SINGLE-TRACK ASYNCHRONOUS PIPELINE TEMPLATE

by

Marcos Ferretti

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(ELECTRICAL ENGINEERING)

August 2004

Dedication

To my wife.

ACKNOWLEDGMENTS

This work would not be possible without the vision and knowledge of my advisor, Peter A. Beerel. His support and guidance were essential for the development of new ideas. Also, he and his wife, Janet, were always available for help, creating a friendly and productive environment.

I would like to thank my friends, from the USC Asynchronous CAD/VLSI Group, for all these years of invaluable suggestions, comments and friendship: Sunan Tugsinavisut, Recep O. Ozdag, Hoshik Kim, Sangyun Kim, Jay Moon, Shantanu Awasthi and Pankaj Golani.

I am especially grateful to the USC professors, in particular, Massoud Pedram, Won Namgoong, Roger Zimmermann and Kian Kaviani, and to the USC Electrical Engineering staff, Tim Boston, Diane Demetras, Lisa I. Connell and Marylee Reynolds, for their patience and crucial help. Also, I am grateful to all my UNICAMP professors, including my former MS advisor, José Antonio Siqueira Dias, for all their efforts to teach me the foundations of electrical engineering and their support for my continuing studies at USC.

I would like to acknowledge the generous support from National Science Foundation Grant CCR-0086036 and the gifts from TRW, Fulcrum Microsystems and the MOSIS Educational Program.

Thanks are due to Helen Thompson, Sam Reynolds, Wes Hansford, and all the MOSIS support people for the great cooperation and effort to meet our needs.

I also would like to thank Andrew Lines for his encouraging comments and discussions, Mike Moacanin and Jeremy Boulton for helping with the temperature measurements.

It is important to acknowledge Prof. Rogerio C. Leite, Sergio C. Leite, Ichiro Aoki, Ricardo R. Maciel and Newton E. Fujii, whom I had the opportunity to work with. They contributed to shape my views of business, politics and friendship. They also encouraged me to pursue new knowledge and goals.

My deepest gratitude is reserved to my wife, Lilione, whose love, confidence and understanding never failed to me. I could never have done this work without her.

Lastly, I would like to thank my parents, Raul and Aurora, and my siblings, Nair, Santo e Geraldo, for supporting me throughout my life and for being there when I needed them.

Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This PhD dissertation presents the single-track full-buffer (STFB) templates for a new fast family of fine-grain high-performance asynchronous pipeline building blocks based on the single-track protocol. A demonstration design, implemented using our STFB standard cell library designed for MOSIS TSMC 0.25 μm process, is presented and analyzed. It includes a 64-bit prefix adder and achieves 1.45 GHz.

The STFB template does not require control wires outside of the datapath and the data is 1-of-N encoded. With a forward latency of 2 transitions and a cycle time of only 6 transitions for most of the configurations, the new family can run up to 2 GHz using the MOSIS TSMC 0.25 μm process. This is significantly faster than all known quasi-delay-insensitive (QDI) templates and has less timing assumptions than the recently proposed ultra-high-speed GasP bundled-data circuits.

STFB functional blocks can offer three times higher throughput requiring half of the area when compared with QDI circuits. In particular, they are advantageous when the distance between two consecutive data tokens is small, as found in loops with multiple tokens, shared resources or small loops with one token.

The template-based approach makes designing STFB blocks simple. Designing complex pipelined circuits using STFB blocks can use the same flow and cad as any channel-based asynchronous architecture. Physical design may in fact be easier than in QDI-based circuits because there are fewer wires between blocks – i.e., there is no acknowledgement wire. There is one constraint, however, in order to satisfy the timing assumptions, the channel load needs to be bounded and, since the STFB channels are

point-to-point connections (no fork in the wires), this bounding is achieved by simply limiting the maximum wire length between STFB pipeline stages.

1 INTRODUCTION

As CMOS manufacturing technology scales into deep and ultra-deep sub-micron design, problems with process and within die variations, clock skew, clock distribution, and on-chip communication in high-speed synchronous designs are becoming increasingly difficult to overcome [12], warranting the exploration of alternative design approaches. In particular, asynchronous design is emerging as an increasingly viable alternative.

In synchronous design, the clock signal is used to synchronize the state update across the system, while in asynchronous designs, there is no global synchronization and all the blocks are data-driven as shown in Figure 1. The clock signal controls the exact moment when the latches should sample the input data. In order to guarantee that the data is stable when sampled, the clock period should account for the worst-case delay including clock skew and all physical variations.

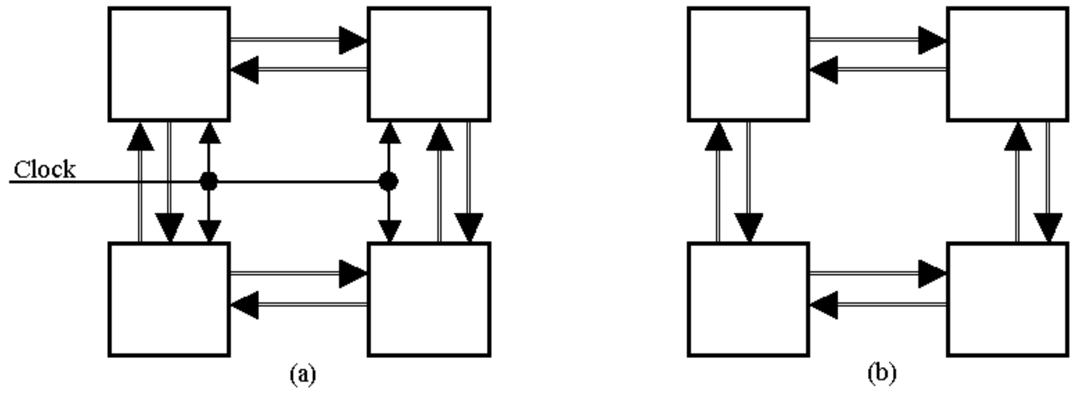


Figure 1 – Synchronous blocks with clock (a) and asynchronous blocks (b).

1.1 Asynchronous Design

The performance of asynchronous circuits is not limited by any global signal and the activity of each stage is data driven, which facilitates the following advantageous characteristics:

1) *No clock distribution and no clock skew.* Clock skew is defined as the time difference between the occurrence of the real clock edge and the desired clock edge. This difference must be measured and minimized to ensure correct operation and that performance does not significantly suffer. The problems of clock distribution and clock skew minimization are becoming increasingly significant as the technology scales, and within die variations increase, and as more complex system-on-chip (SoC) designs with higher clock frequencies are expected by the market place. The clock distribution network is also responsible for a considerable amount of the consumed power, representing 20–50% of the total power on a chip [36][14] and efforts to reduce its contribution to total power are on-going.
2) *Low power consumption.* Although asynchronous circuits in general have more control overhead, blocks that have no data to process remain completely inactive, providing the equivalent of perfect clock-gating [40]. In particular, clock gating in synchronous circuits is an *ad hoc* method of obtaining the same result and is manageable only at a coarse grain level [33]. Consequently, many asynchronous chips have demonstrated significantly lower dynamic power dissipation than their synchronous counterparts [40][20]. That said, it

should be noted that the problem of increasing static power dissipation due to higher leakage currents in state-of-the-art processes is a common problem to both synchronous and asynchronous circuits and one for which there is active research in both domains.

3) *Average case performance.* The data-driven nature of asynchronous circuits implies that the performance is a function of the data being processed and can be measured as an average over time. In fact, by optimizing for the common case, some asynchronous circuit's average performance can be dramatically higher than its worst-case performance. There are two ways this average case performance may take shape. First, the asynchronous architecture may be designed to take advantage of the input statistics of the data, such as the presence of small numbers. Secondly, the asynchronous physical design may focus on critical cycles in the design and allow longer narrower wires between less critical blocks. In contrast, the synchronous circuit's clock frequency must be adjusted to accommodate the worst-case computation [57][37]. Consequently, some asynchronous circuits have demonstrated significantly better average case performance than the worst-case performance of their synchronous counterparts [37].

4) *Easing of global timing issues.* Moreover, as the technology moves into deep sub micron, wire delays will require several clock cycles to propagate information across the chip and multiple clock domains may need to communicate in a SoC design. Asynchronous interfaces can be used to shell

encapsulate the synchronous blocks and all the communications can be done using latency-insensitive asynchronous channels [7][8].

5) *Automatic adaptation to physical properties.* Synchronous designs have to adjust their clock frequency to cover variations in fabrication process, temperature and power supply. Asynchronous designs, on the other hand, naturally adapt to this conditions and the speed variation in any path will not affect the functionality of the system.

6) *Improved EMI.* In synchronous systems, most of the circuit activity occurs around the clock edge, causing a concentration of energy in the clock harmonics. In asynchronous, the activity is uncorrelated, which produces a more distributed noise spectrum with lower peak noise [56]. This characteristic may be very important for SoC and mixed-mode designs.

Among the numerous asynchronous design styles being developed, template-based fine-grain pipelines have demonstrated very high performance [26][47][34][42][43][44]. Template-based approaches have the advantage of removing the need for generating, optimizing, and verifying specifications for complex distributed controllers, which is both difficult and error-prone [57]. Various templates tradeoff latency, cycle time, and robustness to timing. One of the most robust is the quasi-delay-insensitive (QDI) template proposed by Lines [26]. One of the most aggressive is the ultra-high-speed GasP [47]. GasP offers high throughput but requires a bundled data design style that involves additional timing margins and assumptions that must be verified during physical design and that introduces higher latency through the data path than even the QDI templates, possibly yielding lower system performance.

The single-track full-buffer (STFB) templates presented here and in [18], use 1-of-N data encoding and two-dimensional pipelining instead of single-rail encoding and fine-grain pipelining used by GasP. They have two key advantages. First, they remove the GasP bundling constraint, making them easier to design and verify. Second, they reduce forward latency by 58% at the cost of a 26% slower cycle time compared to GasP. The overall performance impact of this tradeoff depends on characteristics of the system. In particular, if the system is latency-critical, where the performance is determined by how fast an individual data token flows through the system, a STFB system can be significantly faster than the comparable GasP system despite having local cycle times that are somewhat larger.

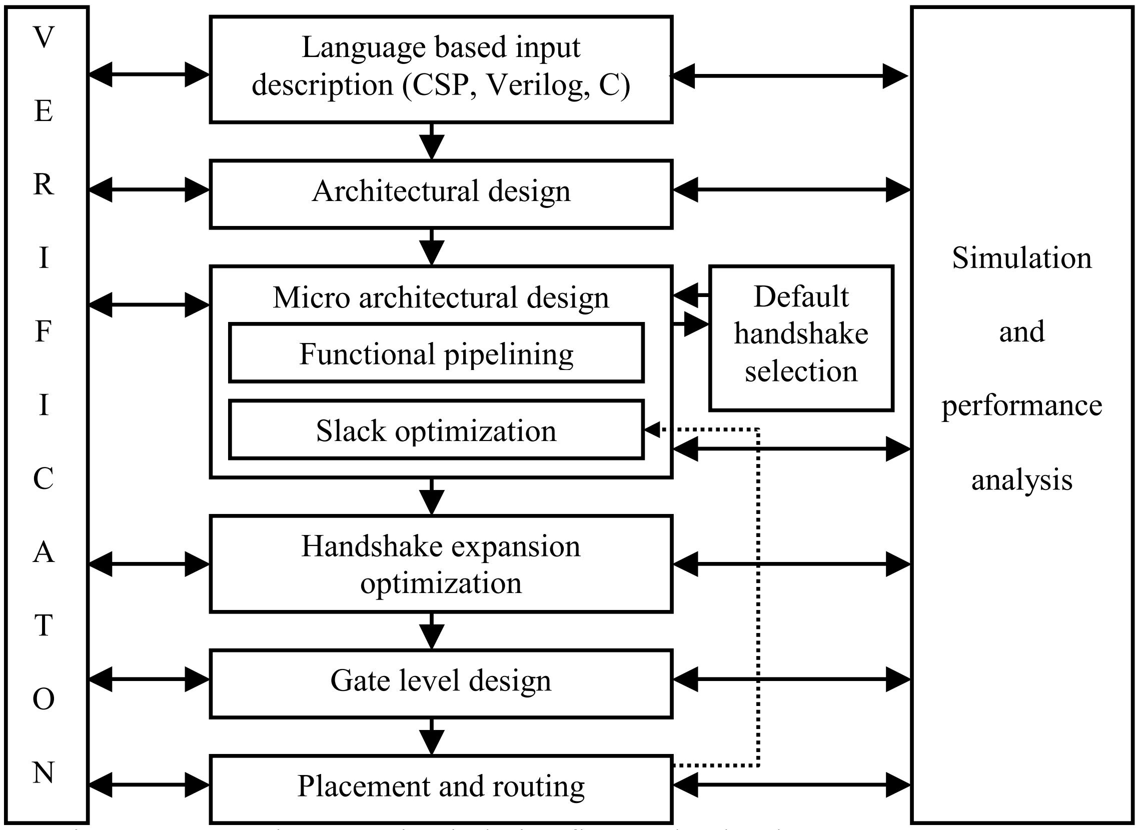
1.2 Test structures

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.

The input circuit allows loading 129 9-stage rings that are used to continuously feed the adder with two 64-bit operands and one bit carry in. The 64-bit prefix adder structure processes all the inputs simultaneously and generates the 64-bit sum and the carry out with throughput of 1.4GHz. The output circuit is a programmable sampler that forwards results to the pins at manageable rates without slowing down the adder.

1.3 Design flow

The USC Asynchronous CAD and VLSI group and the Columbia Asynchronous group are working together to define a complete asynchronous circuit design methodology that will offer automated tools for design of both high-performance and low-power asynchronous circuits. The diagram shown in Figure 2 shows the main steps of the design flow. We will be able to start with a language based model, such as CSP [30] and Verilog [10], as the input description of the desired top-level functionality of the chip and may contain information about the constraints on power, energy consumption, throughput, latency, chip area, etc.


VERIFICATION
Language based input description (CSP, Verilog, C)
Architectural design
Micro architectural design
Functional pipelining
Slack optimization
Default handshake selection
Handshake expansion optimization
Gate level design
Placement and routing
Simulation and performance analysis
Figure 2 – Asynchronous circuit design flow under development.

In this initial description, however, it is not necessary for the designer to inform any detail regarding internal structure or the specific asynchronous protocols to be used in the circuit under development. The next step, the basic architecture design, identifies the number and relative characteristics of the basic blocks in the design (register files, ALUs, multipliers, etc.). We plan to automate this step by adapting variations in classical high-level synthesis, i.e., *schedulin*g, *resource sharin*g, and *bindin*g. In the next step, the micro-architecture design, the designer can choose to implement the architecture with various methods ranging from fine grain pipelines template-based using delay insensitive cells or the STFB templates, presented in this work, to components utilizing bounded delay assumptions with no fine grain pipelining. Once defined the micro-architecture design style, various optimizations can be applied, namely selection of the handshaking protocol, defining the level of pipelining, and slack optimization for pipelined designs. With this micro-architecture, the next step is to identify critical components and perform *handshaking optimization* to achieve higher performance and lower power. Based on the final micro-architecture, a gate or transistor level design can be generated. This can be done either automatically, using new template-based synthesis techniques that our group is creating, or manually. Finally, placement and routing can be applied basically the same way as for synchronous circuit design. This step may require buffer insertion, due to long wires, which would loop back to slack optimization step in an iterative way.

At every step in the design process, verification and performance analysis tools are used to verify the correct functionality and the overall performance. The focus this work is the generation of new templates for template-based design, as well as to help develop the above CAD frame for the automated design of asynchronous systems.

1.4 Contribution of this work

Our main objective is to present our novel high-performance asynchronous pipeline stages, the Single-Track Full Buffer (STFB) templates, which offers high throughput requiring only 6 to 10 transitions per cycle. To accomplish this we implemented:

1) *A set of linear and non-linear STFB stages*. These templates are freely available through MOSIS Educational Program into a library of standard cells with schematic, layout and symbol views, allowing their easy use (see appendix A).
2) *Implementation of a demonstration chip*. A 64-bit prefix adder and its test structures were designed and implemented, using the MOSIS TSMC 0.25 μm technology, in order to demonstrate the advantage of the small cycle time and modularity offered by the STFB templates as well the flexibility and easy of use of conventional (synchronous) back-end design flow to implement a STFB asynchronous design.

1.5 Organization

The remainder of this work is organized as follows. Section 2 provides relevant background information. Sections 3 and 4 describe our proposed 1-of-N templates in detail. Latency and throughput analysis of STFB buffers with QDI buffers are compared in Section 5. The demonstration test chip is presented on Section 6 followed by conclusions drawn in Section 7.

2 BACKGROUND

In the absence of the clock, providing global synchronization, masking logic hazards, and signaling the end of each computation step, asynchronous circuits operate using event-driven logic. In particular, asynchronous circuits are often decomposed into processing blocks that communicate data (called *tokens*) through asynchronous channels. This decomposition facilitates re-using asynchronous blocks and simplifies the design of complex systems.

2.1 Asynchronous channels

An asynchronous channel is a bundle of wires and a protocol to communicate data across the wires from a sender to a receiver. Figure 3 shows three different types of channels.

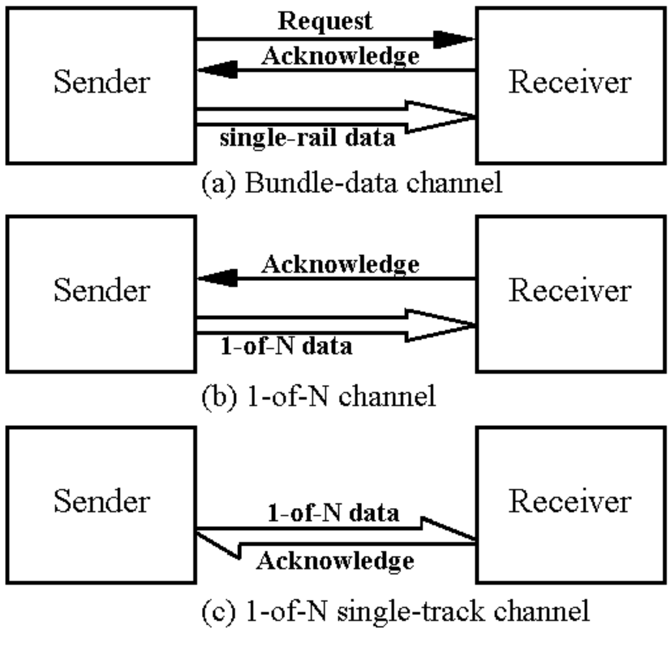


Figure 3 - Asynchronous channels.

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 by a matched delay line that is larger than the sender's computation delay plus some margin.

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

In the 1-of-N channel, the receiver detects the presence of the token from the data itself and, once it no longer needs the data, 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.

Berkel et al. [3] proposed single-track handshake circuits to control medium-grain bundled-data pipelines. Sutherland et al. [47] later developed faster single-rail GasP circuits to control fine-grain bundled-data pipelines. Nyström [34] recently 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 than STFB.

Figure 4 illustrates a single-wire single-track channel. The sender waits for the wire to be low (“ready”) before sending a request by driving the wire high (“busy”). After the receiver detects the wire is high and consumes the data, it drives the wire low.

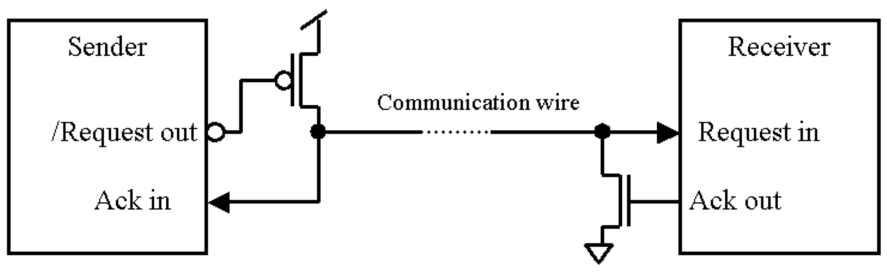


Figure 4 - Single-track protocol typical connection.

Note that “transceivers” can also be implemented using the single-track wire to transport data in both directions if, for every communication event, it is well defined which block will send and which will receive [3]. Similarly, mutually exclusive transmitters and receivers may be connected to the same wire [3]. These possibilities, however, were not covered in our STFB template for the sake of modularity, reliability and performance.

2.2 QDI weak-condition half-buffer (WCHB)

Figure 5 illustrates a well-known dual-rail buffer implementation called weak-condition half-buffer (WCHB) in [26]. L and R identify the left and right environments, 0 and 1 identify the false and the true rails respectively, and "e" identifies the enable signals (high means "ready" and low means "acknowledge"). After reset, L0, L1, R0 and R1 are low while Le and Re are high. Data arrives by one of the left inputs (Lx) rising. This will cause Sx to go low, which will drive the corresponding output Rx high and the left enable Le low. The left environment then will lower Lx while the right environment receives the data Rx and lowers Re. The buffer then raises Le and lowers Rx. The cycle completes when the right environment re-asserts Re. Note that for clarity reset circuitry and staticizers are not typically shown. Note also that the generation and reset of the output token implies that the corresponding input token has been consumed and reset, respectively, a property called weak conditioned in [26] and weak indicatability in [33].

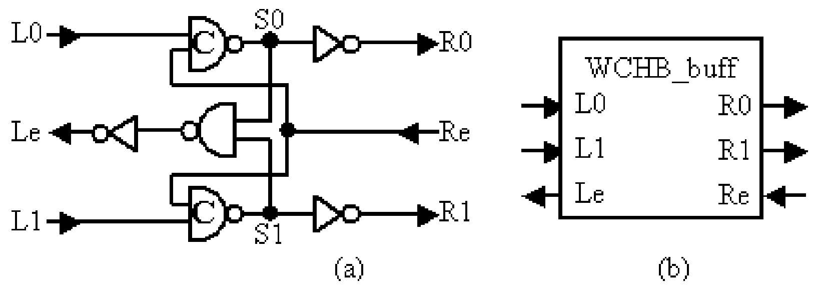


Figure 5 - QDI WCHB buffer: (a) schematic and (b) symbol.

We can derive an estimate of cycle time by counting the number of gate delays or *transitions* in a cycle of operation. The WCHB buffer is faster than other QDI buffers, having a forward latency (*fw*) of 2 transitions, a backward latency (*bw*) of 3 transitions and cycle time of only 10 transitions. However, for more complex processing blocks with many inputs, WCHB is not recommended because it generally requires too many stacked PMOS transistors, making it slower than alternative templates.

2.3 GasP bundled data

Figure 6 shows the GasP circuit where, after reset, L, R, and A are high. When L is driven low by the left environment, the self-resetting NAND will fire, driving A low. This will restore L, activate the data latches, and drive R low, propagating the signal and avoiding re-evaluation until after R is restored high by the right environment. The self-resetting NAND will restore itself by driving A high after 3 transitions. The output of the NAND controls the latches in a parallel single-rail datapath.

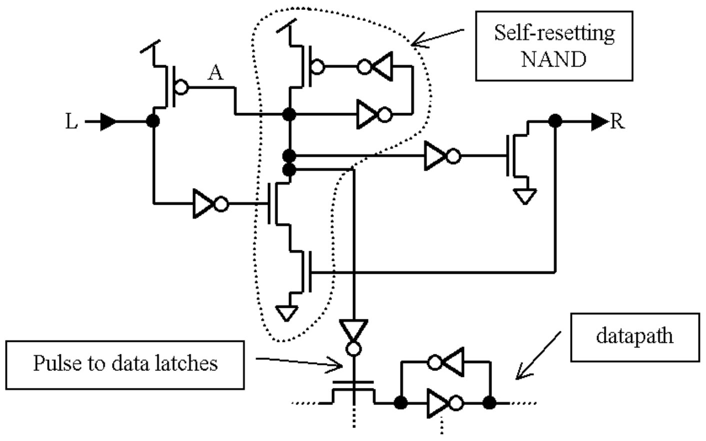


Figure 6 - GasP diagram.

GasP circuits take 4 transitions to forward data and 2 transitions to reset, i.e., 2 transitions to move a “bubble” (or a “blank”) backwards. Of the 4 transitions forward latency, approximately two transitions are required for latency through the latches and satisfying setup/hold times leaving approximately two transitions for computation. Note that the control circuit itself makes up the delay line and that it is the datapath designer's responsibility to pipeline the datapath to match the control circuit delay while satisfying all setup/hold times and time margin due to process variations.

2.4 Fine-grain vs. two dimensional pipelining

The QDI and GasP templates represent a fundamental dichotomy in pipelining philosophy. The GasP design targets standard datapath widths of, for example, 32-bits. In fact, GasP circuits can be viewed as a complex method of distributing a clock that naturally facilitates gated clocking. Consequently, GasP bundled timing constraint captures many of the same problems as clock distribution and clock skew since it has a global timing assumption that all the 32-bits in the width of the data path will be valid when the request arrives. The QDI templates, on the other hand, are generally applied to small datapaths, say 4 bits, and wider datapaths are made up of a two-dimensional array of communicating blocks [11][28][29]. The motivation of limiting individual QDI templates such as the WCHB to small datapaths is to keep the completion-sensing overhead to a minimum, thereby facilitating reasonable throughput while preserving robustness to timing. For our circuits, as we will see below, it also implies we must guarantee only local timing assumptions, which are easier to test and verify than a wide data-path bundle data constrain.

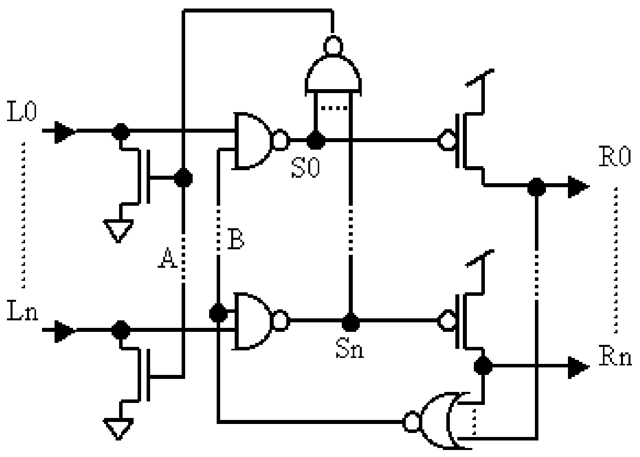
The completion of a wide datapath, if needed, can be pipelined across several pipeline stages using a technique called pipelined completion sensing [11][28][29]. Similarly, the broadcasting of a control signal affecting the entire datapath can be pipelined to avoid having a large completion tree for the acknowledgement signals. In this way, two-dimensional pipelines can have a cycle time that is independent of datapath width.

Moreover, the WCHB, along with other QDI templates, generally have significantly lower latency than their GasP template counterparts because they do not suffer from the latch delay and setup/hold times. Replicating the control circuits for each row (slice of bits) of the two-dimensional array, however, may result in increased area and power.

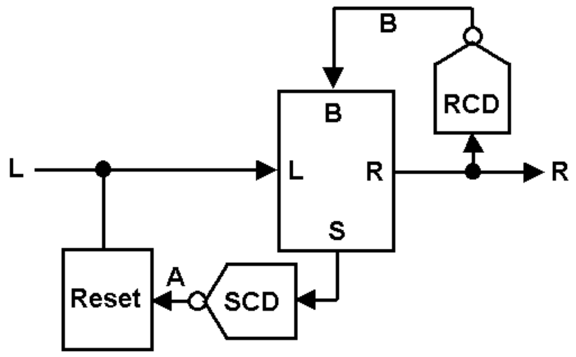
3 SINGLE-TRACK FULL-BUFFER CIRCUITS

3.1 STFB buffers

In asynchronous design, buffers are used to balance pipelines for performance-driven *slack matching* [26] or simply storing data. Figure 7 illustrates our 1-of-N STFB buffer template and its block diagram. When one of the n inputs (Lx) is driven high by the left environment, the corresponding NAND gate will drive Sx low, thereby driving both the corresponding Rx and “A” (the “**A**cknowledgement” signal) high. “A” going high causes Lx to reset low, enabling the left environment to send a new token. Meanwhile, Rx going high causes the “B” (“**B**usy”) signal to lower, restoring Sx high and preventing the NANDs to re-fire even if a new token arrives. The restoring of Sx, in turn, resets “A”. The cycle completes when the right environment lowers Rx, resetting “B” low, and allowing a new data token to be processed. Since distinct tokens can simultaneously be at the left and right environments, the template is said to be a full buffer and have capacity (slack) of 1 token per buffer.



(a)



(b)

Figure 7 - 1-of-N STFB buffer: (a) schematic and (d) block diagram.

As shown in the block diagram, the gate that drives "A" (Acknowledge) is called SCD (State Completion Detector) because it detects that the internal state of the template has captured the input token. The gate that drives "B" is called RCD (Right Completion Detector) because it detects that the output token has been sent to the right environment. The SCD is responsible for the reset of the input token and the RCD enables the main block to operate when the output channel is clear. Note that the generation of the output token *indicates* [30][48] that the corresponding input token was valid and consumed. However, the reset of output tokens is caused by the right environment and does not indicate that the input tokens have reset. Consequently, we call the STFB buffer, along with most STFB logic templates, *semi-weak-conditioned*. As such, there is a timing assumption that the template must reset the input channel before "A" is de-asserted.

Figure 8 shows, as an example, a dual-rail STFB buffer. Figure 9 shows an optimized version in which the static NAND gates driving S0 and S1 are merged into one dual-rail dynamic gate that is reset only by the "B" signal. Figure 10 shows a similarly optimized 1-of-4 STFB buffer circuit and symbol.

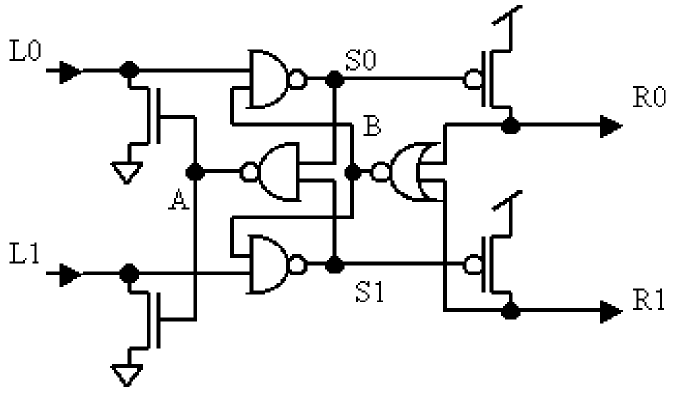


Figure 8 - Dual-rail STFB buffer.

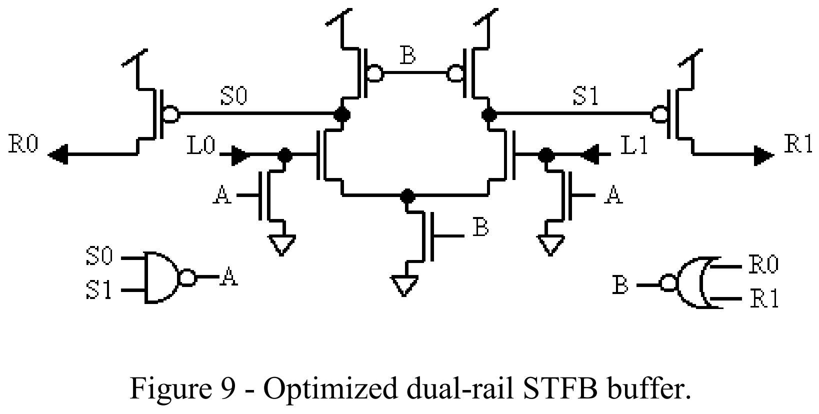


Figure 9 - Optimized dual-rail STFB buffer.

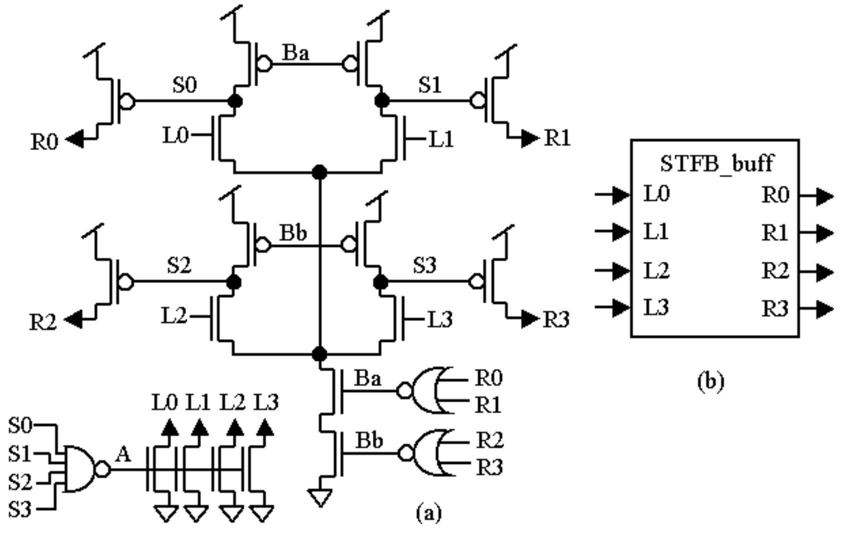


Figure 10 - Optimized 1-of-4 STFB buffer.

STFB buffers have a cycle time of 6 transitions. This is 40% faster than WCHB and the same as GasP. The latency is 2 transitions, which is the same as WCHB and half that of GasP.

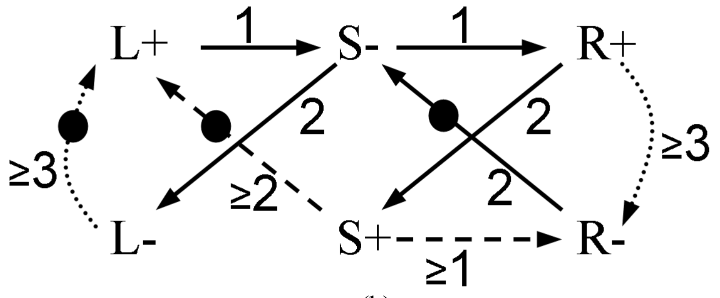
The STFB buffer, however, has higher complexity than both WCHB and GasP buffers. Compared to WCHB buffer, including required staticizers and reset circuit [26], the STFB buffer has 7 more transistors. This increased complexity, however, is mitigated by the fact that the proposed STFB buffer is a full buffer (i.e., has slack of 1), while WCHB is a half buffer (slack of ½). Moreover, the STFB buffer does not require the acknowledge wires (Le/Re), which may represent a significant saving in area and routing effort, and allow the implementation of more complex functions, which would require to move to PCHB since WCHB is used only for buffers. In addition, the power consumption per communication of STFB buffer is potentially lower than WCHB buffer since each communication requires half the number of wire transitions.

Compared to a GasP buffer with a standard 32-bit datapath, the area and power consumption of a STFB pipeline may be higher because the two-dimensional STFB pipeline will be made up of many buffers in parallel and each buffer will have its control circuit overhead.

Figure 11 shows the handshaking expansion (HSE) equation and the signal transition graph (STG) for the presented buffers. The notation “+”, “↑” and “-”, “↓” represent the rising and falling of the signals respectively. The left and right environments drive the dotted arrows and the dashed arrows represent timing constraints. The arrows are annotated with delays in terms of transitions. The greater than or equal sign (“ ”) reflects a timing assumption, which states that the separation between identified events is at least the specified number of transitions.

$$\text{STFB buffer} \equiv *[[\neg R \wedge L \rightarrow R\uparrow];\ L\downarrow]$$

(a)



(b)

Figure 11 - STFB buffer: (a) HSE (handshaking expansion) and (b) STG (signal transition graph).

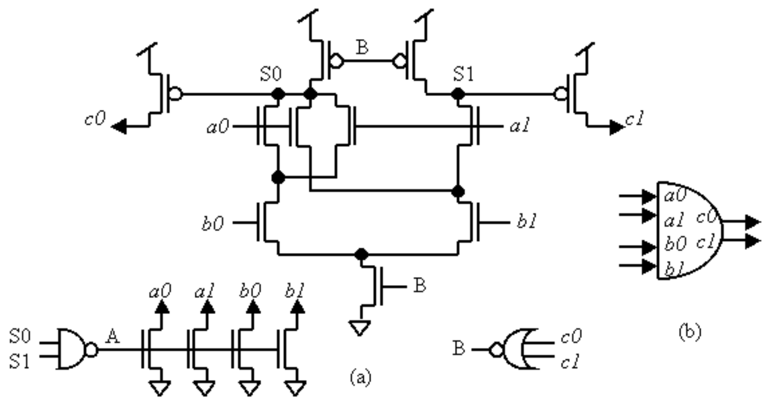
As can be deduced from the STG, the STFB buffer has somewhat tight timing constraints. In particular, the timing margin between the tri-stating of an output wire (one transition after S+) and the earliest time the environment can reset the wire (R-) is zero. Moreover, the timing margin between tri-stating of an input wire (two transitions after S+) and the earliest time the left environment can drive the wire (L+) is also zero. In particular, if these margins are violated, significant short circuit current may occur during the transitioning of the line. In addition, it is assumed that three transitions are sufficient to fully discharge/charge a line. To accommodate these constrains, the channel load needs to be bounded. This is achieved by limiting the wire length of the channels, which can be easily verified after the placement and routing phase. Moreover, automated static timing analysis tools are under development to further improve the design robustness and sign-off process. Unless otherwise noted, these timing constraints apply to all subsequent examples.

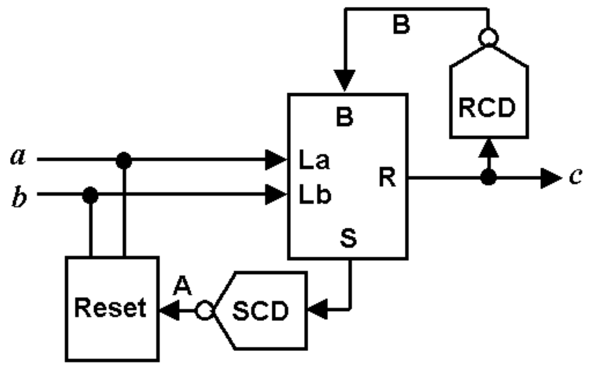
3.2 STFB forks and joins

This section covers a variety of non-linear pipelines stages that involve multiple input and/or multiple output channels and can perform more complex logic functions. While we focus on two dual-rail (1-of-2) inputs/outputs, templates that handle more channels and/or 1-of-N encoding are natural extensions.

3.2.1 Dual-rail STFB semi-weak-conditioned AND

Figure 12 illustrates an STFB AND stage and its block diagram that performs $c = a*b$, where a and b are dual-rail single-track inputs and c is the dual-rail single-track output.





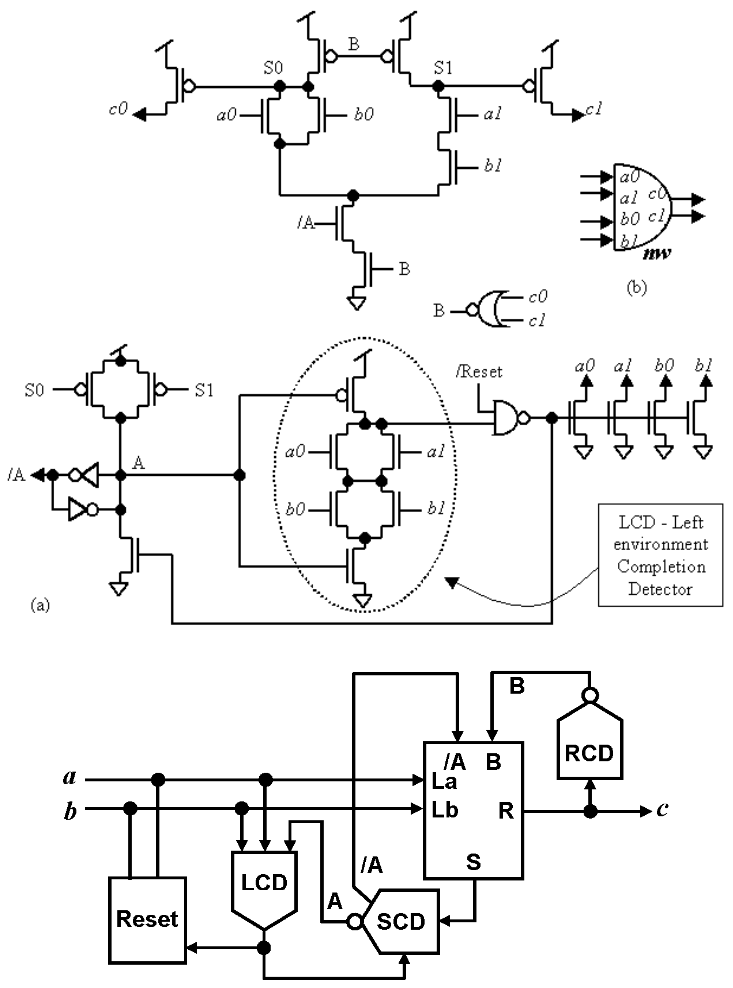
(c)

Figure 12 - SFTB semi-weak-conditioned AND: (a) schematic, (b) symbol, and (c) block diagram.

All the inputs are "acknowledged" by the signal "A" as soon as S0 or S1 goes low. For S1, this happens when *a1* and *b1* are high. For S0, *a0* or *b0* driven low is sufficient to define the logic result, but the circuit explicitly waits for one of the three input combinations 00, 01, and 10 to arrive before lowering S0. In this way, the evaluation of S0 also implies that both tokens (*a* and *b*) arrived, guaranteeing that the acknowledgement does not precede the arrival of a late token, making this gate semi-weak-conditioned.

3.2.2 Dual-rail STFB non weak-conditioned AND

Figure 13 shows a non weak-conditioned AND stage and its block diagram. This circuit generates a zero result token as soon as one of the inputs is zero even if the other input has not arrived. When all the inputs are finally present, however, the stage sends an acknowledgement to all inputs.



(c)

Figure 13 - Non weak-conditioned STFB AND: (a) schematic, (b) symbol, and (c) block diagram.

To do this, while forwarding the early zero result, the gate's SCD (State Completion Detector) sets "A" high, which will disable the logic for future evaluations by keeping "/A" low and will hold the information that an acknowledge is pending. When the LCD (Left environment Completion Detector) detects that all input tokens are present, the acknowledge signal is passed to the transistors that will “consume” the data at the inputs and “A” is reset to zero. This will restore “/A” high and the gate will be ready to evaluate again. This LCD structure adds two transitions to the cycle time but loosens the timing margin between S- and resetting the inputs (corresponding to L- in Figure 11) by two gate delays.

Notice that for multiple inputs, this gate has a much simpler NMOS transistor stack than the weak-conditioned STFB AND.

3.2.3 Dual-rail STFB OR and STFB XORs

By re-arranging the transistors in the evaluation stack (main block), different logic functions may be implemented within the STFB template. A dual-rail STFB OR performs the logic operation: $c = a+b$, where a and b are dual-rail single-track inputs and c is the dual-rail single-track output. This function can be implemented either with semi-weak-conditioned logic or with non-weak-conditioned logic simply by rearranging the transistors in the NMOS stack of the AND circuits presented in Section 3.2.1 and 3.2.2.

Similarly, the dual-rail STFB XOR performs the logic operation: $c = a \oplus b$, where a and b are dual-rail single-track inputs and c is the dual-rail single-track output. The STFB XOR, however, must be semi-weak-conditioned, because, for any XOR gate, all input token values must be known before the output value could be computed.

3.2.4 *Dual-rail STFB non-conditional merge*

The non-conditional merge operation concatenates the incoming data from different mutually exclusive input channels. Figure 14 shows a 2-to-1 non-conditional merge circuit, symbol, and block diagram.

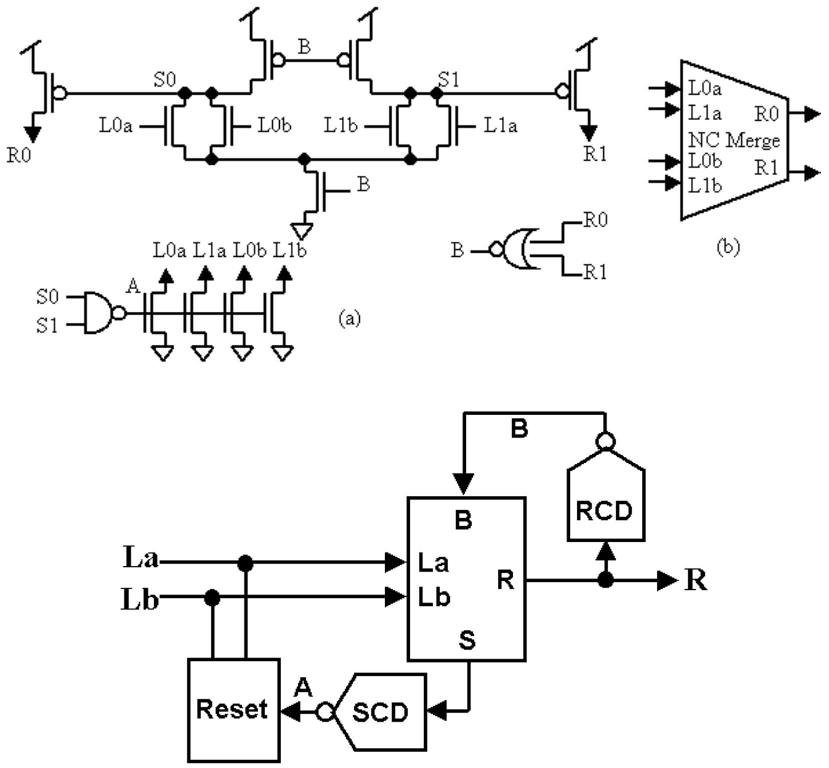


Figure 14 - STFB NCMerge: (a) schematic, (b) symbol, and (c) block diagram.

3.2.5 *Dual-rail STFB fork*

The fork operation consists of replicating the incoming data to several different paths if all output paths are ready. Otherwise, the input data must wait.

Figure 15 shows the 1-to-2 fork stage. Notice that the four-input NOR gate (with a stack of four PMOS transistors) driving B slows down the STFB fork performance. To speed-up the B signal, however, we can use 2 two-input NOR gates to generate Ba and Bb, and replace the B NMOS transistors with stacked Ba and Bb NMOS transistors (similar to what is shown in Figure 10).

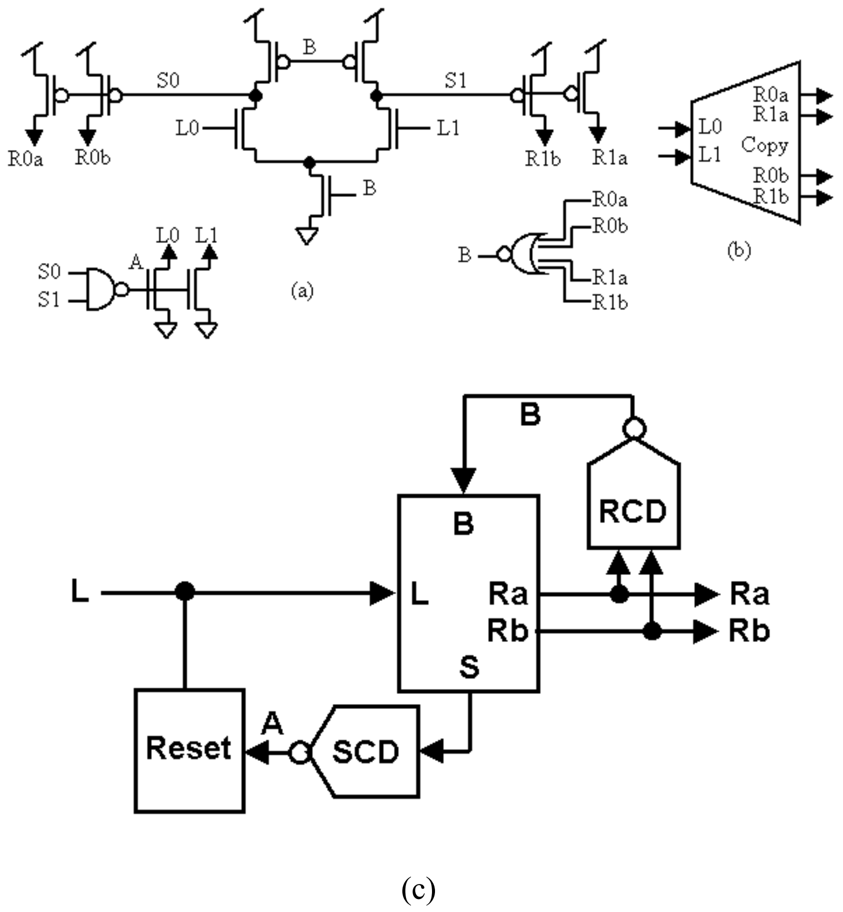


Figure 15 - STFB copy: (a) schematic, (b) symbol, and (c) block diagram.

3.2.6 *Dual-rail STFB full adder*

This is an example of STFB computational stage. To implement a full adder (STFB FA) we need to compute the sum and the carry out before resetting the inputs. As illustrated in Figure 16 and Figure 17, this can be done with a three-input XOR and a three input majority (MAJ) gate. The XOR generates the sum (s=a+b+ci) and the

MAJ generates the carry out (*co*=MAJ(*a,b,ci*)). Figure 18 shows the block diagram of the STFB FA.

In this structure, the carry evaluates as soon as enough inputs arrive to define the correct output value but the acknowledgement waits for both outputs to be generated which, because the sum is an XOR gate, implicitly means that all inputs have arrived. Note that the acknowledgement circuitry adds two gate delays to the cycle time but also loosens the timing margin between S- and resetting the inputs by two gates.

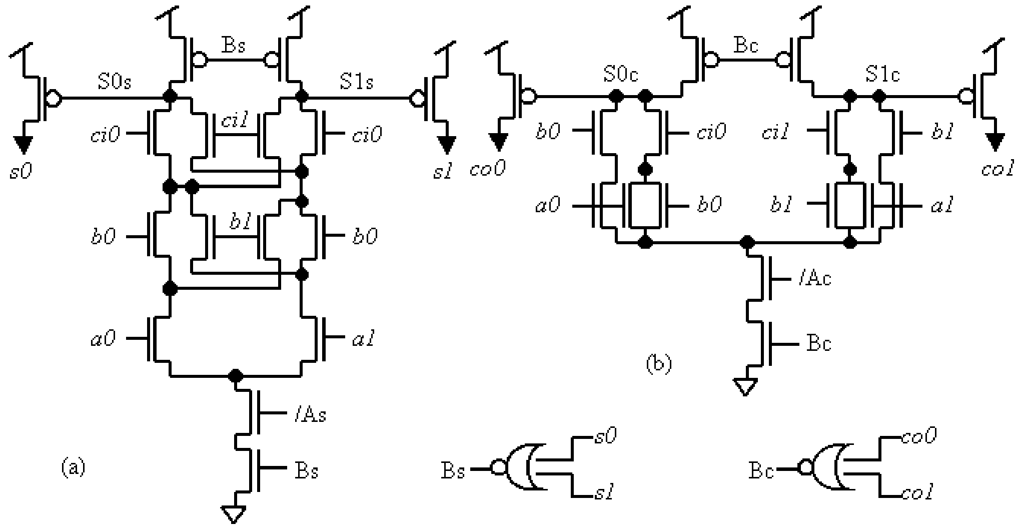


Figure 16 - STFB FA: (a) XOR and (b) majority gates.

The long nmos stacks in the sum and carry circuits can be reduced by one transistor by removing the transistors controlled by /As and /Ac and making As and Ac new inputs of their respective RCD NOR gates.

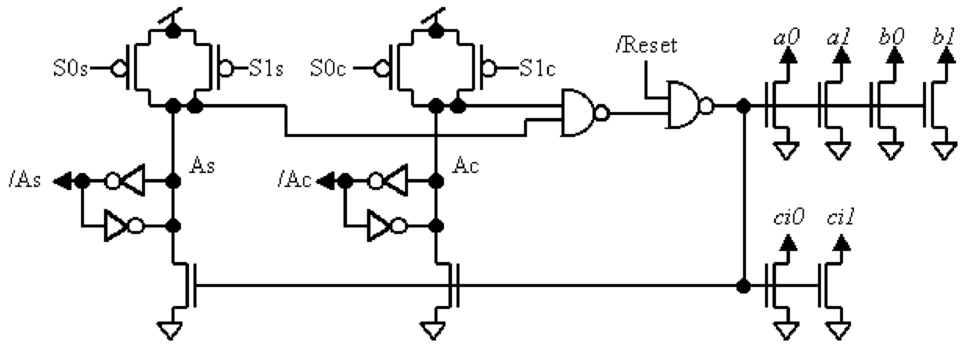


Figure 17 - STFB FA acknowledgement circuit.

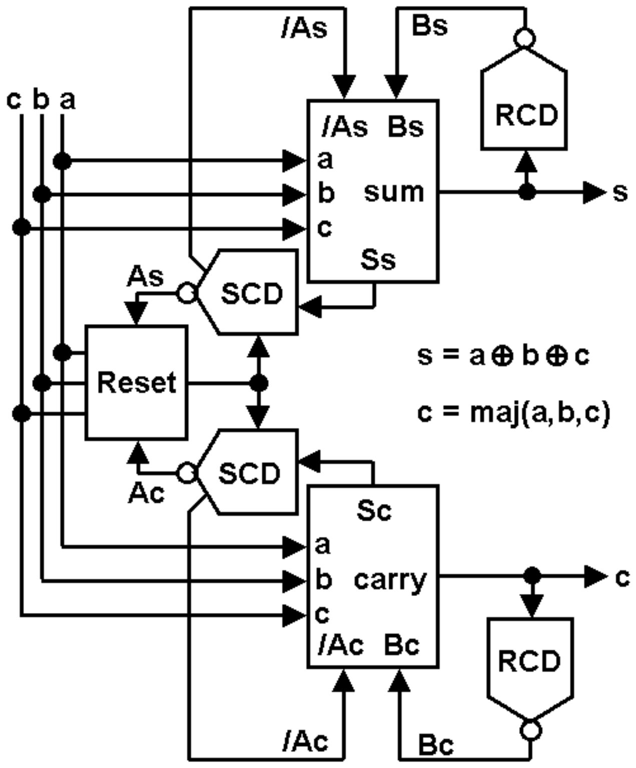


Figure 18 - STFB FA block diagram.

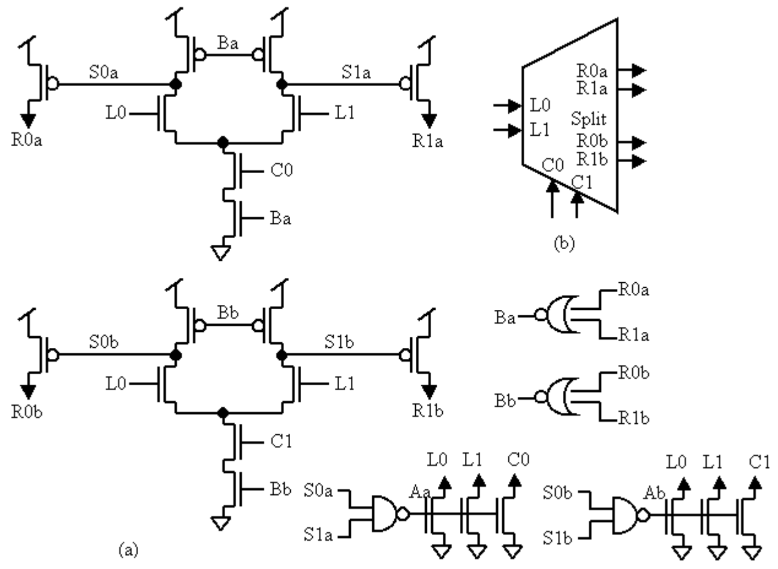
3.3 *STFB conditional stages*

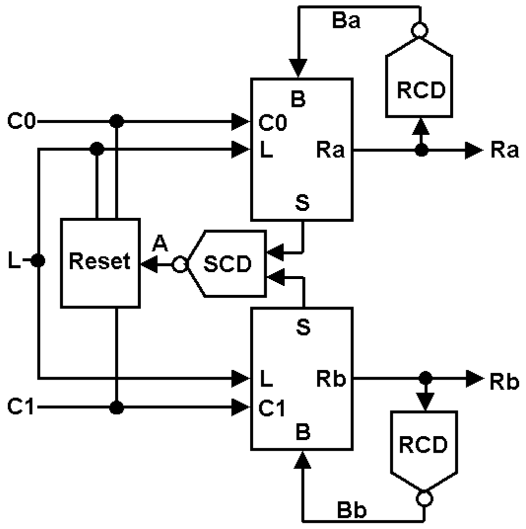
This Section covers a variety of stages in which input and/or output channels are conditionally read or written.

3.3.1 Dual-rail STFB split

The split operation consists of forwarding incoming tokens to one of two output channels based on the value of a control (C) channel. If the chosen output path is busy, the data must wait. Note that the micropipeline version of this block, which samples the control signal rather than consuming it, is called a *select* [46].

Figure 19 shows the 1-to-2 STFB split circuit, symbol, and block diagram. In this example, when C is low (C0 = 1), L is directed to Ra and, when C is high (C1 = 1), to Rb. Interestingly, the STFB split allows a token to be forwarded to one channel even if the other channel is busy (each output has its own RCD), which increases the degree of parallelism.



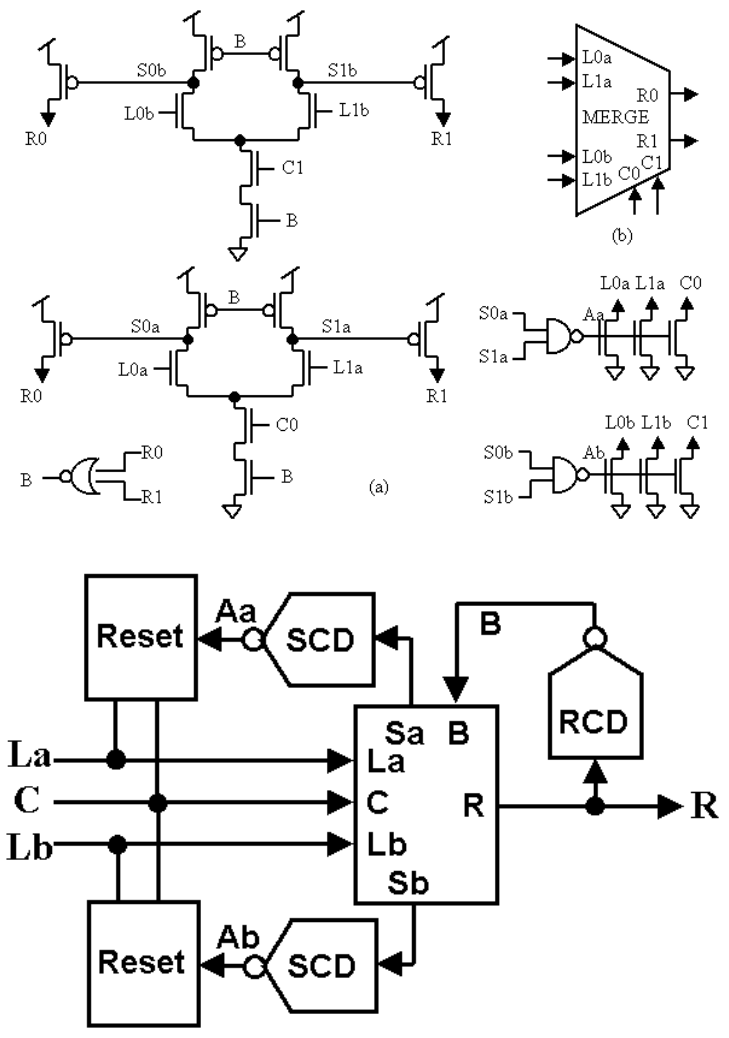


(c)

Figure 19 - STFB split: (a) schematic, (b) symbol, and (c) block diagram.

3.3.2 Dual-rail STFB merge

The merge operation consists of choosing one of the incoming tokens based on the value of a control (C) input. If the output path is busy, the input and control tokens must wait. After forwarding the data, the control token is also consumed.



(c)

Figure 20 - STFB Merge: (a) schematic, (b) symbol, and (c) block diagram.

Figure 20 shows the 2-to-1 merge circuit, symbol, and block diagram. When C is low (C0 = 1), La is directed to R and, when C is high (C1 = 1), Lb is directed to R.

3.3.3 Dual-rail STFB one bit memory

Figure 21 shows a STFB one-bit memory stage. The circuit of has a static memory unit (two inverters), an input (L), an output (R), and a control channel (C). If the control input is low (C0=1), the memory content is transferred to the output (R) and C0 is consumed. If the control input is high (C1=1), the memory is written with the L input value and both, C1 and L, are consumed.

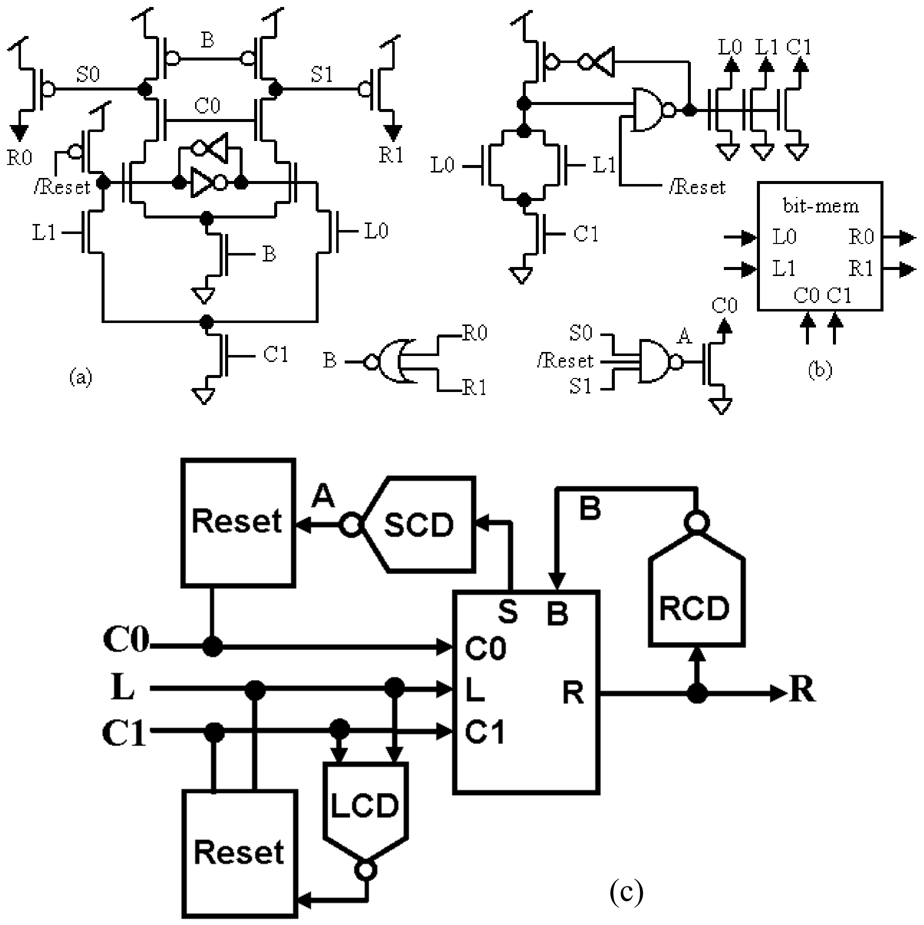


Figure 21 - STFB 1-bit memory: (a) schematic, (b) symbol, and (c) block diagram.

Notice that the control signal flows only trough the channel C, which guarantees the read and write operations are executed in the requested order. Also, there is a timing assumption that the 3 transitions of the write operation are long enough to set the memory value.

3.4 Auxiliary stages

This Section covers bit generators used to generate a stream of tokens, bit buckets to consume unwanted tokens, converters between single-track and four-phase protocols, and staticizer/reset circuitry.

3.4.1 Four-phase to STFB converters

The "transmitter" circuit, illustrated in Figure 22, is our proposed interface between four-phase asynchronous logic and STFB. In this circuit, if Le is high and the right environment is ready, a data arriving from the left environment will be transmitted to the right environment and the signal Le will be set low. This also disables the buffer, avoiding re-transmitting the same data after the right environment consumes it. Le will remain low until both inputs return to zero (four-phase protocol). When this happens, Le is set high and the transmitter is ready for the next data.

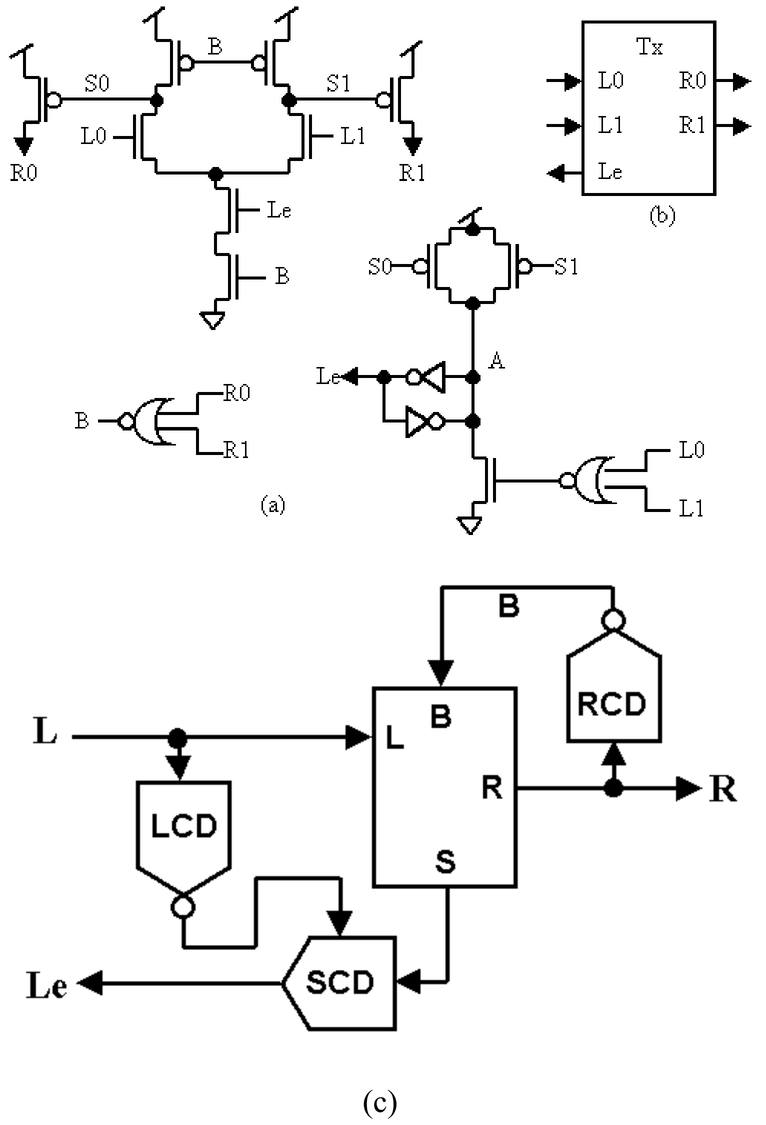
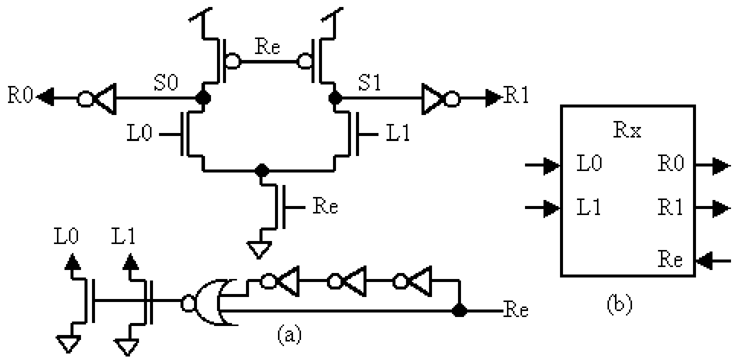
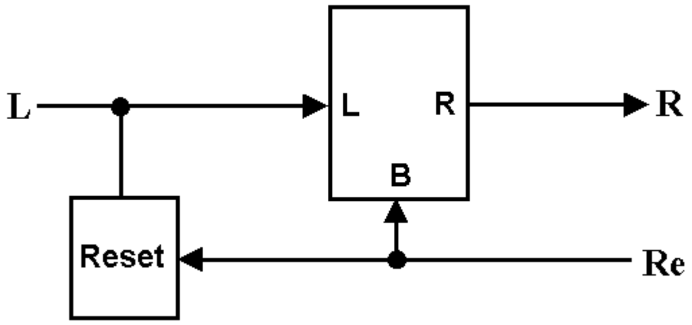


Figure 22 - STFB Tx: (a) schematic, (b) symbol, and (c) block diagram

The "receiver" circuit, illustrated in Figure 23, is our proposed interface between STFB and four-phase asynchronous logic. In this circuit, if Re is high (the right environment is ready), a data from the left environment will be received and the buffer will wait for the signal Re to be set low. When Re goes low, a three gate-delay pulse is generated to consume the left environment data and the receiver is reset (R0 and R1 goes low). While Re is low, R0 and R1 are reset and no new data is received (four-phase protocol). When Re returns to high, the receiver is ready for the next data.





(c)

Figure 23 - STFB Rx: (a) schematic, (b) symbol, and (c) block diagram.

The cycle time of these converters is 10 transitions when connected to WCHB buffers, which matches the WCHB buffer cycle time.

3.4.2 *Dual-rail STFB bit generators and bit buckets*

A bit generator creates a data token every time the line is empty, while a bit bucket consumes unwanted tokens. Both are also useful in test circuitry. The proposed STFB bit generator and bit bucket are shown in Figure 24.

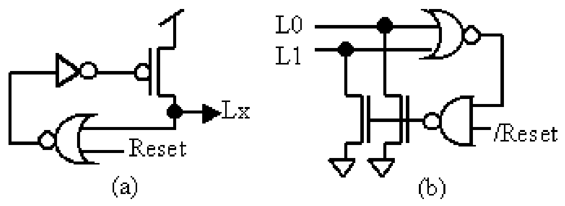


Figure 24 - STFB bit (a) generator and (b) bucket.

3.4.3 Channel initializer

Some circuits, such as loops, may require some form of initialization that cannot be done by a bit generator since it is required just once. One approach is to modify the pipeline stage that needs to be initialized and, instead of simply reset the input wires during the reset phase, place a valid token at its input. This requires a new design and layout of that stage. Another approach is to use an external drive circuit to pull a wire up during a short 3 transitions to "inject" a token in the line after the /Reset signal is deasserted (rise edge of the /Reset signal). Figure 25 shows our channel initializer circuit and symbol. It is an edge to pulse converter with open-drain PMOS driver. The value *i* represents the injected value in the channel after reset.

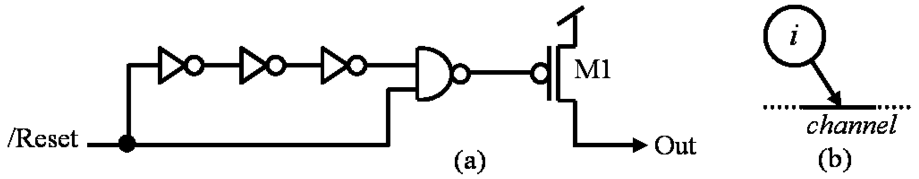


Figure 25 - Channel initializer (a) schematic and (b) symbol.

Since the STFB stages are very fast, we must take care not to use the channel initializer in two consecutive channels to avoid one token overrunning the other. Rather, for neighboring channels that require initialization, we propose to use modified stages.

Another approach is to add a non-conditional merge stage in the pipeline, by replacing a buffer for example, with one input connected to the pipeline and use to other input to insert the initialization tokens we want. This method was used in our demonstration design as described below.

4 STFB STANDARD-CELL DESIGN

In this chapter we present a number of implementation issues of the STFB standard-cell design. 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.

4.1 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 9.

4.2 Balanced response

Symmetrized transistor stacks are utilized to perform the SCD and RCD functions inside the cell. Figure 26 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.

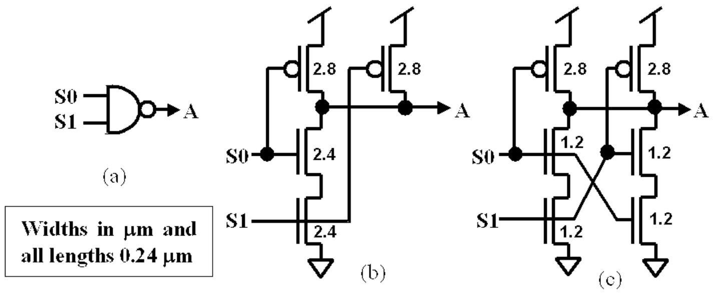


Figure 26. Sub-cell NAND2B_28_12: (a) symbol, (b) conventional diagram and (c) implemented balanced input diagram.

4.3 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 27. If the output channel is empty, the "B" signal is high, "R" is low, and "NR" is 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.

M6 and M7 are responsible to fight leakage and charge-sharing. When the output channel is empty, all output rails are low, "B" is high, and thus M7 alone is active. On the other hand, when one output rail is high, "B" is low, and M6 fights leakage and holds "S" high. For this output rail that is high, M6 and M5 are active, while for all other output rails, M6 and M7 transistors are active. M7 can be much smaller than M6 because while "B" is high, the risk of charge-sharing problems is dramatically reduced as the internal node C at the bottom of the N-stack is actively driven low and thus its capacitance cannot contribute to charge-sharing.

Compared to the original template [18], this template also improves robustness to charge sharing in the N-stack because this output sub-cell now has a lower switching threshold voltage of the "S" signal. In the initial template, M1 was driving the line without M2 and M3, which made the activation threshold of the "S" signal approximately 0.5V (i.e., Vtp) below the power supply voltage (V_{DD}). By adding M2 and M3, the activation threshold of "S" is much lower (around 60% of V_{DD}).

The introduction of M5 also yields a significant performance improvement allowing longer maximum wire length when compared with the initially proposed template [18]. In particular, M5, controlled by the staticizer inverter ("NR" signal), quickly asserts "S" after its output rail is driven high. This enables M6 to be smaller, thereby reducing the load on the "B" signal enabling a faster cycle-time.

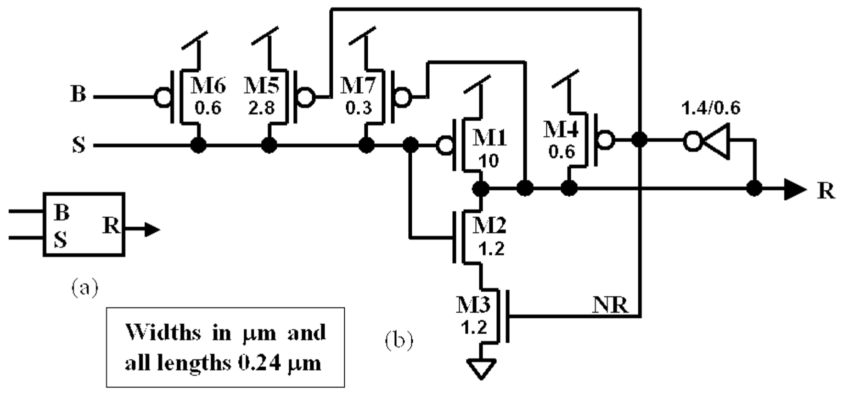


Figure 27. Sub-cell STFB_POUT (a) block diagram and (b) schematic.

4.4 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 28. 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.

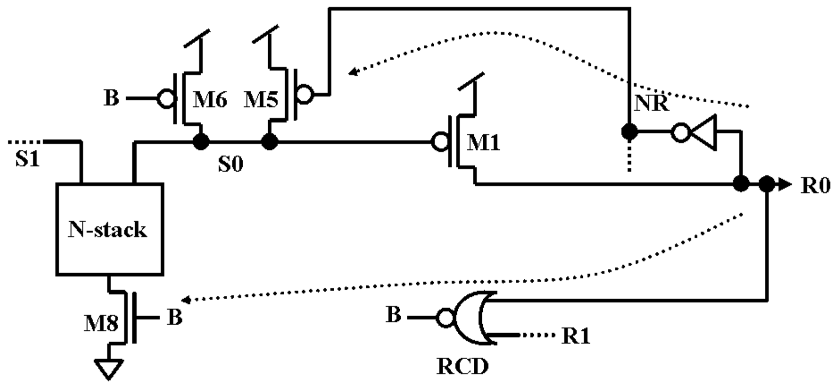


Figure 28. B and NR simultaneous activation.

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 29, 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.

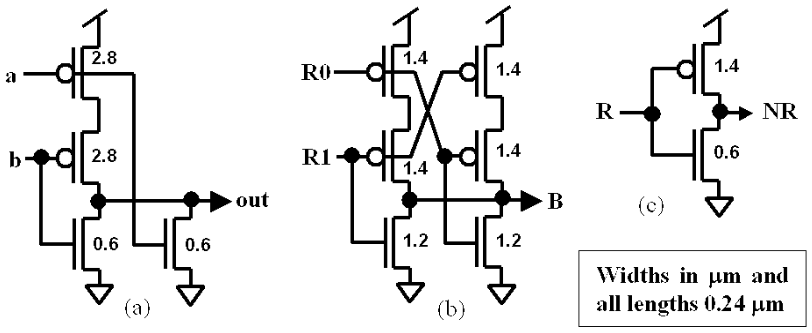


Figure 29. (a) conventional 2-input NOR, (b) balanced RCD and (c) staticizer inverter.

4.5 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 [18]. However, a 3-input NAND gate is much less efficient than a 2-input one. Figure 30.a shows the initially proposed 3-input SCD, where a 3-input NAND gate controls the reset transistors. Figure 30.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.

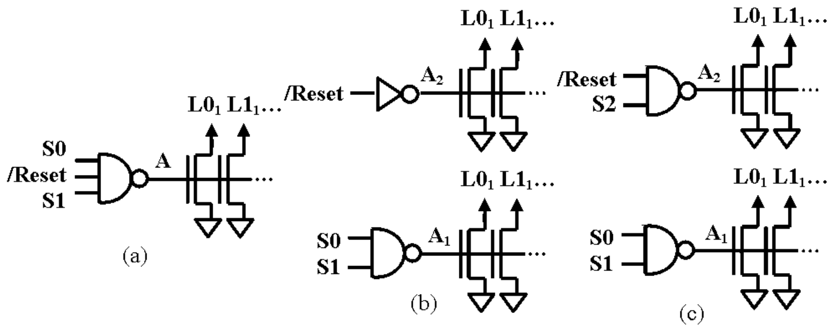


Figure 30. SCD and reset (a) initially proposed and the implemented (b) 1-of-2 and (c) 1-of-3.

4.6 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 31 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 $I_{dp} = I_{peak}/2$ [39].

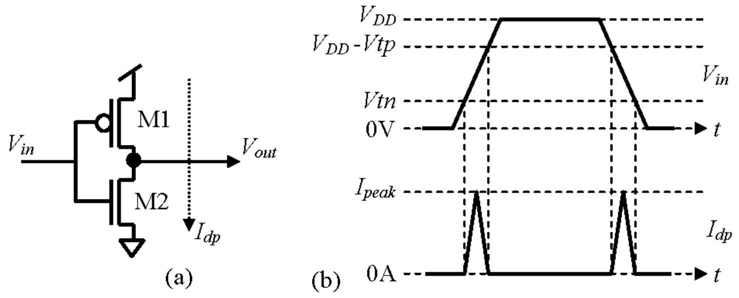


Figure 31. (a) inverter and (b) direct-path current.

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 32 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.

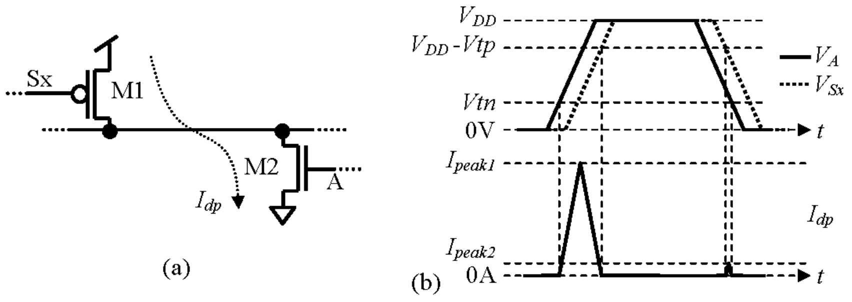


Figure 32. (a) STFB output/input drivers and (b) direct-path current if $V_A \neq V_{Sx}$.

Figure 33 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.

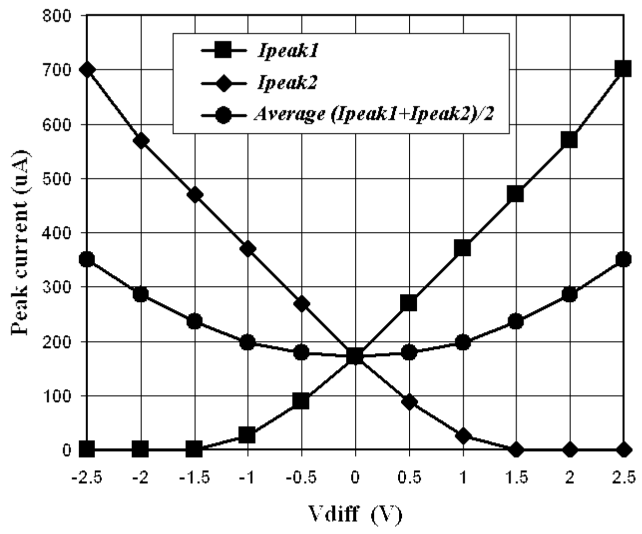
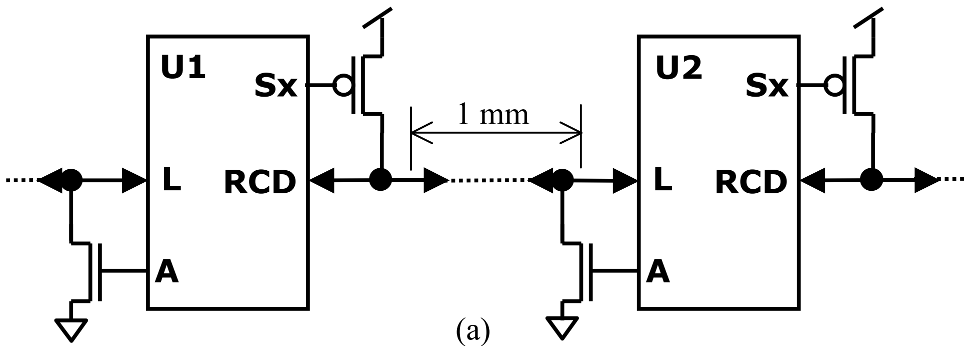


Figure 33. Peak direct-path current versus the PMOS-NMOS gate voltage difference.

Therefore, since we can size the drivers of V_A and V_{Sx}, we may avoid most of the I_{dp} even using our six-transitions STFB template. This careful sizing allows the state signal "Sx" of one stage not to overlap the acknowledge signal "A". This can be illustrated by a simulation of four STFB buffer (U0, U1, U2 and U3), where between U1 and U2 there is a 1 mm long wire and between U0 and U1, and U2 and U3, there is a very short wire as on Figure 34, Figure 35 and Figure 36.



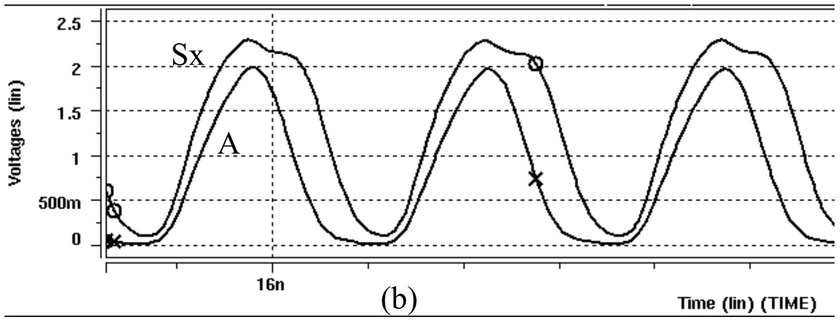


Figure 34 – (a) Two consecutive STFB buffers at full-throughput with 1mm long wire between them and (b) “Sx” (U1) and “A” (U2) signals (V_{DD} = 2.5V).

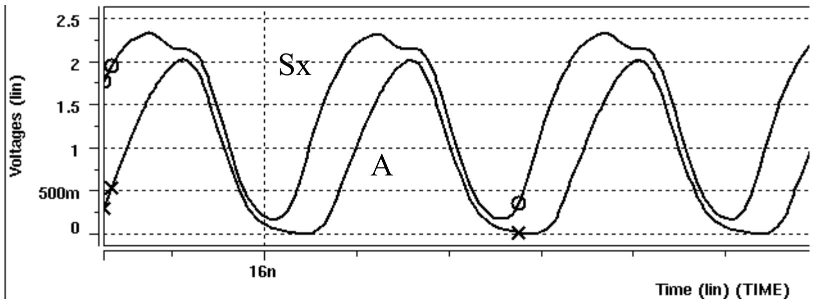


Figure 35 – Left side stage “Sx” (U0) and “A” (U1) signals with a very short wire between U0 and U1 (V_{DD} = 2.5V).

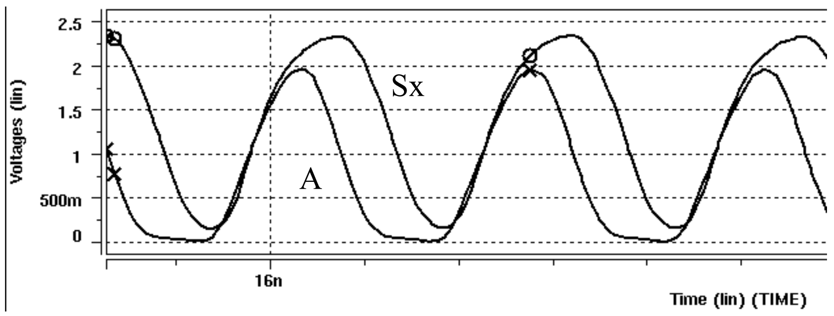


Figure 36 - Right side stage “Sx” (U1) and “A” (U0) signals with a very short wire between U1 and U2 ($V_{DD} = 2.5V$).

4.7 Reset tree

As the circuits grow in complexity and number of stages, special care needs to be taken with the /Reset signal to avoid the destruction of any token that reaches a stage that is still being reset (reset skew). Also, the /Reset rising edge needs to be fast to guarantee that all the stages connected to that Reset line are operational when the process starts. One option is connect all the stages /Reset wires to a big driver that would reset all stages effectively simultaneously. Another alternative (less brute-force) is to create a balanced reset tree of inverters where, at the leafs of the tree would be connected to all the bit generators, channel initializers, STFB Tx (see Section 3.4.1) and initialized stages and passive stages would be connected to leafs that have two or more fewer inverters from the root. This allows the passive stages to come out of reset at least two or more transitions earlier than their active counterparts, providing a reset margin ensuring the passive stages are ready to accept tokens from their active counterparts.

4.8 Noise margin

As for any family of digital circuits, we need to consider the STFB templates reliability to noise. We use the worst-case analytical analysis described in [12], and applied in [58] and [15], with the intended process (TSMC 0.25µm) parameters, where the minimum transistor size used in our circuits are: Wn = 0.6 µm and Wp = 1.4 µm for the minimum width of the NMOS and PMOS transistors respectively. For this analysis, we are using the transistor sizing strategy described on Section 4.1.

Figure 27 shows the STFB output stage where the state signal "S" is hold high by the transistor M7. This means that the NMOS transistor stack has to over-power the state pull-up transistors M7 in order to lower the respective state "S". Therefore, a high level input signal (V_{IH}) needs to be higher than 0.75V, which is bigger than just the NMOS threshold voltage (V_{tn} = 0.53V). If M7 were stronger, V_{IH} would be higher (close to half of the power supply voltage: V_{DD} / 2). However, this would also slow down the circuit and increase the direct-path current for every operation.

Noise can cause a signal V_S, the ideal correct input value, to be perceived by the receiver circuit as $V_R = V_S + V_N$, where V_N is additive noise. If V_S = 0 V, the worst-case noise must be smaller than V_{IH} (0.75V). For $V_S = V_{DD}$, the worst case noise must be smaller than half V_{DD} to avoid change the "state" of the staticizer holding the line. To be reliable we need to have a signal-to-noise ratio (SNR) bigger than one for both cases as shown in equations (1) and (2).

$$SNR_L = \frac{V_{IH}}{V_N} \qquad (1)$$

$$SNR_H = \frac{1}{2} \cdot \frac{V_{DD}}{V_N} \tag{2}$$

A good part of the system-created noise is proportional to the signal amplitude swing, which means that increasing V_{DD} will not improve the *SNR*. Therefore, we will analyze the noise as shown in equation (3).

$$V_N = K_N . V_{DD} + V_{NI} \tag{3}$$

where, $K_N . V_{DD}$ represents the noise sources that are proportional to V_{DD} (2.5V) such as cross talk and signal-induced power supply noise, and V_{NI} represents the noise sources that are independent of the signal amplitude such as receiver offsets and unrelated power supply noise.

Table 1 - Noise source analysis

Parameter	**Definition**	**Value**
K_C	Cross talk coupling coefficient for two 100 µm long 0.4 µm wide metal 4 wire with 0.5 µm spacing	0.1
$Attn_{CP}$	PMOS staticizer cross talk noise attenuation.	0.97
$Attn_{CN}$	NMOS staticizer cross talk noise attenuation.	0.88
K_{PS}	Power supply noise due to signal switching.	5% [58]
K_{NP}	*Worst case:* $K_{NP} = Attn_{CP}.K_C + K_{PS}$	0.147
K_{NN}	*Worst case:* $K_{NN} = Attn_{CN}.K_C + K_{PS}$	0.138
Rx_O	Next stage input offset	0.1 V
Rx_S	Next stage sensitivity	0
PS	Power supply noise (5% [58] of 2.5V)	0.125 V
$Attn_{PS}$	Power supply noise attenuation	1
Tx_O	Output offset	0
V_{NI}	Worst case: $V_{NI} = Rx\_O + Rx\_S + Attn_{PS}.PS + Tx\_O$	0.23 V
V_{NP}	Worst case noise: $V_{NP} = K_{NP}.V_{DD} + V_{NI}$	0.60 V
V_{NN}	Worst case noise: $V_{NN} = K_{NN}.V_{DD} + V_{NI}$	0.58 V
SNR_H	Worst case $SNR_H = 1.25 / V_{NN}$	2.2
SNR_L	Worst case $SNR_L = 0.75 / V_{NP}$	1.3

Table 1, shows the parameters used in our analysis. The meaning of each parameter is detailed below:

K_C : The cross talk coupling coefficient K_C is estimated by the equation below:

$$K_C = \frac{C_C}{C_O + C_C} \tag{4}$$

where, C_C is the parasitic coupling capacitance between the "aggressor" and the "victim" wires, and C_O is the capacitance between the "victim" wire and the substrate including the input and output capacitance of the stages connected by this wire. For a 100 μm long, with spacing of 0.5 μm, and 0.4 μm wide wire implemented using metal 4 with in the TSMC 0.25 μm process, and connecting the output of an STFB buffer to an input of another STFB buffer, we have, approximately, the wire to substrate capacitance C_W = 2.5 fF, the STFB buffer output capacitance (including staticizer) C_{out} = 37.7 fF, and the STFB buffer input capacitance C_{in} = 17.4 fF. Therefore, we estimate: C_O = C_W + C_{in} + C_{out} = 2.5 + 37.7 + 17.4 = 57.6 fF. Since the capacitance between two metal 2 wires, for a wire spacing of 0.5 μm, is $6.45\text{x}10^{-2}$ fF/μm, we estimate C_C = 6.45 fF, resulting K_C = 0.1.

$Attn_C$: The static driver cross talk noise attenuation $Attn_C$ should be near half if the line were continuously driven. However, STFB stages actively drive the line high during 3 transitions, and low during 3 transitions. This means that, unless the pipeline is running at full throughput (6 transitions per token), the output staticizers are holding the line when it is not being actively driven. To compute a worst-case scenario, we considered the victim line hold by the staticizer, while the aggressor is actively driven.

We can compute $Attn_{CN}$ = Rsn/(Rsn + Rdp) and $Attn_{CP}$ = Rsp/(Rsp + Rdn), where Rs is the staticizer impedance and Rd is the driver impedance. For the STFB buffer we have Rsn = 6.9 Ω, Rdn = 0.82 Ω, Rsp = 24.2 Ω and Rdp = 0.94 Ω, resulting $Attn_{CN}$ = 0.88 and $Attn_{CP}$ = 0.97, which means almost no attenuation. In other words, the current staticizers are very weak and make little difference with respect to the noise.

K_{PS} : The power supply noise due to signal switching K_{PS} is assumed to be 5% as in [58].

Rx_O: The next stage input offset *Rx_O* is the difference between the nominal V_{IH} and the minimum V_{IH} expected (reducing V_{IH} reduces our noise margin), estimated to be < 0.1V.

Rx_S: The next stage sensitivity *Rx_S* represents the extra voltage range required over V_{IH} in order to properly activate the next stage. This, in fact, would improve our noise margin since it would require a final V_{IH} closer to $V_{DD}/2$. Also, in our STFB stage, once the driven state (S0 or S1) is low enough to activate the PMOS driver, the stack pull-up became weaker and the switching point is very abrupt due to the positive feedback. Therefore we selected 0V, meaning that the stage will react immediately once V_{IH} is reached.

PS: The power supply noise unrelated to signal switching *PS* is assumed to be 5% as in [58].

$Attn_{PS}$: The power supply noise attenuation $Attn_{PS}$ is 1, meaning: no attenuation (worst-case) assuming V_{IH} is independent of the power supply.

Tx_O: The output sensitivity *Tx_O* represents variation in the output voltage, which is 0V for full-swing (rail-to-rail) circuits.

The final SNR_L is 1.3 for the two 100 µm parallel lines. For 300 µm lines, the SNR_L would be approximately equal to one, the safe limit for the worst-case SNR. Although this analysis is very conservative, based on it, we dedicated extra care in the layout and post-layout verification to avoid malfunctions due to noise issues.

However, this analysis is limited to 0.25 µm or bigger technologies since it does not take into account the line resistance effect, which is very important for deep sub-micron processes. For these processes, a more robust single track protocol is needed, and we propose the static single-track (SST) protocol as described below.

4.9 Static single-track protocol

For deeper sub-micron technologies, the impact of increased wire resistance must be addressed. In particular, dynamic long-distance wires are very dangerous because staticizers are generally too weak to combat coupling noise in the presence of highly resistive wires. Naive solutions include shielding the at-risk wires, increasing the size of staticizers, and/or increased the spacing between wires, all of which have substantial costs in area, power, and/or performance. This section introduces a novel Static Single-Track (SST) protocol that addresses these issues by continuously driving the wire at only a marginal cost in area, power, and performance.

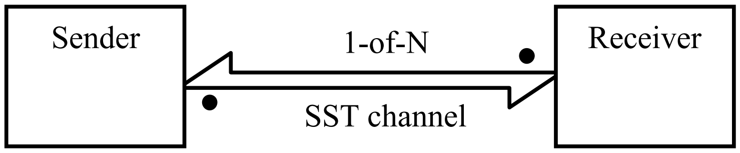


Figure 37. 1-of-N Static Single-Track asynchronous channel.

Figure 37 shows the 1-of-N SST channel block diagram. This new asynchronous communication protocol can be described by two main operations modes that each communication block has during the hand shaking through the single-track: the drive and the hold modes (indicated by the half arrow head and the dot, respectively). For the SST protocol, each communication stage has the ability to change a wire logic level by strongly driving it towards the one logic level during a bounded time interval, and the same block is responsible to strongly hold the same wire, as long as necessary, if the wire reaches the opposite logic level. Therefore, although it is a single-track channel, there is no use of weak staticizers to hold the logic level in the wires, whatever is the wire logic level, inclusive during transitions, there is always a strong drive path as if it were statically driven, and there is no fight between the drive and the hold phases. Initially, we called SST the “no fight” protocol [17]. Moreover, for high-resistive wires, this protocol may improve performance by seamlessly allowing the single-track wire to be strongly driven on both ends towards the same direction as explained below.

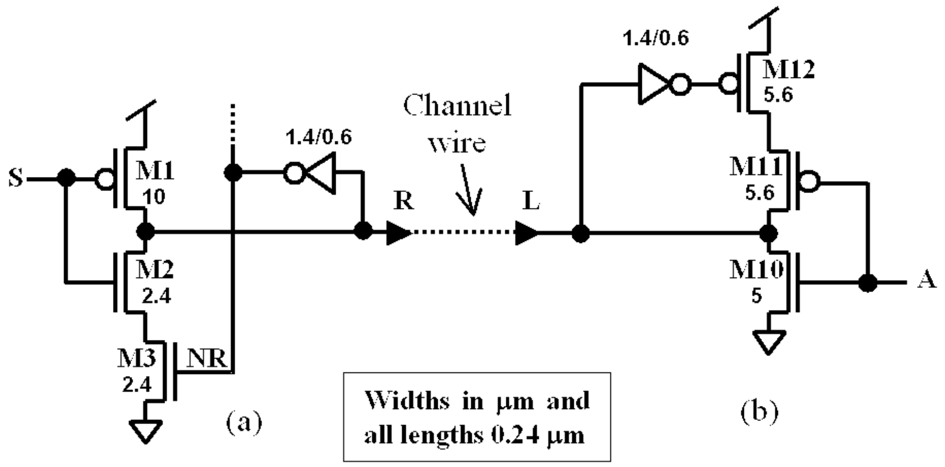


Figure 38. Static Single-Track channel drivers implementation: (a) sender and (b) receiver "drive-and-hold" circuits.

One proposed implementation of SST line driver is shown in Figure 38. The active drivers are M1 and M10. The additional transistors M2 and M11 ensure that there is no fight during transitions of the wire, allowing M3 and M12 to be as large as desired to combat coupling noise. Therefore, each side of the wire has complementary "drive-and-hold" circuits.

In particular, let us explain how M3 and M12 act to continuously drive the channel wire. Consider first the case in which the sender side "S" is high and "A" is low. In this case, the line can be low (for example after reset) or high (a token is stalled on the channel). While it is low, M2 and M3 strongly keep the line low, whereas when the line is high, M11 and M12 strongly keep the line high. Conversely, when the sender side "S" is low and "A" is low, M1 actively drives the wire high. Lastly, when "A" is high M10 actively drives the wire low. Thus, in all cases, there is a strong path from the wire to a power supply.

The ability to drive the wire continuously is counter-intuitive to the single-track protocol in which both the sender and receiver go tri-state after sending/receiving a token. The key leap is to realize that the sender/receiver can also be responsible for actively driving the line **before** sending/receiving the token, and this is the great accomplishment of the SST protocol. It may also be instructive to view this novel driver as a combinational staticizer of a dynamic inverter in which the feedback inverter is duplicated and the N and P portions are split between sender and receiver sides.

For deep-sub-micron high-resistive wires, the SST protocol may improve the driving characteristics of long wires because, once a transition is detected, the "hold" side is activated, helping to fully drive the wire. This unique characteristic may significantly contribute to overcome long wires impedance and noise related issues.

Lastly, it should be emphasized that the SST protocol is not specific to STFB circuits, but can also be applied to other single-track circuits, including GasP [47] and ASTPL [34].

4.10 Timing margin: The ten transitions STFB template

Figure 40 shows an alternative 10 transitions STFB template, and Figure 39 its STG. This template offers a self-reset 3-transitions active output (S- to S+ period), which is independent of the output wire load, one transition of margin for R+ to hold S+, and two transitions of margin between the drive/reset phase of the output/input single-track wires.

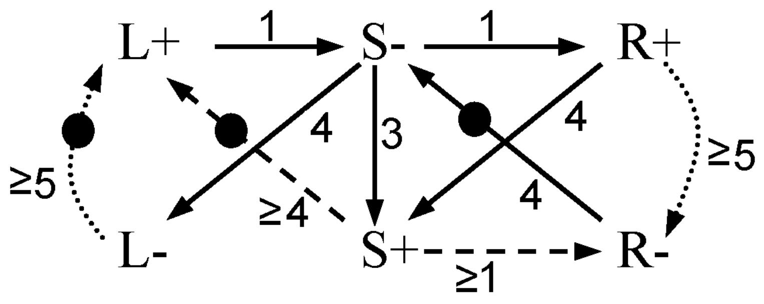


Figure 39 - 10-transistions STFB signal transition graph (STG).

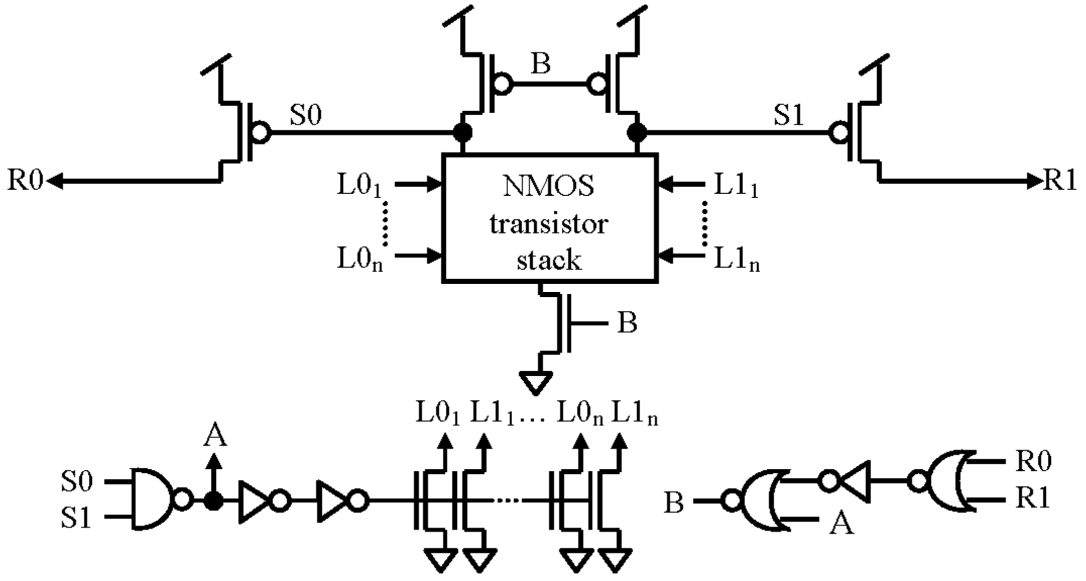


Figure 40 - 10-transitions STFB template.

As we can see, in the 10-transitions STFB template, two more gates (2 transitions) are added in the A and B signal paths, and the signal A is used to “self-reset” the states S0 and S1 by lowering B. Once the output has a token, the B signal is hold low even after A is restored low. These extra transitions increase the template cycle time to 10 transitions, while the active and reset phase are still 3 transitions long, which results in 2 transitions (2 gate-delay) margin on each side of the template drive (input/output).

The price for these margins is a slightly more complex circuit, which may not be much difference for complex stages, for instance, the full-adder (Figure 16 and Figure

17), which already has 8-transitions, and a 67% slower cycle time, when compared with the 6-transitions template. However, for latency critical systems ("token-limited" as shown on Figure 35) the 10-transitions STFB template offers the same performance as the 6-transitions one. Moreover, 10-transitions STFB is still much better than most of the QDI templates [26], which would require 14 to 18 transitions per cycle, and it can be used in conjunction to the 6-transitions templates since the active and reset phase has the same duration (3 transitions) on both templates.

For complex stages with many inputs and outputs, for 1-of-4 stages for example, the 10-transistions template may have some of the added inverters in the SCD and RCD changed to NAND/NOR gates allowing easy handling of multiple tracks. For example, a 10-transistion 1-of-4 STFB stage could have two 2-input NOR gates connected through a 2-input NAND gate in order to perform the RCD function.

However, in order to emphasize the advantages of the small cycle time offered by our circuits, especially in situations where we need high throughput, and we have small loops with a single token or big loops with multiple tokens, we plan to concentrate our research on the 6-transition STFB template family.

5 THROUGHPUT ANALYSIS AND COMPARISON

5.1 Introduction

The performance of a circuit can be optimized based on different metrics, such as: throughput, energy, latency, area, etc. In this thesis, we concentrate our efforts to optimize throughput [26], while latency, area estimation and the $E\tau^2$ [49] will also be used for comparison. Although more complex STFB stages offer a bigger advantage when compared with QDI stages with the same functionality, most of the time, we will compare STFB buffers with WCHB buffers, which are the fastest QDI stages.

The $E\tau^2$ metric is the product of the energy (E) times the square of cycle time (τ). This metric is approximately independent of the power supply voltage (V_{DD}) since E is proportional to V_{DD}^2 and τ is proportional to V_{DD}^{-1}, and it allows us to compare different designs even when they are running at different power supply voltages or when the energy and throughput of one pipeline are both higher than another. The higher the $E\tau^2$ metric, the less efficient is the pipeline, which means more energy per processed token.

5.2 Pipeline optimization

The static capacity (S) of a pipeline ("static slack" [26]) is given by the stage static capacity ($s = 1$ for full-buffer and $s = ½$ for half-buffer stages) times the number of stages (N) in the pipeline, as shown in the equation below.

$$S = s.N \tag{5}$$

Reducing the capacity of a pipeline may cause deadlock of the system, if the capacity of the pipeline is not greater than the number of tokens it must contain. Increasing the capacity of the pipeline does cannot introduce errors in a large class of asynchronous systems called "capacity elastic" ("slack elastic"). This class, however, does not include most systems that perform some type of arbitration. In particular, when arbiters exist, care must be taken to ensure that adding pipeline stages does not introduce deadlock.

We can add capacity to a pipeline by changing the individual stages capacity (change *s* from ½ to 1), which can be done in the QDI templates, or by just adding buffers to the pipeline. Since the STFB templates are already full-buffer ($s = 1$), we can adjust the STFB pipeline capacity only by buffer insertion or removal.

For a given pipeline with *N* stages, we want to analyze the throughput (*t*) as a function of the number of tokens in the pipeline (*x*), as shown below. Both *t* and *x* are averages and it is assumed that the pipeline is running at steady state. This means that the throughputs measured at the both ends of the pipeline are equal and approximately constant over time.

$$t = f(x) \tag{6}$$

The average forward latency (*fw*) of a stage in the pipeline can be measured in seconds or number of transitions, and represents the time it takes for a token to move forward through an empty pipeline stage. If the pipeline is empty and we introduce one token in the pipeline ("token limited" or "forward latency limited" operation), it takes *Fw* (seconds or transitions) for it to arrive at the end of the pipeline, we can compute the *fw* average of all the stages in the pipeline as:

$$fw = \frac{Fw}{N} \tag{7}$$

The throughput of a "token limited" pipeline can be estimated by the number of tokens in the pipeline (*x*) divided by the total forward latency (*N.fw*), as shown below.

$$t(x) = \frac{x}{N.fw} \tag{8}$$

The average backward latency (*bw*) of a stage in the pipeline can also be measured in seconds or number of transitions, and represents the time it takes for a "bubble" (or "hole") to move backward through a pipeline stage. Assuming that the pipeline is full, if we introduce one bubble at the output of the pipeline ("bubble limited" or "backward latency limited" operation), as the bubble moves backward, the tokens move forward one stage at the time. It takes *Bw* (seconds or transitions) for it to arrive at the beginning of the pipeline and we can compute the *bw* average of all the stages in the pipeline as:

$$bw = \frac{Bw}{N} \tag{9}$$

The throughput of a "bubble limited" pipeline can be estimated by the number of bubbles in the pipeline, which is the static capacity minus the number of tokens in the pipeline (*S* - *x*), divided by the total backward latency (*N.bw*), as shown below.

$$t(x) = \frac{S - x}{N.bw} \tag{10}$$

Notice that the throughput is zero if the pipeline is completely empty ($x = 0$) or completely full ($x = S$). If we start with an empty pipeline and we increase x, $t(x)$ will also increases until the internal cycles of the stages or internal cycles in the pipeline (non linear pipeline, i.e. with loops, forks and joins) limits the peak throughput (T). At this point, x is called the "minimum dynamic capacity" ($d_{\min}$) ("minimum dynamic slack" [26]) and $t = f(d_{\min}) = T$. If we keep increasing x, T will remain the same until the backward latency start limiting the throughput. At this point, x is called the "maximum dynamic capacity" ($d_{\max}$) ("maximum dynamic slack" [26]) and, at this point we still have: $t = f(d_{\max}) = T$. If we keep increasing the pipeline occupancy x, the throughput will decrease towards 0. The graph $t = f(x)$ is a "trapezoid".

However, for our templates used within linear pipelines, we have no internal cycles that can limit the stage handshake, and there will be only one optimum number of tokens in the pipeline that maximizes the throughput (one optimum x). Resulting: $d_{\min} = d_{\max} = d$, which is simply called the "dynamic capacity" (d) ("dynamic slack" [26]) of the pipeline, where $t = f(d) = T$. The graph $t = f(x)$ is now a "triangle" as shown in Figure 41.

The average cycle time (τ) of a pipeline can be obtained by the inverse of the peak throughput (T), which happens when $d_{\min} \leq x \leq d_{\max}$. For an optimized homogeneous linear pipeline, τ is also the stage cycle time, which can be extracted from the STG diagram of that stage, in terms of number of transitions, by counting the number of transitions required by the stage and its neighbors to complete one cycle of operation.

Therefore, at the peak throughput (T) we have:

$$t = \frac{S - d_{\max}}{N.bw} = \frac{d_{\min}}{N.fw} = T = \frac{1}{\tau} \tag{11}$$

We can, then rearrange (11) to the equations below:

$$\frac{1}{N.fw} = \frac{T}{d_{\min}} \quad \text{and} \quad \frac{1}{N.bw} = \frac{T}{S - d_{\max}} \tag{12}$$

Now, from equations (8), (10), (11) and (12) we can describe $f(x)$ as:

$$\begin{aligned} f(x) &= T\frac{x}{d_{\min}} \quad \text{for } 0 \le x \le d_{\min} \\ f(x) &= T \quad \text{for } d_{\min} \le x \le d_{\max} \\ f(x) &= T\frac{S - x}{S - d_{\max}} \quad \text{for } d_{\max} \le x \le S \end{aligned} \tag{13}$$

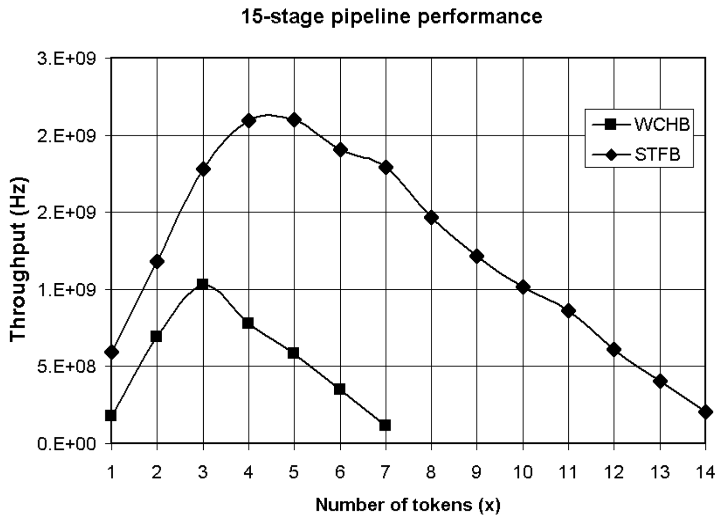
Notice that, the real throughput $t \le f(\mathrm{x})$, since $t = f(\mathrm{x})$ only for stead state throughput.

Also, from equation (11), we can find the relations:

$$d_{\min} = \frac{N.fw}{\tau} \quad \text{and} \quad d_{\max} = S - \frac{N.bw}{\tau} \tag{14}$$

If we analyze fw, bw and τ, as number of transitions, for just one pipeline stage (N = 1), we will find the dynamic capacity of the STFB 6-transitions templates as d_{STFB_6t}

= 2/6 = 1/3. This means that, to reach maximum throughput, we need just 3 stages for every token. For the WCHB and the STFB 10-transitions templates we have: d_{WCHB_10t} = 2/10 = 1/5, which imply that we need 5 stages per token to reach maximum throughput. For the QDI Pre-Charge Half-Buffer (PCHB) and Pre-Charge Full-Buffer (PCFB) [26], we have d_{PCHB_14t}= 2/14 = 1/7 and have d_{PCFB_12t}= 2/12 = 1/6, requiring seven and six stages per token respectively.



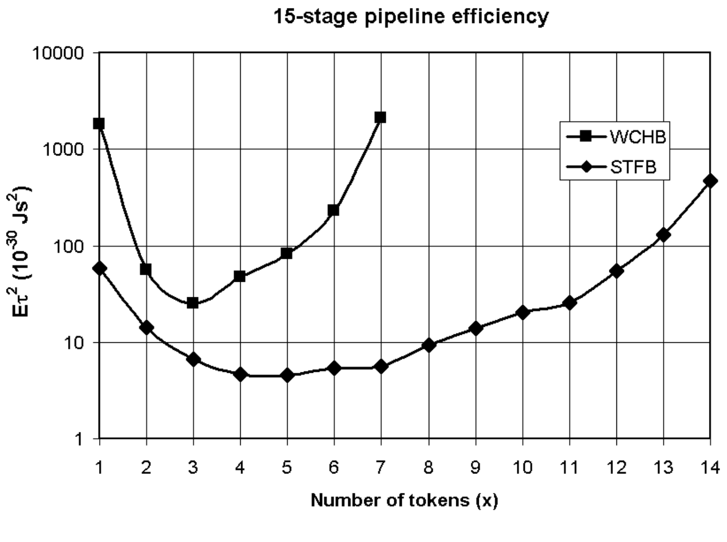


Figure 41 - Comparison of two 15-buffer pipelines: (top) throughput and (bottom) $E\tau^2$ metric versus pipeline occupancy (*x*).

Figure 41 shows the simulation results of two 15-buffer pipelines implemented with WCHB and STFB buffer templates. The transistor sizing strategy for both templates was the same used in our library and demonstration design: twice minimum size strength for the N/P-stack and eight times the minimum size for the line drivers. Different transistor sizing can lead to somewhat different results, but the presented theory would still apply. Although, the theoretical up-slope of the triangle graph ("token limited" region) should be the same for all pipeline with the same overall forward latency (*Fw*), or pipelines with the same number of stages (*N*) and templates with the same forward latency (*fw* = 2 for STFB, WCHB, PCHB and PCFB), and that we could estimate the slope by *T/d*, which is 1/(*N.fw*), we can see that STFB performance is higher since it has smaller forward latency in terms of time due to its domino logic style N-stack. The down-slope of the triangle graph ("bubble limited" region) is determined by the overall backward latency (*Bw*) and this line cross the *x* axis where $x = S$. The $E\tau^2$ graph also indicates that the better efficiency of STFB is evident (by a factor of 10 at peak throughput). This metric allows us to say that the STFB pipeline could match the WCHB speed (by lowering the power supply voltage) and would require much less energy per token to perform the same job.

However, the theoretical model, described above, shows that the maximum throughput (T) is equal to the inverse of the cycle time ($1/\tau$). This clearly demonstrates that the small cycle time of the STFB will offer higher throughput for the same number of stages or equivalent throughput with much less stages (less area and power).

Therefore, to take advantage of the STFB templates, we need to select applications that require steady state ultra-high throughput. This is a non-trivial task, since it is usually very difficult to feed/read a pipeline so fast. Because of that, we selected the 64-bit prefix adder with an input and output circuitry that allows it to run at full throughput as described on Section 6. Since the STFB has twice and three times the throughput of WCHB and PCHB respectively, we believe that STFB can be easily used to implement shared resources, saving area and power, and to alleviate bottle-necks on a mix-template design.

6 THE EVALUATION AND DEMONSTRATION CHIP

6.1 Introduction

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.

6.2 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 [22][23]:

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

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:

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

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 [22][23][5], 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 [22], 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:

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

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

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

$$(K_{j-2^x+1:j}, P_{j-2^x+1:j}, G_{j-2^x+1:j}) =$$

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

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

Step 4 (1 stage deep)

Compute:

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

$$c1_{n-1} = G_{0:n-1} + P_{0:n-1} c1_{-1}$$
$$c0_{n-1} = K_{0:n-1} + P_{0:n-1} c0_{-1}$$

Figure 42 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 43 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).

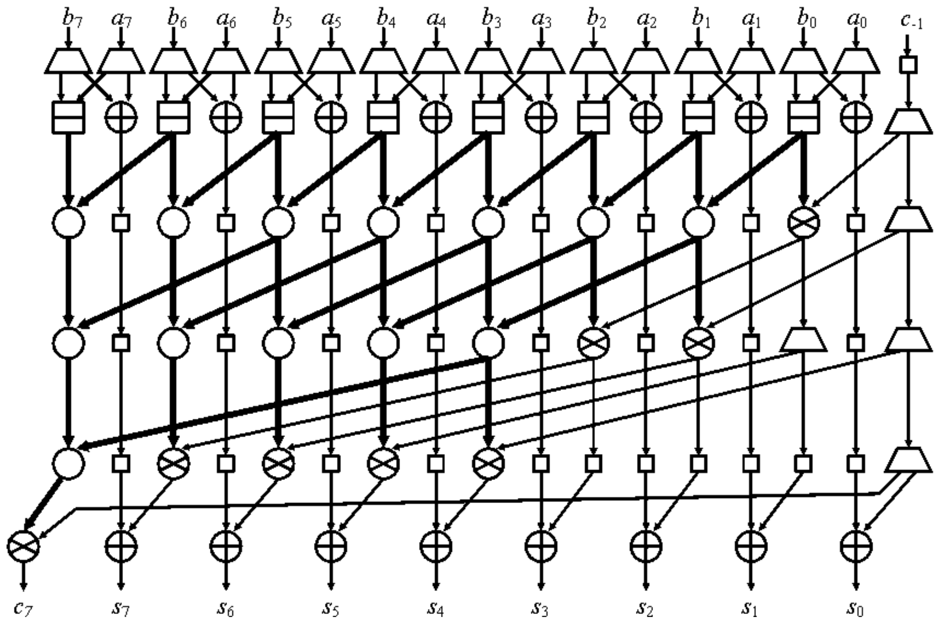

b_7 a_7 b_6 a_6 b_5 a_5 b_4 a_4 b_3 a_3 b_2 a_2 b_1 a_1 b_0 a_0 c_{-1}
c_7 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Figure 42. 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 43. Pipeline stages utilized in the adder.

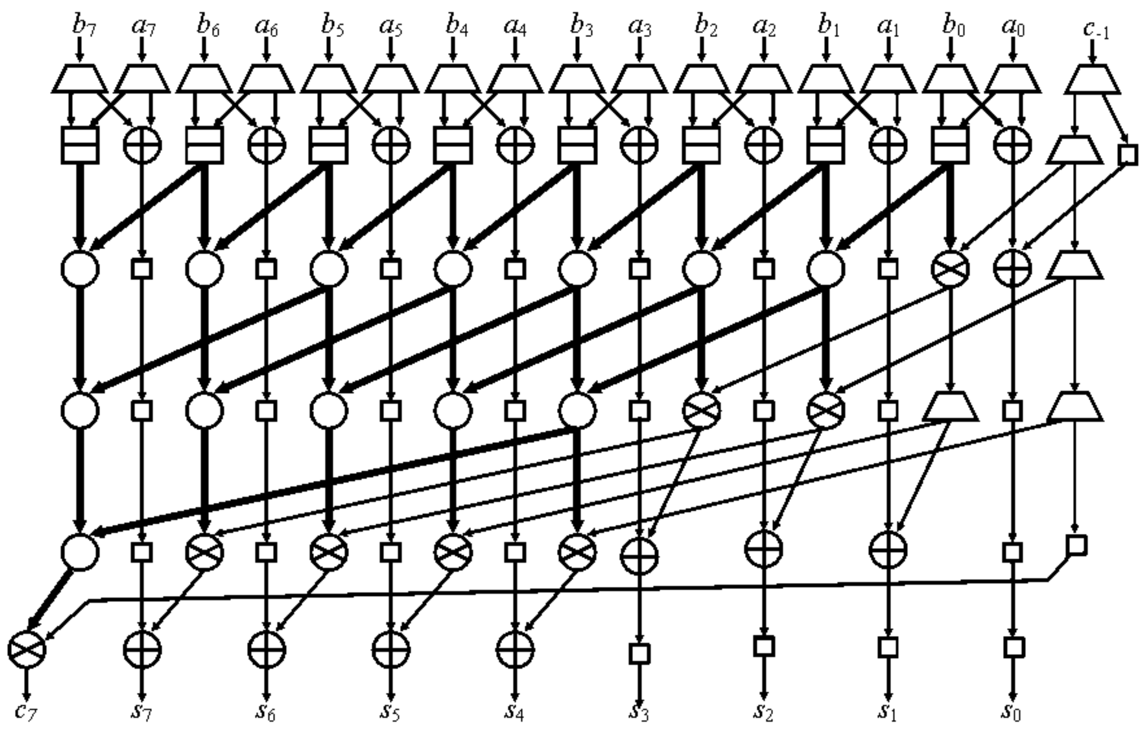


Figure 44. 8-bit async. prefix adder optimized.

Figure 44 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. This fork 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 six 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. In addition, 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.

Figure 45 shows the implemented 64-bit STFB prefix adder schematic and some input and output details. Notice that we opted to capture a “flat” schematic in order to simplify the visualization of the connections and the reset tree distribution. The last level of connections requires wires that are, at least, half of the adder long, and after place & routing resulted in wires as long as 800 μm. These long wires and complex STFB stages reduced the adder throughput when compared with a pipeline of just buffers close together. Simulation results indicate that a ring of STFB buffers can run above 2 GHz, while the 64-bit STFB prefix adder achieved 1.4 GHz under the same conditions. Appendix B has the complete schematics of our demonstration chip.

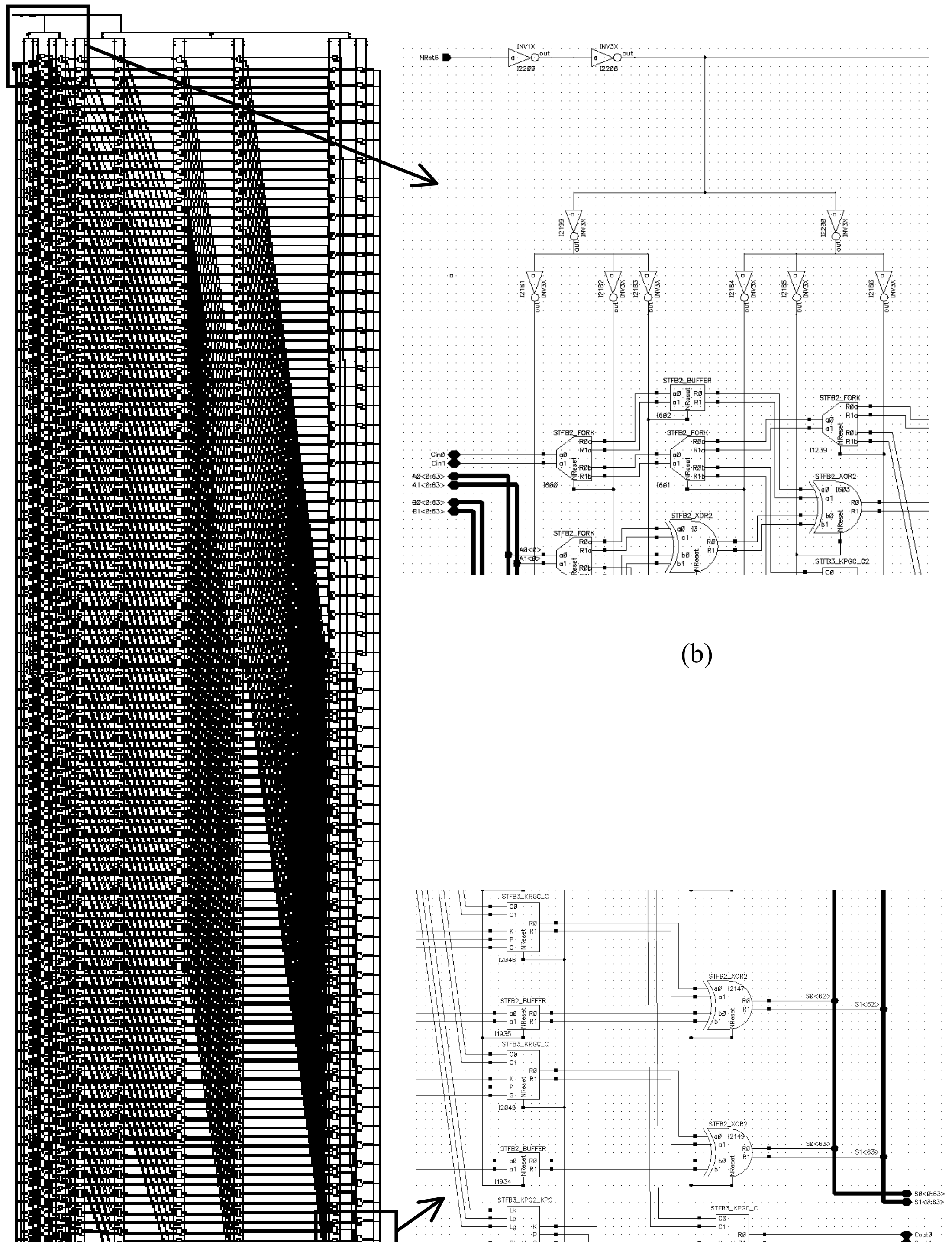


Figure 45. (a) 64-bit STFB Prefix Adder schematic, (b) input and (c) output details.

6.3 The input circuitry

The input circuitry loads and continuously repeats a test pattern to be fed into the adder. The INPUTGEN129BY9 block is composed of single-rail to single-track converters, split circuits and 129 9-stage rings (two 64-bit numbers and carry in).

Figure 46 shows the input generator block, where eight bits of data and four bits of address are converted from single-rail to single-track. The 4-bit address directs the 8-bit data to one out of 16 specific 9-stage ring groups that will be used to continuously generate the 64-bit A and B operands. This addressing operation is necessary due to pins limitations in the demonstration chip design. In addition, the carry-in pattern is converted from single-rail to single-track and loaded in its specific ring to supply C_{-1} to the adder. There are two load control lines (not shown), one for the 12-bit data-address set and another for the carry in signal.

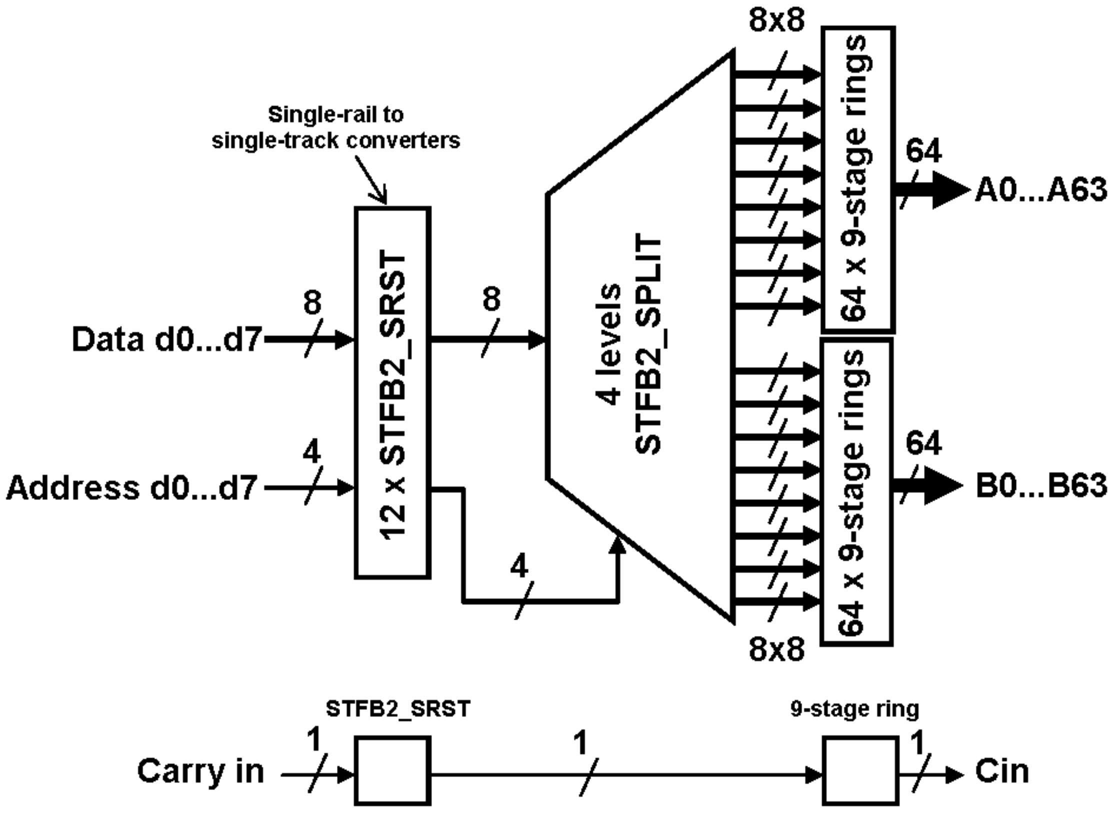


Figure 46. INPUTGEN129BY9 block diagram.

Figure 47 shows the 9-stage ring diagram, where we used seven buffers, one fork, one merge, one xor, and the controlled bit-generator (square with the letters BG). Although the rings support up to seven tokens each, the maximum throughput of the ring is achieved with 3 tokens.

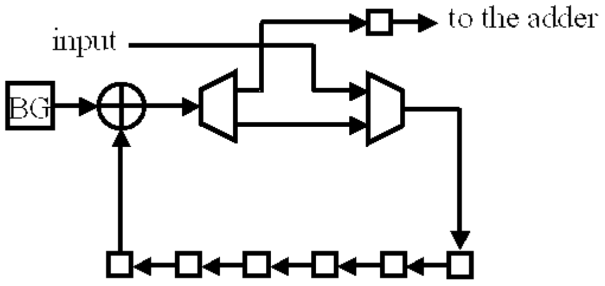


Figure 47. 9-stage ring utilized in the input circuitry.

After the tokens are loaded, the BG cell is enabled with the "GO" signal (not shown). 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. In this design we have three independent signals to control the inversion of A, B and C_{-1}.

6.4 The output circuitry

In order to test the adder running at full throughput, we implemented a programmable output circuitry that samples the 65-bit result (64-bit sum and one bit carry out), forwarding to the output pins one out of n results ($0 \leq n \leq 7840$). The SAMPLER65BY1000 circuit is implemented with three 30-stage rings each of them connected to a 65-bit split structure. The 30-stage rings are similar to the 9-stage ring in Figure 47, they simply have 27 buffers in the loop instead of just seven, and they can be individually loaded with a sequence of tokens.

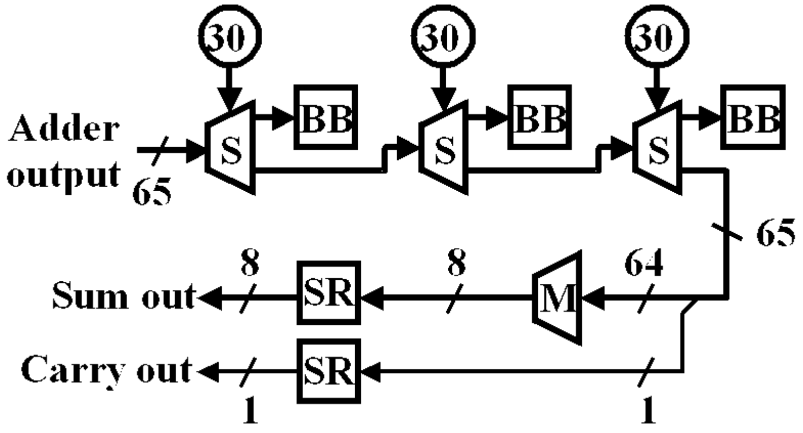


Figure 48. SAMPLER65BY1000, MUX 64 to 8 and single-rail converters block diagram.

Figure 48 illustrates the sampler circuit where the split stages (S), controlled by the 30-stage rings, direct the input token to a bit-bucket (BB), where the token is destroyed, or to the next split. The 65-bit output of the last split has the sampled result that is going to be send to the output pins. The carry out is separated converted to single-rail and sent to its exclusive pin. The 64-bit sum is sent to a MUX that routes to the output one byte at the time, starting for the most significant one (big-ending). Again, this routing procedure is necessary due to pins limitations in the demonstration chip.

The 30-stage rings can run at full throughput if we load them with 10 tokens each. This would also result in a sample rate of 1 out of 1000 results. For example, if we load all three rings with “1000000000”, we would sample the first result, the 1001st, the 2001st and so on. If we load the first ring with “0100000000” and the others with “1000000000”, we would sample the second result, the 1002nd, the 2002nd and so on. Therefore, with this sampler architecture, we can choose which results we want to see.

Moreover, we can change the sample rate by loading the rings with different number of tokes. We need to be careful, however, in order not to slow down the adder if we want to check its performance at full throughput. If the first ring is loaded with ten tokens, we can load the other two with 28 tokens, yielding a sampling rate up to one out of 7840 results without limiting the adder throughput.

Notice that, like the input circuitry in section 6.3, all the output circuit is implemented using STFB stages and, if the external test circuit is slow in consuming the output results, the input circuit and the adder will slow down to accommodate the consumer and no sampled data will be lost.

6.5 The chip layout

Figure 49 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 70%, 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.

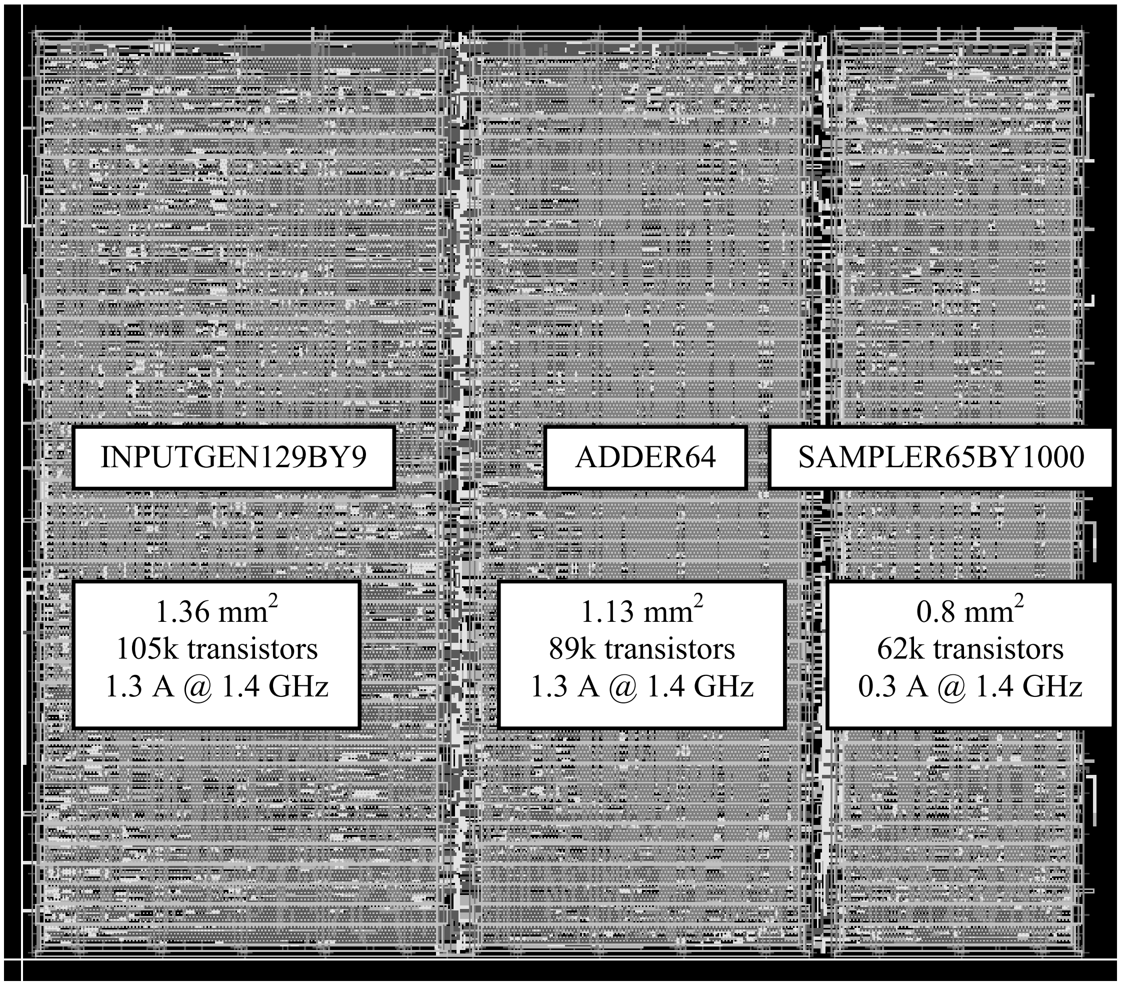


Figure 49. The input, adder and sampler block layout with respective areas, transistor counts and simulated current and throughput.

6.6 Power Distribution and EM

Figure 50 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 1.3A each (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 (0.3A average). 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).

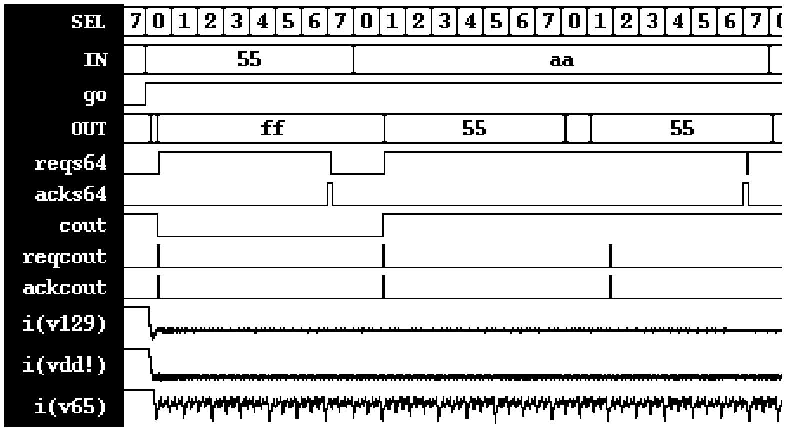


Figure 50. Typical simulation output.

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. Also, 14 pins where allocated to V_{DD} and 14 to GND, 7 pairs placed on each side of the three blocks.

6.7 Simulation results

Table 2 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 2. 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

6.8 Comparisons

Table 3 shows a comparison of some STFB pipeline stages with PCHB stages and static standard cell CMOS gates (referred as "static"). The latency and cycle time are written in terms of number of transitions. The static 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 3. STFB, PCHB and CMOS comparison.

Function	Cell	Latency	Cycle Time	Area (μm^2)	Area ratio
Buffer	STFB	2	6	415	4.5
	PCHB	2	14	726	7.9
	static	2	-	92	1
2-input AND/OR	STFB	2	6	472	4.6
	PCHB	2	14	968	9.3
	static	2	-	104	1
2-input XOR	STFB	2	6	472	2.6
	PCHB	2	14	1048	5.7
	static	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 static 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.

6.9 Demonstration chip implementation and test

Figure 51 shows the fabricated demonstration chip (ASYNC1b) layout, where the STFB blocks are placed on top under a power grid implemented with metal 5. Due to the expected high current, 14 pins of V_{DD} and 14 pins for GND where distributed on both sides of the design. The second part of the chip is a completely independent circuit implementation of the sequential decoder algorithm [35].

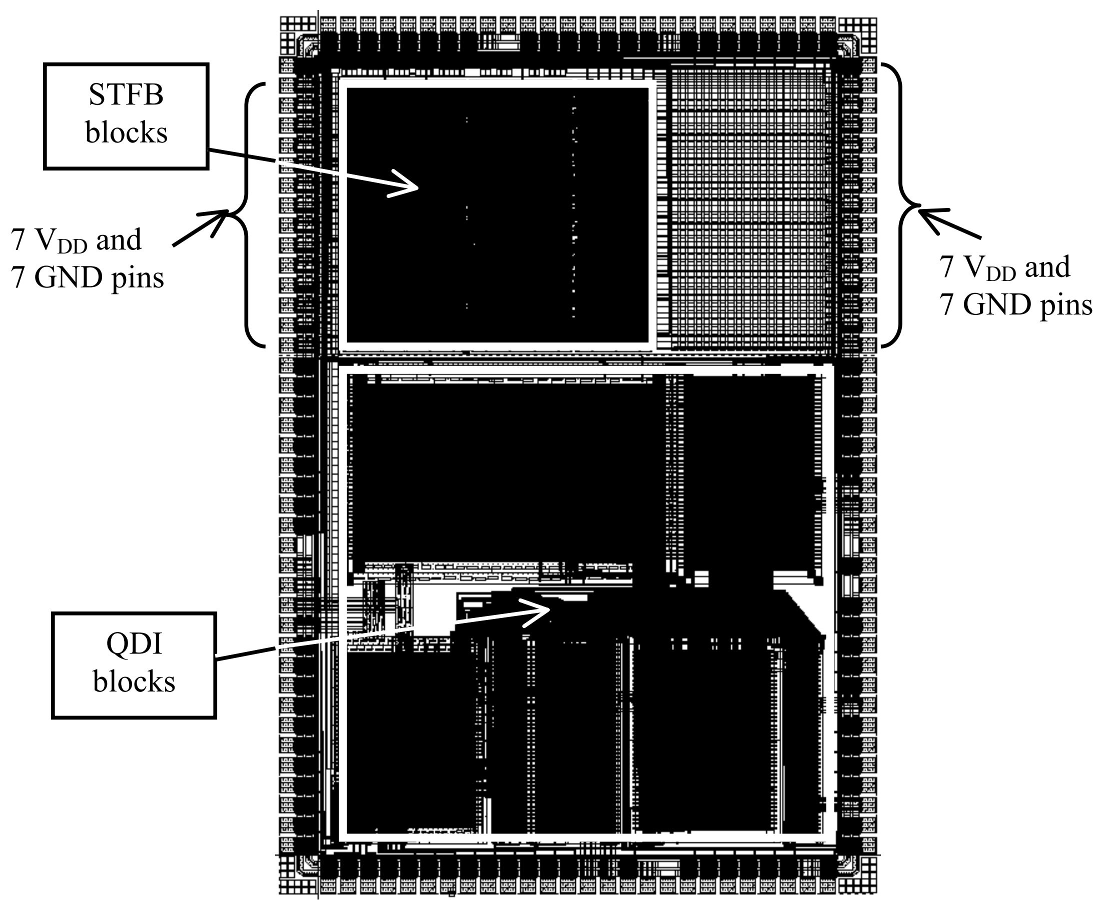


Figure 51. ASYNC1b layout has 20.5 mm^2 and 132 pins.

Figure 52 shows a picture of the implemented chip. Noticed that the STFB blocks are completely covered by the alternated metal 5 power lines. The package utilized is a ceramic 132 pin PGA (Pin Grid Array) where the STFB circuits are using 28 power supply pins (14 V_{DD} and 14 GND), 12 bi-directional pins, 13 input pins and 3 supply pins for the pads (3.3V, 2.5V and VSS). The total STFB pins are 56, and the remainders 72 are used by the QDI part of the ASINC1b chip.

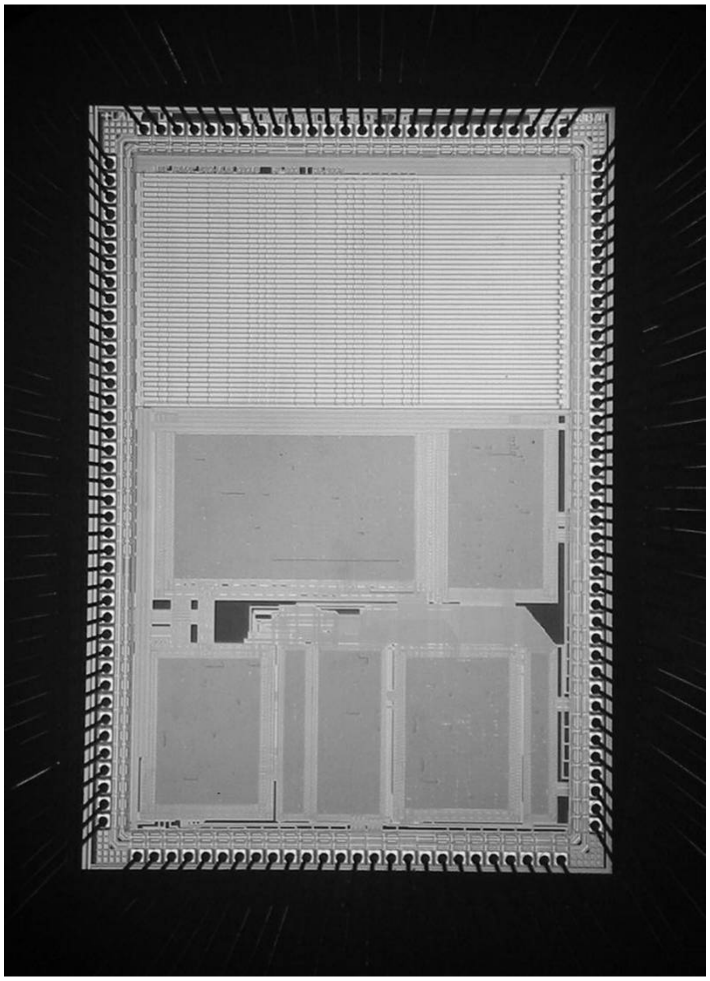


Figure 52. ASYNC1b demonstration chip (die photo).

The Figure 53 shows the demonstration chip on the evaluation board. The evaluation board disables the QDI part of the chip and it uses a FPGA (not shown) to setup and run the STFB part. The FPGA is a Xilinx XC2S100 Spartan-II on a Xess XSA prototyping board. The software utilized to program the FPGA are the Xilinx ISE version 6 and the Xess tools package. Once programmed, the FPGA loads the STFB input block with the operands, sets the sample rate in the output block and runs the ASYNC1b chip by acknowledging all the requests as they come out of the chip.

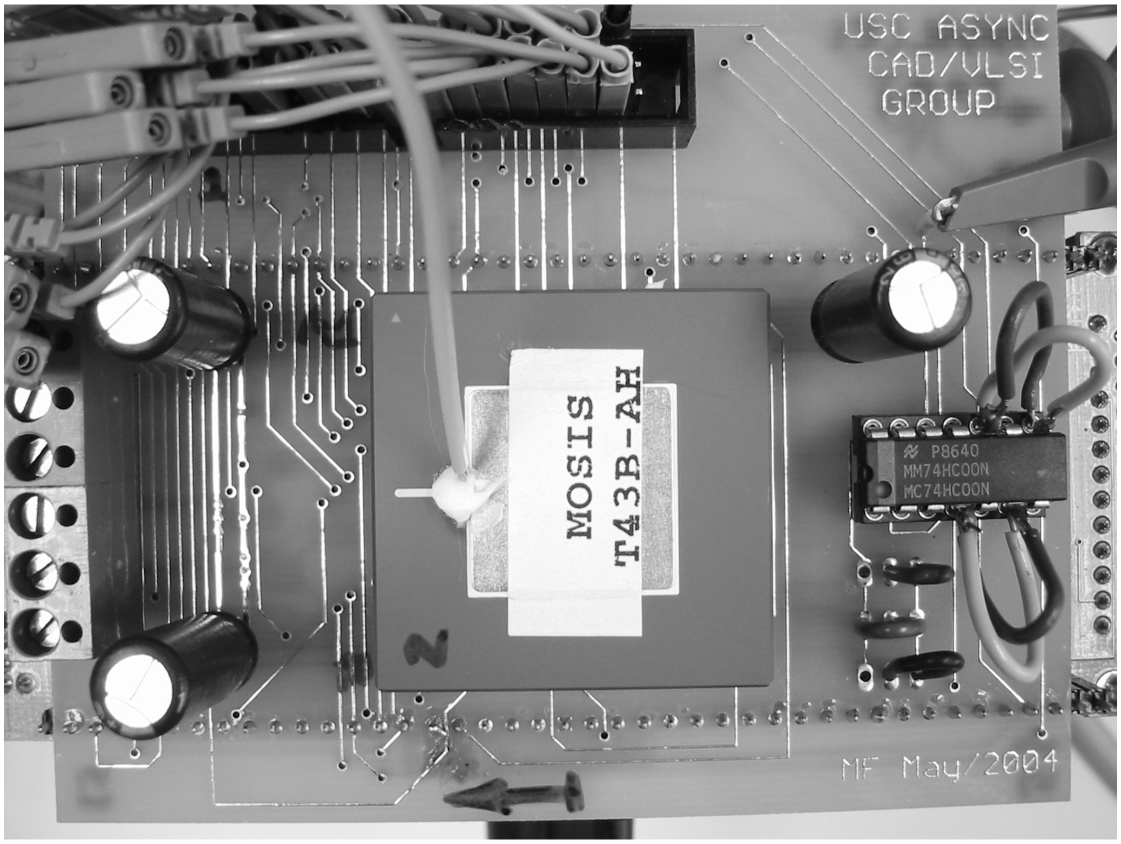


Figure 53. Demonstration chip on the test board.

An oscilloscope (Tektronix TD210) is used to check the byte and carry acknowledges as shown below. This allows an easy check of the chip throughput since one *carry out* is outputted at every sampled result. One multimeter is used to measure the temperature on top of the package while another displays the on-chip voltage. The current is measured by the power supply (Agilent E3610A). A 24-channel logic analyzer (Link Instruments LA-2124) is used to capture the waveforms, which allow checking the initialization and operation of the demonstration chip. The ceramic package thermal coefficient with no wind is 29^{o}C/W [38]. With a fan blowing air close to the chip, we estimated a thermal coefficient of ~20^{o}C/W. This means that the die temperature is ~20^{o}C higher than the air temperature if the power dissipation is one watt and there is a fan blowing air over the package. Since we have a power dissipation of more than 5W, the die temperature would be too high without the fan and the fan is used to keep the package and die temperature in manageable ranges.

Figure 54 shows the test setup with the fan. Notice that the temperature on top of the package is 40^{o}C (the room temperature was around 23^{o}C), the on-chip voltage at 2.5V and the V_{DD} current at 2.26A. The estimated die temperature is around 130^{o}C.

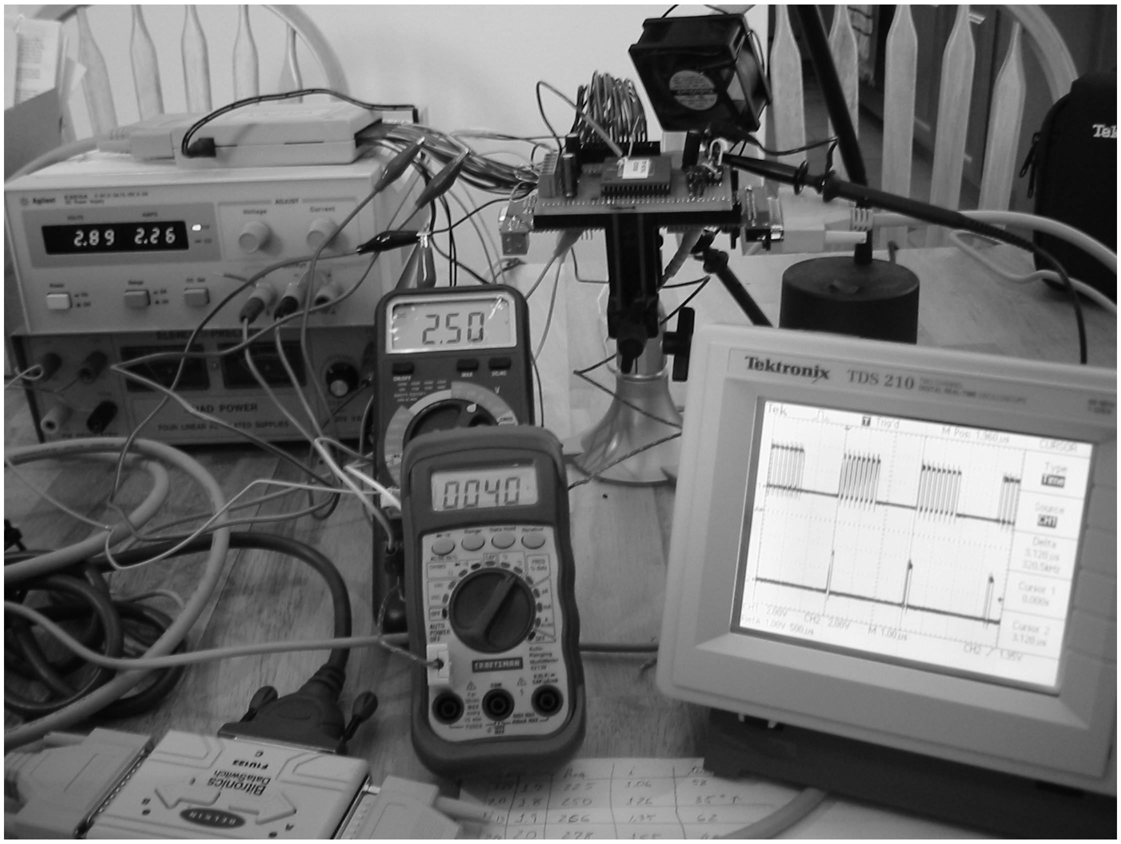


Figure 54. Test chip and equipment setup.

6.10 Test results

Figure 55 shows the measured waveforms of the chip number 3 (all 40 samples delivered by MOSIS were numbered sequentially for tracking purposes). Notice that the channel 1 shows the carry out acknowledge produced by the FPGA at every request from the test chip. The channel 1 frequency, 313 kHz, indicates that the 64-bit adder is running at 1.25 GHz since the sample rate was set to 1:4000. The channel 2 signal shows the acknowledge of the result which is outputted one byte at the time requesting eight consecutive acknowledges of 200ns each (5 MHz).

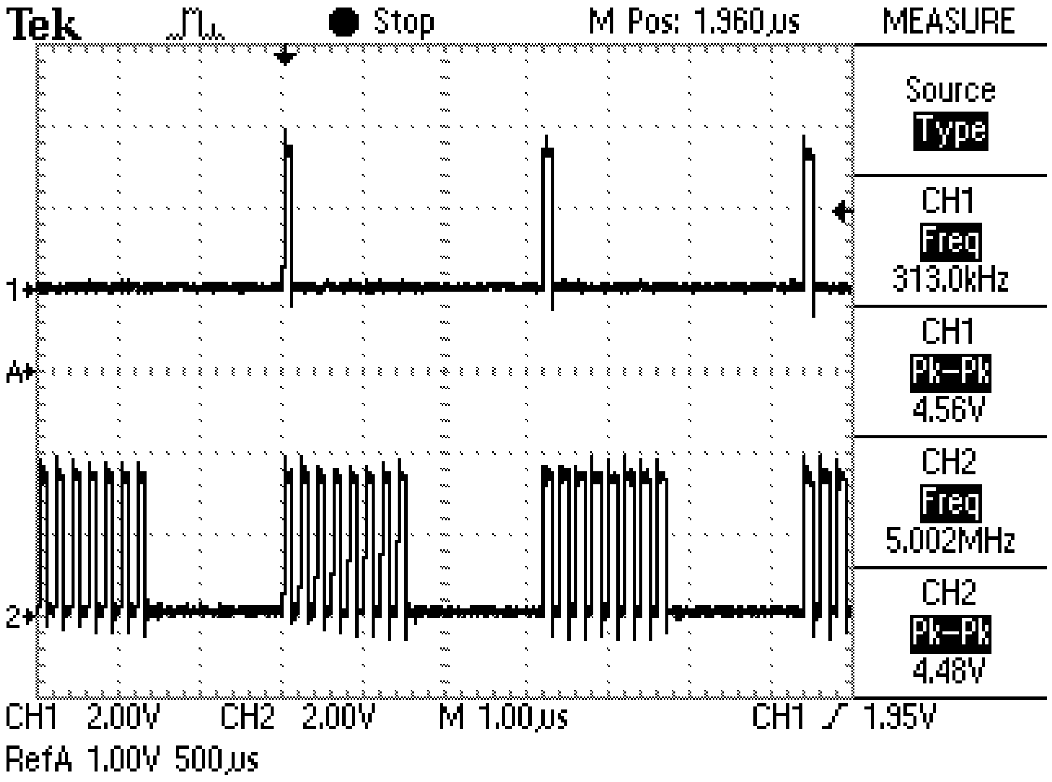


Figure 55. Chip#3 at 1.25GHz (2.5V on-chip, 2.26A, 40^{o}C package, fan at 1.5")

The sampler rings, as explained on Section 6.4, may be programmed with different number of tokens in order to allow a different sample rate, making possible to sample all the possible results. Figure 56 shows the loading phase of the chip after the rising edge of the reset (*NRst*) signal. Notice that by loading the Ring0 with 11 tokens and the Ring1 and Ring2 with 19 tokens we have a sample rate of 1:3971.

Also, since the input rings are much faster than the adder, we can load three carries, three 64-bit operands for A and four 64-bit operands for B, resulting in 12 combinations as shown on Table 4, without reducing the adder throughput.

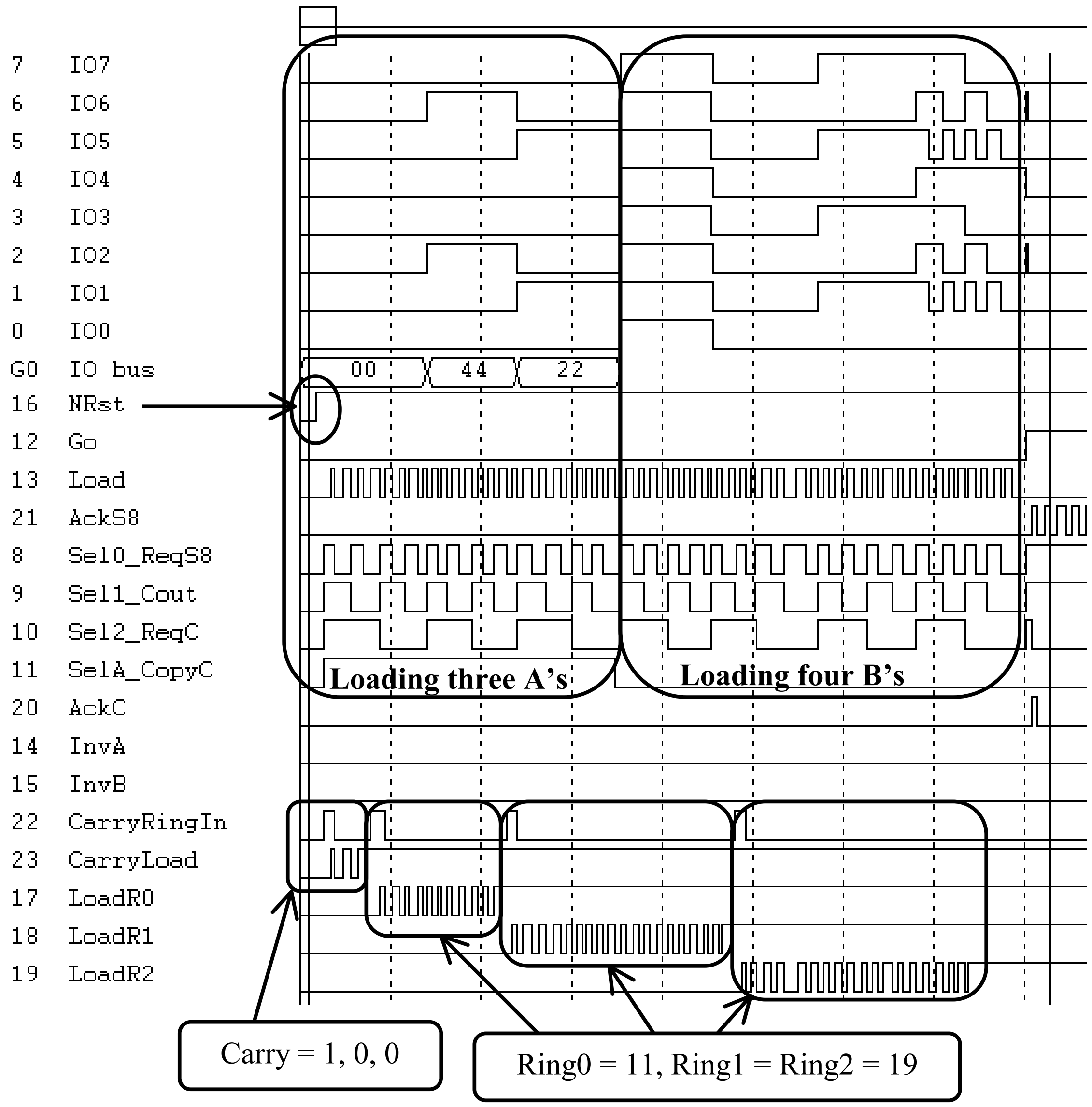


Figure 56. Logic Analyzer capture wave form of the loading sequence.

Figure 57 shows the operation of the demonstration chip. After the rising of the "Go" signal, the input rings start feeding the adder continuously while the output rings sample the results allowing the first result to go out, then the 3972nd , the 7943rd, and so on. Each 64-bit result is multiplexed to 8-bit output starting with the most significant byte, which allows us to easily check the correctness of the output using the logic analyzer.

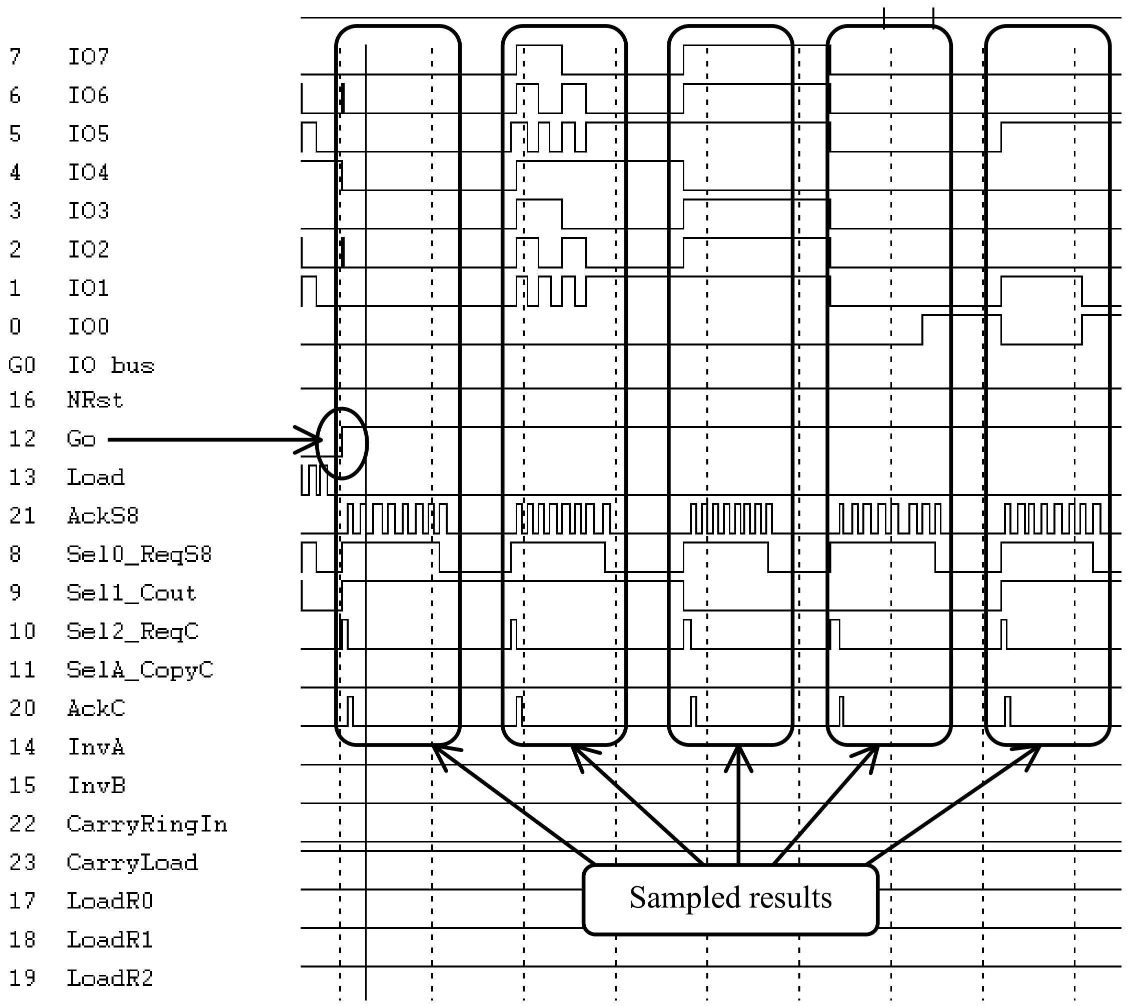


Figure 57. Logic Analyzer capture wave form of the running mode.

Table 4 shows the test case 042-F0AF, where there are 3 operands for A and Carry and 4 operands for B. Table 5 shows the sum and carry result sequence for this test case with a sample rate of 1:3971. The demonstration chip results values and sequence are correct as expected.

Table 4. Example of loaded operands used for test: sequence 042-F0AF.

loading	A	B	C
1	0000000000000000	FFFFFFFFFFFFFFFF	1
2	4444444444444444	0000000000000000	0
3	2222222222222222	AAAAAAAAAAAAAAAA	0
4	-	FEDCBA9876543210	-

Table 5. Sequence of output results from 042-F0AF test case (sample 1:3971).

Input	A	B	C	S	Cout	Sample
1	0000000000000000	FFFFFFFFFFFFFFFF	1	0000000000000000	1	1
2	4444444444444444	0000000000000000	0	4444444444444444	0	12
3	2222222222222222	AAAAAAAAAAAAAAAA	0	CCCCCCCCCCCCCCCC	0	11
4	0000000000000000	FEDCBA9876543210	1	FEDCBA9876543211	0	10
5	4444444444444444	FFFFFFFFFFFFFFFF	0	4444444444444443	1	9
6	2222222222222222	0000000000000000	0	2222222222222222	0	8
7	0000000000000000	AAAAAAAAAAAAAAAA	1	AAAAAAAAAAAAAAAB	0	7
8	4444444444444444	FEDCBA9876543210	0	4320FEDCBA987654	1	6
9	2222222222222222	FFFFFFFFFFFFFFFF	0	2222222222222221	1	5
10	0000000000000000	0000000000000000	1	0000000000000001	0	4
11	4444444444444444	AAAAAAAAAAAAAAAA	0	EEEEEEEEEEEEEEEE	0	3
12	2222222222222222	FEDCBA9876543210	0	20FEDCBA98765432	1	2

Table 6 shows some performance measurements with chip #3 for different supply voltages. Notice that the voltage drop from the power supply to the voltage inside the chip is significant due to the high level of current required. The "on-chip" voltage is measured by two supply pins (one V_{DD} and another GND, pins B01 and C03) that are connected to a voltmeter instead of the power supply. This means that the entire chip current is supplied through 13 pins of V_{DD} and 13 pins of GND, which represents about 170 mA per pin at full throughput (2.5V, 1.28 GHz). The "on-chip" voltage is a good estimative of the adder supply voltage, however, due to the high current levels, we estimate that the voltage on top of the adder to be around 0.1V below of the "on-chip" value.

Table 6. Measurements of chip #3 with fan at 1.5" distance.

Voltage Supply	Voltage On-chip	Current (A)	Output Freq (MHz)	Package temp (oC)	Power (W)	die temp(oC)	Eτ2 (10^{-27} Js^2)
1.88	1.7	1.11	952	29	1.9	61	2.19
2.00	1.8	1.28	1020	29	2.3	69	2.17
2.12	1.9	1.43	1084	30	2.7	77	2.13
2.25	2.0	1.58	1136	31	3.2	86	2.16
2.38	2.1	1.73	1188	33	3.6	96	2.17
2.51	2.2	1.87	1204	34	4.1	105	2.36
2.63	2.3	2.00	1248	35	4.6	115	2.37
2.76	2.4	2.14	1264	37	5.1	126	2.54
2.88	2.5	2.27	1280	38	5.7	137	2.71
3.02	2.6	2.42	1300	39	6.3	149	2.86

Figure 58 shows the measurements in a graphic format and compare the results with and without fan.

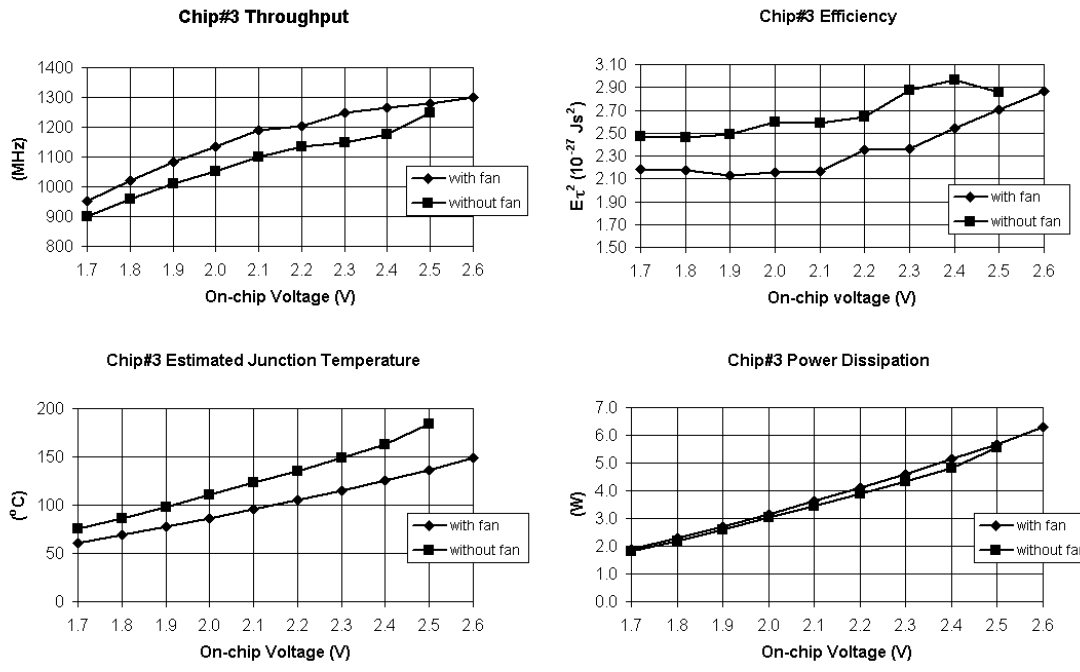
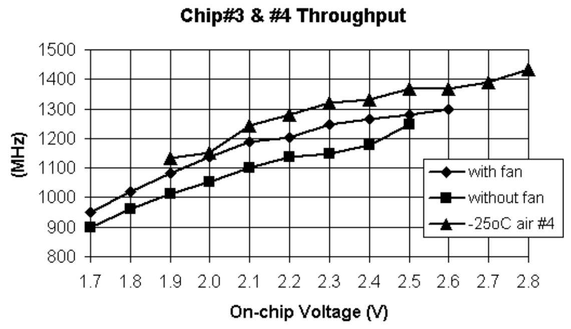


Figure 58. Graphics of chip #3 measurements.

The measurements of the chip operation without fan were performed without waiting for the temperature to stabilize since the package temperature was raising fast and some irreversible damage could have been made to the chip if more time were allowed. Notice that, the cooler operation yields higher throughput and is more efficient (lower $E\tau^2$) since the power dissipation is about the same. The junction temperature was estimated based on the ambient temperature and assuming the thermal coefficients of 20°C/W with fan and 29°C/W without the fan [38].

Comparing with simulation results, we can see that the performance is close but below to the TT-2.5V-25°C simulation case. However, for the real chip test, we have to consider that the die temperature is much higher and that the voltage on top of the adder is smaller than 2.5V due to the voltage drop on the real power grid. Taking these effects into account, the performance of the real design is as expected.

Thanks to Fulcrum Micro-Systems, we were able to further evaluate the temperature influence on the circuit performance. Fulcrum's precision forcing temperature system is a machine that blows air at controlled temperature over the device under test. We setup the air temperature to -25°C and we estimated the junction temperature to be between 0 to 10°C.



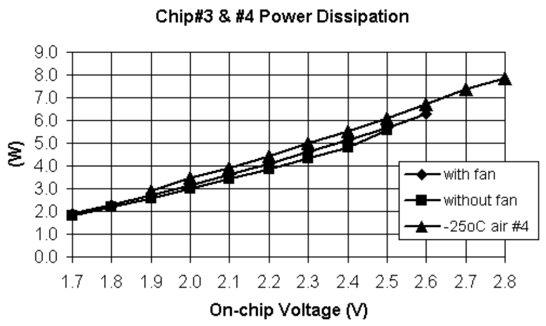


Figure 59. Chip #4 (under -25°C air flow) compared with chip #3 results.

Figure 59 shows the higher performance of the cooler chip, reaching 1.45 GHz. Since the die temperature is close to the simulated at typical condition, the performance also gets close. However, the voltage drop in the real power grid still remains.

The “on-chip” voltage range of our test was limited by the operation of the chip. Voltages above 2.6V and below 1.7V cause the chip to stop running when tested just with the fan. The reason for the upper limit is likely to be on-chip noise inducing or killing tokens, which causes a complete halt of the circuit. The lower limit is likely due to the assumption that the three transitions active phase will be enough to discharge the line (the charge operation has the staticizer and RCD feed-back to compensate, this is not the case for the SCD). The lower supply voltage would make the reset transistors weak and tokens would be left on the long channels clogging the pipeline and halting the circuit.

The overall performance of the chip is very good and its operation is stable. The tested samples were used continuously for several hours at full throughput without presenting wrong operations or detectable performance variation.

7 CONCLUSIONS

STFB templates are proposed for high-speed area-efficient asynchronous non-linear pipeline design. A freely available STFB standard cell library using TSMC 0.25 μm technology was generated and posted with MOSIS Educational Program. A complete STFB design with 260,000 transistors is successfully implemented and tested reaching 1.45 GHz.

The STFB templates use 1-of-N data encoding single-rail hand-shaking to avoid timing assumptions based on bundling constraints that are often hard to analyze, to guarantee during design, and to verify after layout. The templates have higher throughput than the fastest known QDI templates and have lower latency than the most aggressive GasP templates. Consequently, for systems that are latency-critical, STFB templates may yield a significant performance advantage.

Implementation issues and performance analysis methodology are presented. The timing constraints and noise margin are discussed, and the performance of the STFB templates is compared with QDI templates. The small cycle time of the STFB templates is thoroughly analyzed. This small cycle time allows the STFB circuits to operate at very high throughputs with small distances between consecutive data tokens, resulting in smaller and faster circuits than their QDI alternatives. The energy per operation is also advantageous as demonstrated by a comparison of $E\tau^2$ metrics.

The demonstration design includes the input generating circuit, a 64-bit prefix adder and a programmable position and rate output sampler circuit. All these circuits were implemented using our STFB standard cell library in a conventional back-end flow, which resulted in a simple, fast and efficient design process that can be easily understood by synchronous designers.

The demonstration design chip exploits the advantages of the small STFB cycle time. The input circuit uses 129 9-stage rings, which are examples of high speed loops processing multiple data tokens. The 64-bit prefix adder represents a high-complexity design with large STFB stages operating with dual rail and 1-of-3 channels. The sampler circuit uses multiple rings running at different rates. Also, all the support logic to load the operands and unload the results is implemented with STFB stages.

As continuation of this work, changes can be made in the template in order to improve the noise margins. In particular, if a smaller feature size process is targeted, different transistor sizing and/or the use of the Static Single-Track SST protocol should be explored. The 10-transitions STFB template may also be used to improve reliability over process variations due to its self reset characteristics and time margins.

REFERENCES

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital System Testing & Testable Design*. Wiley-IEEE Press, 1993.

[2] W. J. Bainbridge and S. B. Furber, "Delay Insensitive System-on-Chip Interconnect using 1-of-4 Encoding", 7th International Symposium on Asynchronous Circuits and System, pp: 118 – 126, Salt Lake City, Utah, USA 2001.

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

[4] K. Bernstein, K. M. Carrig, C. M. Durham, P. R. Hansen, D. Hogenmiller, E. J. Nowak and N. J. Rohrer, *High Speed CMOS Design Styles*, Kluwer Academic Publishers, Norwell, Massachusetts, USA 1998.

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

[6] E. Brunvand, "Translating Concurrent Communicating Programs into Asynchronous Circuits", PhD Thesis Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA 1991.

[7] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of Latency-Insensitive Design", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 20, no. 9, pp.1059-1076, September 2001.

[8] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Coping with Latency on SoC Design", IEEE Micro Magazine, pp.24-35, September-October 2002.

[9] L. P. Carloni, K. L. McMillan, A. Saldanha and A. L. Sangiovanni-Vincentelli, "A Methodology for Correct-by-Construction Latency Insensitive Design", Proceedings of the International Conference on Computer-Aided Design, 1999.

[10] M. D. Ciletti, *Modeling, Synthesis and Rapid Prototyping with the Verilog HDL*, Prentice-Hall, Upper Saddle River, New Jersey, USA 1999.

[11] U. V. Cummings, A. M. Lines and A. J. Martin, "An Asynchronous Pipelined Lattice Structure Filter." Proc. of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp.126-133, November 1994.

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

[13] A. Davis and S. M. Nowick, "An Introduction to Asynchronous Circuit Design", Technical Report UUCS-97-013, University of Utah, Salt Lake City, Utah, USA Sept. 19, 1997.

[14] D. Duarte, V. Narayanan and M. J. Irwin. "Impact of technology scaling in the clock system power," Proceedings for IEEE Computer Society Annual Symposium on VLSI – ISVLSI, pp. 59-64, 2002.

[15] M. Ferretti and P. A. Beerel, "Low-Swing Signaling Using Dynamic Diode Connected Driver", 27th European Solid-State Circuits Conference ESSCIRC, Villach, Austria, September 2001.

[16] M. Ferretti and P. A. Beerel, "Low-Swing Signaling Using Dynamic Diode Connected Driver", Submitted to IEEE Transactions on VLSI System.

[17] M. Ferretti and P. A. Beerel, "Asynchronous 1-of-n Logic Using Single-Track Protocol", CENG Technical Report No. 01-03, Department of Electrical Engineering – Systems, University of Southern California, Los Angeles, California, USA, July 12th 2001.

[18] M. Ferretti and P. A. Beerel, "Single-Track Asynchronous Pipeline Templates Using 1-of-N Encoding", Design Automation & Test in Europe Conference DATE, Paris, France, March 2002.

[19] M. Ferretti, R. O. Ozdag and P. A. Beerel, "High Performance Asynchronous ASIC Back-End Design Flow Using Single-Track Full-Buffer Standard Cells", 10th Symposium on Asynchronous Circuits ASYNC, Herssonissos, Crete, Greece, April 2004.

[20] H. Gageldonk, D. Baumann, K. Berkel, D. Gloor, A. Peeters and G. Stegmann, "An Asynchronous low-power 80c51 microcontroller", Proceedings International Symposium on Advanced Research on Asynchronous Circuits and System, pp: 96 – 107, 1998.

[21] S. Ghosh, Hardware Description Languages, IEEE Press Series on Microelectronics Systems, Piscataway, New Jersey, USA 2000.

[22] 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.

[23] 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.

[24] S. Hauck, “Asynchronous Design Methodologies: An Overview”, Proceedings of the IEEE, Vol. 83, No. 1, pp. 69-93, January 1995.

[25] I. Koren, *Computer Arithmetic Algorithms*. A. K. Peters Ltd., 2001.

[26] A. M. Lines, “Pipelined Asynchronous Circuits”, Master Thesis, California Institute of Technology, June 1998.

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

[28] 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.” Proceedings of ARVLSI, pp. 164-181, 1997.

[29] A. J. Martin, M. Nyström and C. G. Wong, “A 100-MIPS GaAs Asynchronous Microprocessor.” IEEE Design & Test of Computers, Volume: 11, Issue: 2, pp. 43-49, 1994.

[30] A. J. Martin. Synthesis of asynchronous VLSI circuits. In J. Straunstrup, editor, *Formal Methods for VLSI Desi*gn, chapter 6, pp. 237–283. North-Holland, 1990.

[31] C. J. Myers, *Asynchronous Circuit Design*, John Wiley and Sons, July 2001.

[32] C. D. Nielsen. “Evaluation of Function Blocks for Asynchronous Design,” Proceedings of ACM, pp:454-459, September 1994.

[33] L. S. Nielsen and J. Sparso, “Designing Asynchronous Circuits for Low Power: An IFIR Filter Bank for Digital Hearing Aid,” Proceedings of the IEEE, vol. 87, no. 2, pp. 268-281, February 1999.

[34] M. Nyström, “Asynchronous Pulse Logic”, PhD Thesis Dissertation, California Institute of Technology, May 14, 2001.

[35] R. O. Ozdag and P. A. Beerel, “A Channel Based Asynchronous Low Power High Performance Standard-Cell Based Sequential Decoder Implemented with QDI Templates”, 10th Symposium on Asynchronous Circuits ASYNC, Herssonissos, Crete, Greece, April 2004.

[36] J. Pangjun and S.S. Sapatnekar. “Low-Power Clock Distribution Using Multiple Voltages and Reduced Swings,” IEEE Transaction on VLSI Systems, Vol. 10, No. 3, pp:309-318, June 2002.

[37] A. Peeters and K. Berkel, “Synchronous Handshake Circuits”, 7th International Symposium on Asynchronous Circuits and System, pp: 86 – 95, Salt Lake City, Utah, USA 2001.

[38] PGA132L Package Handbook supplied by MOSIS (pkg-pga132l-char.pdf), May/14/1993

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

[40] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev. "RAPPID: An asynchronous instruction length decoder." Proceedings for the International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp. 60–70, April 1999.

[41] C. L. Seitz. "System timing," in C. A. Mead and L.A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

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

[43] M. Singh and S. M. Nowick, "MOUSETRAP: Ultra-High-Speed Transition-Signaling Asynchronous Pipelines." ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU-2000), Austin, TX, December 2000.

[44] M. Singh and S. M. Nowick, "MOUSETRAP: Ultra-High-Speed Transition-Signaling Asynchronous Pipelines." Proceedings of the IEEE International Conference on Computer Design (ICCD-01), Austin, TX, September 2001.

[45] K. Soumyanath, S. Borkar, C. Zhou, B. Bloechel, "Accurate On-Chip Interconnect Evaluation: A Time Domain Technique", Symposium on VLSI Circuits Digest of Technical Papers, pp. 116-117, 1998.

[46] I. Sutherland, "Micropipelines", Communications of the ACM, vol. 32, #6, pp: 720-738, June 1989.

[47] I. Sutherland and S. Fairbanks, "GasP: A Minimal FIFO Control", Proceedings of 7th International Symposium on Asynchronous Circuits and System, pp: 46 – 53, Salt Lake City, Utah, USA 2001.

[48] I. Sutherland, B. Sproull and D. Harris, *Logical Effort*, Morgan Kaufmann Publishers, Inc., San Francisco, USA 1999.

[49] J. Teifel, D. Fang, D. Biermann, C. Kelly, R. Manohar, "Energy-Efficient Pipelines", 8th International Symposium on Asynchronous Circuits and System, Manchester, UK, April 2002.

[50] TSMC 0.25μm Logic 1P5M Salicide 2.5V, 2.5/3.3V Spice Models, Document No. TA-1099-6001 (T-025-LO-SP-005) Revision 2.2, TSMC Taiwan Semiconductor Manufacturing Co., Ltd., March 2001.

[51] TSMC 0.25μm Logic 1P5M Salicide 2.5/3.3V Design Rule, Document No. TA-1099-4003 (T-025-LO-DR-001) Revision 2.2, TSMC Taiwan Semiconductor Manufacturing Co., Ltd., October 2000.

[52] V. I. Varshavsky (editor), *Self-Timed Control of Concurrent Processes : The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, January 1990.

[53] T. E. Williams, "Self-Timed Rings and Their Application to Division", Technical Report No. CSL-TR-91-482, Department of Electrical Engineering and Computer Science, Stanford University, Stanford, California, USA 1991.

[54] T. E. Williams, "Performance of Iterative Computation in Self-Timed Rings", Journal of VLSI Signal Processing, vol. 7, pp. 17-31, 1994.

[55] K. Y. Yun and D. L. Dill, "Automatic Synthesis of Extended Burst-Mode Circuits: Part I (Specification and Hazard-Free Implementations)", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 2, pp. 101-117, Feb. 1999.

[56] K. Y. Yun and D. L. Dill, "Automatic Synthesis of Extended Burst-Mode Circuits: Part II (Automatic Synthesis)", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 2, pp. 118-132, Feb. 1999.

[57] 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 Systems, vol. 6, no.4, pp. 643–655, December 1998.

[58] H. Zhang, V. George and J. M. Rabaey, "Low-Swing On-Chip Signaling Techniques: Effectiveness and Robustness", IEEE Transactions on VLSI System., vol. 8.3, pp. 264-272, June 2000.

APPENDIX A: STFB STANDARD CELL LIBRARY

This appendix is a copy of the freely available STFB standard cell library documentation. It was re-formatted to fit inside the dissertation margins and the layout pictures were removed due to non-disclosure issues.

You can find more information about the USC Asynchronous libraries at:

http://jungfrau.usc.edu/AsyncLib.htm

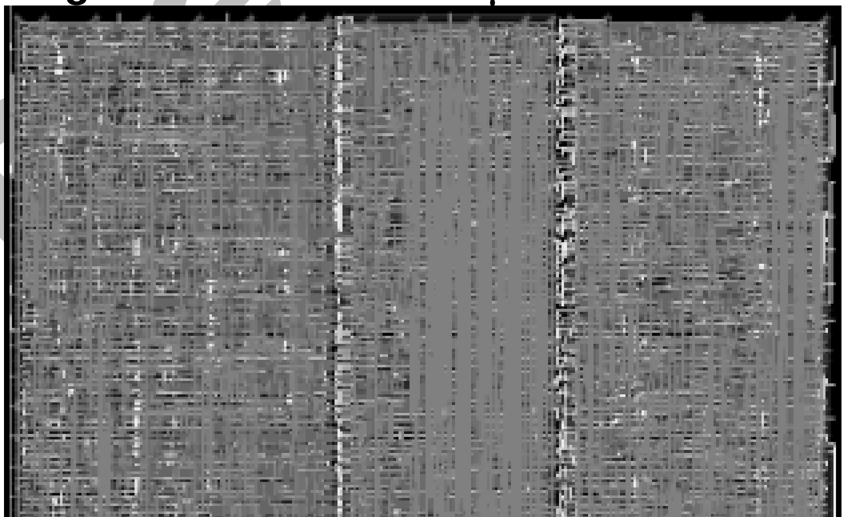
University of Southern California

Department of Electrical Engineering Systems

Asynchronous CAD/VLSI Group

Asynchronous CMOS Single-Track Full-Buffer Standard Cell Library

Designed for: TSMC 0.25 μm CMOS Process



TERMS AND CONDITIONS

TRADEMARKS

Contents

Asynchronous CMOS Single-Track Full-Buffer Standard Cell Library

Designed for: TSMC 0.25 μm Process

REVISIONS

Revision	Date	From	Description
0.1	27/Oct/03	M. Ferretti	Original
0.2	06/Nov/03	M. Ferretti	Added introduction
0.3	24/Jun/04	M. Ferretti, S. Awasthi, P. Pankaj	Added sub-cell STFB_POUTMERGE and cells STFB2_MERGE, STFB2_MERGENC and STFB2_BITGENSINGLE.

INTRODUCTION

This is the companion documentation of the Single-Track Full-Buffer library presented in [2] and [4], designed for the TSMC 0.25 µm process, and made available through [5].

Standard Cell Specifications

Standard-cell specifications are the physical constraints utilized during the custom layout of the cells. 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 (P&R) feasible.

Routing Grid

The routing grids are the positions available for the P&R tool to place the routing wires and connections to the cell's IO pins. The pins specifications need 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.

Also, to increase the number of routing positions available over the cell, the routing grid is offset with respect to the cell border by half of a grid space. Since the width and height of the cell are, respectively, multiples of the horizontal and vertical grid spaces, there are half-grid offsets on all sides of the cell as shown in Figure 1.

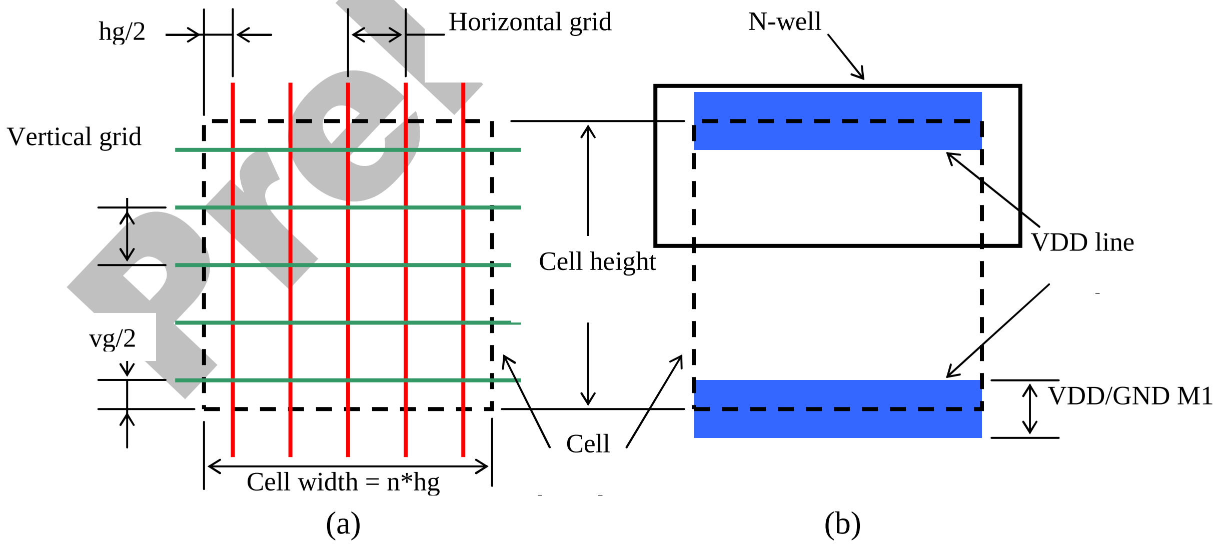


Figure 1 - Diagram of the utilized cell grid (a), cell height, N-well and power lines dimensions (b).

Cell dimensions, Power lines and N-well

The cell height and width are multiples of the horizontal and vertical grids respectively. The height of our cells is 12.8 µm, which corresponds to 16 horizontal grid steps. The width of our cells varies from 0.9 to 78.3 µm (1 to 87 vertical grid steps).

Power lines (VDD and GND) don't need to be in the grid since they are connected by abutting the cells and by a plan-power phase. Also, the N-well needs to be continuous throughout the cell, even when it is not been used, to avoid DRC errors after placement.

Using the Cell Library

The cells in this library where designed to be used with Cadence™ tools. The P&R tool utilized is Silicon Ensemble™.

Library Sections

The sub-cells, presented in Section 0, are used as building blocks of the more complex STFB cells shown in Section 0. The sub-cells cannot be directly utilized for P&R. However, they save time and reduce errors when designing the bigger STFB cells.

The support cells, presented in Section 0, can be used as regular cells in the P&R since they satisfy the standard cells specifications. They perform basic logic functions, inverters, nand and nor gates, and there are two special cells: FILL and FILLCAP3.

Placement

Usually, to allow a good placement, the row utilization factor is set to be between 60 to 85% during the floor-planning step. This means that, after the placement of the cells required in the application circuitry, there will be some empty space in each row. By utilizing the FILLCAP3 as the first filler cell, we can add bypass capacitance throughout the circuitry (approximately 55 fF per filler cell). Then, we can utilize the FILL cell, as the second filler cell, to close all the remaining gaps and avoid any DRC errors related to power lines and N-well continuity.

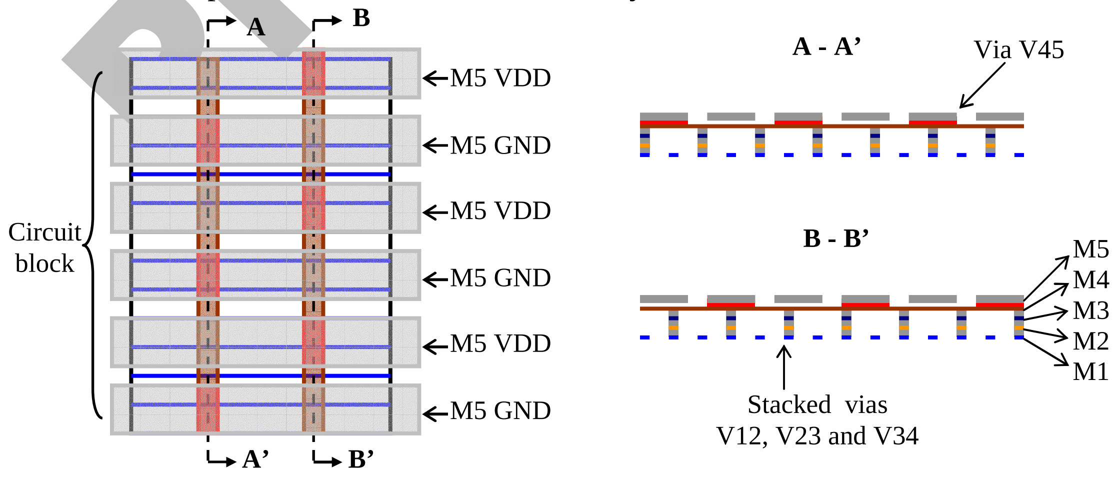


Figure 2 - Example of M5 and M4 stripes used for power distribution.

Power-planning

Metal 5 is usually the thickest metal layer and it is mainly utilized for power distribution. During the power-planning phase, besides a conventional power ring around the circuit block, spaced vertical M4 stripes can be utilized to connect the VDD and GND lines while allowing enough space for M4 vertical routing wires. Then, on top of the entire circuit, wide horizontal M5 stripes can be placed connecting VDD and GND to the M4 stripes as shown in Figure 2. This is an efficient way to distribute the power minimizing voltage drop (IR-drop) and Electro-Migration effects.

Routing

The preference routing directions are horizontal, for the metal layers M1, M3 and M5, and vertical, for M2 and M4. Since the STFB cells, shown in Section 0, are complex cells, M2 was utilized for horizontal connections inside the cell. This was a compromise solution in order to keep M3 and M4 free for routing while using M1 and M2 inside the cells. Notice that, M2 utilized inside the cell is placed in the horizontal grid allowing an easy way to define most of the cell's pins.

REFERENCES

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

[2] 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.

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

[4] M. Ferretti and P. A. Beerel, "High Performance Asynchronous ASIC Back-End Design Flow Using Single-Track Full-Buffer Standard Cell", Submitted to ASYNC'2004.

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

SUB-CELLS

Sub-cells were utilized as building blocks for the remaining cells in this library. STFB cells are usually designed as a group of sub-cells and some specific circuitry.

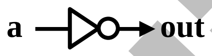
INV_14_06

Description

Name:	INV_14_06
Function:	Minimum size inverter sub-cell.
Dimensions:	12.8 x 1.98 μm
Logic equation:	out = $\overline{a}$
Truth table:	

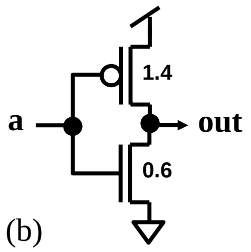
a	out
0	1
1	0

Schematic (b) and symbol (a)



(a)

Widths in μm and all lengths 0.24 μm



(b)

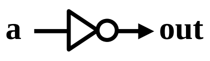
INV_28_12

Description

Name:	INV_28_12
Function:	Twice minimum size inverter sub-cell.
Dimensions:	12.8 x 1.98 μm
Logic equation:	out = $\bar{a}$
Truth table:	

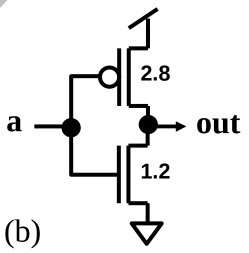
a	out
0	1
1	0

Schematic (b) and symbol (a)



(a)

Widths in μm and all lengths 0.24 μm



(b)

NAND2B_28_12

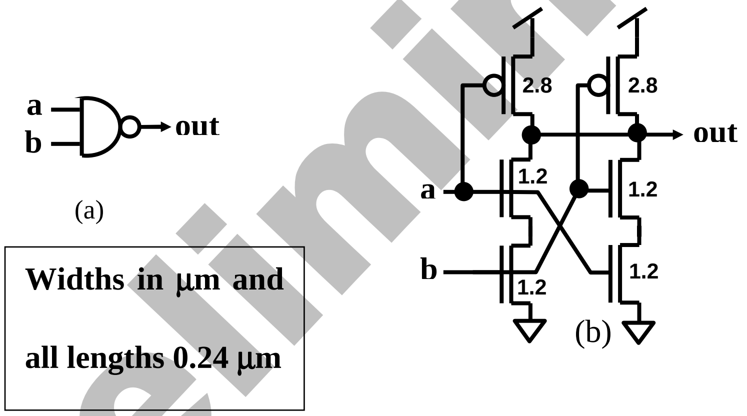
Description

Name:	NAND2B_28_12
Function:	Symmetrized 2-input twice minimum size NAND gate.
Dimensions:	12.8 x 4.56 µm
Logic equation:	out = $\overline{a \cdot b}$

Truth table:

a	b	out
0	x	1
x	0	1
1	1	0

Schematic (b) and symbol (a)



NAND2B_56_24

Description

Name: NAND2B_56_24

Function: Symmetrized 2-input 4 times minimum size NAND gate.

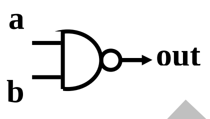
Dimensions: 12.8 x 4.56 μm

Logic equation: $\text{out} = \overline{a \cdot b}$

Truth table:

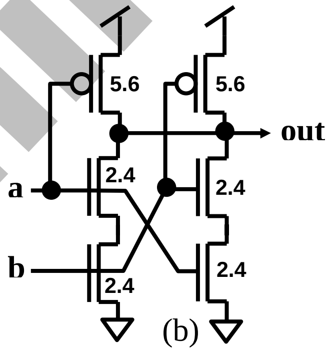
a	b	out
0	x	1
x	0	1
1	1	0

Schematic (b) and symbol (a)



(a)

Widths in μm and all lengths 0.24 μm



(b)

NAND3_28_12

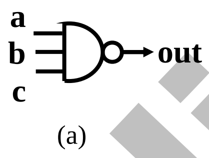
Description

Name:	NAND3_28_12
Function:	3-input 2 times minimum size NAND gate.
Dimensions:	12.8 x 3.93 µm
Logic equation:	out = $\overline{a \cdot b \cdot c}$

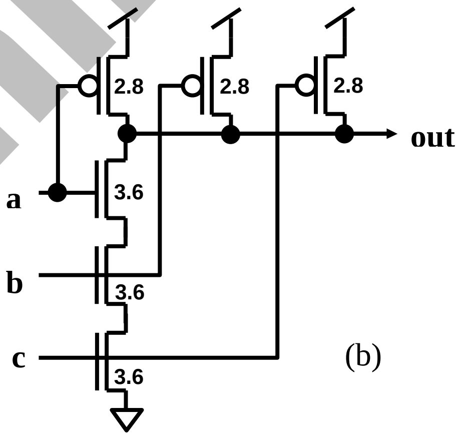
Truth table:

a	b	c	out
0	x	x	1
x	0	x	1
x	x	0	1
1	1	1	0

Schematic (b) and symbol (a)



Widths in µm and all lengths 0.24 µm



NOR2B_14_12

Description

Name: NOR2B_14_12

Function: Symmetrized 2-input minimum size NOR gate.

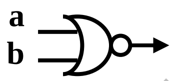
Dimensions: 12.8 x 4.24 μm

Logic equation: $out = \overline{a + b}$

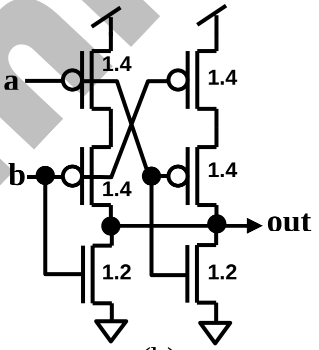
Truth table:

a	b	out
1	x	0
x	1	0
0	0	1

Schematic (b) and symbol (a)



(a)



(b)

Widths in μm and all lengths 0.24 μm

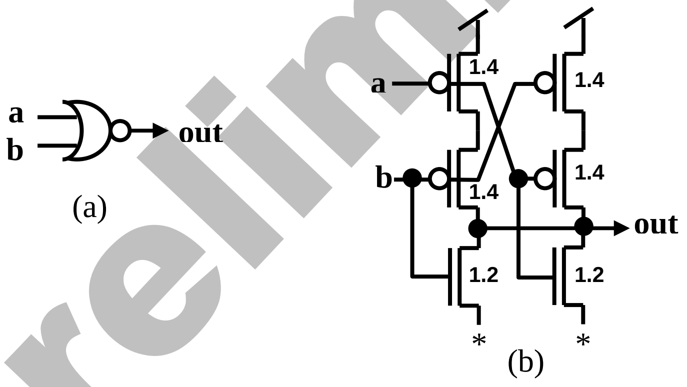
NOR2B_14_12FORK

Description

Name:	NOR2B_14_12FORK
Function:	Symmetrized 2-input minimum size NOR gate, prepared to be used in a dual output channel cell.
Dimensions:	12.8 x 4.24 μm
Logic equation:	$\text{out} = \overline{a + b}$
Truth table:	

a	b	out
1	x	0
x	1	0
0	0	1

Schematic (b) and symbol (a)



* To be connected to GND when this sub-cell is used.

Widths in μm and all lengths 0.24 μm

NOR2B_14_12OD

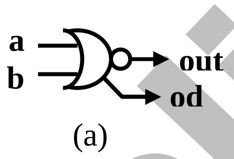
Description

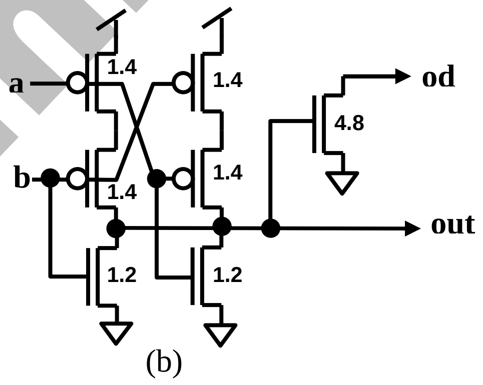
Name:	NOR2B_14_12
Function:	Symmetrized 2-input minimum size NOR gate with open-drain output.
Dimensions:	12.8 x 4.24 μm
Logic equation:	$\text{out} = \overline{a + b}$

Truth table:

a	b	out	od
1	x	0	z
x	1	0	z
0	0	1	0

Schematic (b) and symbol (a)





Widths in μm and all lengths 0.24 μm

NOR3B_14_12

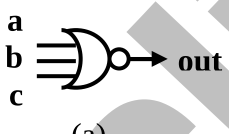
Description

Name:	NOR3B_14_12
Function:	Symmetrized 3-input minimum size NOR gate.
Dimensions:	12.8 x 6.3 μm
Logic equation:	$\text{out} = \overline{a + b + c}$

Truth table:

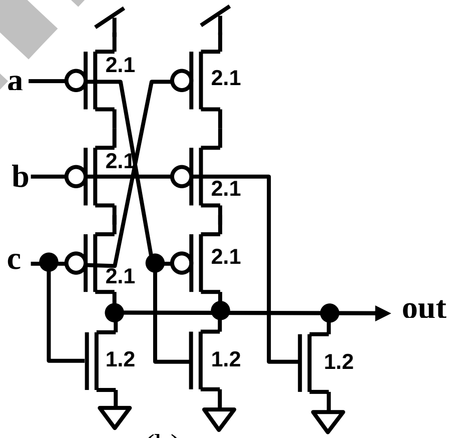
a	b	c	out
1	x	x	0
x	1	x	0
x	x	1	0
0	0	0	1

Schematic (b) and symbol (a)



(a)

Widths in μm and all lengths 0.24 μm



(b)

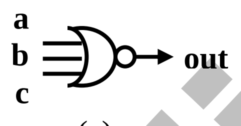
NOR3B_14_12FORK

Description

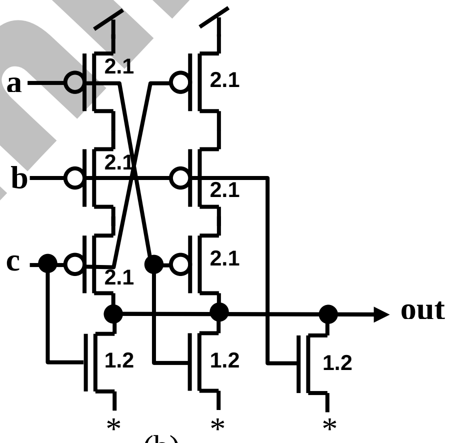
Name: NOR3B_14_12FORK
Function: Symmetrized 3-input minimum size NOR gate.
Dimensions: 12.8 x 5.52 μm
Logic equation: $\text{out} = \overline{a + b + c}$
Truth table:

a	b	c	out
1	x	x	0
x	1	x	0
x	x	1	0
0	0	0	1

Schematic (b) and symbol (a)



(a)



(b)

Widths in μm and all lengths 0.24 μm

* To be connected to GND when the sub-cell is used.

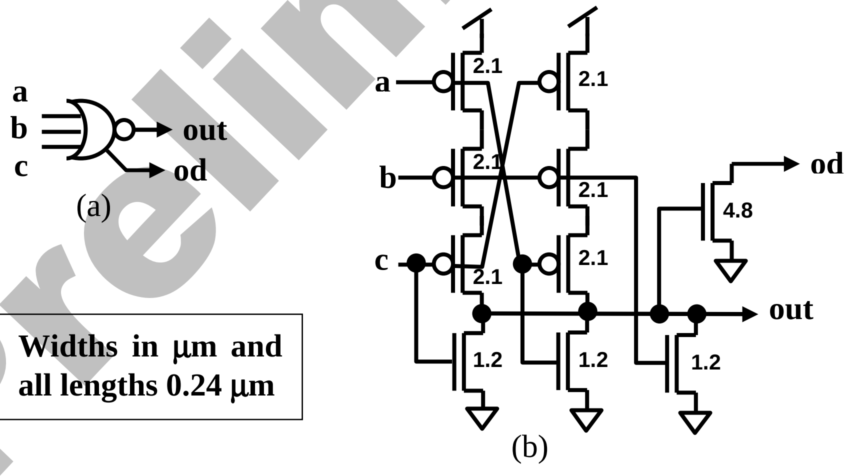
NOR3B_14_12OD

Description

Name:	NOR3B_14_12OD
Function:	Symmetrized 3-input minimum size NOR gate with open drain output.
Dimensions:	12.8 x 5.52 μm
Logic equation:	$\text{out} = \overline{a + b + c}$
Truth table:	

a	b	c	out	od
1	x	x	0	z
x	1	x	0	z
x	x	1	0	z
0	0	0	1	0

Schematic (b) and symbol (a)



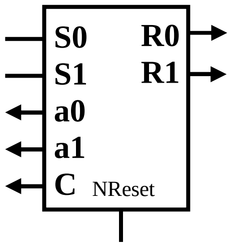
Widths in μm and all lengths 0.24 μm

STFB2_CORE2I

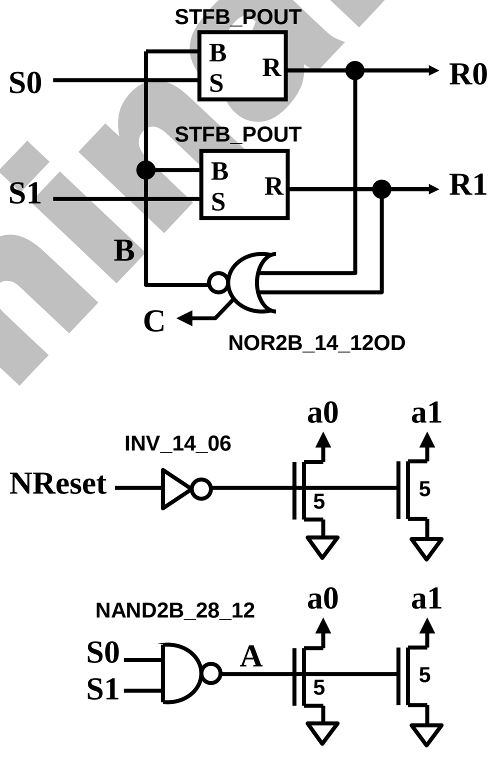
Description

Name:	STFB2_CORE2I
Function:	Core sub-cell for a STFB stage with 1-input and 1-output dual-rail channels.
Dimensions:	12.8 x 29.61 μm

Schematic (b) and symbol (a)



(a)



Widths in μm and all lengths 0.24 μm

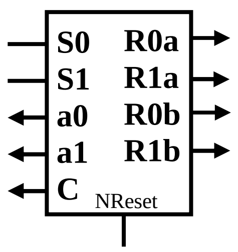
(b)

STFB2_CORE2I4O

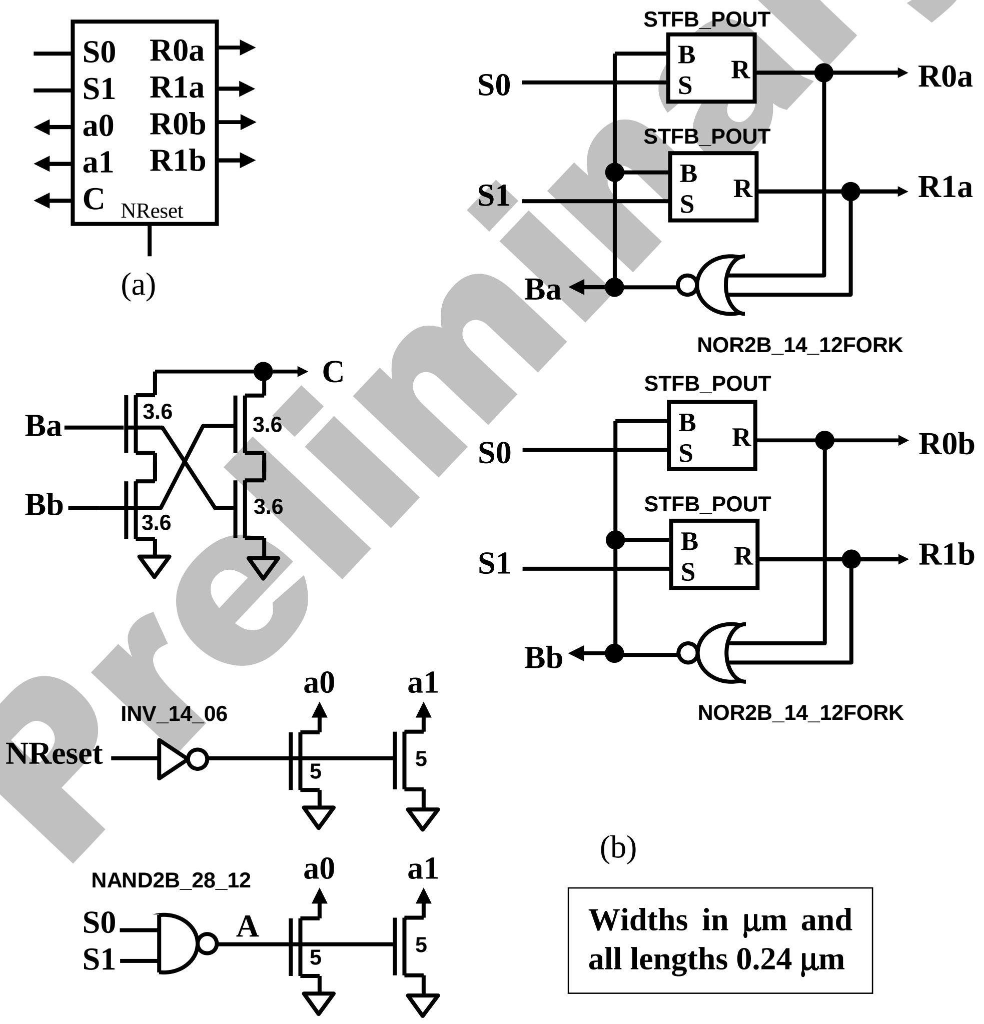
Description

Name:	STFB2_CORE2I4O
Function:	Core sub-cell for a STFB stage with 1-input and 2-output dual-rail channels.
Dimensions:	12.8 x 48.13 µm

Schematic (b) and symbol (a)



(a)



(b)

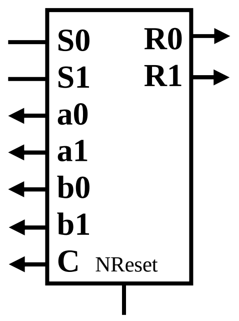
Widths in µm and all lengths 0.24 µm

STFB2_CORE4I

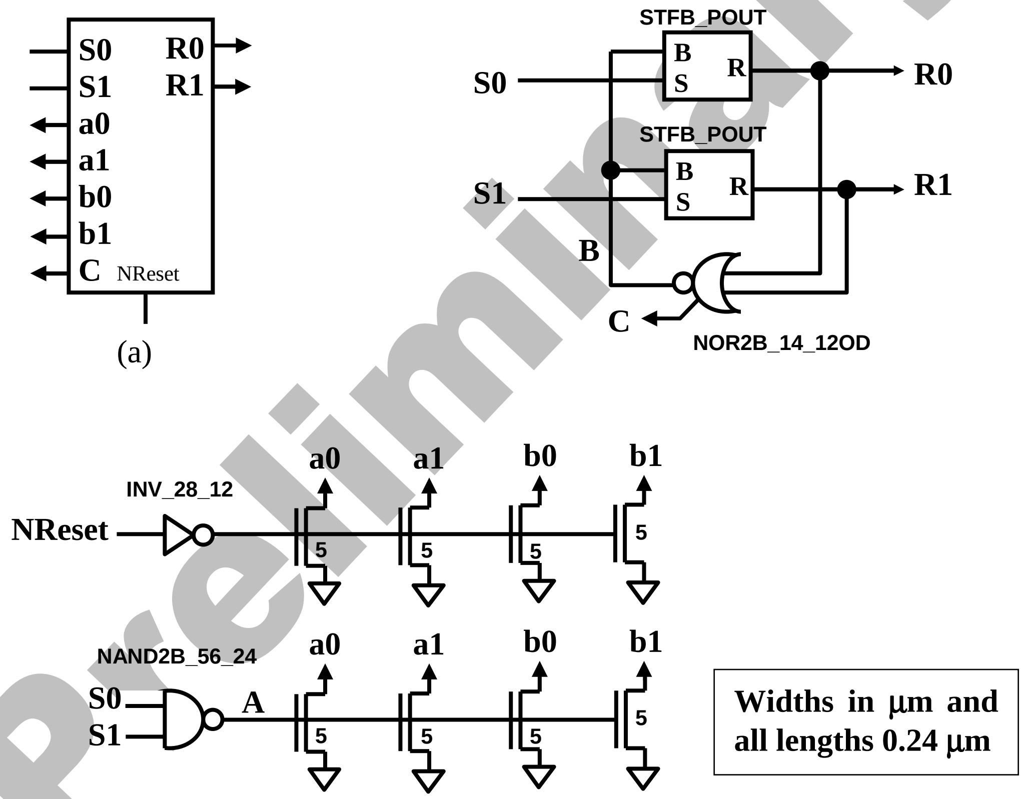
Description

Name:	STFB2_CORE4I
Function:	Core sub-cell for a STFB stage with 2-input and 1-output dual-rail channels.
Dimensions:	12.8 x 31.58 μm

Schematic (b) and symbol (a)



(a)



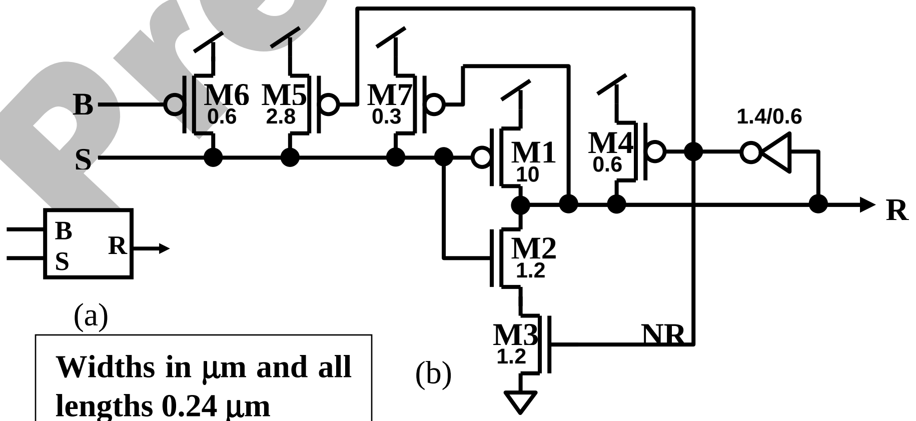
(b)

STFB_POUT

Description

Name:	STFB_POUT
Function:	Single-track output driver with staticizer.
Dimensions:	12.8 x 7.14 μm
Pins:	"R" drives one of the 1-of-N wires in the output channel.
	"S" is driven low by the N-stack driving "R" high.
	"B" is driven low when the output channel is "busy".
Operation:	The STFB_POUT sub-cell includes the staticizer structure and three PMOS transistors utilized to restore the state input ("S") high. If the output channel is empty, "R" is low, the "B" signal is high, 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" high, turning off M1.

Schematic (b) and symbol (a)



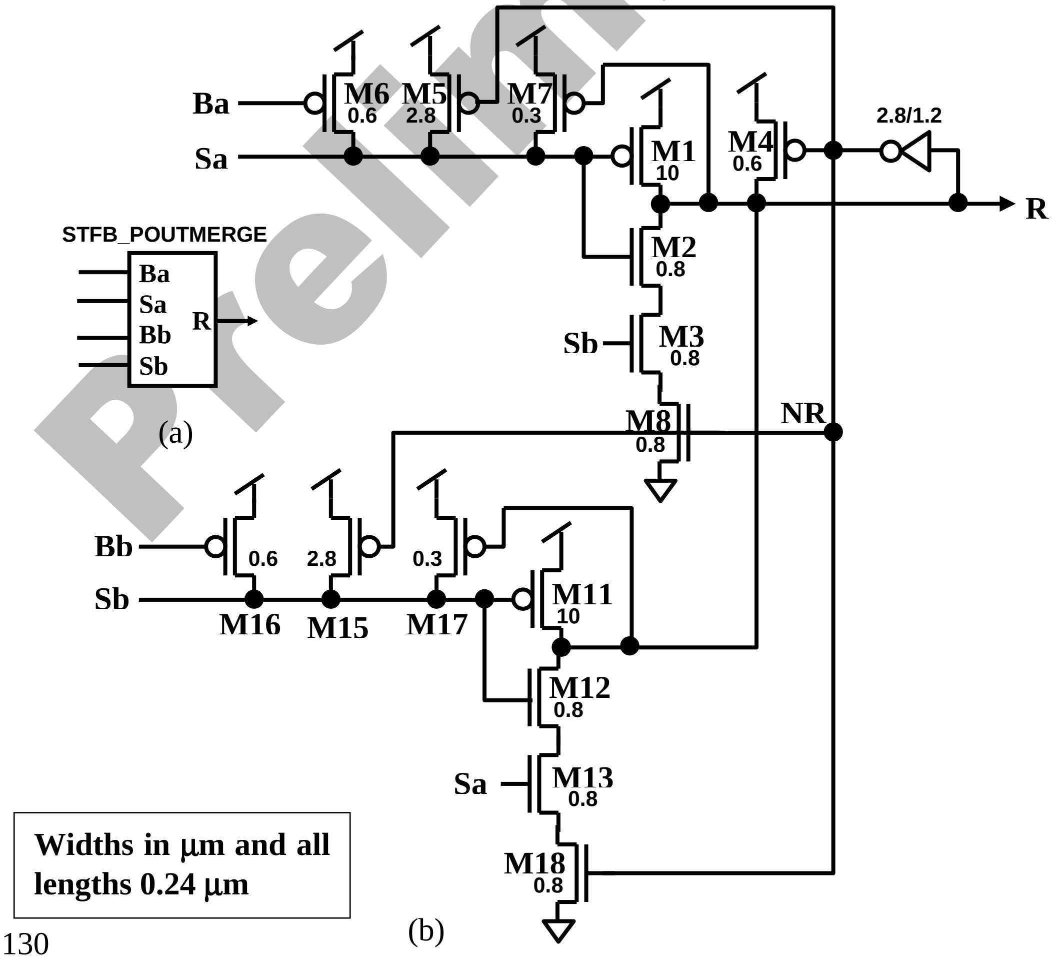
Widths in μm and all lengths 0.24 μm

STFB_POUTMERGE

Description

Name:	STFB_POUTMERGE
Function:	Single-track double output driver with staticizer.
Dimensions:	12.8 x 11.18 µm
Pins:	"R" drives one of the 1-of-N wires in the output channel.
	"Sa and Sb" are driven low by the respective N-stack to drive "R" high.
	"Ba and Bb" are driven low when the output channel is "busy".
Operation:	Same as STFB_POUT sub-cell but with two independent set of inputs (S,B), allowing the merge of two states to one output channel.

Schematic (b) and symbol (a)



(a)

(b)

SUPPORT CELLS

These are the cells that were utilized as support circuitry to the STFB cells. They are designed to be used in the automated P&R flow.

FILL Cell

Description

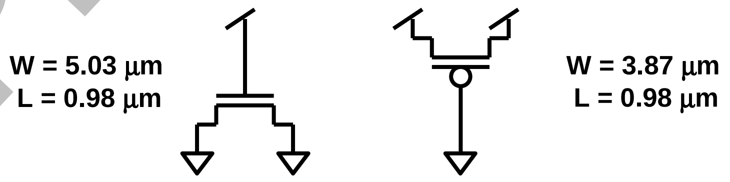
Name:	FILL
Function:	To avoid DRC errors, the filler cell allows a continuous VDD, GND, N-well and implants.
Dimensions:	12.8 x 0.9 μm

FILLCAP3

Description

Name:	FILLCAP3
Function:	To reduce the power supply ripple, this cell inserts two bypass capacitors implemented with transistors [1]. Also, to avoid DRC errors, it allows a continuous VDD, GND, N-well and implants. The total capacitance per cell is 55 fF.
Dimensions:	12.8 x 2.7 μm

Schematic



INV1X

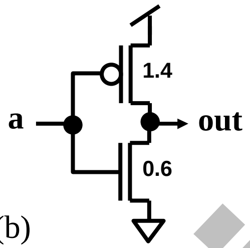
Description

Name:	INV1X
Function:	Minimum size inverter.
Dimensions:	12.8 x 2.7 μm
Logic equation:	out = $\overline{a}$
Truth table:	

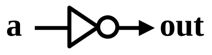
a	out
0	1
1	0

Schematic (b) and symbol (a)

Widths in μm and all lengths 0.24 μm



(b)



(a)

INV3X

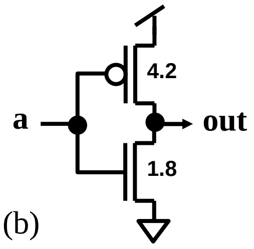
Description

Name:	INV3X
Function:	Three times minimum size inverter.
Dimensions:	12.8 x 2.7 μm
Logic equation:	out = $\overline{a}$
Truth table:	

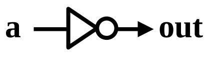
a	out
0	1
1	0

Schematic (b) and symbol (a)

Widths in μm and all lengths 0.24 μm



(b)



(a)

INV12X

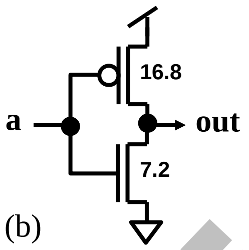
Description

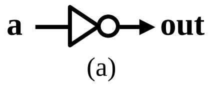
Name:	INV12X
Function:	12 times minimum size inverter.
Dimensions:	12.8 x 6.3 μm
Logic equation:	out = $\overline{a}$

Truth table:

a	out
0	1
1	0

Schematic (b) and symbol (a)





Widths in μm and all lengths 0.24 μm

NAND2X2

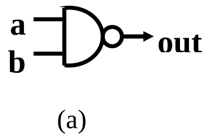
Description

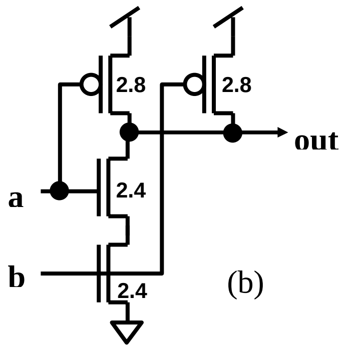
Name:	NAND2X2
Function:	2-input 2 times minimum size NAND gate.
Dimensions:	12.8 x 3.6 μm
Logic equation:	out = $\overline{a \cdot b}$

Truth table:

a	b	out
0	x	1
x	0	1
1	1	0

Schematic (b) and symbol (a)





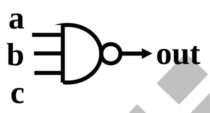
NAND3X2

Description

Name:	NAND3X2
Function:	3-input 2 times minimum size NAND gate.
Dimensions:	12.8 x 4.5 μm
Logic equation:	$\text{out} = \overline{a \cdot b \cdot c}$
Truth table:	

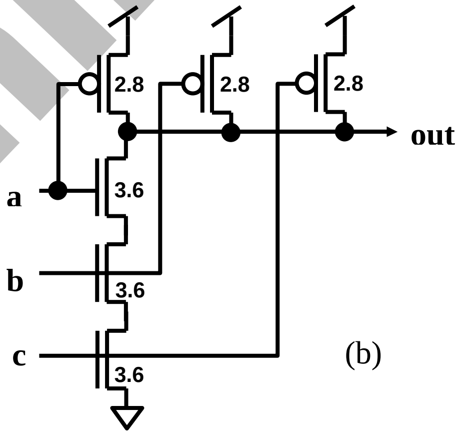
a	b	c	out
0	x	x	1
x	0	x	1
x	x	0	1
1	1	1	0

Schematic (b) and symbol (a)



(a)

Widths in μm and all lengths 0.24 μm



(b)

NAND4X2

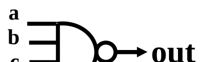
Description

Name:	NAND4X2
Function:	4-input 2 times minimum size NAND gate.
Dimensions:	12.8 x 5.4 µm
Logic equation:	$\text{out} = \overline{a \cdot b \cdot c \cdot d}$

Truth table:

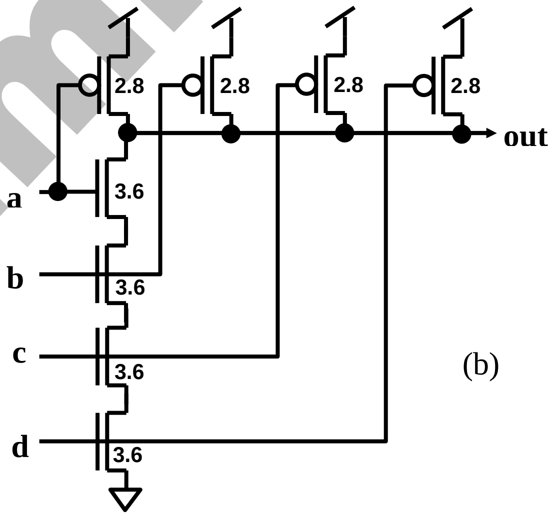
a	b	c	d	out
0	x	x	x	1
x	0	x	x	1
x	x	0	x	1
x	x	x	0	1
1	1	1	1	0

Schematic (b) and symbol (a)



(a)

Widths in µm and all lengths 0.24 µm



(b)

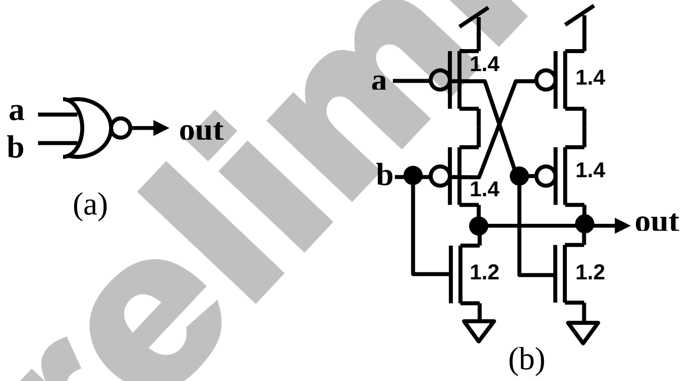
NOR2BX1

Description

Name: NOR2BX1
Function: Symmetrized 2-input minimum size NOR gate.
Dimensions: 12.8 x 4.5 μm
Logic equation: $\text{out} = \overline{a + b}$
Truth table:

a	b	out
1	x	0
x	1	0
0	0	1

Schematic (b) and symbol (a)



Widths in μm and all lengths 0.24 μm

NOR3BX1

Description

Name: NOR3BX1

Function: Symmetrized 3-input minimum size NOR gate.

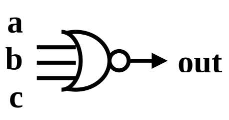
Dimensions: 12.8 x 6.3 µm

Logic equation: $out = \overline{a + b + c}$

Truth table:

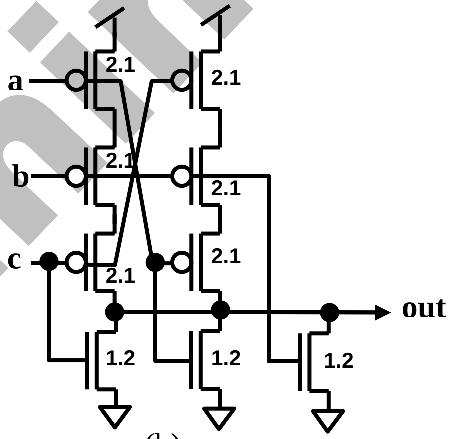
a	b	c	out
1	x	x	0
x	1	x	0
x	x	1	0
0	0	0	1

Schematic (b) and symbol (a)



(a)

Widths in µm and all lengths 0.24 µm



(b)

TRISTATE3X

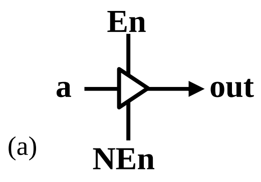
Description

Name:	TRISTATE3X
Function:	3 times minimum size tristate buffer.
Dimensions:	12.8 x 9.9 μm
Logic equation:	If enabled (En = 1 and NEn = 0): out = a
	If disabled (En = 0 and NEn = 1): out = z

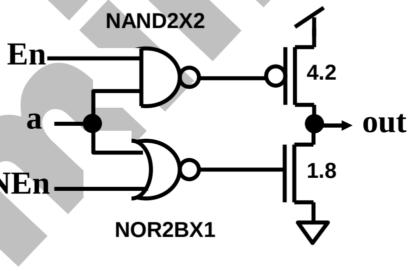
Truth table:

En	NEn	a	out
1	0	0	0
1	0	1	1
0	1	x	z

Schematic (b) and symbol (a)



(a)



(b)

Widths in μm and all lengths 0.24 μm

STFB STANDARD CELLS

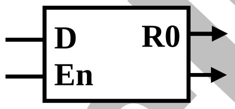
This are the STFB cells utilized for automatic P&R.

STFB2_BITGEN

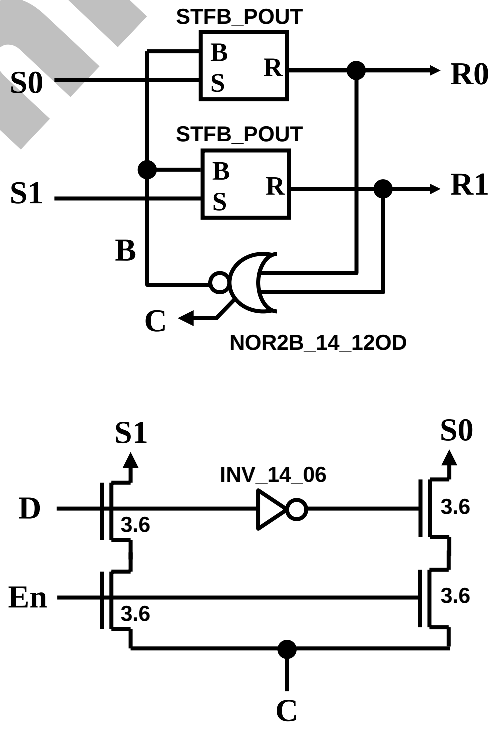
Description

Name:	STFB2_BITGEN
Function:	Generates dual-rail single-track tokens.
Dimensions:	12.8 x 25.2 µm
Pins:	D: single-rail data input.
	En: enable signal.
	R0-R1: output dual-rail single-track channel.
Operation:	If En = 1, then the value of D is used to continuously generate tokens of the same value.

Schematic (b) and symbol (a)



(a)



(b)

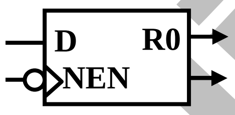
Widths in µm and all lengths 0.24 µm

STFB2_BITGENSINGLE

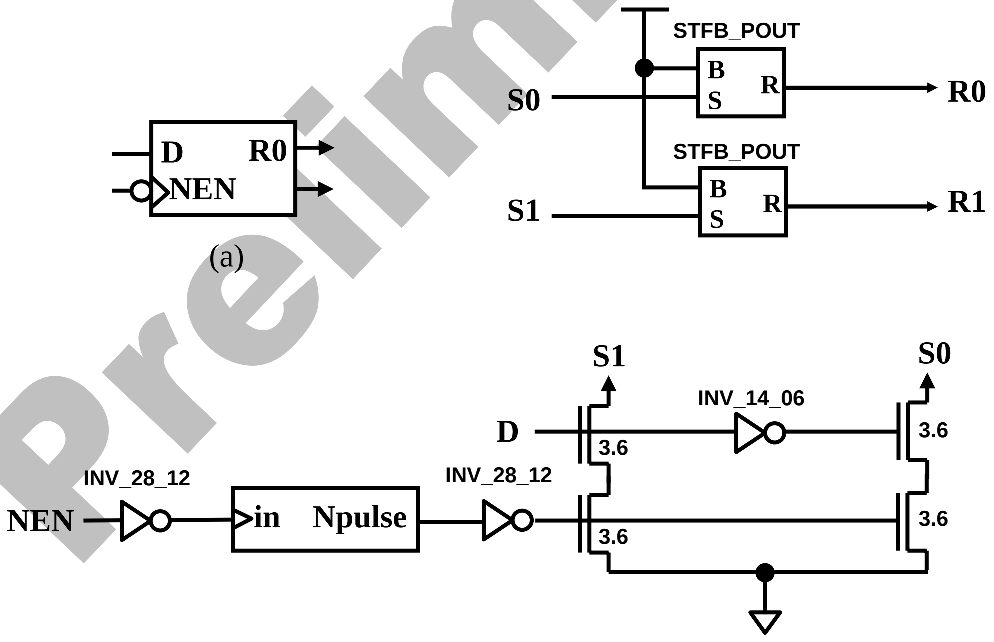
Description

Name:	STFB2_BITGENSINGLE
Function:	Generates one dual-rail single-track token at the falling edge of the NEN signal.
Dimensions:	12.8 x 36 μm
Pins:	D: single-rail data input. NEN: falling edge enable signal. R0-R1: output dual-rail single-track channel.
Operation:	If NEN = ↓, then the value of D is used to generate a single token of the same value as D. It is assumed that the output channel is empty.

Schematic (b) and symbol (a)



(a)



(b)

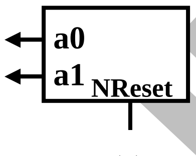
Widths in μm and all lengths 0.24 μm

STFB2_BUCKET

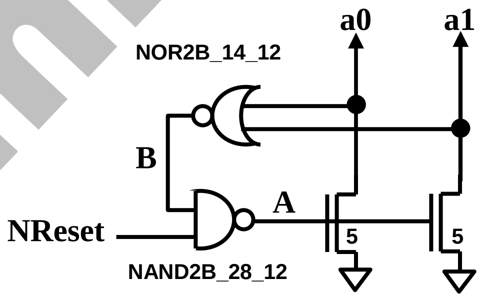
Description

Name:	STFB2_BUCKET
Function:	Consumes dual-rail single-track tokens.
Dimensions:	12.8 x 25.2 μm
Pins:	a0-a1: dual-rail data input. NReset: active low reset.
Operation:	If NReset = 0, then drives the dual-rail input (a0-a1) low. If NReset = 1, then consumes any dual-rail token that arrives at the input (a0-a1).

Schematic (b) and symbol (a)



(a)



(b)

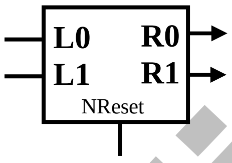
Widths in μm and all lengths 0.24 μm

STFB2_BUFFER

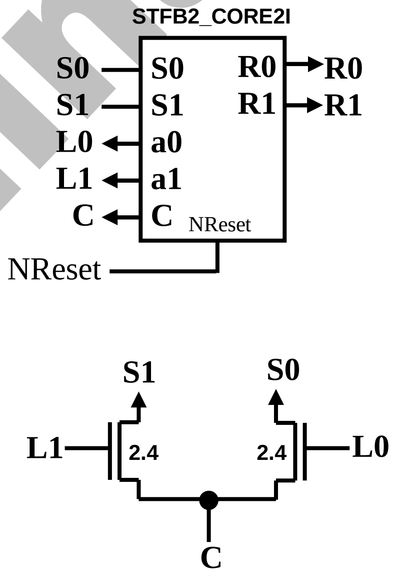
Description

Name:	STFB2_BUFFER
Function:	Copies the input token to the output, both dual-rail single-track channels.
Dimensions:	12.8 x 32.4 μm
Pins:	L0-L1: input dual-rail single-track channel. R0-R1: output dual-rail single-track channel. NReset: active low reset.
Operation (HSE):	STFB2_BUFFER ≡ ∗[[¬R∧L→R↑]; L↓]

Schematic (b) and symbol (a)



(a)



Widths in μm and all lengths 0.24 μm

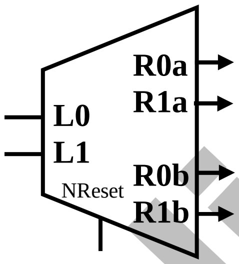
(b)

STFB2_FORK

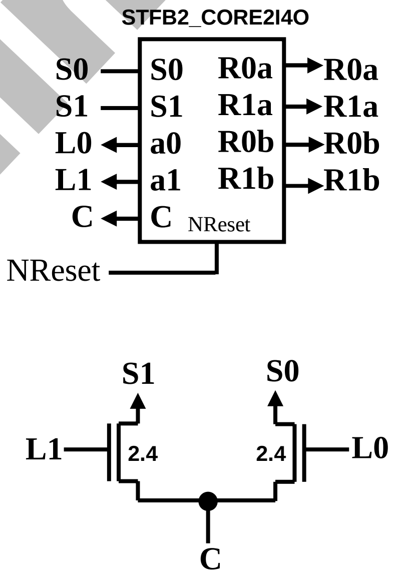
Description

Name:	STFB2_FORK
Function:	Copies the input token to the two outputs, all dual-rail single-track channels.
Dimensions:	12.8 x 50.4 µm
Pins:	L0-L1: input dual-rail single-track channel. R0a-R1a: output dual-rail single-track channel. R0b-R1b: output dual-rail single-track channel. NReset: active low reset.
Operation (HSE):	STFB2_FORK ≡ ∗[[¬Ra∧¬Rb∧L→Ra↑, Rb↑]; L↓]

Schematic (b) and symbol (a)



(a)



(b)

Widths in µm and all lengths 0.24 µm

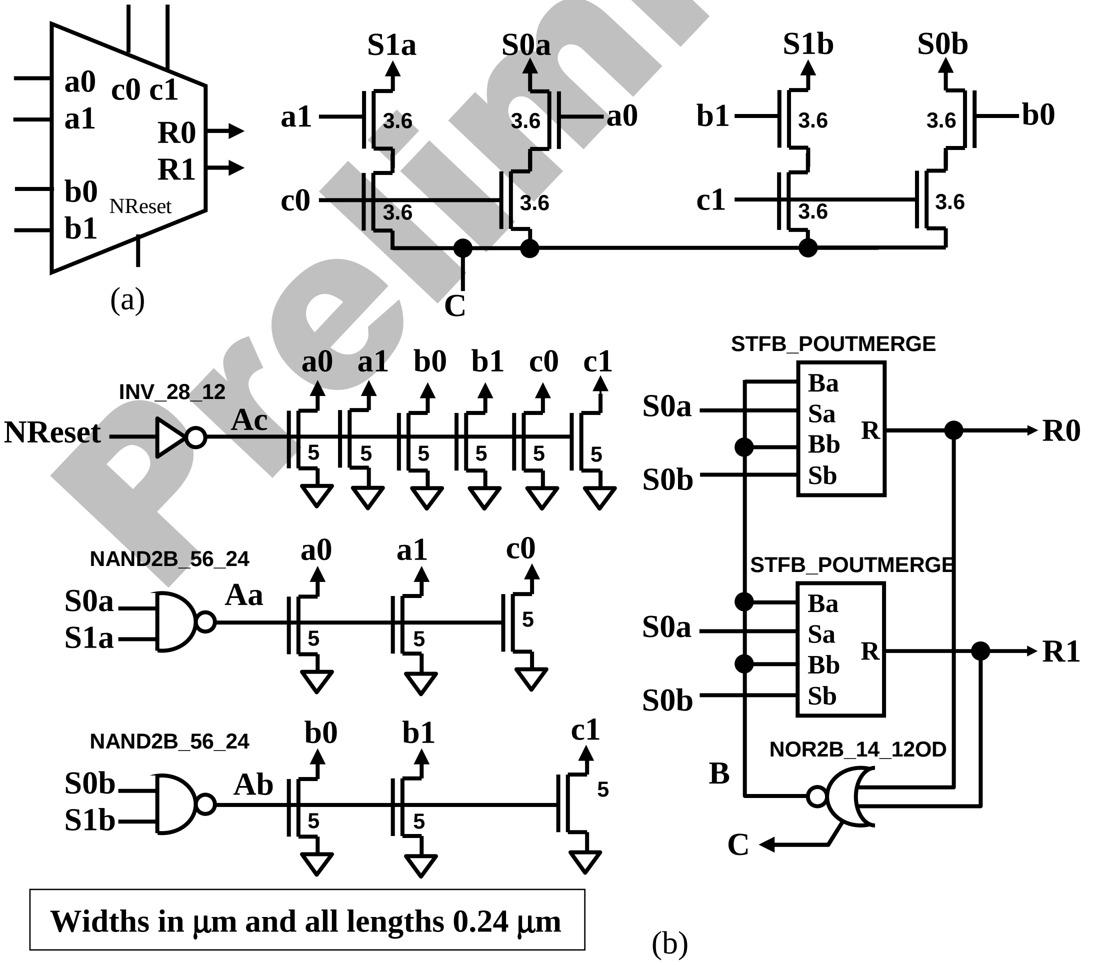
STFB2_MERGE

Description

Name:	STFB2_MERGE
Function:	Based on the value of a control token, forward one of the input tokens to the output.
Dimensions:	12.8 x 51.3 μm
Pins:	a0-a1: input dual-rail single-track channel. b0-b1: input dual-rail single-track channel. c0-c1: input dual-rail single-track channel. R0-R1: output dual-rail single-track channel. NReset: active low reset.
Operation (HSE):	STFB2_MERGE

≡ $*[[\ \neg R \wedge a \wedge c0 \rightarrow R\uparrow];\ a\downarrow;\ c0\downarrow\ \|\ [\neg R \wedge b \wedge c1 \rightarrow R\uparrow\ ];\ b\downarrow;\ c1\downarrow]$

Schematic (b) and symbol (a)

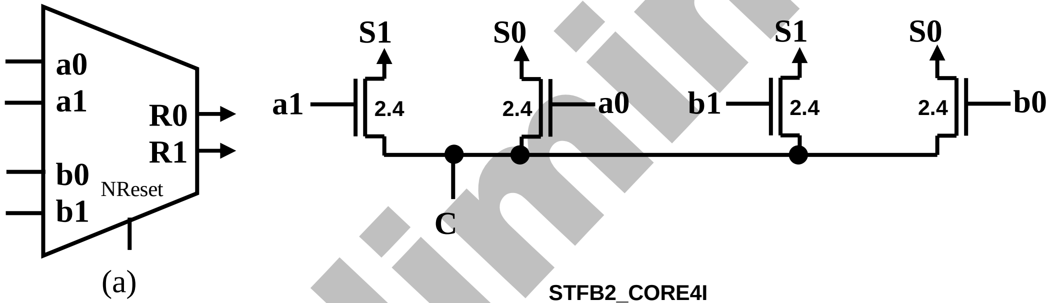


STFB2_MERGENC

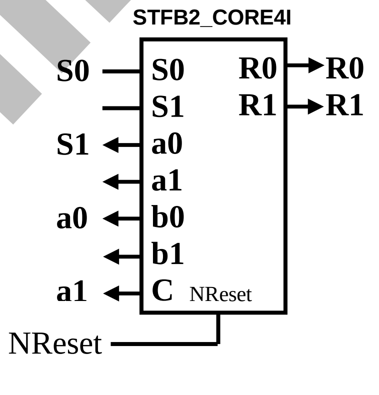
Description

Name:	STFB2_MERGENC
Function:	Forward any input tokens to the output. The input tokens must be mutually exclusive.
Dimensions:	12.8 x 36 μm
Pins:	a0-a1: input dual-rail single-track channel. b0-b1: input dual-rail single-track channel. R0-R1: output dual-rail single-track channel. NReset: active low reset.
Operation (HSE):	STFB2_MERGENC ≡ ∗[[¬R∧(a | b)→R↑]; a↓; b↓]

Schematic (b) and symbol (a)



(a)



(b)

Widths in μm and all lengths 0.24 μm

STFB2_SPLIT

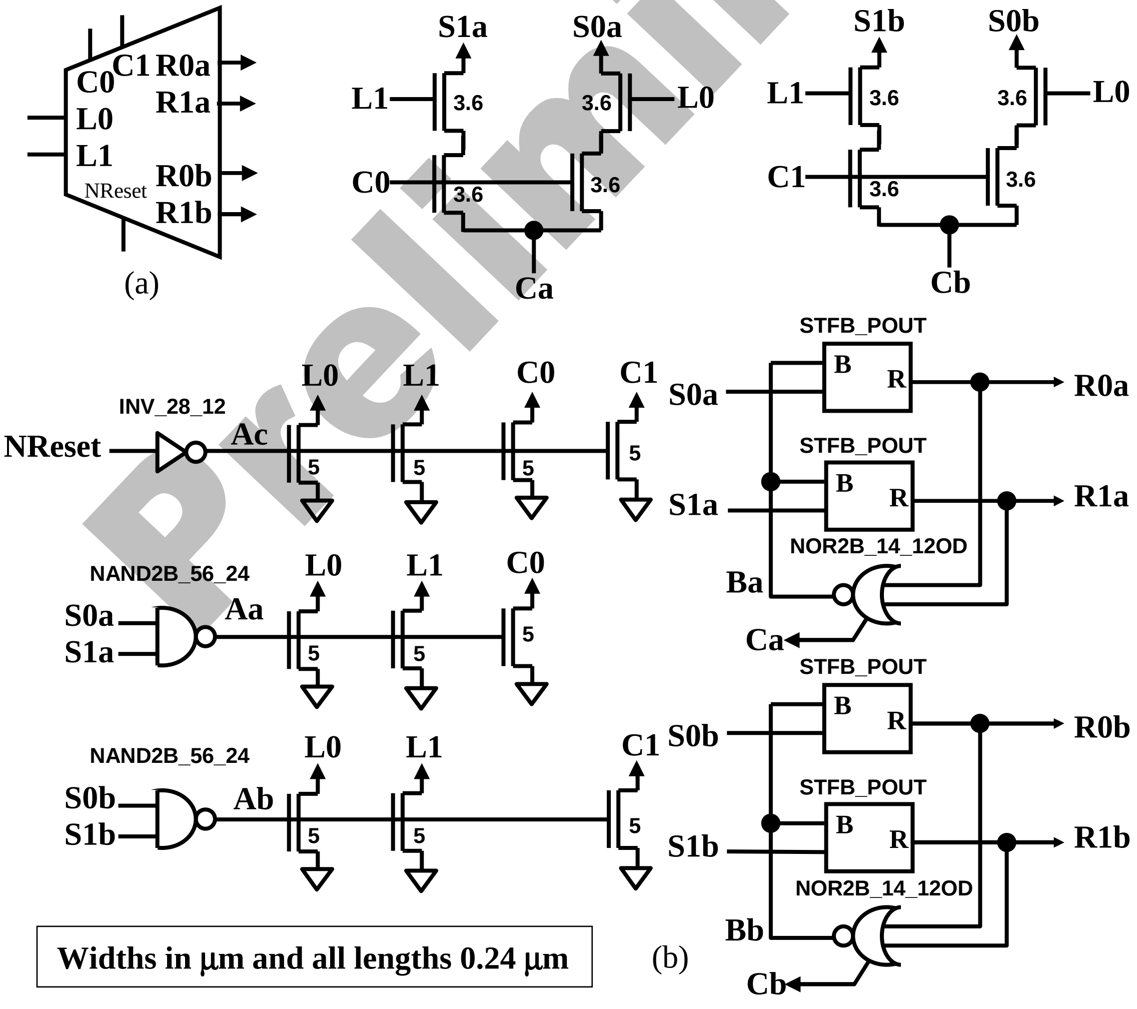
Description

Name:	STFB2_SPLIT
Function:	Copies the input token to the one of the outputs based on a control token, all dual-rail single-track channels.
Dimensions:	12.8 x 63.9 μm
Pins:	L0-L1: data input dual-rail single-track channel. C0-C1: control input dual-rail single-track channel. R0a-R1a: output dual-rail single-track channel. R0b-R1b: output dual-rail single-track channel. NReset: active low reset.

Operation (HSE):

STFB2_SPLIT ≡ ∗[[¬Ra∧C0∧L→Ra↑ | ¬Rb∧C1∧L→Rb↑]; L↓, C↓]

Schematic (b) and symbol (a)



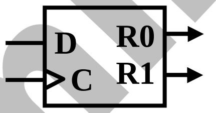
Widths in μm and all lengths 0.24 μm

STFB2_SRST

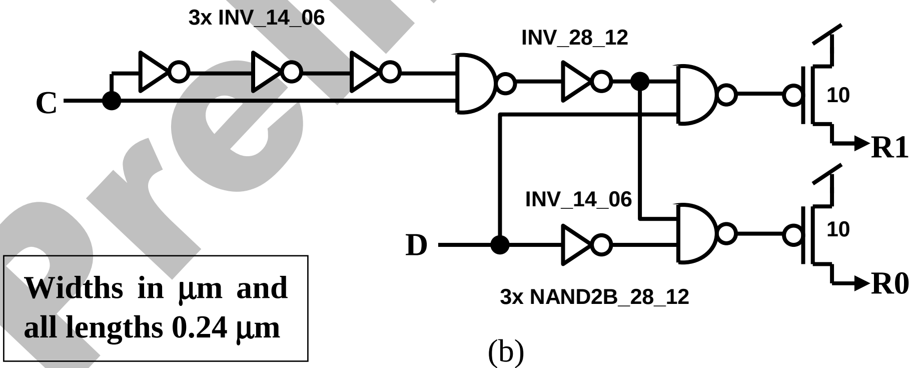
Description

Name:	STFB2_SRST
Function:	Converts a single-rail value to a single-track token.
Dimensions:	12.8 x 30.6 μm
Pins:	D: single-rail data input.
	C: single-rail control input.
	R0-R1: output dual-rail single-track channel.
Operation:	At positive edge of the control signal (C), the value of the data (D) is converted to dual-rail single-track (R0-R1).

Schematic (b) and symbol (a)



(a)



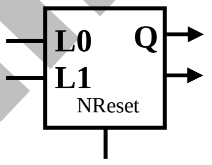
(b)

STFB2_STSR

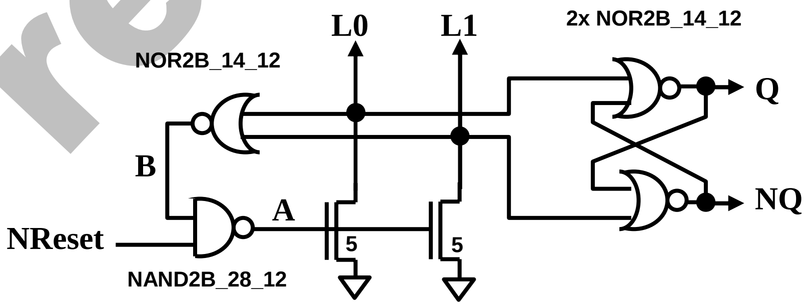
Description

Name:	STFB2_STSR
Function:	Converts a dual-rail single-track token to a single-rail value.
Dimensions:	12.8 x 21.6 μm
Pins:	L0-L1: dual-rail single-track data input. NReset: active low reset. Q: single-rail data output. NQ: complemented single-rail data output. R0-R1: output dual-rail single-track channel.
Operation:	When a token arrives in the dual-rail single-track channel (L0-L1), its value is utilized to set the single-rails outputs (Q and NQ) and the token is consumed.

Schematic (b) and symbol (a)



(a)



(b)

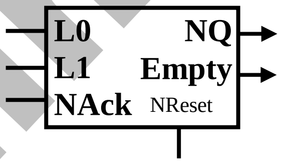
Widths in μm and all lengths 0.24 μm

STFB2_STSRALIGN

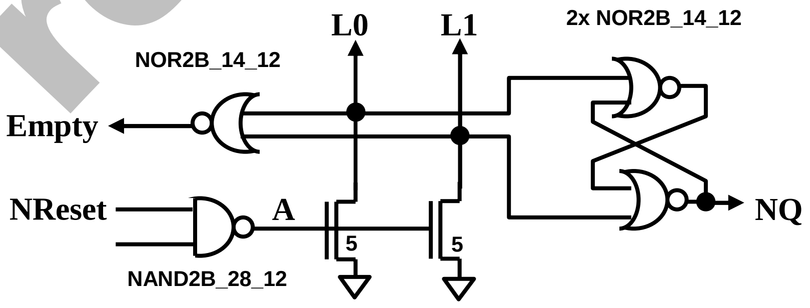
Description

Name:	STFB2_STSRALIGN
Function:	Converts a dual-rail single-track token to a single-rail value and waits for a command to consume the token.
Dimensions:	12.8 x 21.6 µm
Pins:	L0-L1: dual-rail single-track data input. NReset: active low reset. Q: single-rail data output. NQ: complemented single-rail data output. R0-R1: output dual-rail single-track channel.
Operation:	When a token arrives in the dual-rail single-track channel (L0-L1), its value is utilized to set the single-rails outputs (Q and NQ) and the token is consumed.

Schematic (b) and symbol (a)



(a)



(b)

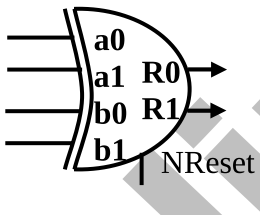
Widths in µm and all lengths 0.24 µm

STFB2_XOR2

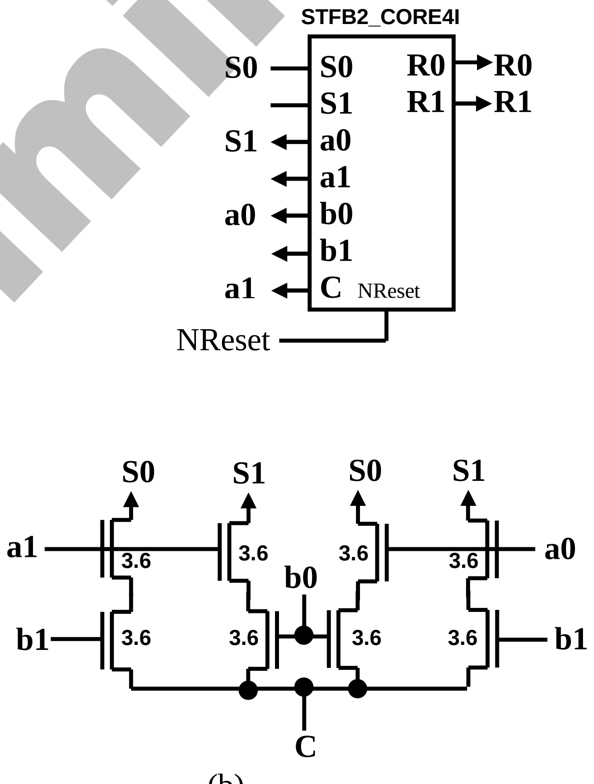
Description

Name:	STFB2_XOR2
Function:	Performs the exclusive-or operation on the input tokens and generate an output token, all dual-rail single-track channels.
Dimensions:	12.8 x 36.9 µm
Pins:	a0-a1: "a" input dual-rail single-track channel. b0-b1: "b" input dual-rail single-track channel. R0-R1: output dual-rail single-track channel. NReset: active low reset.
Operation (HSE):	STFB2_XOR2 ≡ $*[[\neg R \wedge a \wedge b \rightarrow R\uparrow]; a\downarrow\ b\downarrow]$

Schematic (b) and symbol (a)



(a)



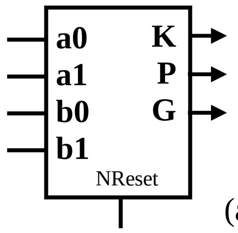
(b)

STFB3_AB_KPG

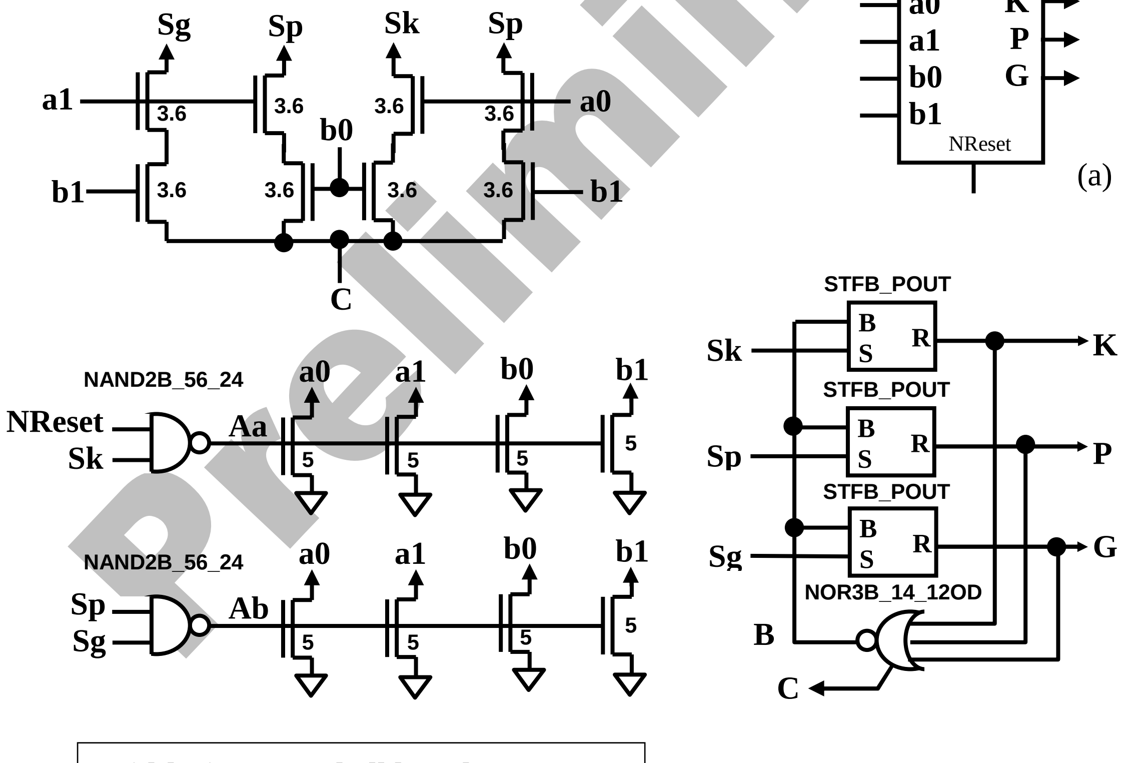
Description

Name:	STFB2_AB_KPG
Function:	Generates one 1-of-3 token form two dual-rail input tokens.
Dimensions:	12.8 x 49.5 μm
Pins:	a0-a1: “a” input dual-rail single-track channel. b0-b1: “b” input dual-rail single-track channel. KPG: 1-of-3 single-track output channel. NReset: active low reset.
Operation:	K = a0.b0; P = a1.b0 + a0.b1; G = a1.b1

Schematic (b) and symbol (a)



(a)



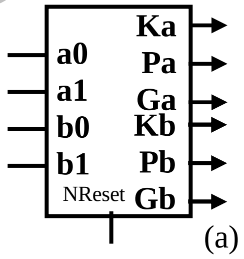
(b)

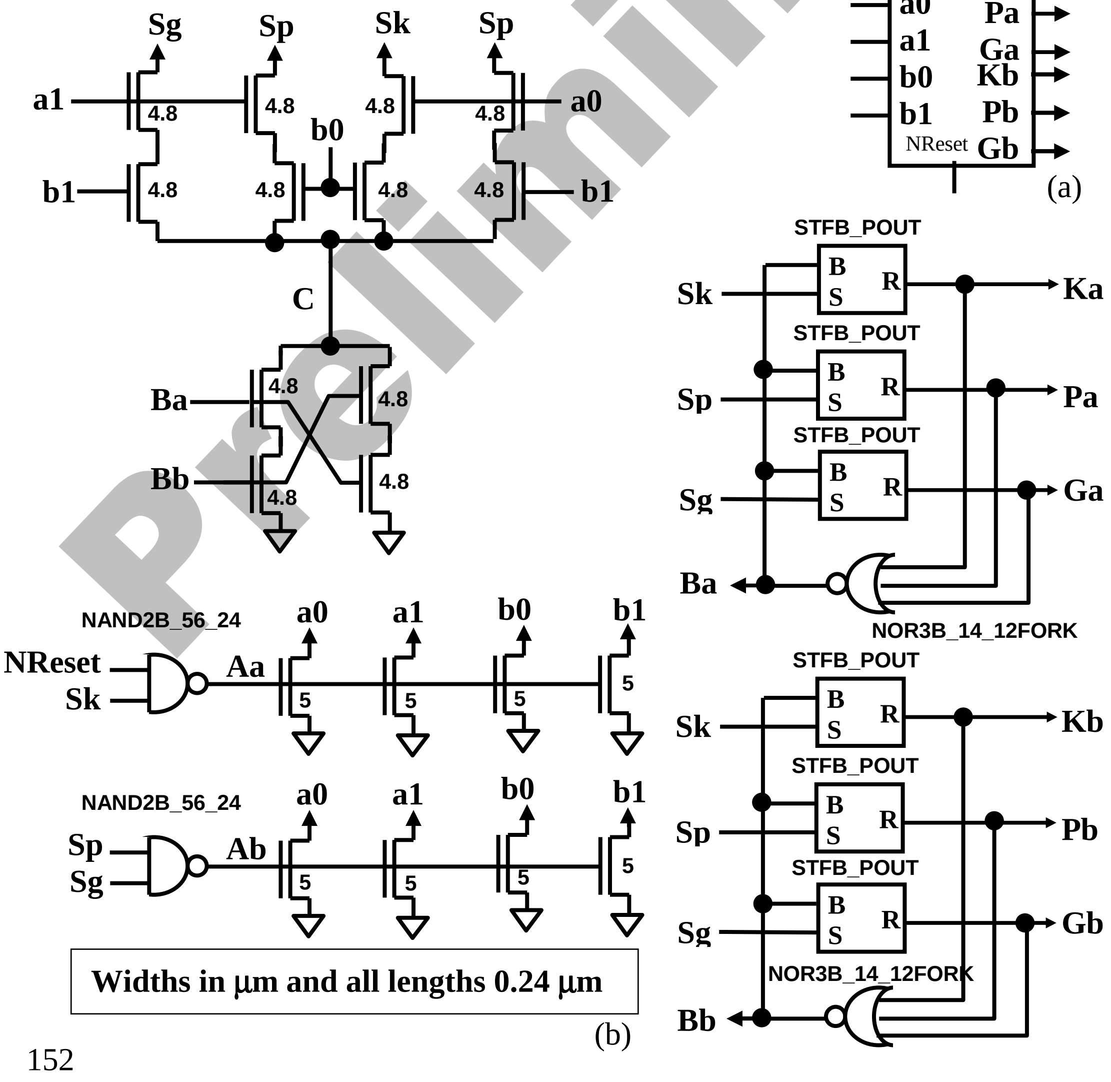
STFB3_AB_KPG2

Description

Name:	STFB2_AB_KPG2
Function:	Generates two 1-of-3 token form two dual-rail input tokens.
Dimensions:	12.8 x 77.4 μm
Pins:	a0-a1: "a" input dual-rail single-track channel. b0-b1: "b" input dual-rail single-track channel. KaPaGa: 1-of-3 single-track output channel. KbPbGb: 1-of-3 single-track output channel. NReset: active low reset.
Operation:	K = a0.b0; P = a1.b0 + a0.b1; G = a1.b1

Schematic (b) and symbol (a)





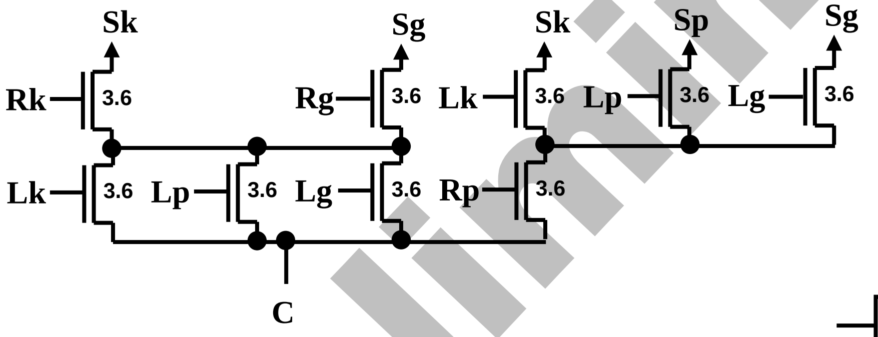
Widths in μm and all lengths 0.24 μm

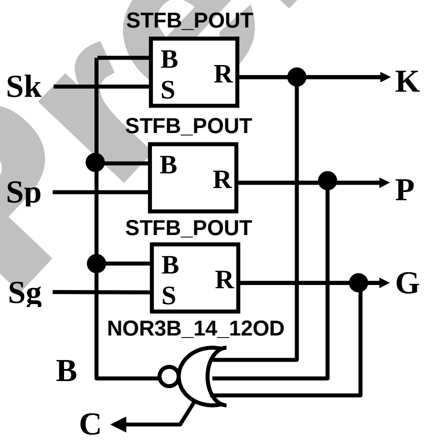
STFB3_KPG2_KPG

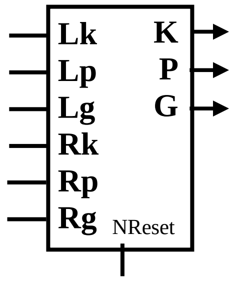
Description

Name:	STFB2_KPG2_KPG
Function:	Generates one 1-of-3 token form two 1-of-3 input tokens.
Dimensions:	12.8 x 49.5 μm
Pins:	LkLpLg: "Left" input 1-of-3 single-track channel. RkRpRg: "Right" input 1-of-3 single-track channel. KPG: 1-of-3 single-track output channel. NReset: active low reset.
Operation:	K = Rk+Rp.Lk; P = Rp.Lp; G = Rg+Rp.Lg

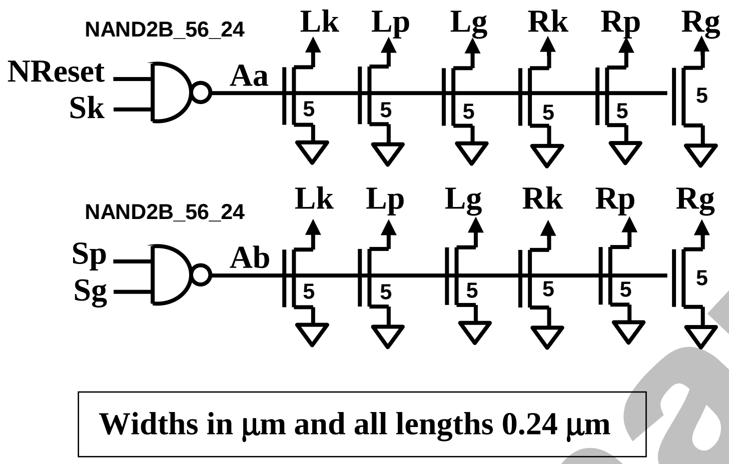
Schematic (b) and symbol (a)







(a)



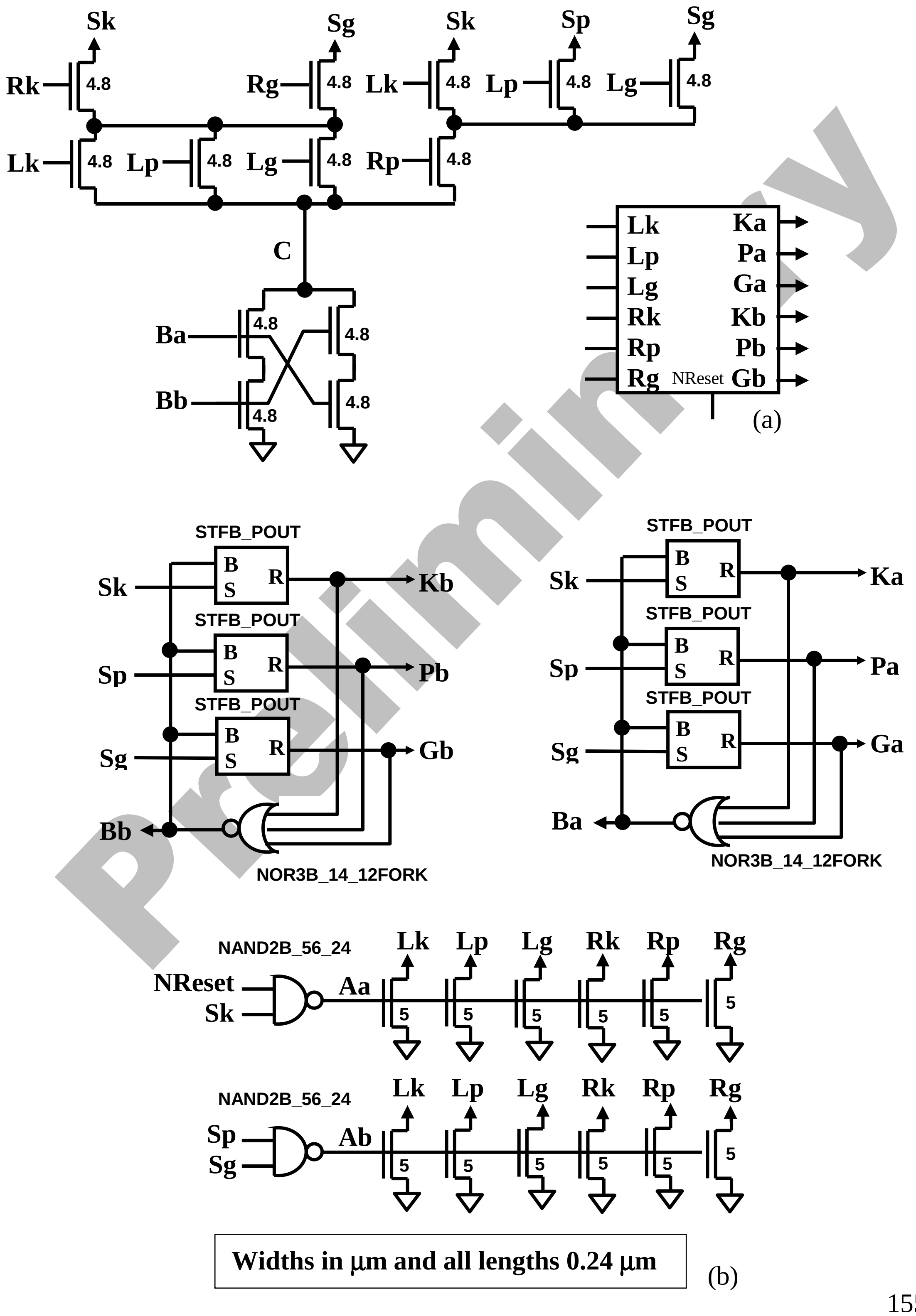
(b)

STFB3_KPG2_KPG2

Description

Name:	STFB2_KPG2_KPG
Function:	Generates two 1-of-3 tokens form two 1-of-3 input tokens.
Dimensions:	12.8 x 78.3 m
Pins:	LkLpLg: "Left" input 1-of-3 single-track channel.
	RkRpRg: "Right" input 1-of-3 single-track channel.
	KaPaGa: 1-of-3 single-track output channel.
	KbPbGb: 1-of-3 single-track output channel.
	NReset: active low reset.
Operation:	K = Rk+Rp.Lk; P = Rp.Lp; G = Rg+Rp.Lg

Schematic (b) and symbol (a)

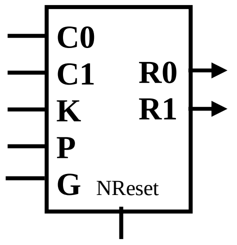


STFB3_KPGC_C

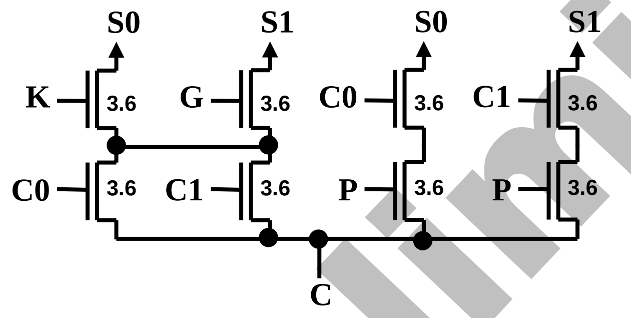
Description

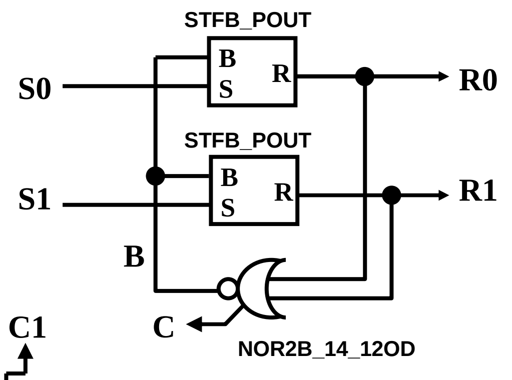
Name:	STFB2_KPGC_C
Function:	Generates one dual-rail token form one 1-of-3 and one dual-rail input tokens.
Dimensions:	12.8 x 39.6 μm
Pins:	C0-C1: “Carry” input dual-rail single-track channel.
	KPG: “kpg” input 1-of-3 single-track channel.
	R0-R1: “Carry out” output dual-rail single-track channel.
	NReset: active low reset.
Operation:	R0 = K.(C0+C1)+P.C0; R1 = G.(C0+C1)+P.C1

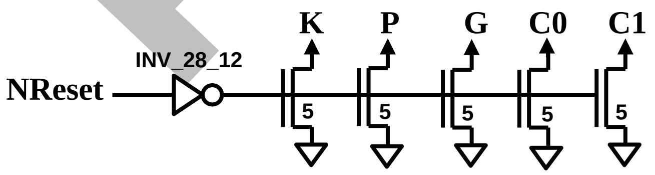
Schematic (b) and symbol (a)

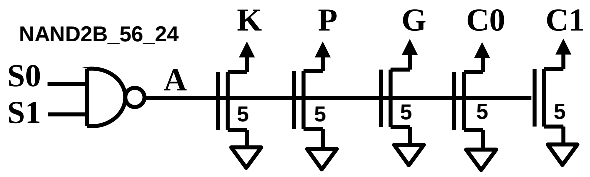


(a)









(b)

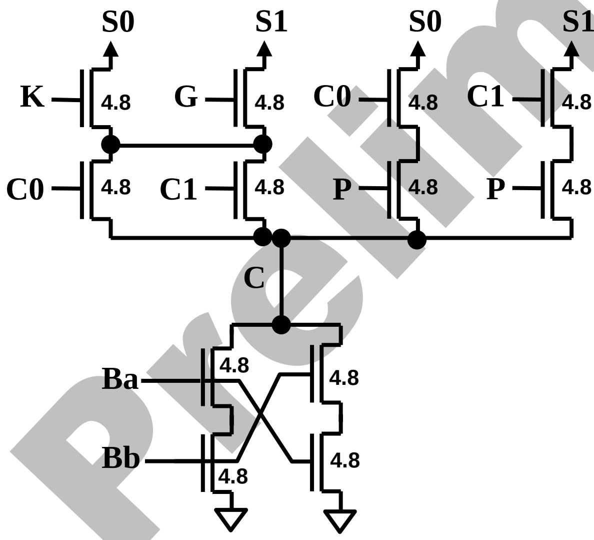
Widths in μm and all lengths 0.24 μm

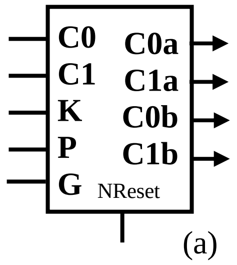
STFB3_KPGC_C2

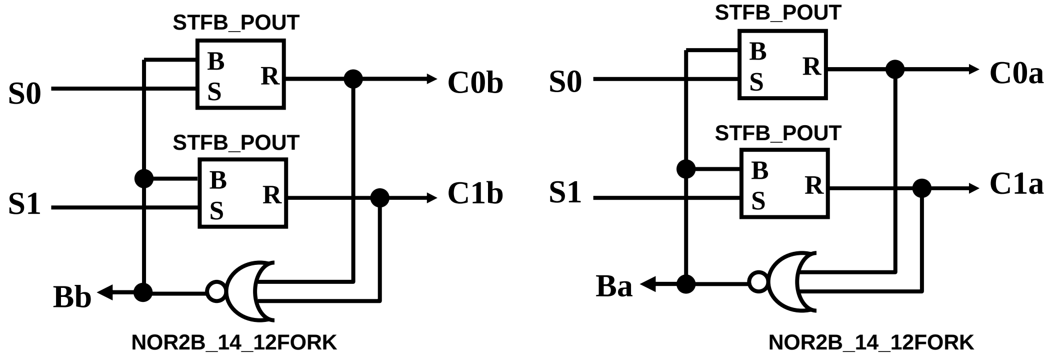
Description

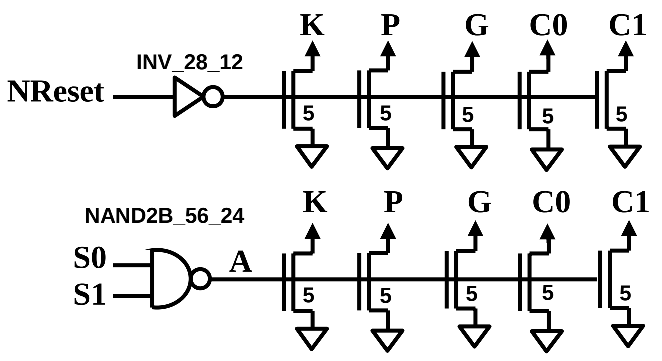
Name:	STFB2_AB_KPG2
Function:	Generates two dual-rail token form one 1-of-3 and one dual-rail input tokens.
Dimensions:	12.8 x 59.4 µm
Pins:	C0-C1: “Carry” input dual-rail single-track channel.
	KPG: “kpg” input 1-of-3 single-track channel.
	C0a-C1a: “Carry out” output dual-rail single-track channel.
	C0b-C1b: “Carry out” output dual-rail single-track channel.
	NReset: active low reset.
Operation:	C0a/b = K.(C0+C1)+P.C0; C1a/b = G.(C0+C1)+P.C1

Schematic (b) and symbol (a)









Widths in μm and all lengths 0.24 μm

(b)

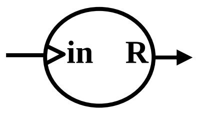
STFB_CHINIT

Description

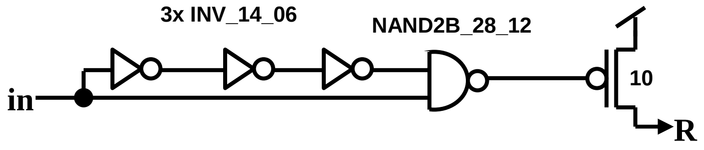
Name:	STFB_CHINIT
Function:	Inserts a token in a 1-of-N single-track channel.
Dimensions:	12.8 x 14.4 μm
Pins:	in: single-rail trigger input.
	R: open-drain output to be connected to a wire in a single-track channel.
Operation:	At positive edge of the control signal (in), the open-drain output (R) inserts a token in a 1-of-N single-track channel.

Schematic (b) and symbol (a)

Widths in μm and all lengths 0.24 μm



(a)



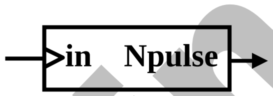
(b)

STFB_NPULSE

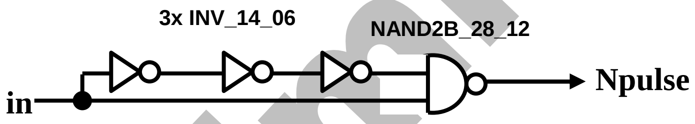
Description

Name:	STFB_NPULSE
Function:	Generates a 3-transistion negative pulse.
Dimensions:	12.8 x 14.4 µm
Pins:	in: single-rail trigger input.
	Npulse: single-rail output.
Operation:	At positive edge of the control signal (in), the single-rail output (Npulse) stays low during 3-transitions (3 gate-dalays).

Schematic (b) and symbol (a)



(a)



(b)

APPENDIX B: DEMONSTRATION CHIP SCHEMATICS

This appendix shows the schematics used to implement the demonstration chip ASYNC1b. All levels are expanded down to basic cells shown in the appendix A.

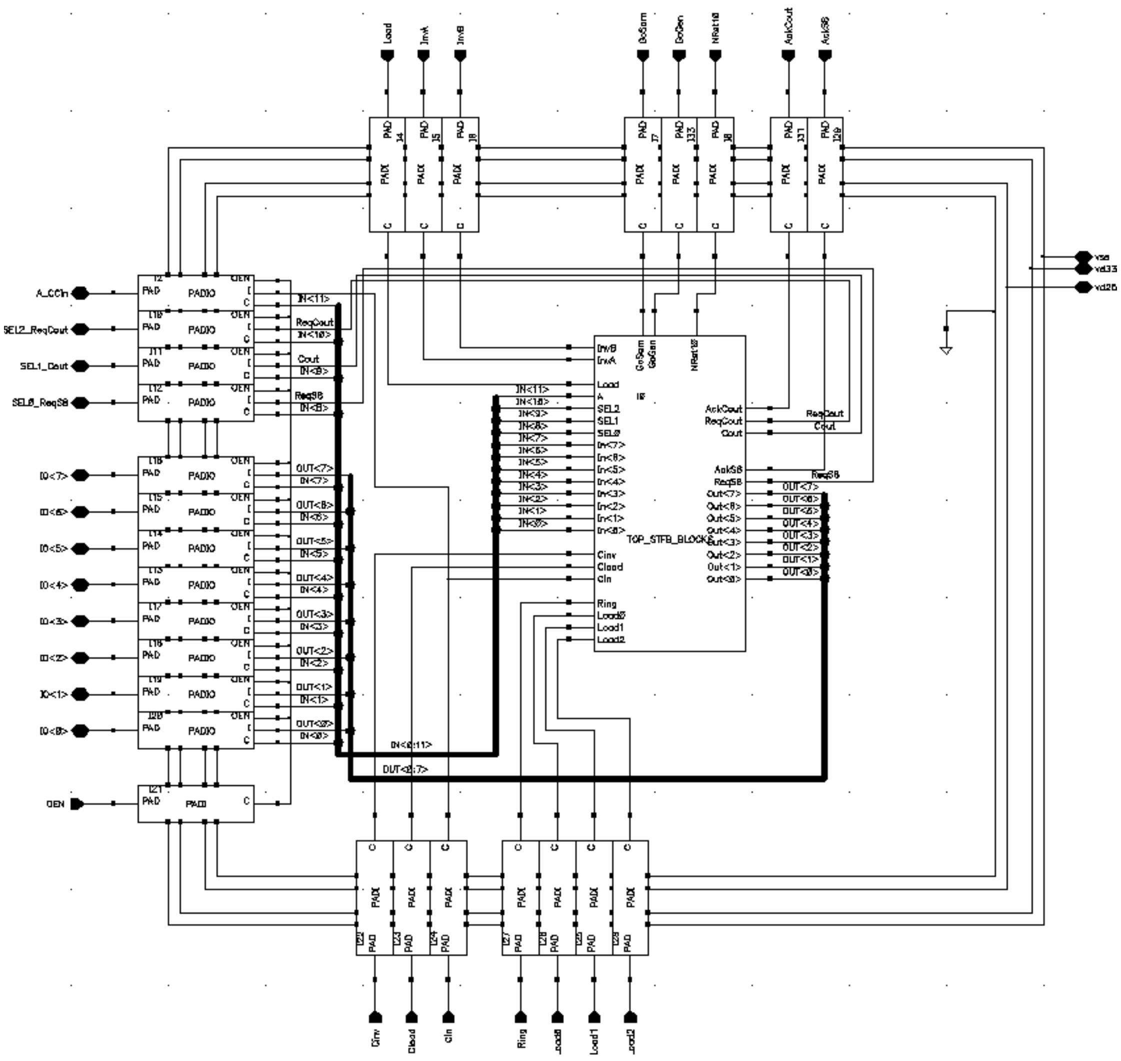


Figure 1. (STFBCHIP) Top level schematic with pads for LVS of the STFB circuits.

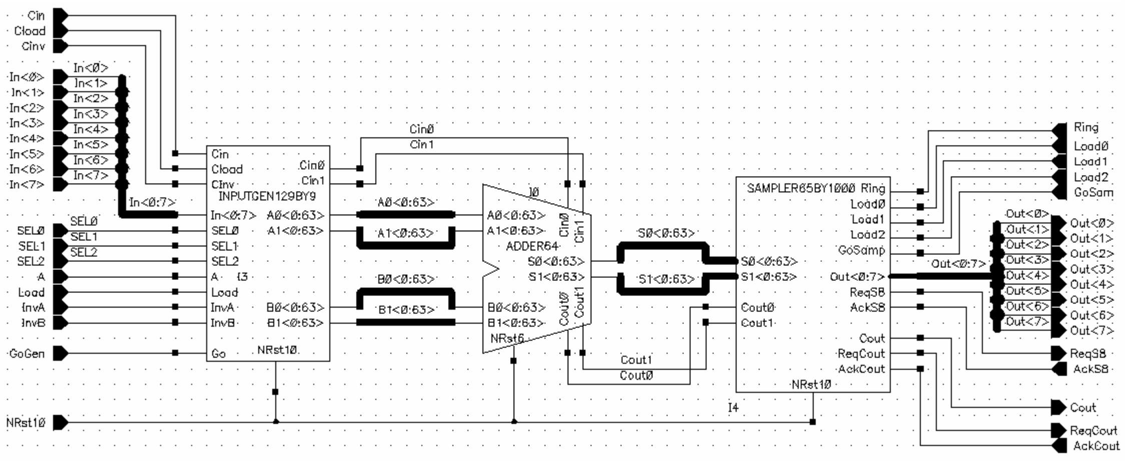


Figure 2. (STFBBLOCKS) Main STFB circuit blocks.

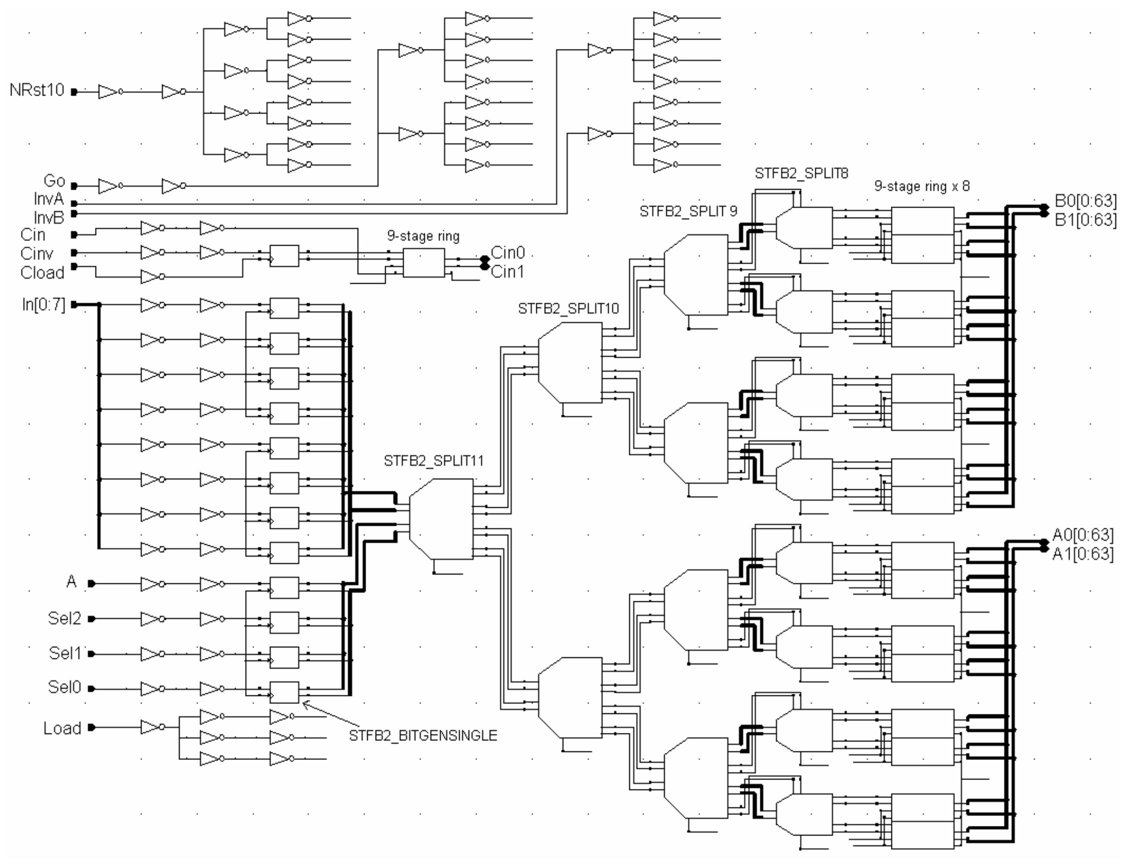


Figure 3. (INPUTGEN129BY9) 129-bit input generating block.

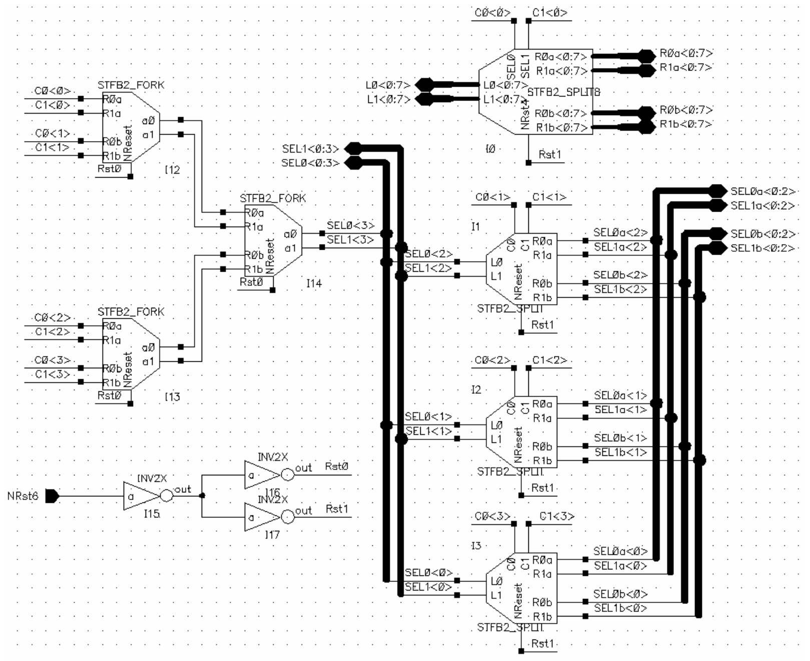


Figure 4. (STFB2_SPLIT11) 11-bit split.

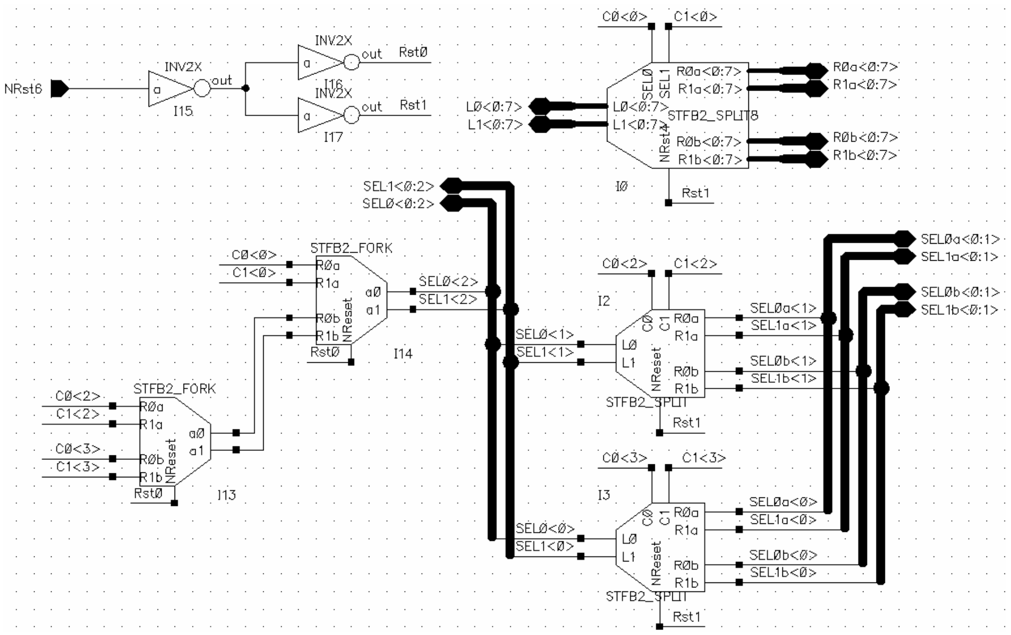


Figure 5. (STFB2_SPLIT10) 10-bit split.

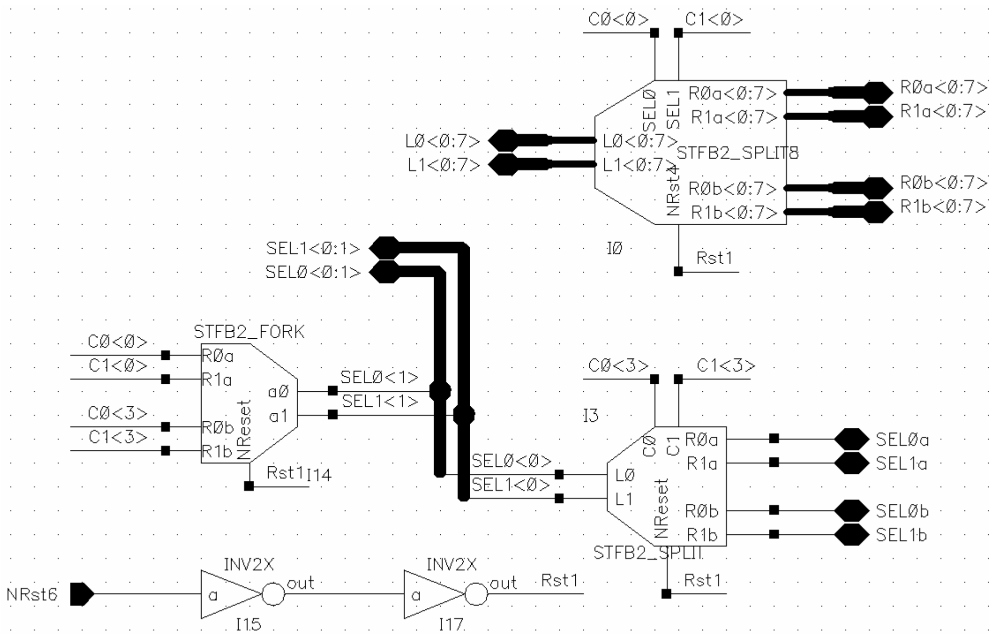


Figure 6. (STFB2_SPLIT9) 9-bit split.

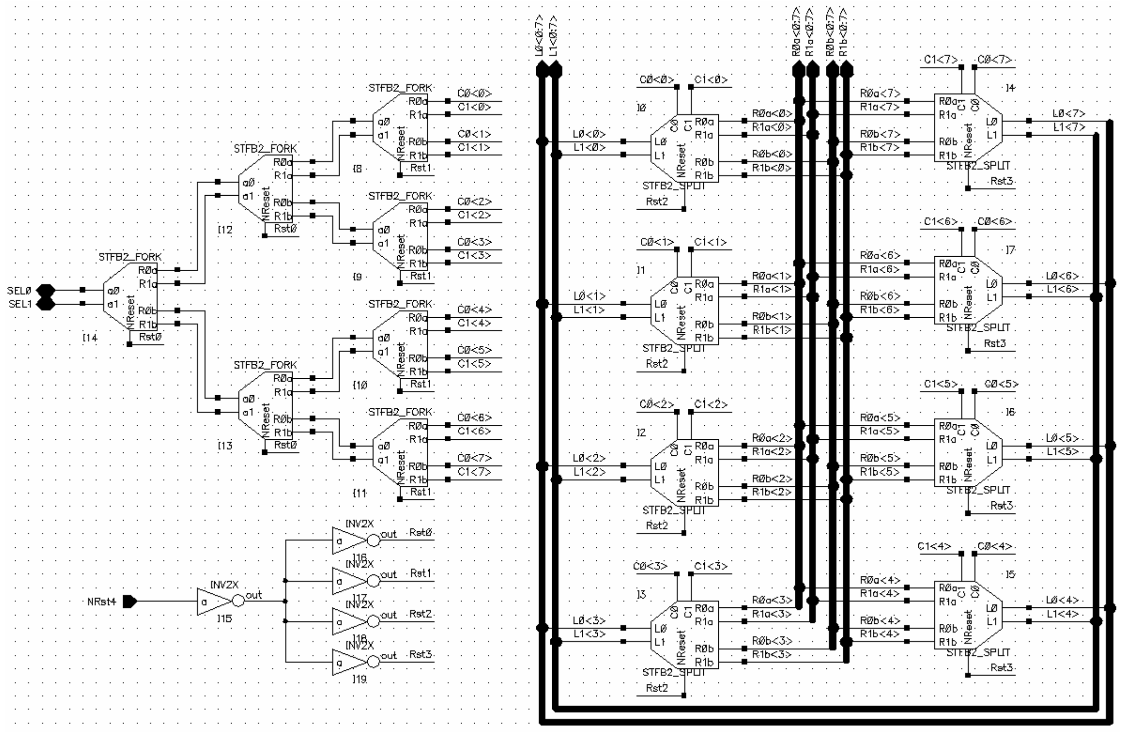


Figure 7. (STFB2_SPLIT8) 8-bit split.

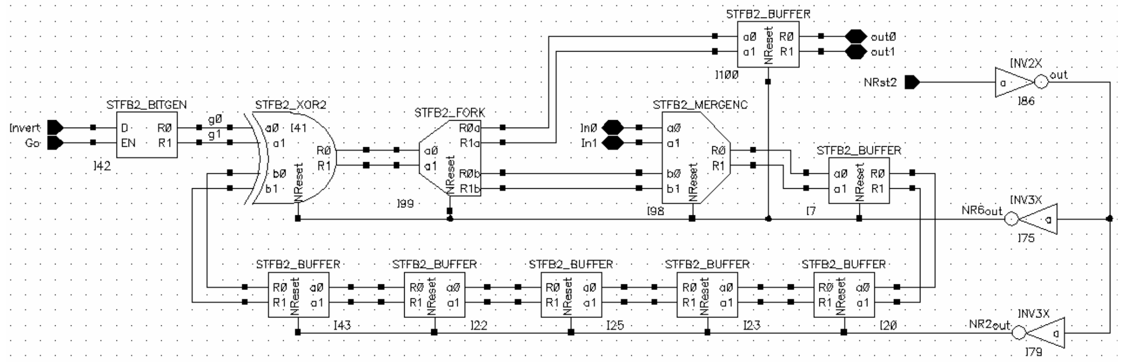


Figure 8. (STFB2_RING9) 9-stage ring.

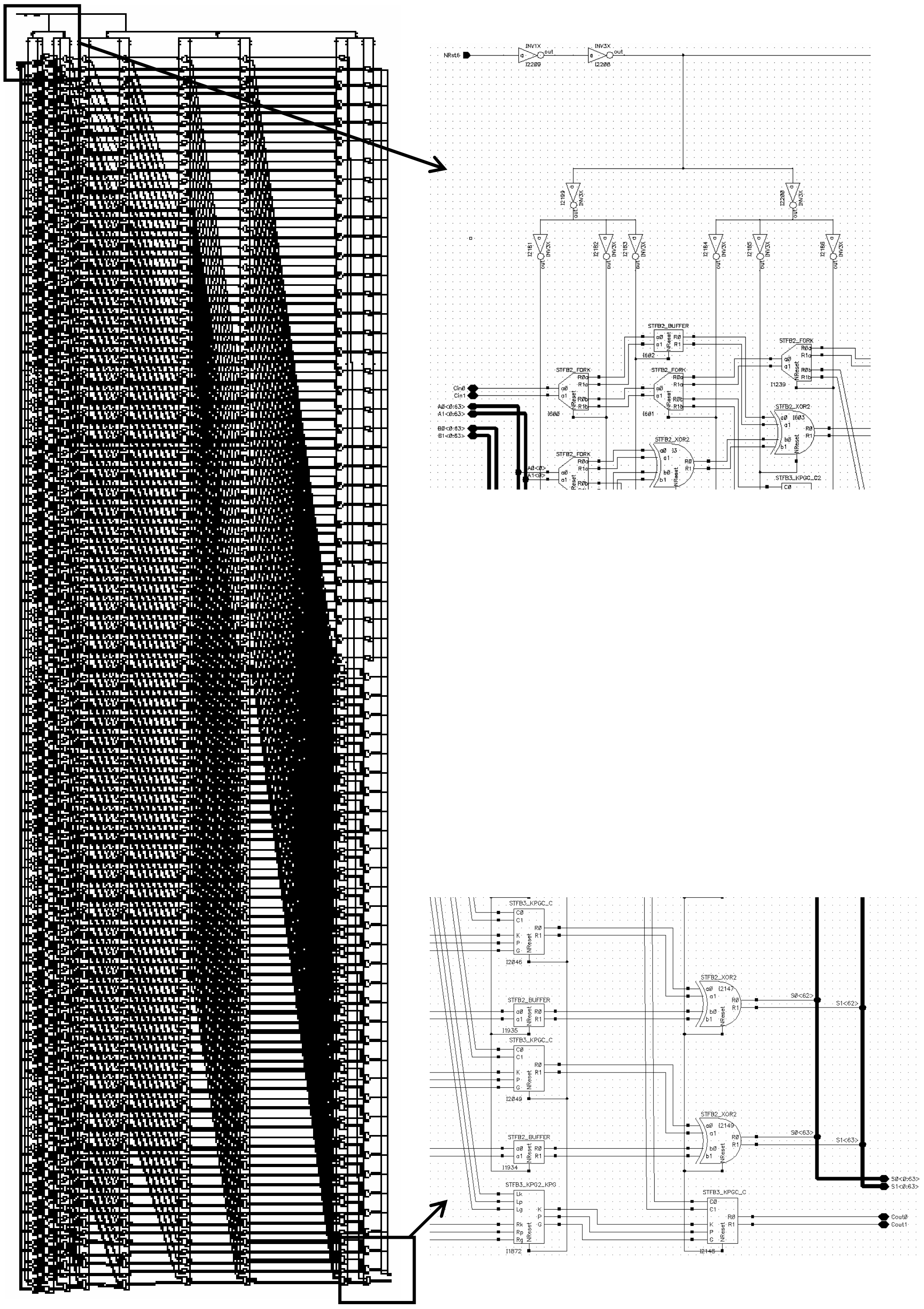


Figure 9. (ADDER64) 64-bit STFB prefix adder schematic (with input and output details).

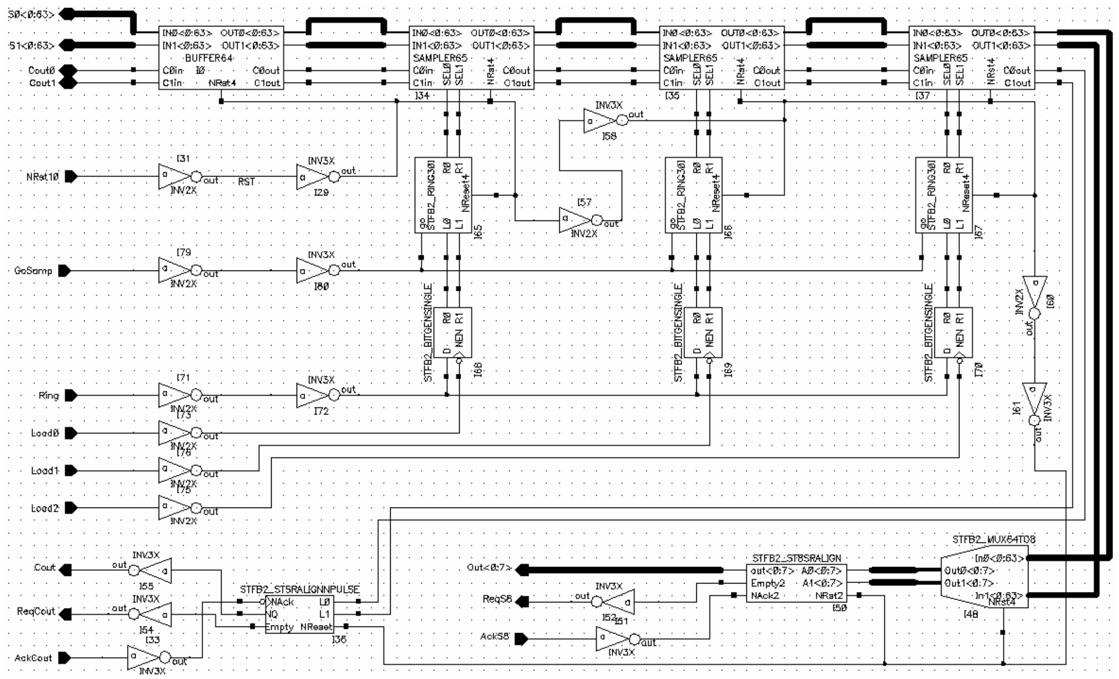


Figure 10. (SAMPLER65BY1000) Output sampler schematic.

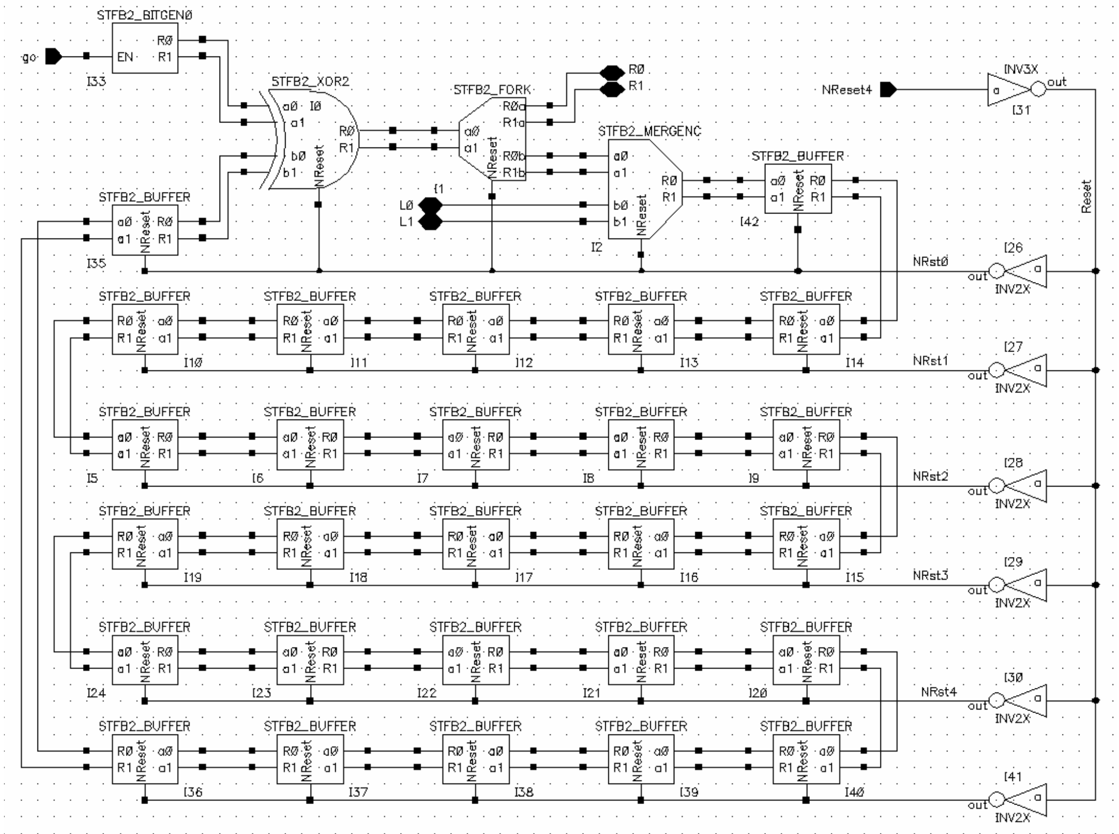


Figure 11. (STFB2_RING30) 30-stage STFB ring.

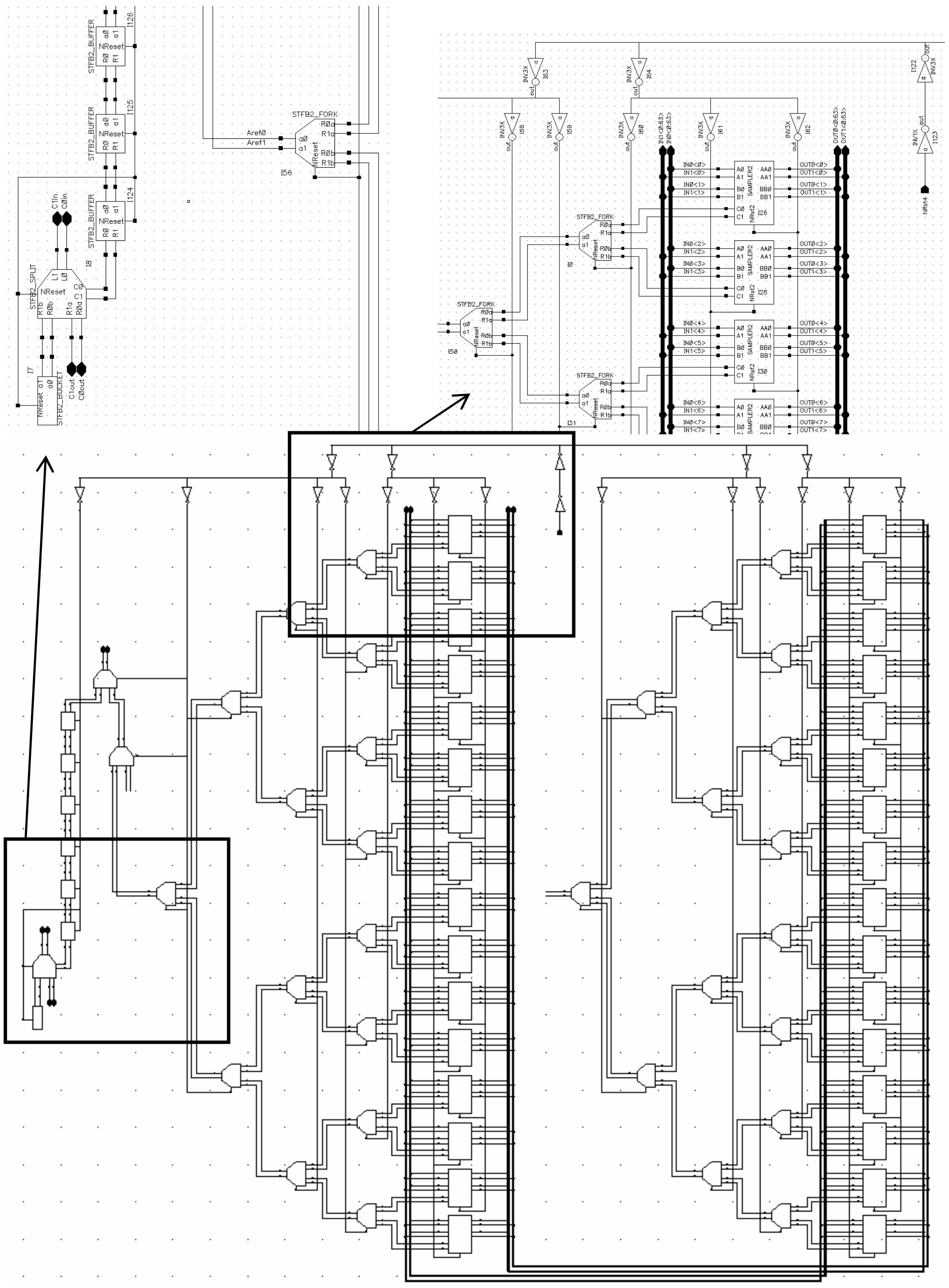


Figure 12. (SAMPLER65) 65-bit split schematic and some details.

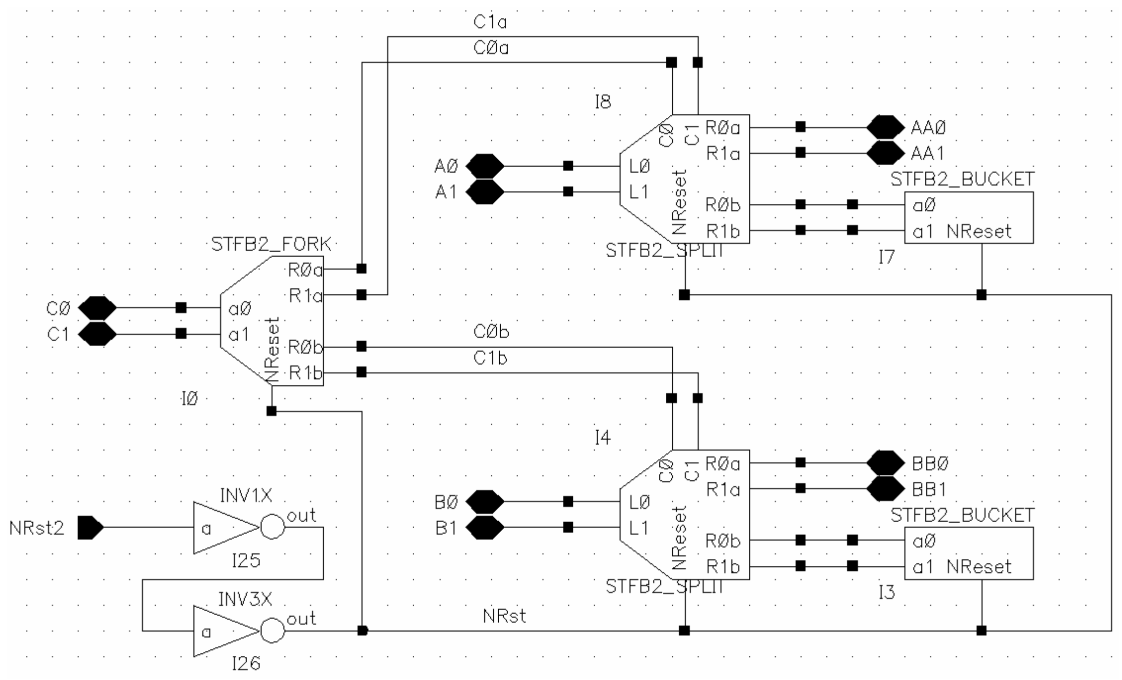


Figure 13. (STFB_SAMPLER2) 2-bit sampler schematic.

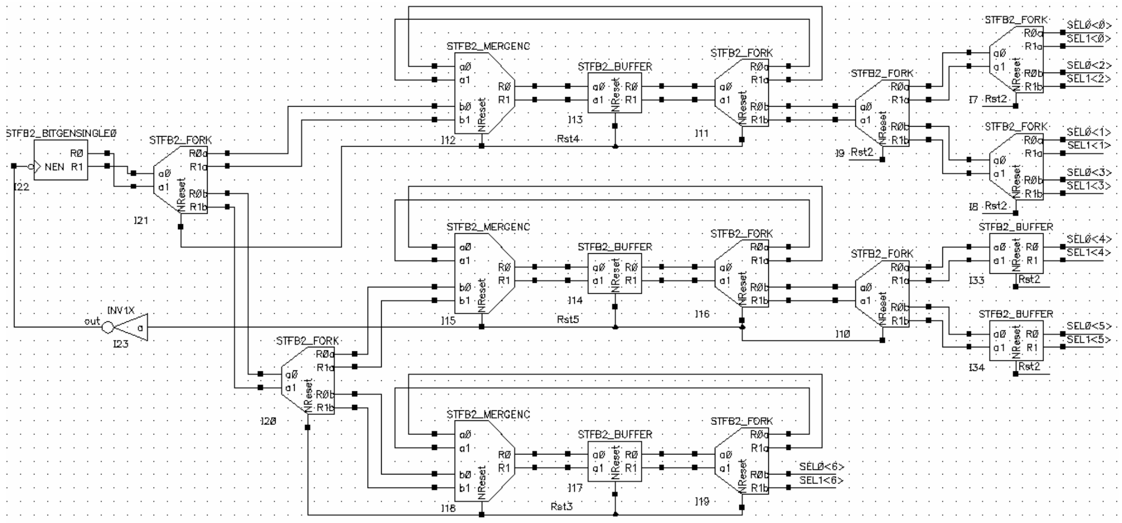


Figure 14. (STFB2_MUX64TO8) 3-bit counter implemented with self-initialized rings.

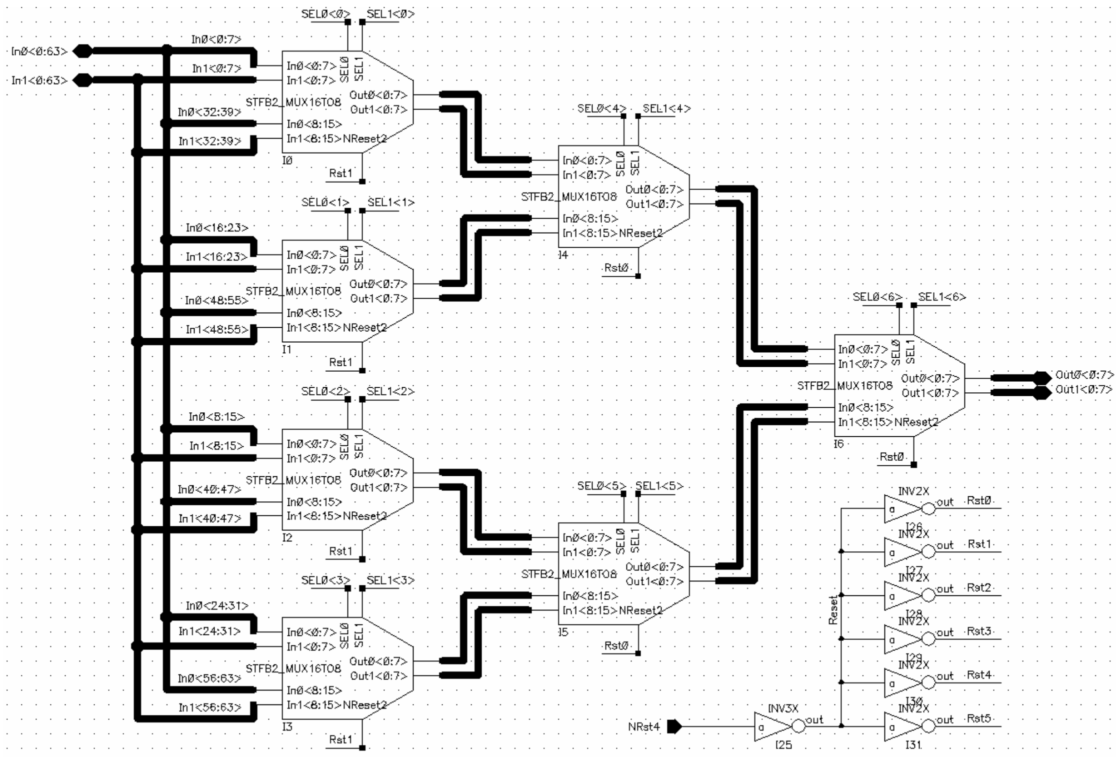


Figure 15. (STFB2_MUX64TO8) 64 to 8-bit merge tree.

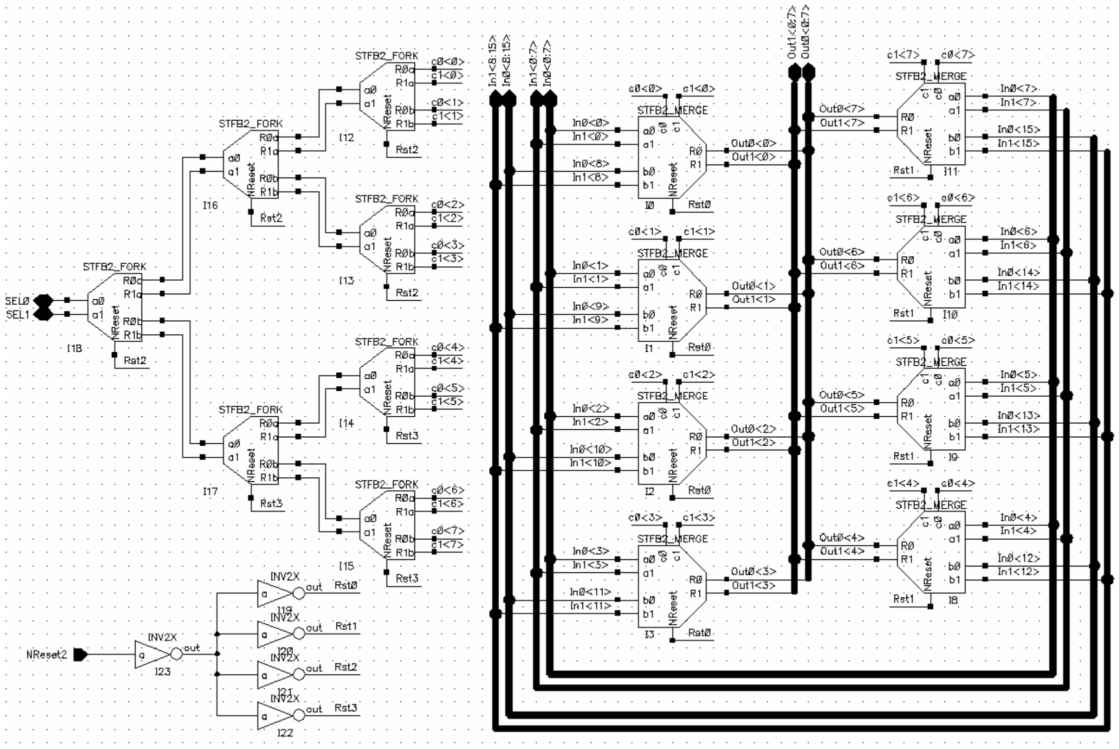


Figure 16. (STFB2_MUX16TO8) 16 to 8-bit merge schematic.

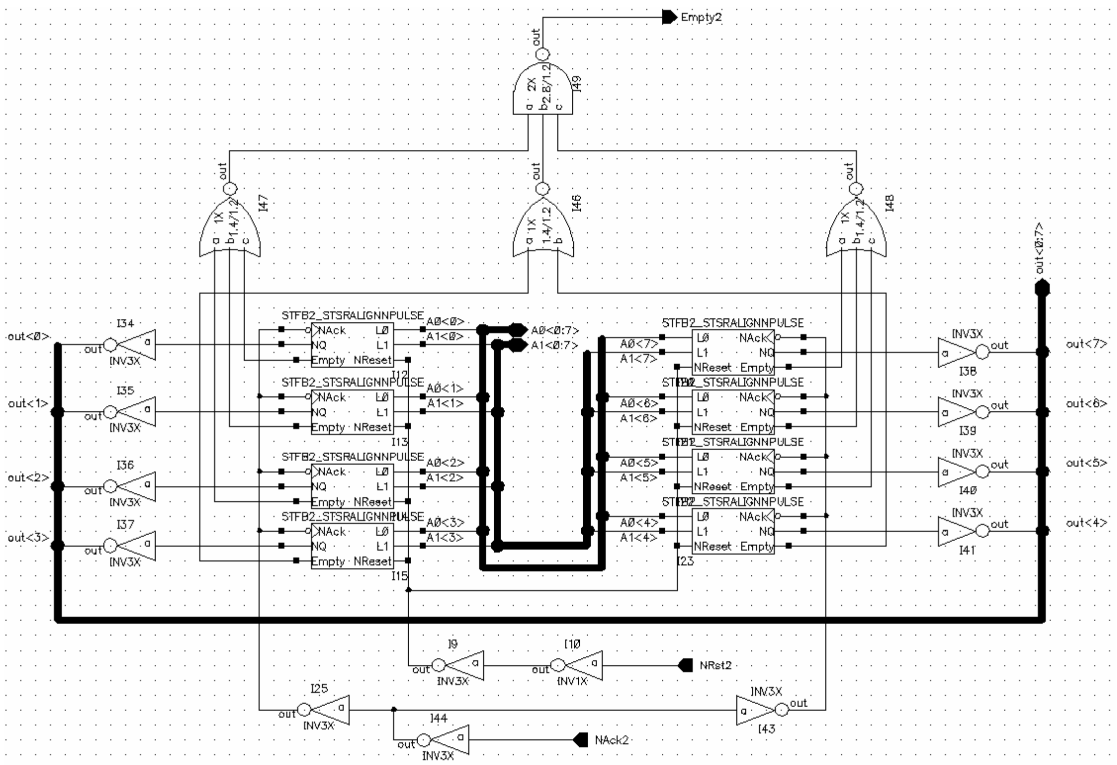


Figure 17. (STFB2_ST8SRALIGN) 8-bit single-track to single-rail conversion schematic.