTW12, were synthesized using Synopsys' Design Compiler using a TI 45nm process. DTW has been shown to be a quite effective and powerful technique for processing time series data acquired from sensors in wearable computers [22], [23]. These three blocks are used in a wearable computing platform operating at VDD = 200 mV. Several ISPD 2012 benchmarks were also synthesized in the same way.

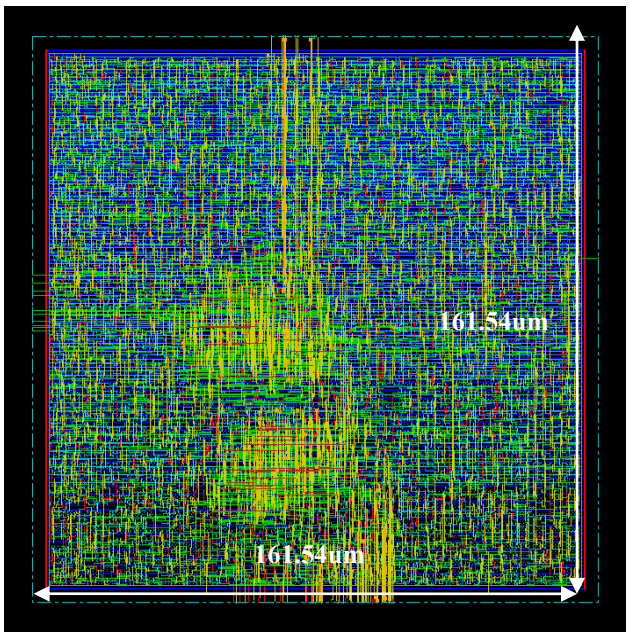

Fig. 5. Post P&R layout view DTW12

The number of FF levels and the area overhead is shown in Table 1. The components of the post-layout area for the synchronous and asynchronous circuits are shown in Fig. 4. Most of the FF levels in the DTW blocks represent shift registers. For shift register chains, the flip-flops are decomposed as in Fig. 1b as we do not need a programmable delay element for these flip-flop chains (there is enough delay in the hand-shaking circuitry). Thus, the so-called agree signals in Fig. 2a are not used. In this case, the AND gate producing $Rout_x$ in Fig. 2a has as inputs the *fanout Aout's* and the request in (Rin_x). The area overhead is unusually large for the DTW blocks due to the predominance of shift registers. Fig. 5 shows the post-layout view of the DTW12 block.

The DMA benchmark had the minimum overhead of 4.4% and the USB had the maximum of 106%. The very unusual area overhead for the USB benchmark is because it has only an average of 36 combinational logic cells per FF level. The asynchronous circuit has low area overhead if the number of combination cells is large compared to the number of FF levels. This is readily apparent in Table 1. The area of the latches in the TI 45 nm standard cell library is 66% of the area of the flip-flops.

Fig. 6 shows the throughput delay overhead for the new asynchronous approach. For the seven benchmarks, the overhead ranges from 2.5% to 17.4%, with an average of about 10%. This analysis was conducted for a supply voltage of 1.1V under nominal conditions. For truly sub-threshold operation, the asynchronous approach has an appreciable advantage in throughput since the guardband

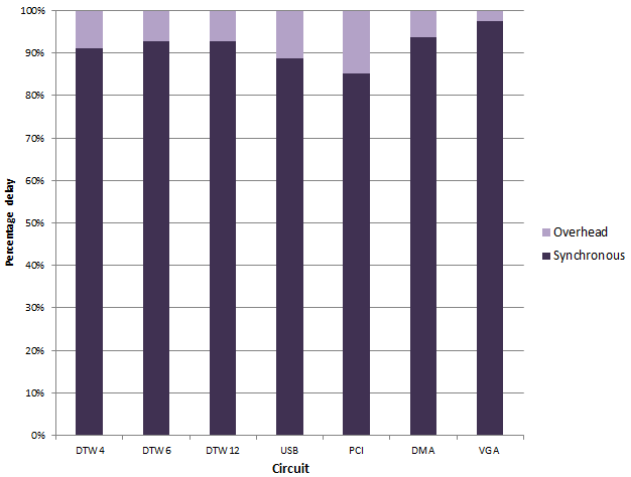

Fig. 6. Throughput Delay Overhead

needed for a synchronous implementation is at least 10X above the nominal delay for that supply voltage.

Extensive Monte Carlo analysis was performed for the DTW blocks using the process mismatch models provided for the Texas Instruments 45nm process, as well as for the various slow and fast process corners. For each run, the on-the-fly determined bounding delay element register values were sufficient to assure continuously correct outputs. It was observed that the stage delays vary substantially over the Monte Carlo runs. An example is shown in Fig. 7 for the first FF level of the DTW4 block. The frequency distribution of the initial count values (or ICVs in Fig. 2c) for this level for 1000 runs is shown in Fig. 7. Each count value is equivalent to roughly 3 gate delays in the local oscillator (shown in Fig. 2b).To demonstrate the resiliency of our design we induced severe supply variations; VDD was a sinusoid signal where the period of oscillation was longer than the worst-case path delay path, and where the peak-to-peak variation was ±10% (or 27.5mV) of the power supply voltage (275mV). In all cases, the circuits operated correctly.

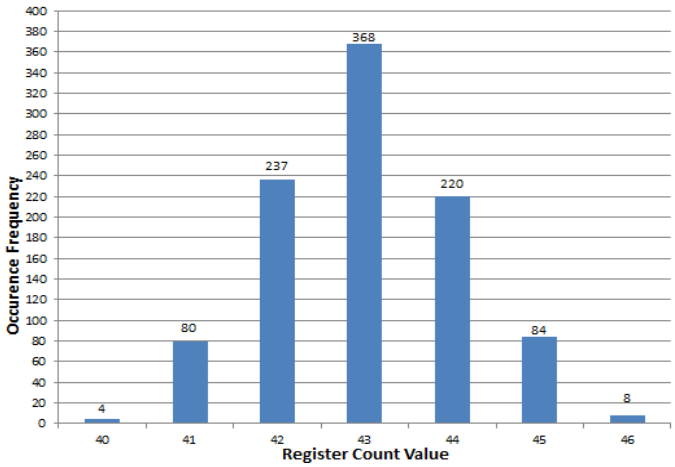

Fig. 7. ICV distribution for DTW4 FF level 1 for 1000 Monte Carlo runs.

VI. Conclusion

A new delay-bounded asynchronous logic technique aimed at maximizing reliability at very low voltages has been proposed. Compared to previous asynchronous logic approaches, the area and nominal delay overheads are small. Conventional standard cell libraries and conventional logic synthesis tools are used, as the new asynchronous logic circuits are generated in a so-called de-synchronization manner. That is, the circuit is transformed to an asynchronous version in a plug-compatible manner, meaning that the asynchronous circuit functions in exactly the same way as the synchronous version, with flip-flops in the same positions within the circuitry. As a consequence, the same built-in self-test circuitry is used. The bounding delay elements used by the asynchronous controller feature programmable delays that are updated during on-the-fly operation of the circuit, resulting in strong resiliency even at low voltages and with extreme process, voltage and temperature variations. Several benchmark circuits were implemented with the new asynchronous design flow using the 45nm TI process. Monte Carlo analysis demonstrates the expected resiliency. Compared to the equivalent synchronous circuits, the asynchronous versions have area overheads averaging 40%, although much smaller for large circuits. Nominal delay overheads average 10%.

Acknowledgment

This work was supported in part by the National Science Foundation, under grants CNS-1150079 and CNS-1138396, and the STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding organizations.

References

[1] Mohammad-Mahdi Bidmeshki, and Roozbeh Jafari, “Low Power Programmable Architecture for Periodic Activity Monitoring,” *Proc. of ICCPS*, April 2013.

[2] Reza Lotfian and Roozbeh Jafari, “An Ultra-Low Power Hardware Accelerator Architecture for Wearable Computers Using Dynamic Time Warping,” *Proc.* of *DATE*, March 2013.

[3] Karl M. Fant and Scott A. Brandt, NULL Convention LogicTM System, US patent 5,305,463, April 19, 1994.

[4] A.M. Lines and S. Fairbanks, “GasP: a minimal FIFO control,” *Proc. of ASYNC*, 2001, pp. 46 – 53.

[5] S. Schuster et al., “Asynchronous interlocked pipelined CMOS circuits operating at 3.3-4.5 GHz,” *Proc. ISSCC*, 2000, pp. 292 – 293.

[6] R.O. Ozdag and P.A. Beerel, “High-Speed QDI Asynchronous Pipelines,” *IEEE Proc. of ASYNC*, 2002, pp. 13 – 22.

[7] I.E. Sutherland, “Micropipelines,” *Communications of the ACM*, vol. 32, no. 6, 1989, pp. 720 – 738.

[8] Jens Sparso, Asynchronous circuit design: A Tutorial, Technical University of Denmark, 2006, http://www.imm.dtu.dk/~jsp/.

[9] G.N. Hoyer, G. Yee, and C. Sechen, “Locally Clocked Pipelines and Dynamic Logic,” *IEEE Trans. on VLSI*, vol. 10, 2001, pp. 58 – 62.

[10] S.B. Furber and P. Day, “Four-phase micropipeline latch control circuits,” *IEEE Trans. VLSI*, vol. 4, no. 2, 1996, p. 247.

[11] Montek Singh and Steven M. Nowick “MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines,” *IEEE Trans. on VLSI*, vol. 15, no. 6, 2007, pp. 684 – 698.

[12] T.E. Williams and M.A. Horowitz, “A zero-overhead self-timed 160ns 54b CMOS divider,” *Proc. ISSCC*, 1991, p. 98 .

[13] Ik Joon Chang, Sang Phill Park and Roy, K., "Exploring Asynchronous Design Techniques for Process-Tolerant and Energy-Efficient Subthreshold Operation," *IEEE Journal of Solid-State Circuits*, vol. 45, no.2, pp. 401,410, Feb. 2010.

[14] Crop, J., Fairbanks, S., Pawlowski, R. and Chiang, P., "150mV subthreshold Asynchronous multiplier for low-power sensor applications," *Proc. VLSI-DAT,* p. 254, April 2010.

[15] Xiaofei Chang, Yong Lian, "A Quasi-Delay-Insensitive Dual-Rail low-pass filter working in subthreshold region," *Proc.* of *BioCAS,* pp. 214-217, Nov. 2010.

[16] A. Arthurs, Jia Di, "Analysis of ultra-low voltage digital circuits over process variations," *Proc. of SubVT, 2012 IEEE* , pp. 1-3, Oct. 2012.

[17] Tsung-Te Liu, Louis Alarcon and Jan Rabaey. "Towards an Ultra Low-Energy Computation with Asynchronous Circuits," Poster, BWRC Winter 2012 Retreat, Jan. 2012.

[18] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, “Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications”, *IEEE Trans. on CAD*, vol. 25, no. 10, 2006 , pp. 1904 – 1921.

[19] N. Andrikos, L. Lavagno, D. Pandini, and C. Sotiriou, “A Fully-Automated Desynchronization Flow for Synchronous Circuits,” *Proc. of DAC*, June 2007, pp. 982 – 985.

[20] T. Austin, D. Blaauw, T. Mudge, K. Flautner, "Making typical silicon matter with Razor," *Computer*, vol. 37, no. 3, pp. 57- 65, Mar 2004.

[21] D. Ernst et al., "Razor: circuit-level correction of timing errors for low-power operation," *IEEE Micro,* vol. 24, no. 6, p. 10, Nov. 2004.

[22] Tormene, P., Giorgino, T., Quaglini, S. and Stefanelli, M., “Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation,” *Artificial Intelligence in Medicine*, Vol. 45, January 2009, Pages 11-34.

[23] Liu, R., Zhou, J., Liu, M., Hou, X., “A wearable acceleration sensor system for gait recognition,” ICIEA, 2007, pp. 2654-2659.