High Performance Asynchronous ASIC Back-End Design Flow Using Single-Track Full-Buffer Standard Cells

Marcos Ferretti, Recep O. Ozdag, Peter A. Beerel
Department of Electrical Engineering Systems
University of Southern California
Los Angeles, CA 90089 – USA
ferretti@usc.edu, ozdag@usc.edu, pabeerel@usc.edu

Abstract

This paper presents a back-end design flow for high performance asynchronous ASICs using single-track full-buffer (STFB) standard cells and industry standard CAD tools to perform schematic capture, simulation, layout, placement and routing. This flow is demonstrated and evaluated on a 64-bit asynchronous prefix adder and its test circuitry. The STFB standard cells provide low latency and fast cycle-times at the expense of some timing assumptions. This paper demonstrates that, by controlling top-block sizes and/or wire length within the place & route flow, ultra-high-performance circuits can be automatically designed. In particular, in the TSMC 0.25 μm process our post-layout STFB standard-cell 64-bit asynchronous prefix adder requires 0.96 mm², offers a latency of 2.1 ns, has a throughput of 1.4 GHz, and operates at five process corners as well as a wide-range of temperatures and voltages.

1. Introduction

As CMOS manufacturing technology scales into deep and ultra-deep sub-micron design, problems with clock skew, clock distribution, on-chip variations, and on-chip communication in high-speed synchronous designs are becoming increasingly difficult to overcome [1], warranting the exploration of alternative design approaches. In particular, asynchronous design is emerging as an increasingly viable alternative.

Among the numerous asynchronous design styles being developed [3], template-based fine-grain pipelines have demonstrated very high performance [5][6][7][8][9]. Template-based approaches also have the advantage of removing the need for generating, optimizing, and verifying specifications for complex distributed controllers, which is both difficult and error-prone [2], the automation of which is an area of significant research [17].

Various templates tradeoff latency, cycle time, and robustness to timing. The most robust is the quasi-delay-insensitive (QDI) templates proposed by Lines [5]. One of most aggressive is the ultra-high-speed GasP [7]. GasP offers high throughput but requires a bundled data design style that involves additional timing margins and assumptions that must be ensured and verified during physical design. In addition, the delay elements needed to address these timing assumptions often increase the forward latency of the blocks, which may significantly impact overall system performance. We recently proposed the single-track full-buffer (STFB) templates [10] which use 1-of-N data encoding to provide a practical tradeoff between performance and robustness. It uses two-dimensional pipelining to achieve similar throughput to GasP with fewer timing assumptions and lower latency.

In this paper, we propose a back-end design flow to support the automated design of STFB-based functional blocks and/or chips with standard commercial tools. In fact, to our knowledge, other back-end flows for template-based fine-grain pipelines involve more labor-intensive semi-automated full-custom flows [18][19] or have adopted the use of existing low-performance standard cell libraries [20]. Moreover, our STFB library and the QDI library utilized in a high performance sequential decoder chip [21] are among the first standard-cell libraries for template-based designs that have been made available (through the MOSIS Educational Program) [22], allowing more widespread adoption of this technology.

This paper demonstrates and evaluates this standard-cell-based flow on a 64-bit asynchronous prefix adder and its test circuitry. In particular, in the TSMC 0.25 μm process our STFB standard-cell 64-bit asynchronous prefix adder requires 0.96 mm², offers a latency of 2.1 ns and has a throughput of 1.4 GHz. Moreover, post-layout simulations show that it operates safely at five process corners as well as a wide-range of temperatures and voltages.

The remainder of this paper is organized as follows. Section 2 reviews asynchronous channels and STFB templates. Section 3 presents details of the transistor-level design of the STFB cells. Section 4 describes the asynchronous library and ASIC design flows. Section 5 details the proposed test chip. Section 6 presents simulation results, Section 7 discusses area, cycle time, and latency comparisons with QDI and synchronous counterparts, and Section 8 draws some conclusions.

2. Background

This section reviews asynchronous channels and introduces the single-track full-buffer (STFB) template.

2.1. Asynchronous Channels

An asynchronous channel is a bundle of wires and a protocol to communicate data across the wires from one pipeline stage (the sender) to another one (the receiver). Figure 1 shows three different types of channels.

The bundled-data channel has the advantage that the data is single-rail encoded (the same used in synchronous design) but is dependent on the timing assumption that the data is valid when the request signal is asserted. The request signal is typically driven through a delay line with a delay matched to the sender's computation delay plus some margin.

Alternatively, in a 1-of-N channel, the data (token) value is 1-of-N encoded where N wires are used to transmit N possible data values by asserting exactly one wire at a time. A *blank* or NULL is encoded by de-asserting all wires. 1-of-2 (dual-rail) and 1-of-4 encodings are most common and both effectively use two wires per bit to encode the data.

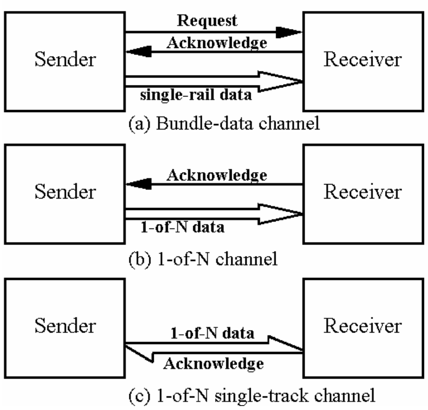


Figure 1. Asynchronous channels.

In the 1-of-N channel, the receiver detects the presence of the token from the data itself and, once the data is no longer needed, it acknowledges the sender. In the typical four-phase protocol, the sender then removes the data by resetting all wires and waits for the acknowledgement to be de-asserted before sending another token.

In the 1-of-N single-track channel, the receiver detects the presence of the token, as in the 1-of-N channel, but is also responsible for consuming it (by resetting all the wires). The sender detects that the token was consumed before sending another token.

Related designs include that from Berkel et al. [4] who proposed single-track handshake circuits to control medium-grain bundled-data pipelines. Sutherland et al. [7] later developed faster single-track GasP circuits to control fine-grain bundled-data pipelines. Nyström [8] also proposed a dual-rail (1-of-2) single-track template based on self-resetting pulsed-logic circuits like GasP but which requires significantly more transistors and is significantly slower. STFB templates, introduced in [10], offer GasP-like performance with template-based flexibility, allowing the utilization of conventional CAD tools.

2.2. STFB templates

Figure 2 shows a typical STFB cell's block diagram. When there is no token in the right channel (R) (the channel is empty), the Right environment Completion Detection block (RCD) asserts the "B" signal, enabling the processing of a next token. In this case, when the next token arrives at the left channel (L) it is processed lowering the state signal "S", which creates an output token to the right channel (R) and causes the State Completion Detection block (SCD) to assert "A", removing the token from the left channel through the Reset block. The presence of the output token on the right channel resets the "B" signal which activates the two PMOS transistors at the top of the N-stack, restoring "S", and deactivates the NMOS transistor at the bottom of the N-stack, as shown in Figure 3, disabling the stage from firing while the output channel is busy.

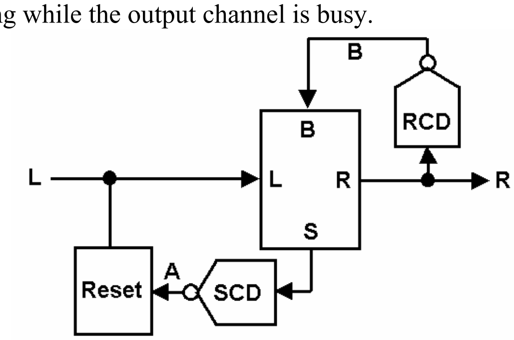


Figure 2. Typical STFB block diagram

Figure 3 shows a simplified schematic of the STFB dual-rail template. The NOR gate in this figure is the RCD, the NAND gate is the SCD and the NMOS transistor stack defines the cell's main function. Note that the NMOS transistor stack is designed to be semi-weak-conditioned in that it will not evaluate until all expected input tokens arrive [10].

The cycle time of the STFB template is 6 transitions and the forward latency is 2 transitions. This implies that the peak pipeline throughput can be achieved with just three stages per token, which allow the implementation of high performance small rings. The full-buffer characteristic of STFB stage refers to the capacity of each stage to hold up to one token.

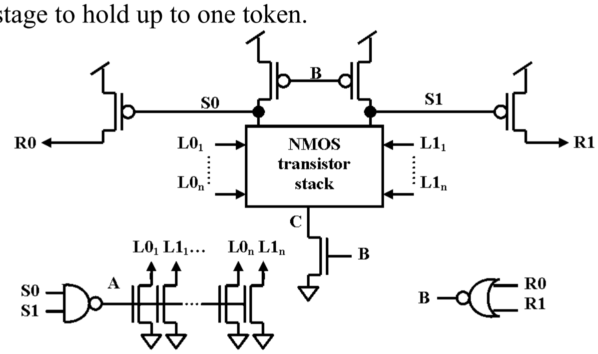


Figure 3. Simplified dual-rail STFB template.

3. STFB Standard-Cell Design

This section describes the transistor-level optimization implemented to improve performance and reliability in a standard-cell environment. Due to the timing assumptions in the STFB template, the transistor level design of each cell and sub-cell was done manually and checked through extensive SPICE simulation as described below.

3.1. Transistor sizing strategy

An important characteristic of the STFB architecture is that all the channels are point-to-point channels. This means that there are no forked wires and the channel load is a function of the wire length and the next stage input capacitance. Consequently, since the fanout is always one, the variance on output load is even more dominated by the variation in the wire-lengths than is typical in synchronous designs. Therefore, our initial version of the library introduced here adopts a single-size strategy for each STFB function. The chosen size is reasonable to safely drive, with adequate performance, a buffer load through up to a 1 mm long wire with 0.4 µm width and 0.5 µm spacing. This implies that we can place and route a block as big as 0.5x0.5 mm with essentially no special routing constraints. Larger blocks can also be implemented as long as the wires are constrained to be smaller than this limit. Longer wires would result in poor transition times that could compromise timing assumptions and thus functionality. In the future, special CAD tools to automatically add STFB pipelined buffers within the P&R flow could also accommodate longer connections.

Although the TSMC 0.25 µm process allows somewhat smaller transistors, we choose, as our minimum NMOS transistor width 0.6 µm and minimum PMOS transistor 1.4 µm. Also, we assumed, as a basis for the STFB cells creation, that the strength of the main N-stack should be, at least, twice of the minimum size NMOS. This means that the width of each NMOS transistor in the N-stack should be k*1.2 µm, where k is the number of transistors in the path to drive the state to ground. For example: for a 2 transistors path, the width of each N-stack transistor should be at least 2.4 µm.

We use, for sizing, a known practical rule that one inverter can drive efficiently four to five times its own input load. By hand calculation we determined that, because the main N-stack has twice the strength of a minimum size inverter, it can safely drive a capacitance load equivalent to 20 µm of "gate width", which is sufficient to drive the output transistor and the SCD as shown in Figure 3.

3.2. Balanced response

Symmetrized transistor stacks are utilized to perform the SCD and RCD functions inside the cell. Figure 4 shows a 2-input NAND gate where the NMOS transistor stack of the conventional diagram is cut in the middle and symmetrized to allow the same time response for both inputs. This approach minimizes the data influence in the cell timing behavior.

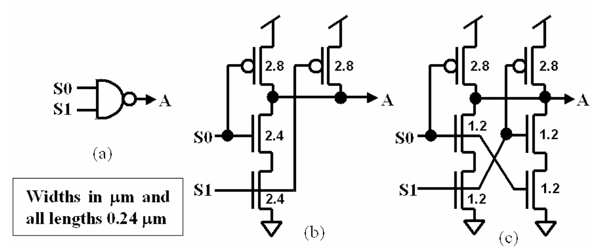


Figure 4. Sub-cell NAND2B_28_12: (a) symbol, (b) conventional diagram and (c) implemented balanced input diagram.

3.3. Output sub-cell STFB_POUT

The output driver sub-cell STFB_POUT is utilized in all STFB cells. It includes the staticizer structure and three PMOS transistors utilized to restore the state input ("S") high as illustrated in Figure 5. If the output channel is empty, the "B" signal is high, "R" is low, and "NR" is high. During this time, M7 alone fights leakage and holds "S" high. At the same time, M2 and M3 hold "R" low. When "S" is driven low, the output driver PMOS transistor M1 drives the output "R" high, which makes the minimum size inverter drive "NR" low, deactivating M3 and activating M4 and M5. The RCD (not shown) will also make the "B" signal fall, activating M6. M4 will hold the line high while M5 and M6 drive “S” back high, turning off M1.

Notice that M6 is controlled by the “B” signal from the RCD and its main function is to avoid any misfire caused by charge-sharing in the N-stack when a token is still present at the output (i.e., while the output channel is busy). Also, M5, which is controlled by the staticizer inverter (“NR” signal), is responsible to quickly assert “S” after firing.

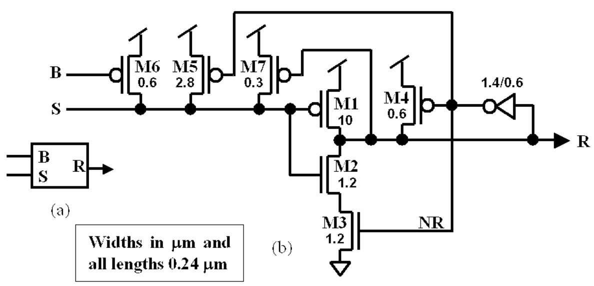


Figure 5. Sub-cell STFB_POUT (a) block diagram and (b) schematic.

This output stage topology offers a significant performance improvement allowing longer maximum wire length when compared with the initially proposed template [10]. It also improves robustness to charge sharing in the N-stack because this output sub-cell now has a lower switching threshold voltage.

3.4. The RCD sizing

The NOR gate in the STFB template (RCD) is also implemented as a symmetrized gate and it is responsible to drive the “B” signal low no later than the signal “NR” goes low in order to disable the N-stack and restore the signal “S”, as shown in Figure 6. This is an internal timing constraint that needs to be met to avoid the short-circuit current that would be caused by attempting to restore “S” while the N-stack is still enabled.

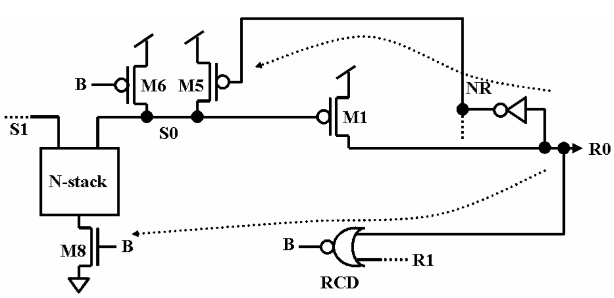


Figure 6. B and NR simultaneous activation.

This timing assumption is satisfied by reducing the load connected to the RCD output (W_{M6} = 0.6 μm, which is good enough to fight N-stack charge sharing) and by transistor sizing as shown in Figure 7, where the NMOS transistors of the balanced RCD are 1.2 μm wide, while, for a regular minimum sized NOR gate, we would use 0.6 μm.

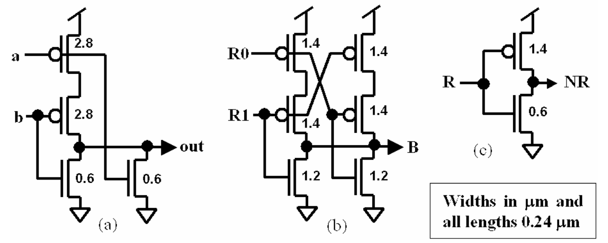


Figure 7. (a) conventional 2-input NOR, (b) balanced RCD and (c) staticizer inverter.

3.5. Input channel reset transistors

In the STFB template, the input token is consumed by driving the input channel wires low. It is done when the signal “A”, generated by the SCD block, activates a set of 5 μm wide NMOS transistors connected to each input wire. Also, to initially reset the entire circuitry, a global “/Reset” (active low reset) signal is used to force all channels low. Initially this signal was simply added as one input to the SCD block [10]. However, a 3-input NAND gate is much less efficient than a 2-input one. Figure 8.a shows the initially proposed 3-input SCD, where a 3-input NAND gate controls the reset transistors. Figure 8.b and c show the implemented reset structure, which uses 2-input NAND gates, allowing a smaller load on the states (“S0”, “S1”, “S2”) and offering a better performance of the SCD for dual-rail and 1-of-3 channels. Notice that the added transistors share the same drain connections, which results in a marginal increase in area and input capacitance for the STFB stage.

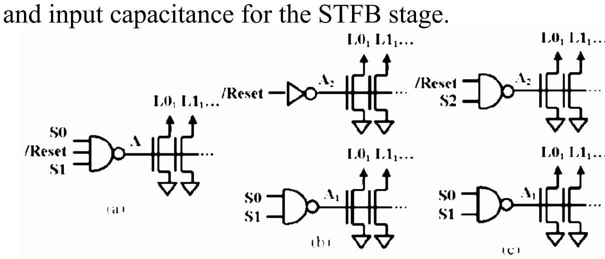


Figure 8. SCD and reset (a) initially proposed and the implemented (b) 1-of-2 and (c) 1-of-3.

3.6. Direct-path current analysis

A perceived problem with STFB designs is the amount of direct-path current, also known as short-circuit current, caused by violations of the timing constraint associated with tri-stating a wire before the preceding/succeeding stage drives it. This section analyzes this constraint in detail.

Figure 9 shows a conventional CMOS driver where both the PMOS and the NMOS transistor gates are connected together implementing an inverter. This means that during the rise (t_r) and fall (t_f) time of the input voltage (V_{in}) both transistors will be briefly active, allowing a direct-path current from V_{DD} to ground. Since this current has an approximate triangular shape, we can estimate the direct-path current as $I_{dp} = I_{peak}/2$ [11].

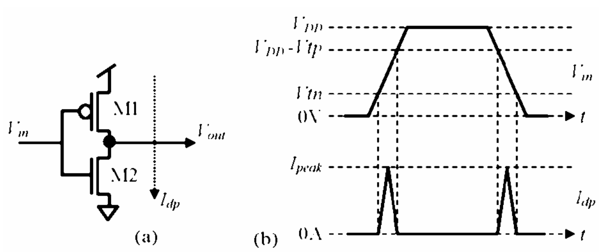


Figure 9. (a) inverter and (b) direct-path current.

For our STFB pipeline stages, the NMOS transistor gate is connect to signal "A", and the PMOS transistor gate is connected to "Sx" (one of the "states"). Figure 10 shows this implementation and the direct-path current if V_A happens earlier than V_{Sx}. If the voltage difference ($V_{diff} = V_A - V_{Sx}$) is zero, the STFB stage I_{dp} is similar to a conventional inverter. However, if one of the voltage transitions occurs ahead of the other, i.e., V_{diff} is different than zero, we may observe a higher peak current during one transition and a smaller peak current during the next transition, or vice-versa.

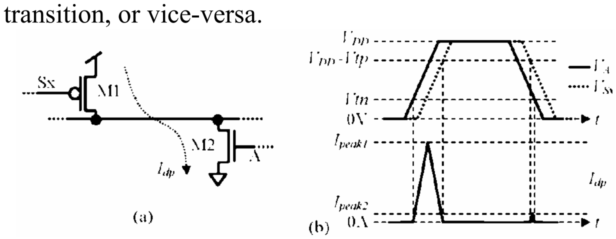


Figure 10. (a) STFB output/input drivers and (b) direct-path current if $V_A \neq V_{Sx}$.

Figure 11 shows the peak direct-path current versus the PMOS-NMOS gate voltage difference during an input rise/fall edge ($V_{diff} = V_A - V_{Sx}$). These values were obtained through DC Hspice simulation analysis using typical parameters with double than our minimum-sized transistors. Notice that, assuming that V_A and V_{Sx} have the same shape (both have the same width, rise and fall times), the average peak current is not significantly different than the inverter peak current for $V_{diff} < 1$ V. This means that a considerable difference between V_A and V_{Sx} can be tolerated without a significant jump in power supply consumption.

SPICE simulation also showed that the direct-path current of the STFB templates is no worse than an inverter driving the line, and the timing assumption associated with tri-stating one stage before the other drives the line is not a hard constraint. For our STFB pipeline stages, the time difference between V_A and V_{Sx} is bounded by the wire-length constraint to ensure correct operation.

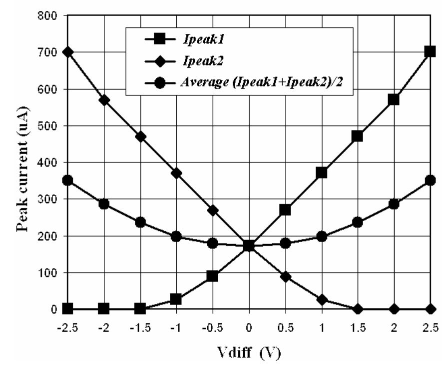


Figure 11. Peak direct-path current versus the PMOS-NMOS gate voltage difference.

4. Back-end design flow

Here we describe the generation of the standard-cell asynchronous library and its utilization in the standard-cell design flow.

4.1. Library design flow

Figure 12 shows the design flow utilized for the creation of the STFB cell library. Each block is described below:

Template specifications are the definitions of the utilized template as described in Section 3 and in [10].

Schematic, symbol and functional (Verilog) cell views are captured using Cadence Virtuoso environment and a text editor. Currently this step is done manually, however, synthesis from the template specifications is an area of future work.

From the **schematic**, netlist SPICE files, that include automatically estimated source-drain geometries, based on gate widths, are generated for **simulation** and for **LVS** (Layout Versus Schematic check using Dracula), which, in turn, provides parasitic capacitance information and the source-drain geometries extracted from layout. Extensive Hspice simulations were used to verify the general operation and performance of all cells pre and post-layout. Schematic and symbol of frequently used sub-cell circuits were created to simplify and speed-up this phase, including a POUT sub-cell, various basic gates, and several common control cores for different numbers of inputs and outputs.

Standard-cell specifications are the physical constraints utilized during the custom layout of the cell. For example, the cell height, power lines width, location of routing grid, etc. These are the same parameters utilized for synchronous cell designs and are necessary to make automated placement and routing feasible. Interestingly, the pins specifications needed to be in the grid and on a metal shape whose width is an even multiple of minor spacing grid steps (0.01 μm) to avoid off-grid error messages in the ASIC P&R phase.

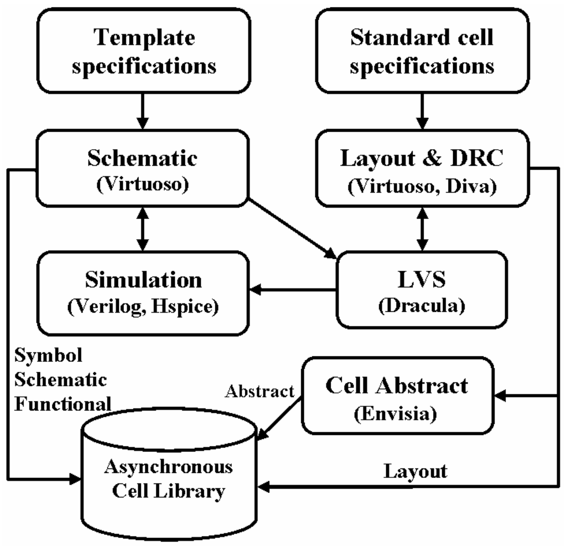


Figure 12. Standard-cell library design flow.

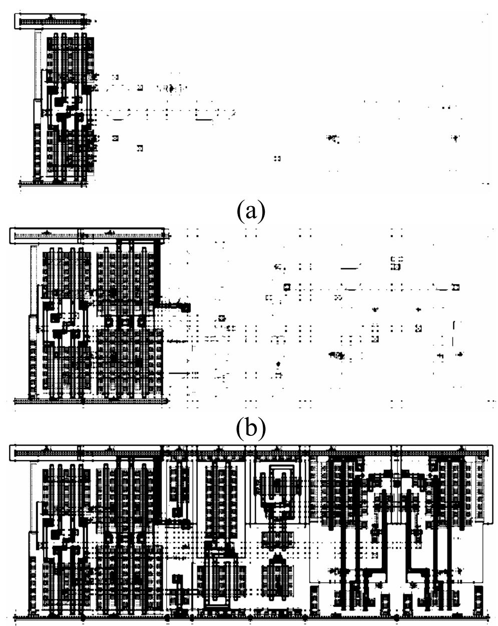
Layout & DRC are the manual physical design steps. To simplify this phase, reducing errors and saving time, sub-cell layouts were created matching the ones described in the **schematic** phase. Therefore, for most of the library cells, the top-level layout views are implemented with a mixture of sub-cells and cell-specific layout. The Diva Design Rule Checker (DRC) verifies that the layout satisfies all process design rules, however, it is also necessary to manually check if the cell complies with the standard-cell specifications mentioned above. Note also that the layout is done such that all cells DRC-cleanly abut, even when horizontally and vertically flipped.

An **abstract** layout view for the cells is generated using the Cadence tool Envisia Abstract Generator. The abstract file is in LEF format and represents the cells physical dimensions and the metal layers with a description of the power lines, input/output pins and metal obstructions. The placement and routing tool uses this file in the ASIC design flow.

The resulting Asynchronous Cell Library is a tree of directories, for the Cadence tools, where the sub-levels are the cells, their views (symbol, schematic, functional and layout) and the abstract file. A preliminary version of the STFB library has been released [22]. It contains all common sub-cells for dual and 1-of-3 rail logic, cells for Buffers, Splits, Merges, BitBuckets, and BitGenerators as well specific cells used in our adder test chip. In the future, Verilog behavioral views of all cells will be completed and input capacitance and delay equations will be characterized and included in the library using the Liberty (.lib) file format [23].

4.2. STFB2_XOR2 cell example

Figure 13 shows the layout of the STFB2_XOR2 cell. This cell is a STFB pipeline stage with two dual-rail input channels and one dual-rail output channel. In our library, this cell has four views: symbol, functional, schematic and layout. The symbol view is used to instantiate the cell in higher level schematics, the functional view is the verilog behavioral description of the cell, the schematic view has the transistor-level schematic of the cell, including the symbols of the sub-cells used to implement this cell, and the layout view, which, similarly to the schematic view, is composed of a cell-specific part and various sub-cells as shown in Figure 13. In this figure, we can see that the STFB2_XOR2 cell includes the 8 input transistors, that define the XOR function, and a STFB2_CORE4I sub-cell, which includes 4 reset transistors and one INV_28_12, one NAND2B_56_24, one NOR2B_14_12OD and two STFB_POUT sub-cells.



(a)

(b)

(c)

Figure 13. STFB2_XOR2 cell layout (a) custom layout and STFB2_CORE4I sub-cell, (b) with STFB2_CORE4I sub-cell expanded, and (c) with all sub-cells expanded.

Notice that, by re-arranging the input transistor connections shown in Figure 13.a, we can easily implement other two-input one-output cells such as STFB2_AND2 and STFB2_OR2.

4.3. Asynchronous ASIC design flow

Once we have STFB standard cells in our cell library, a conventional ASIC design flow can be utilized to generate a high performance asynchronous design as shown in Figure 14. Note that currently the entire design is entered through schematics (synthesis is an area of future work) and each block is sent to P&R and are then wired together in the chip assembly step. Verification can be performed through Verilog cell-level simulation and Nanosim transistor-level simulation.

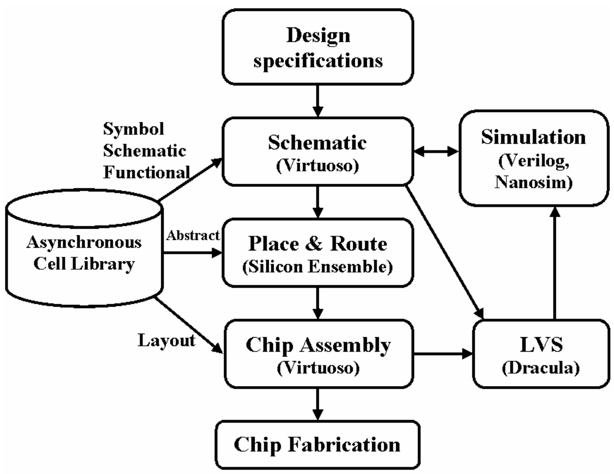


Figure 14. Asynchronous ASIC design fow.

5. The evaluation and demonstration chip

A test chip was designed to validate the design flow as well as the performance of the STFB templates. The central block of the test chip is a 64-bit STFB prefix adder, while the input and output circuitry were designed to feed the adder and sample the results enabling the checking of its performance and correctness at full-throughput.

5.1. The Prefix adder

Given two n-bit numbers A and B in two's complement binary form, the addition operation, $A+B$, can be performed by computing [14][15]:

$$\left.\begin{aligned} g_j &= a_j b_j \\ p_j &= a_j \oplus b_j \\ c_j &= g_j + p_j c_{j-1} \\ s_j &= p_j \oplus c_{j-1} \end{aligned}\right\} \quad 0 \le j < n$$

where, c_{-1} is the adder primary carry input, a_j, b_j and s_j are bits of A, B and the addition result S respectively, g_j is the generate signal and p_j is the propagate signal for the bits at position j.

For an asynchronous 1-of-N implementation, a_j, b_j, c_j and s_j are dual-rail channels, where, for example, $a1_j$ high means $a_j = 1$, and $a0_j$ high means $a_j = 0$. Also, we use the k_j, "kill" signal, to form a 1-of-3 channel (k_j, p_j, g_j). The asynchronous equations become:

$$\left.\begin{aligned} g_j &= a1_j b1_j \\ p_j &= a1_j b0_j + a0_j b1_j \\ k_j &= a0_j b0_j \\ c1_j &= g_j + p_j c1_{j-1} \\ c0_j &= k_j + p_j c0_{j-1} \\ L0_j &= a0_j b0_j + a1_j b1_j \\ L1_j &= a0_j b1_j + a1_j b0_j \\ s0_j &= L0_j c0_{j-1} + L1_j c1_{j-1} \\ s1_j &= L0_j c1_{j-1} + L1_j c0_{j-1} \end{aligned}\right\} \quad 0 \le j < n$$

where, L is the result of $a_j \oplus b_j$ (a_j xor b_j). This means that a_j and b_j need to be duplicated since we need one pair for the carry computation and another for the final sum.

Adapting from the usual synchronous definition [12][16], we define ($K_{j:j}$, $P_{j:j}$, $G_{j:j}$) = (k_j, p_j, g_j) (asynchronous 1-of-3 channel) and:

$$(K_{i:j}, P_{i:j}, G_{i:j}) = (k_j, p_j, g_j) o (k_{j-1}, p_{j-1}, g_{j-1}) o \ldots o (k_i, p_i, g_i)$$

where, $j > i$ and o is the fundamental carry operator adapted to the asynchronous implementation as:

$$(k_j, p_j, g_j) o (k_i, p_i, g_i) = ((k_j + p_j k_i), (p_j p_i), (g_j + p_j g_i))$$

Therefore, at each bit position, the final dual-rail carry can be computed by:

$$c1_j = G_{0:j} + P_{0:j} c1_{-1} \qquad c0_j = K_{0:j} + P_{0:j} c0_{-1}$$

where, $c1_{-1}$ and $c0_{-1}$ define the dual-rail adder primary carry input.

Adapting from [14], the asynchronous addition can be performed in the following steps:

Step 1 (1 stage deep)
Duplicate ($a0_j$, $a1_j$) and ($b0_j$, $b1_j$) $\forall j\ 0 \le j < n$

Step 2 (1 stage deep)
Compute:

$$\left.\begin{aligned} g_j &= a1_j b1_j \\ p_j &= a1_j b0_j + a0_j b1_j \\ k_j &= a0_j b0_j \\ L0_j &= a0_j b0_j + a1_j b1_j \\ L1_j &= a0_j b1_j + a1_j b0_j \end{aligned}\right\} 0 \le j < n$$

Step 3 ($\lceil \log_2 n \rceil$ stages deep)
For $x = 1, 2 \ldots \lceil \log_2 n \rceil$ compute:

$$c1_j = G_{j-2^{x-1}+1:j} + P_{j-2^{x-1}+1:j} c1_{j-2^{x-1}}$$

$$c0_j = K_{j-2^{x-1}+1:j} + P_{j-2^{x-1}+1:j} c0_{j-2^{x-1}}$$

$$\forall j\ 2^{x-1} - 1 \le j < 2^x - 1$$

$$\begin{aligned}(K_{j-2^x+1:j}, P_{j-2^x+1:j}, G_{j-2^x+1:j}) = \\ (K_{j-2^{x-1}+1:j}, P_{j-2^{x-1}+1:j}, G_{j-2^{x-1}+1:j}) \\ o(K_{j-2^x+1:j-2^{x-1}}, P_{j-2^x+1:j-2^{x-1}}, G_{j-2^x+1:j-2^{x-1}})\end{aligned}$$

$$\forall j\ 2^x - 1 \le j < n$$

Step 4 (1 stage deep)
Compute:

$$\left.\begin{aligned} s0_j &= L0_j c0_{j-1} + L1_j c1_{j-1} \\ s1_j &= L0_j c1_{j-1} + L1_j c0_{j-1} \end{aligned}\right\} 0 \le j < n$$

$$c1_{n-1} = G_{0:n-1} + P_{0:n-1} c1_{-1}$$

$$c0_{n-1} = K_{0:n-1} + P_{0:n-1} c0_{-1}$$

Figure 15 illustrates the above steps with an example, an 8-bit asynchronous prefix adder, where, the thin arrows are 1-of-2 (dual-rail) channels and the thick arrows are 1-of-3 channels.

Notice that some STFB pipeline stages must have two versions: one with unique output channel and another with duplicated output channels. This is necessary because we are using point-to-point single-track channels (there are no forks in the wires). The pipeline stages used with their library name are as shown below:

In Figure 16 the STFB2 prefix is used for stages with only dual-rail channels, and STFB3 is used for stages with at least one 1-of-3 channel. In particular, the STFB3_AB_KPG stage implements the *kpg* part of **step 2** (described above) and has two dual-rail input channels (A and B) and one 1-of-3 output channel (KPG). STFB3_AB_KPG2 implements the same functionality but has two 1-of-3 output channels (KPG2). Similarly, cells STFB3_KPG2_KPG and STFB3_KPG2_KPG2 implement the *kpg* part of **step 3** and have two 1-of-3 input channels and one or two 1-of-3 output channels, respectively. In the same manner, the carry generation parts of **step 3** and **4** are implemented by the cells STFB3_KPGC_C and STFB3_KPGC_C2. Finally, **step 1** and the sum parts of **steps 2** and **4** are implemented by STFB2_FORKs and STFB2_XOR2s. The buffers (STFB2_BUFFER) are used for capacity matching ("slack" matching).

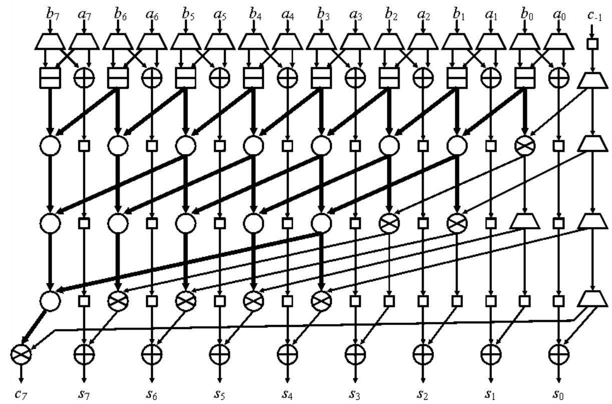


Figure 15. 8-bit asynchronous prefix adder.

STFB2_FORK (fork stage)
STFB2_BUFFER (buffer stage)
STFB2_XOR2 (2-input xor stage)
STFB3_AB_KPG and STFB3_AB_KPG2
STFB3_KPG2_KPG and STFB3_KPG2_KPG2
STFB3_KPGC_C and STFB3_KPGC_C2

Figure 16. Pipeline stages utilized in the adder.

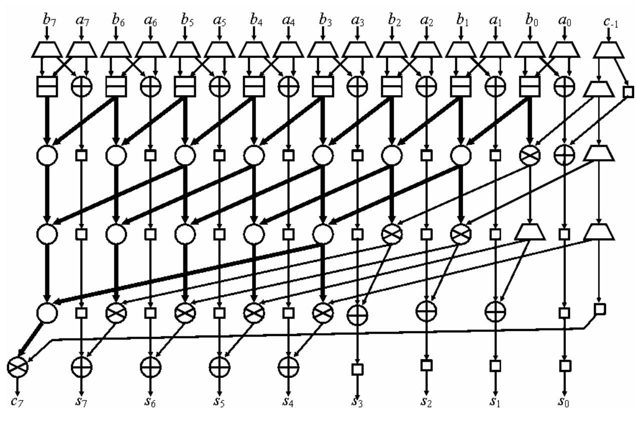


Figure 17. 8-bit async. prefix adder optimized.

Figure 17 shows an optimized version of the 8-bit prefix adder, where the carry input (c_{-1}) is forked at the first step allowing an early computation of s_0 and improving the layout by replacing the bottom fork, which was used previously to supply c_{-1} to s_0 and c_{n-1} (located in two opposite extremes of the adder), with a simple buffer. Also, the xor stages of the first half of the adder, from s_1 to $s_{(n/2)-1}$, can be moved one step earlier. These modifications saved $(n/2)$-2 buffers and simplified the layout.

In this small example, the 8-bit asynchronous prefix adder is 6 levels deep $(2 + \lceil \log_2 n \rceil + 1)$. The implemented 64-bit asynchronous prefix adder is, therefore, 9 levels deep. This means that, after 9 times the forward latency of the STFB templates (9*2 = 18 transitions) the resulting 64-bit plus carry out are available. Also, since the cycle time of the STFB template is just 6 transitions, the 64-bit adder can have up to 3 additions simultaneously being processed (3 tokens in the pipeline) at maximum throughput.

5.2. The input circuitry

The input circuitry generates a test pattern to be fed into the adder. The INPUTGEN129 block is composed of 129 15-stage rings (two 64-bit numbers and carry in).

Figure 18 shows the 15-stage ring diagram, where we have 14 buffers, one fork and one xor, and the square with the letters TI is a token inserter block (not shown) and the square with the letters BG is a controlled bit-generator (not shown). Although the rings support up to 14 tokens each, the maximum throughput of the ring is achieved with 5 tokens.

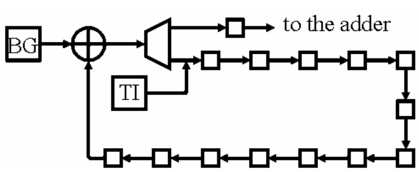


Figure 18. 15-stage ring utilized in the input circuitry.

After the tokens are inserted by the TI cell, the BG cell is enabled. Since, now, the xor stage has one token in each input, it generates a token that enters the fork stage, where one copy of the token is sent to the adder and another is sent back into the ring. If BG is enabled to generate "zero" tokens, the tokens in the ring simply circulate making copies of themselves. If BG is enabled to generate "one" tokens, the tokens in the ring are inverted at every pass through the xor increasing the number of scanned combinations.

5.3. The output circuitry

In order to test the adder running at full throughput, we implemented output circuitry that samples the 65-bit result (64-bit and carry out), forwarding to the output pins one out of 128 results. Then, a much slower external circuit can read and compare the results of the iteration #1, #129, #257, #385, #513,…

If the input generator rings are loaded with 5 tokens (no inversion enabled), the SAMPLER65 block outputs all the 5 results in the order 1, 4, 2, 5 and 3.

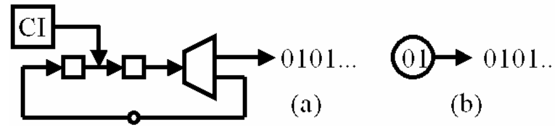


Figure 19. 01 ring (a) circuit and (b) symbol.

Figure 19 shows a 01 ring, where, after reset, the channel initializer (CI) block inserts a "zero" token in the small ring. The output channel of the fork that returns to the ring has both wires inverted (shown as a bubble on the wire) before connect to the first buffer. This will make the token change value at every loop and the circuit output becomes a sequence 010101… Also, notice that this ring has three stages and one token, which, for STFB, means full throughput.

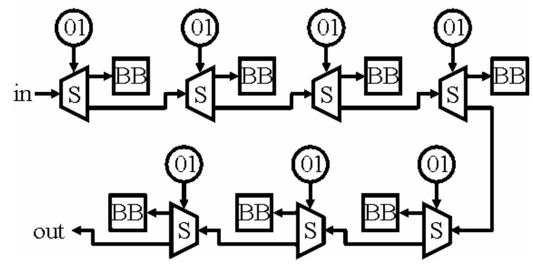


Figure 20. 1:128 sampler diagram.

Figure 20 illustrates a 1:128 sampler circuit where the split stages (S), controlled by 01 rings, direct the input token to a bit-bucket (BB), where the token is destroyed, or to the next split. The SAMPLER65BY128 block, used in our design, has a similar structure for the carry out signal and, for the remaining 64 bits, each of the 01 ring outputs are forked until they reach their respective 64 split stages. Note also that single-track to single-rail converters and their respective control circuits are not shown.

5.4. The chip layout

Figure 21 shows a picture of the laid-out 64-bit STFB asynchronous prefix adder and its auxiliary test circuitry. Each block P&R was performed separately with an area utilization of 80%, the three blocks where forced to have the same height (1.7 mm) and the placement of the adder block pins matched their correspondents in the input and sampler blocks. The total area is 4.1 mm^2.

Notice that, by performing P&R on separated blocks, we significantly reduce the probability of a very long wire that could compromise the performance and the functionality of the design. In fact, post-layout we guaranteed no STFB signal wires were longer than 1 mm. Also, as filler cells, a total of 1.6 nF in bypass capacitors were added.

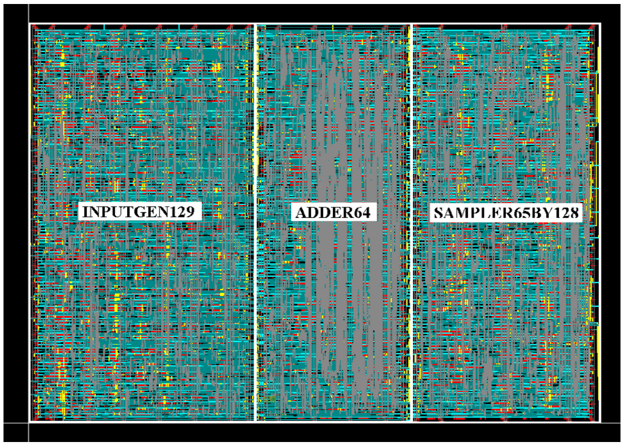


Figure 21. The input, adder and sampler blocks.

5.5. Power Distribution and EM

Figure 22 shows a post-layout Nanosim simulation result (transistor model TT, 25°C and V_{DD} = 2.5V), where we can see the format of each block current. The i(v129) and i(vdd) are the input and the adder block current respectively, and they are almost constant around 1.6 and 1.2A respectively (running at full throughput: 1.4 GHz). The i(v65) is the sampler block current, whose ripple depends on how far the token flows in the split pipeline and varies from 0.2 to 0.6A. The overall current is relatively constant, when compared to synchronous designs, which significantly reduces the need for on-chip bypass capacitors and offers very low Electro-Magnetic Interference (EMI).

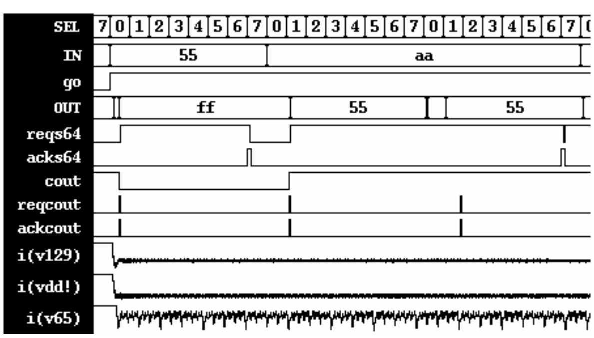


Figure 22. Typical simulation output.

As these designs consume significantly more current than their slower synchronous counterparts, voltage drop (IR drop) and the electromigration over the power lines become important factors. Fortunately, the router supports the insertion of a robust power grid to mitigate these effects.

6. Simulation results

Table 1 shows the simulation results of the five simulated corners. In this table, the conditions consist of the combination of the model library (NMOS and PMOS models: T = typical, S = slow and F =fast), the simulation temperature, and the power supply voltage. I_{av} is the average current of the three blocks when active. Latency is the 64-bit adder propagation time, and Throughput is the number of additions processed per second.

Table 1. Results

Conditions	I_{av}	Latency	Throughput
TT, 25°C, 2.5V	3.3 A	2.1 ns	1.47 GHz
SS, 100°C, 2.2V	1.8 A	3.3 ns	943 MHz
FF, 0°C, 2.7V	4.6 A	1.6 ns	1.95 GHz
SF, 25°C, 2.5V	3.2 A	2.2 ns	1.46 GHz
FS, 25°C, 2.5V	3.2 A	2.2 ns	1.46 GHz

7. Comparisons

Table 2 shows a comparison of some STFB pipeline stages with PCHB stages and static standard cell CMOS gates. The latency and cycle time are written in terms of number of transitions. The CMOS standard cell gates, used in this comparison, were designed under the same standard cell specification utilized for the STFB and PCHB pipeline stages. Also, they are composed of a 2X gate followed by an 8X inverter in order to match driving strengths.

Table 2. STFB, PCHB and CMOS comparison.

Function	Cell	Latency	Cycle Time	Area (μm²)	Area ratio
Buffer	STFB	2	6	415	4.5
	PCHB	2	14	726	7.9
	CMOS	2	-	92	1
2-input AND/OR	STFB	2	6	472	4.6
	PCHB	2	14	968	9.3
	CMOS	2	-	104	1
2-input XOR	STFB	2	6	472	2.6
	PCHB	2	14	1048	5.7
	CMOS	2 or 3	-	184	1

For these basic functions, the area ratio indicates that the STFB stages are approximately 50% smaller than the PCHB stages and about 5 times bigger than a CMOS implementation (not considering the latch/flip-flop and clock-tree overhead required for synchronous designs). Also, excluding the reset wire utilized by both the STFB and PCHB stages, the STFB dual-rail implementation uses 33% less wires than PCHB and just twice the number of wires of the CMOS circuit.

8. Conclusions

This paper introduces a STFB standard-cell library available through the MOSIS Education Program, which facilitates a conventional back-end flow for ultra-high-performance asynchronous blocks. Implementation details of the STFB cells are presented and the flow is demonstrated on several significant size blocks - a 64-bit adder and its test circuitry. Post-layout results show performance of over 1.4 Gigahertz in TSMC's 0.25 μm process. Since the STFB cells can easily be interfaced with other even more robust templates, such blocks may be used to solve performance bottlenecks in a bigger design where ultra-high performance is needed.

9. Acknowledgements

This research has been partially supported by NSF Grant CCR-0086036 and gifts from TRW, Fulcrum Microsystems and the MOSIS Educational Program. Thanks to Jay Moon for his valuable help with the CAD tools, to Sachit Chandra for his help with the design flow and Sunan Tugsinavisut for many helpful discussions.

References

[1] W. J. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge Univ. Press, Cambridge, UK, 1998

[2] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. Dooply, and J. Arceo, "The Design and Verification of a Low-Control-Overhead Asynchronous Differential Equation Solver", IEEE Transactions on VLSI, Dec. 1998

[3] A. Davis and S. M. Nowick, "An Introduction to Asynchronous Design", Univ. of Utah Tech. Rep., Dept. of Computer Science, UUCS-97-013, Sept. 19, 1997.

[4] K. van Berkel, and A. Bink,, "Single-Track Handshake Signaling with Application to Micropipelines and Handshake Circuits", Proc. ASYNC, pp: 122–133, 1996.

[5] A. M. Lines, "Pipelined Asynchronous Circuits", Master Thesis, California Institute of Technology, June 1998.

[6] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee, "The Design of an Asynchronous MIPS R3000 Microprocessor." Proc.of ARVLSI, pp. 164-181, 1997.

[7] I. Sutherland and S. Fairbanks, "GasP: A Minimal FIFO Control", Proc. of ASYNC, pp: 46 – 53, 2001.

[8] M. Nyström, "Asynchronous Pulse Logic", PhD Thesis, California Institute of Technology, May 14, 2001.

[9] M. Singh and S. M. Nowick, "High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths", Proc. of ASYNC, pp: 198 – 209, 2000.

[10]M. Ferretti and P. A. Beerel, "Single-Track Asynchronous Pipeline Templates Using 1-of-N Encoding", Proceedings of DATE, pp: 1008–1015, Paris, France, March 2002.

[11] J. M. Rabaey, *Digital Integrated Circuits*, Prentice Hall Electronics and VLSI Series, New Jersey, USA 1996.

[12] I. Koren, Computer Arithmetic Algorithms, 2nd Edition, A. K. Peters, Natick, MA, USA 2002

[13] R. Manohar, J. A. Tierno, "Asynchronous Parallel Prefix Computation", IEEE Transactions on Computers, pp: 1244 -1252, vol. 47, Nov. 1998.

[14] A. Goldovsky, R. Kolagotla, C.J. Nicol and M. Besz, "A 1.0-nsec 32-bit Tree Adder in 0.25-μm static CMOS", Proc. 42nd IEEE Midwest Symp. on Circuits and Systems, pp: 608 -612, vol. 2, 1999.

[15] A. Goldovsky, H.R. Srinivas, R. Kolagotla and R. Hengst, "A Folded 32-bit Prefix Tree Adder in 0.16-μm static CMOS", Proc. 43rd IEEE Midwest Symp. on Circuits and Systems, pp: 368–373, Lansing MI, August 2000.

[16] R.P. Brent and H. T. Kung, "A regular layout for parallel adders", IEEE Trans. on Computers, C-31, pp: 260-264, March 1982.

[17] Theobald, M. and Nowick, S.M., "Transformations for the synthesis and optimization of asynchronous distributed control", Proc. Design Automation Conference, pp: 263 - 268, June 2001.

[18]U. Cummings, "Terabit Clockless Crowbar Switch in 130 nm", Proc. 15th Hot Chips Conference, August, 2003.

[19]A. J. Martin, M. Nyström, K. Papadantonakis, P. I. Penzes, P. Prakash, C. G. Wong, J. Chang, K. S. Ko, B. Lee, E. Ou, J. Pugh, E. Talvala, J. T. Tong, A. Tura, "The Lutonium: a sub-nanojoule asynchronous 8051 microcontroller", ASYNC'2003.

[20] M. Renaudin, P. Vivet, F. Robin. "ASPRO-216: A Standard-Cell QDI 16-BIT RISC Asynchronous Microprocessor", ASYNC'98.

[21] R. O. Ozdag and P. A. Beerel, "A Channel Based Asynchronous Low Power High Performance Standard-Cell Based Sequential Decoder Implemented with QDI Templates", ASYNC'04.

[22] USC Asynchronous CAD/VLSI Group Standard Cell Library, http://jungfrau.usc.edu/AsyncLib.html, October 2003.

[23] Synopsys, *Liberty User Guide*, Vol. 1 and 2, version 2003.10, October 2003