Comparison of Asynchronous Design Techniques: QDI, GasP, and Single-Track

Mika Ichiki-Welches, Dimitar Dimitrov

April 2017

Abstract

In this report, we compared series quasi-delay insensitive (QDI) weak-conditioned half buffers (WCHB), a GasP full-buffer, and a single-track full buffer. We explain the designs we considered, our respective source and sink designs, transistor sizing decisions, and our results concluding the dominance of STFB design in latency and energy consumption, and of GasP in throughput.

1 Introduction

Within the relatively short lifespan of the study of asynchronous design, a number of compelling topologies have been introduced. In this report, we focus on comparing quasi-delay insensitive (QDI), GasP, and single-track designs. QDI is the only design which is "fully asynchronous." The only timing constraints in that design come from isochronic forks in the logic. The other two designs maintain a large-scale asynchronous architecture, but sacrifice delay insensitivity on a small (single-cell) scale by making simple timing assumptions which are unlikely to be broken.

We quantified these comparisons by implementing full buffers in each topology, comparing series QDI weak-conditioned half buffers (WCHB) designed from PRS, the GasP self-resetting full buffer from [1], and the optimized dual-rail single-track full buffer (STFB) from [2]. We compared these designs over latency, throughput, area, and energy, concluding that STFB was superior in latency and energy consumption, while GasP had the largest throughput of the three designs.

1.1 QDI

Quasi-delay insensitive circuits are currently the most robust asynchronous circuits; they make almost no assumptions on the delays present in wires and other components within the circuit. Instead, QDI circuits rely on handshakes being exchanged between circuit blocks in the form of acknowledgments, which serve to tell a block's environment whether or not it is available to send and receive data.

The communicating hardware processes (CHP) syntax describe such communications between blocks at a high level. The CHP for a buffer cell is *[L?x; R!x], where L is data read from the left environment of the buffer and R is the data sent to the right environment. There are a number of QDI buffers that follow this CHP using different trains of logic. As explained in [3], the QDI buffers that are most commonly used are the weak-conditioned half buffer (WCHB), pre-charge half buffer (PCHB), and pre-charge full buffer (PCFB). For the comparisons in this report, we have focused on WCHB. It being weak-conditioned means that it does not reset its output until its input has been reset. The production rule set (PRS) for WCHB is as follows, where Le and Re are acknowledgments in the reverse sense (1 is empty/ready; 0 is full) to and from the left and right environments respectively, and $L0$, $L1$, $R0$, and $R1$ are the true and false data lines from/to the left and right environments, respectively:

$Re \wedge L1 \;\; \mapsto \;\; R1\uparrow$
$\neg Re \wedge \neg L1 \wedge \neg L0 \;\; \mapsto \;\; R1\downarrow$

$Re \wedge L0 \;\; \mapsto \;\; R0\uparrow$
$\neg Re \wedge \neg L1 \wedge \neg L0 \;\; \mapsto \;\; R0\downarrow$

$\neg R0 \wedge \neg R1 \;\; \mapsto \;\; Le\uparrow$
$R0 \vee R1 \;\; \mapsto \;\; Le\downarrow$

This rule set describes the interactions between the nodes in the WCHB circuit below. In summary, when Re and the incoming data are high, the output data is set high. When Re and the incoming data are low, the output data is sent low. When Re is low and no incoming data is being received, the output data lines ($R1$ and $R0$) are sent low. Le is controlled by the output lines; when no output data is being sent, Le is high (ready) and when either $R0$ or $R1$ are being sent high, Le is low (full).

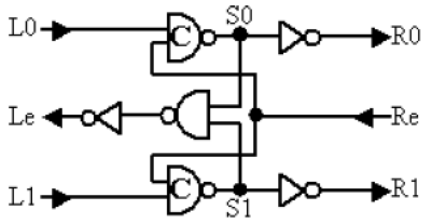

Figure 1: WCHB QDI design

Below is the CMOS-implementable version of the PRS that we translated into our WCHB circuit in LTSpice. It includes an inverted reset line called $\_rst$, which is also inverted for CMOS implementable purposes.

$Re \wedge L1 \wedge \_rst \;\; \mapsto \;\; \_R1\downarrow$
$(\neg Re \wedge \neg L1 \wedge \neg L0) \vee \neg \_rst \;\; \mapsto \;\; \_R1\uparrow$

$Re \wedge Lf_i \wedge \_rst \;\; \mapsto \;\; \_R0\downarrow$
$(\neg Re \wedge \neg L1 \wedge \neg L0) \vee \neg \_rst \;\; \mapsto \;\; \_R0\uparrow$

$\neg R0 \wedge \neg R1 \;\; \mapsto \;\; Le\uparrow$
$R0 \vee R1 \;\; \mapsto \;\; Le\downarrow$

$\_R1 \;\; \mapsto \;\; R1\downarrow$
$\_R0 \;\; \mapsto \;\; R0\downarrow$

For our purposes, we put two WCHB buffers in series to make a more direct comparison to the GasP and STFB full buffers. Our full buffer (two WCHB) schematic can be seen in Figure 2

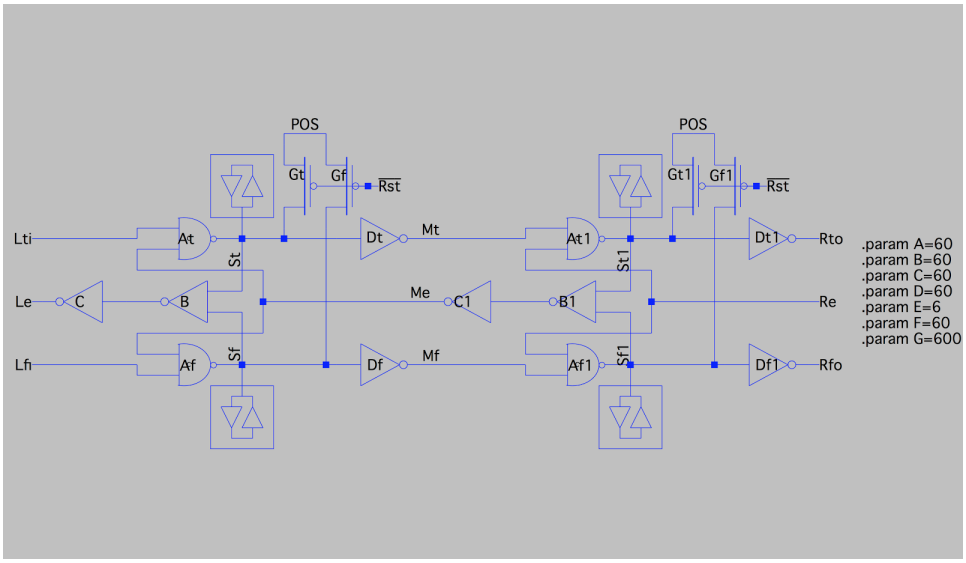

Figure 2: Our implementation of a WCHB full buffer

1.2 GasP

The GasP[1] family of circuits use careful transistor timing to create very fast cells. The GasP architecture is abstracted as a network of PATHs and PLACEs, wherein PATHs connect PLACEs. Each PLACE has a state - FULL or EMPTY. A PATH transmits information from the source PLACE to the destination PLACE when they are FULL and EMPTY respectively. It then flips their states to be EMPTY and FULL. GasP can easily be adapted for a number of binary data lines: a single circuit controls its adjacent PLACEs' states and opens a number of pass transistors to relay the data. A circuit for a simple implementation of this idea is shown in Figure 3. We implemented an 8-stage (8 PLACEs and 8 PATHs) version of this buffer. Our schematic of this buffer can be found in Figure 4.

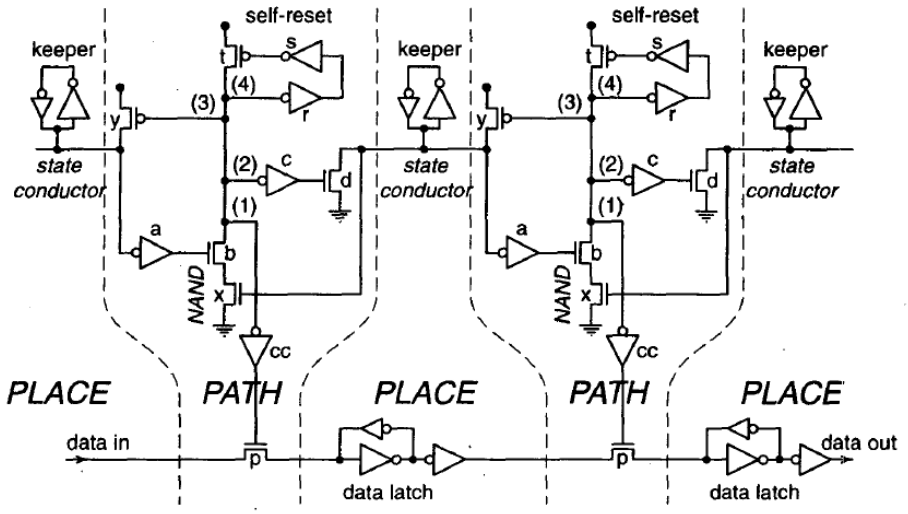

Figure 3: GasP buffer design from [1] with resetting NAND

The state conductors (which are state-holding and are therefore connected to staticizers) encode the state of each PLACE: HIGH means EMPTY and LOW means FULL. The left and right PLACE are read into

[1]a potential improvement on Molnar's "asP*" family of circuits

each path by NMOS transistors `b` and `x` respectively. Since the signal into `b` is inverted by `a`, the series conducts when the left PLACE is LOW (FULL) and the right PLACE is HIGH (EMPTY). This brings the middle rail LOW, which does 4 things (as labeled on the diagram):

1. Using inverter `cc`, it opens pass transistor(s) `p`. This allows the data held in the left PLACE to overwrite the data held in the right PLACE. The data is then stored using a staticizer. GasP is unique in that rather than moving data, it simply allows data to be overwritten (the data that used to be in the left PLACE stays there, but since its state changes, as will be shown, it may now be later overwritten).

2. Using inverter c, it turns on transistor `d` which brings the right state conductor low, setting its PLACE's value to FULL.

3. It turns on transistor `y` which brings the left state conductor high, setting its PLACE's value to EMPTY.

4. It finally triggers the self-reset sub-circuit: it pulls down inverter `r` which pulls up inverter `s` which opens transistor `t` which pulls the middle rail high. Note that the major timing assumption here is that the rail doesn't reset until every other process is finished. A relatively easy way to ensure this is by matching all delays. Since the self-reset takes three transitions, while everything else takes two, that is sufficient to ensure the correct order. Also note that all actions are self-resetting: in addition to `r`, `s` and `t`, `y`, `a` and `b` and `c`, `d` and `x` all form self-resetting loops. This, however, makes sizing somewhat difficult, as will be shown.

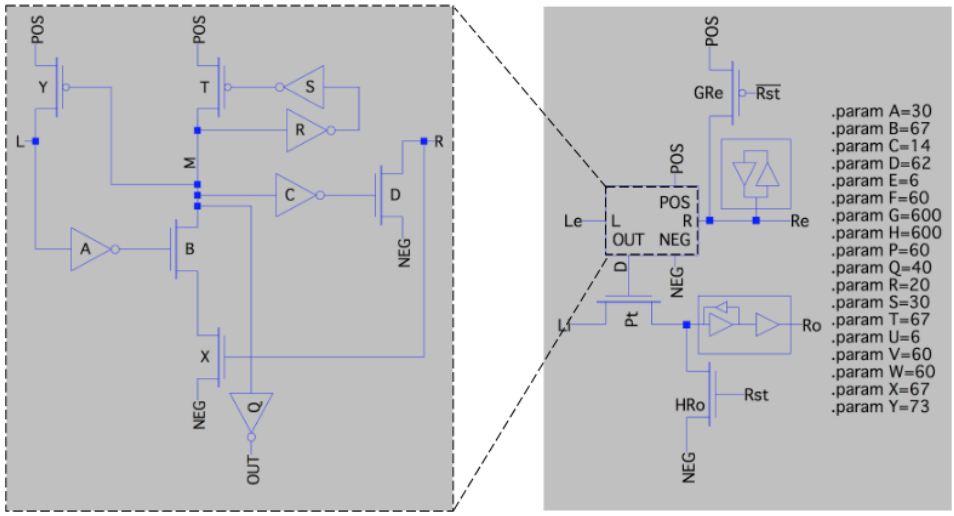

Figure 4: Our implementation of a GasP buffer

1.3 Single-Track

While both QDI and GasP use an acknowledgment line for handshaking, single-track design employs a single line for both the data transmission and the acknowledgment, thus making it single-track. In this case the right environment, in receiving the data, resets the data lines from its left environment. For example, if L0 were to go high, this would drive S0 low, in return pulling R0 high. Meanwhile, S0 having been pulled low would make the output of the NAND gate, A, go high. This serves as an acknowledgement that the input data has been received and pulls L0 back low to reset it. When R0 was sent high, it caused the NOR gate to pull B low. B acts as the inverse of a busy signal, pulling both S0 and S1 high and letting go of R0 and R1 so that they can be set again. When S0 is pulled high, the NAND gate makes A low again, which lets go of L0 and L1 so that they can take in new data.

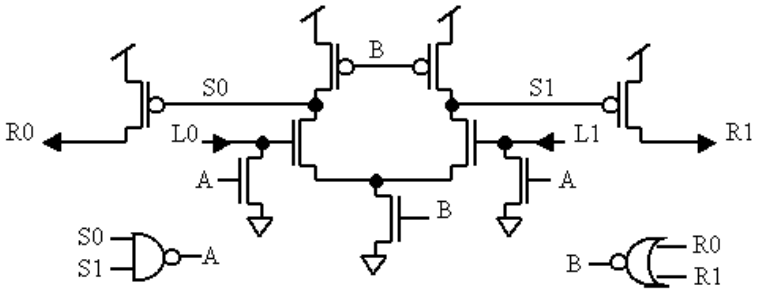

Figure 5: Optimized dual-rail single-track full buffer design from [2]

According to [2], the single-track full buffer (STFB) resulted in half the latency of GasP, and 40% faster cycle time than WCHB, which was comparable to the results we produced. A qualitative benefit of the STFB for us was that it is designed as a framework that can be very easily translated into other types of logic functions. Our STFB schematic can be seen below in Figure 6

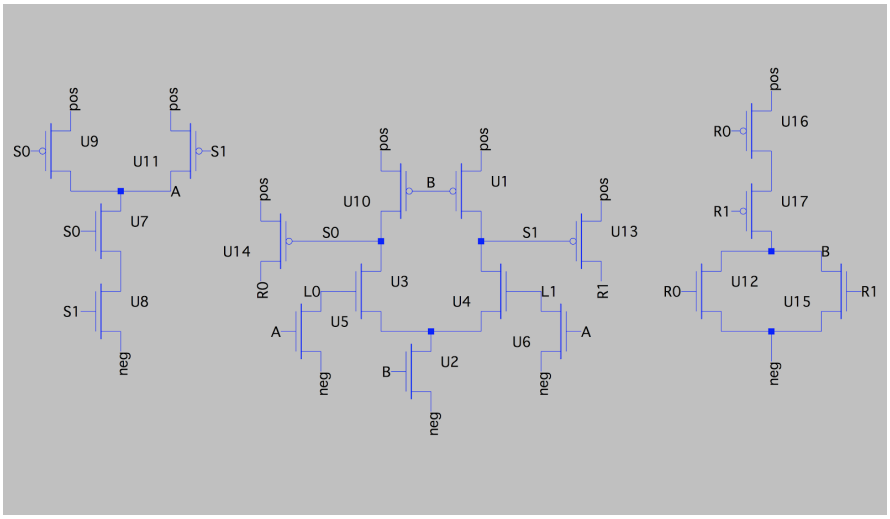

Figure 6: Our implementation of the STFB

2 Setup

For this comparison, we used C5N technology, with λ=0.3μm and a p/n ratio of 1.4.

2.1 Transistor Sizing

One of our challenges included finding proper transistor sizings for the GasP buffer, as Sutherland's sizing decisions were not made explicit in [1]. To do so, we applied the strategy used in [4], which uses Sutherland's theory of logical effort [5] and the relationships between transistors in the network to calculate their optimal sizes. Logical effort is the measure of "how many stages of logic are required for the fastest implementation of any given logic function," where the logical effort of an inverter is one, and the logical effort of any other logic gate describes its input and output driving strengths relative to the inverter. Using the logical efforts in Figure 7, we mapped the relationships between components of the GasP buffer, where L represents the load of the data latch and W represents the load of the state conductor staticizer.

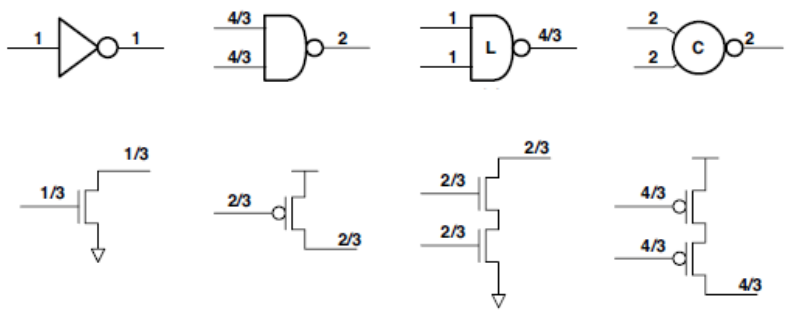

Figure 7

		1	2	3	4	5	6	7	8	9	
y	s1·x1 =	$\frac{2}{3}$·x1 +	1·x2 +		$\frac{2}{3}$·x4						+ W
a	s2·x2 =		1·x2 +		$\frac{2}{3}$·x4						
t	s3·x3 =	$\frac{2}{3}$·x1 +		$\frac{2}{3}$·x3 +		1·x5 +		1·x7 +	1·x8		
b&x	s4·x4 =	$\frac{2}{3}$·x1 +			$\frac{2}{3}$·x4 +	1·x5 +		1·x7 +	1·x8		
q	s5·x5 =					1·x5					+ L
s	s6·x6 =			$\frac{2}{3}$·x3 +			1·x6				
r	s7·x7 =						1·x6 +	1·x7			
c	s8·x8 =								1·x8 +	$\frac{1}{3}$·x9	
d	s9·x9 =		1·x2 +		$\frac{2}{3}$·x4 +					$\frac{1}{3}$·x9	+ W

We then synthesized this chart into two matrices, T and b, where T held the logical efforts described by the nine components' relationships with each other and b represented the loads, L and W, where we've set both to be 60 λ:

$$T = \begin{bmatrix} \frac{2}{3} & 1 & 0 & \frac{2}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{2}{3} & 0 & 0 & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{2}{3} & 0 & 1 & 0 & 1 & 1 & 0 \\ \frac{2}{3} & 0 & 0 & \frac{2}{3} & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{2}{3} & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{1}{3} \\ 0 & 1 & 0 & \frac{2}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} \end{bmatrix} \quad (1)$$

$$b = \begin{bmatrix} 60 \\ 0 \\ 0 \\ 0 \\ 60 \\ 0 \\ 0 \\ 0 \\ 60 \end{bmatrix} \quad (2)$$

If we have a diagonal matrix S in which we define the delays of each component, from $s0$ to $s9$, we can solve the equation below for x, where x is the matrix of transistor sizes in λ.

$$S * x = T * x + b \quad (3)$$

We defined our s values to all be the same, and weighed them by our drive strength ratios, 2.5. Our resulting x array, rounded to the nearest lambda value, was:

$$x = \begin{bmatrix} 73 \\ 30 \\ 67 \\ 67 \\ 40 \\ 30 \\ 20 \\ 14 \\ 62 \end{bmatrix} \tag{4}$$

According to [5], counter leakage and loading from the reverse inverter in the staticizer could be minimized by making it minimum sized, so we sized our staticizers accordingly, with the forward inverters sized at 10x the their reverse counterparts.

2.2 Source and Sink

We included source and sink circuits to feed and receive data from each circuit. For WCHB, we used a two-inverter buffer as a source (8) and a NOR gate as the sink (9), which takes in the true and false output lines R0 and R1 from the last buffer and returns an acknowledgement Re. This is much faster than a cell, so we know that neither is a bottleneck for throughput.

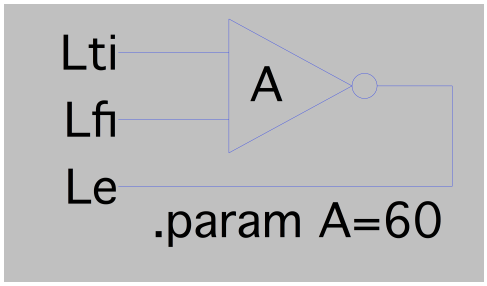

Figure 8

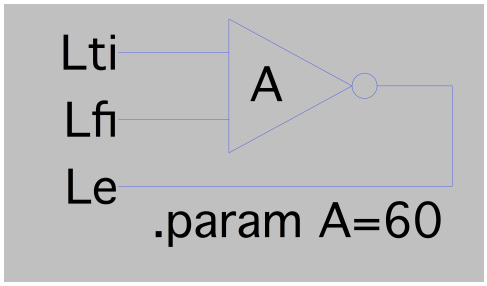

Figure 9

For GasP, we designed a source and sink very similar to the GasP buffer design that we used. We sized them the same way, so we believe they are not the bottlenecks for throughput.

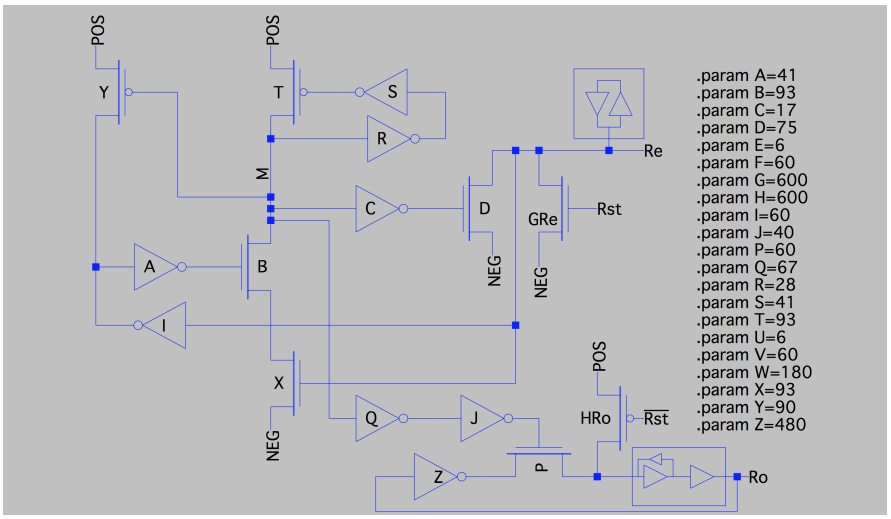

Figure 10

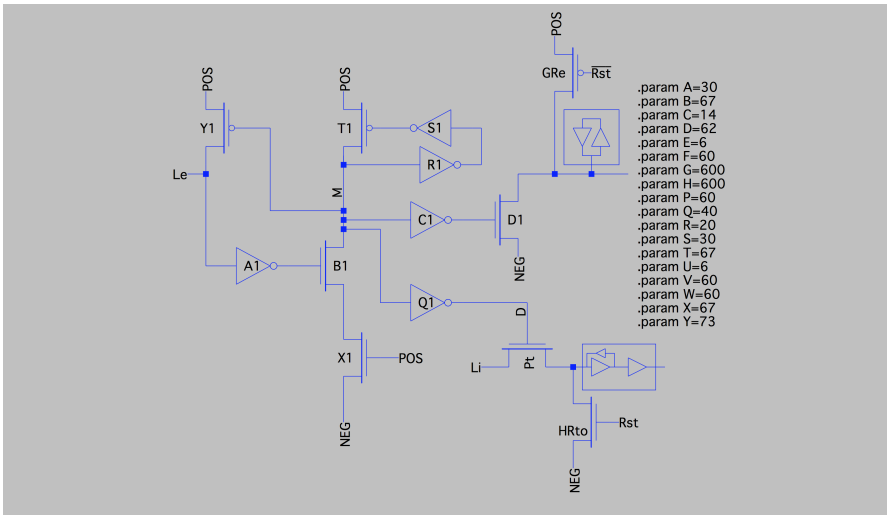

Figure 11

For STFB, we used the single-track transmit (Tx) and receive (Rx) circuits from [2] which connected our WCHB sink and source respectively to our 8-stage buffer chain. Since the WCHB cells are maximally fast, and the Tx and Rx cells are STFB cells, we believe that they are also not bottlenecks for throughput.

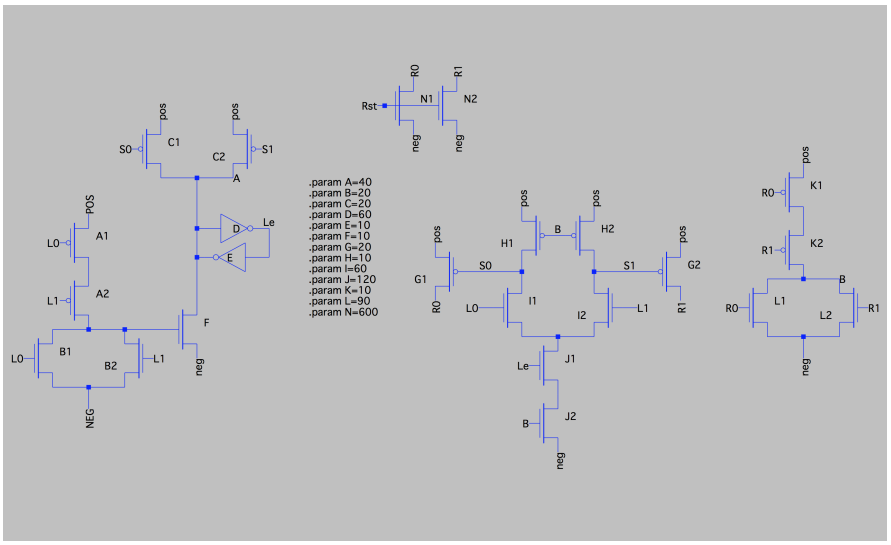

Figure 12

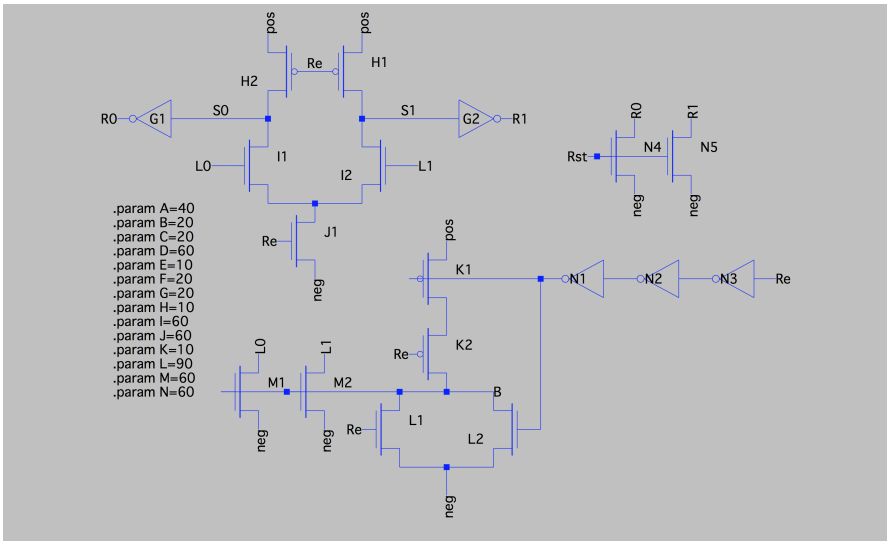

Figure 13

3 Results

Waveforms of middle three acknowledge lines of the buffer chain can be seen in figure 14. The top graph shows WCHB, the middle GasP and the bottom STFB.

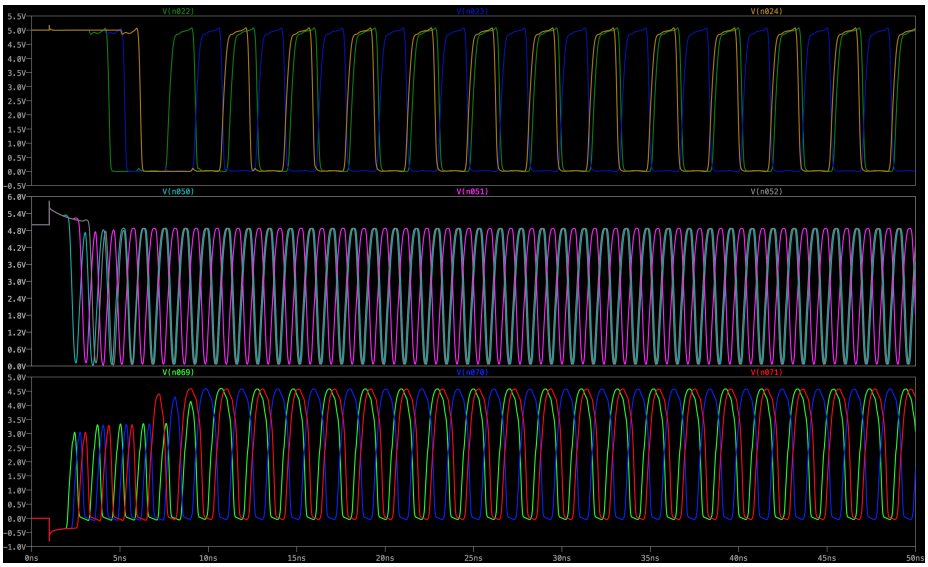

Figure 14

These results are summarized below:

	Throughput	Latency	Area	Energy
WCHB	296Mb/s	783ps	$580\mu m^2$	107pJ
GasP	844Mb/s	562ps	$222\mu m^2$	99.3pJ
STFB	754Mb/s	280ps	$246\mu m^2$	21.4pJ

Energy per transition is calculated as total power divided by the product of the number of stages and the throughput, where total power for WCHB over 8 stages (assuming the power consumption of the source and sink is negligible) was 254mW, the total power for GasP over what we considered 10 stages (8 buffer stages, as well as a GasP source and sink) was 419mW, and total power for STFB over 10 stages (8 buffer stages, Tx, and Rx) was 161mW. Note that the numbers for WCHB describe one full-buffer stage, or two WCHBs.

In terms of latency and energy consumed, STFB proved to be the better design, and its size was comparable to that of GasP, which yielded the fastest throughput. Note that our design for the STFB does not include staticizers, as [2] was not clear on how they sized these components. Despite this bias in our design, we believe that adding them would not affect STFB's relative performance to QDI and GasP.

References

[1] I Sutherland and S Fairbanks. GasP: a minimal FIFO control. In *Asynchronous Circuits and Systems, 2001. ASYNC 2001. Seventh International Symposium on Asynchronous Circuits and Systems, 2001*, pages 46–53, February 2001.

[2] M. Ferretti and P. A. Beerel. Single-track asynchronous pipeline templates using 1-of-n encoding. In *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, pages 1008–1015, 2002.

[3] A. M Lines. Pipelined asynchronous circuits. Technical report, Pasadena, CA, USA, 1998.

[4] J. Ebergen, J. Gainsley, and P. Cunningham. Transistor sizing: how to control the speed and energy consumption of a circuit. In *10th International Symposium on Asynchronous Circuits and Systems, 2004. Proceedings.*, pages 51–61, Apr 2004.

[5] Ivan Sutherland, Bob Sproull, and David Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.