Title: Implementation and Evaluation of a Pipelined Synchronous 8-Bit Carry Select Adder

Author: Aritro Sarkar, MS in Computer Engineering, Stony Brook University, Stony Brook, NY 11794.

I. ABSTRACT

Multi-bit arithmetic operations are widely used and necessary for a variety of digital and analog applications. However, using purely ripple carry adders create greater delay for arithmetic operations to complete. This paper presents the design and evaluation of a **Pipelined Synchronous 8-bit Carry Select Adder (CSLA)**. A CSLA enables faster arithmetic operations by using parallel computation to reduce propagation delay, and performance can be increased by leveraging pipelining. The CSLA was implemented in 45nm technology using the open cell library FreePDK45. The design was implemented using 4-bit ripple carry adders, constructed from single bit adders, and an array of 2:1 multiplexers were used to select the carry signal. The design is pipelined at the primary inputs and outputs to enable higher throughput and clock frequencies. We then validate the performance of our CSLA which can run at 4GHz and consumes 0.96mW of power.

II. INTRODUCTION

There are various adder topologies, where the use between them is dependent on the specific application. In this section, we will cover the design and benefits for multiple topologies.

One of the most common designs is the **Ripple Carry Adder (RCA)**. A RCA contains a series of Full Adders (FA) with each FA used to add two bits along with a carry bit. The carry signal propagates sequentially from one full adder to the next, extending to the N-th FA, where N is the total number of bits. However, delay increases as the number of bits do, as the critical path is greater due to this sequential behavior.

The **Carry Skip Adder (CSKA)** improves upon the Ripple Carry Adder (RCA) by incorporating skip logic to accelerate carry propagation. Instead of allowing the carry to ripple through every full adder, the CSKA computes group propagate signals for blocks of adders. When all bits within a block are in the propagate condition, the incoming carry is allowed to skip the entire block. This behavior shortens the critical path and significantly reduces the overall addition delay compared to the RCA.

In a **Carry Save Adder (CSA)**, three bits are added in parallel at each stage. Unlike conventional adders, the carry is not propagated immediately to the next stage. Instead, the carry is saved locally and passed forward as an additional addend to the subsequent stage. By avoiding carry propagation across stages, this architecture significantly reduces carry-related delay and improves overall computation speed.

For a **Carry Bypass Adder (CBA)**, the adder is partitioned into multiple blocks, each implemented using a ripple-carry adder (RCA). For each block, a bypass (or propagate) signal is generated by combining the bit-level propagate signals within that block, similar to the propagate logic used in carry lookahead adders (CLA). A multiplexer uses this bypass signal to determine whether the carry-in should bypass the block directly or whether the carry-out generated by the RCA should be forwarded to the next block. Based on the bypass condition and the incoming carry, the appropriate carry is propagated to the subsequent stage.

III. CARRY SELECT ADDER DESIGN

A. Block Level Architecture

The eight bit carry select adder is divided into three four bit RCA blocks. The lower bits (a3-a0) and (b3 - b0) are applied to the first four bit adder to generate the sum outputs (s3- s0) along with the carry out (cout). The upper bits (a7-a4) and (b7 - b4) are applied in parallel to the two four bit adders. One of these adders computes the sum assuming a carry in of zero while the other computes the sum assuming a carry in of one. The outputs of these two adders are connected to an array of 2:1 multiplexers. The carry out from the first adder is used as the select signal for the multiplexers. When c4 is low the outputs from the adder with carry in zero are selected and passed to the final output. When c4 is high the outputs from the adder with carry in one are selected. The final carry out c8 is also selected through the multiplexer stage.

To increase the operating frequency of the CSLA, the design is pipelined by inserting synchronous reset D flip-flops (DFF) at the primary inputs and outputs. The critical path starts at the output of the input-stage flip-flop driving the carry-in signal, propagates through the least significant 4-bit ripple-carry adder to generate the carry-select signal, and then passes through the multiplexer selection logic that resolves the most significant sum bit (S7), and finally to the input of the output-stage flip-flop.The block level architecture of our CSLA is shown in Figure 1.

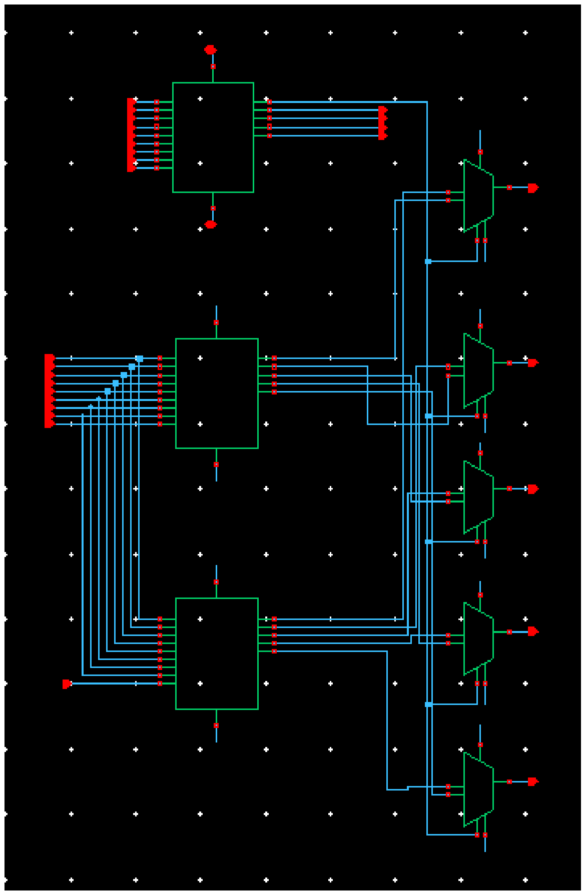
Figure 1: CSLA Block Level Architecture

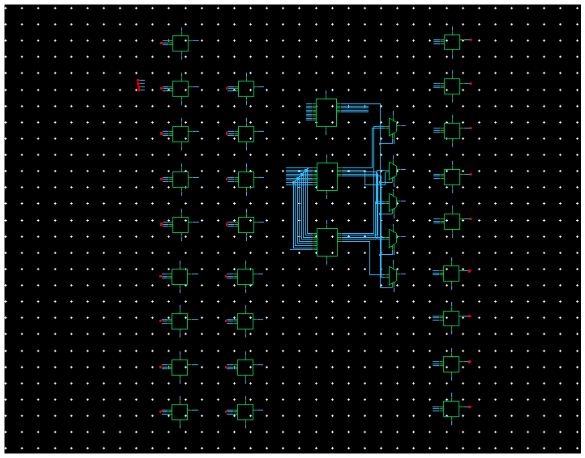
Figure 2: Pipelined CSLA Architecture

B. Flip-Flop

The flip-flop used in this design is a synchronous, transmission-gate–based D flip-flop implemented in a master–slave configuration. The input data D is first gated by the reset logic and then captured by the master latch when the clock is active. This master latch is built using CMOS transmission gates (TG) controlled by complementary clock phases, allowing it to be transparent during one phase of the clock while isolating and holding its value during the opposite phase. The slave latch, driven by the opposite clock phase, captures the master latch output and drives the final output Q through inverters. This two-phase operation ensures edge-triggered behavior, since data is only transferred from input to output on the clock transition.

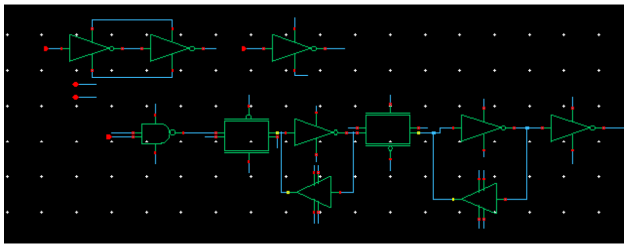

Figure 3: Synchronous D Flip-Flop

C. 1-Bit Adder

The 1-bit adder used in this design is a mirror full adder implemented at the transistor level using complementary CMOS logic. The circuit generates the carry output according to Cout = AB + (A + B)Cin using a fully symmetrical pull-up and pull-down network which makes the logic function self-dual and results in balanced rise and fall delays. A key advantage of this adder is that the carry generation path contains at most two series transistors, reducing worst-case delay which is great for faster arithmetic operations. The sum output is derived from the carry signal and the input operands, and is buffered with an inverter to provide full rail-to-rail swing and sufficient drive strength. It also uses less transistors than other configurations, allowing for less power and area consumption.

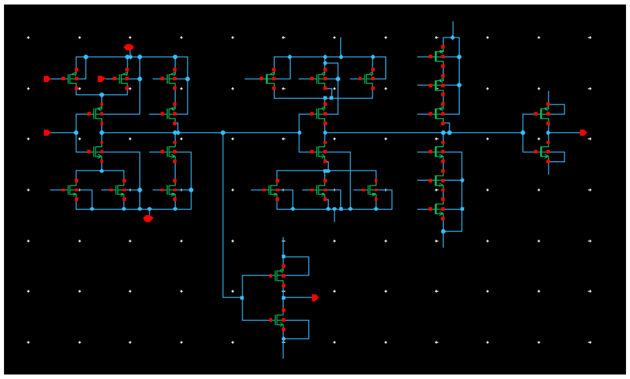

Figure 4: Mirror Full Adder

D. 2:1 Multiplexor

The 2:1 multiplexor used in this design is implemented using CMOS TGs controlled by complementary select signals. The two data inputs, A and B , are each connected to the output through a TG. The select signal S directly controls one TG, while its complement S-bar, generated by an inverter, controls the other, ensuring that only one input is connected to the output at any time. When S=1, the TG associated with A is enabled and B is disabled, but when S=0, the TG associated with B is enabled and A is disabled. Using TG allows the multiplexor to pass both logic 0 and logic 1 levels without threshold voltage degradation, providing full rail-to-rail output swing. The symmetric structure results in balanced delay for both input paths allowing this multiplexor, and utilizes less transistors, allowing it to be a good choice for large designs.

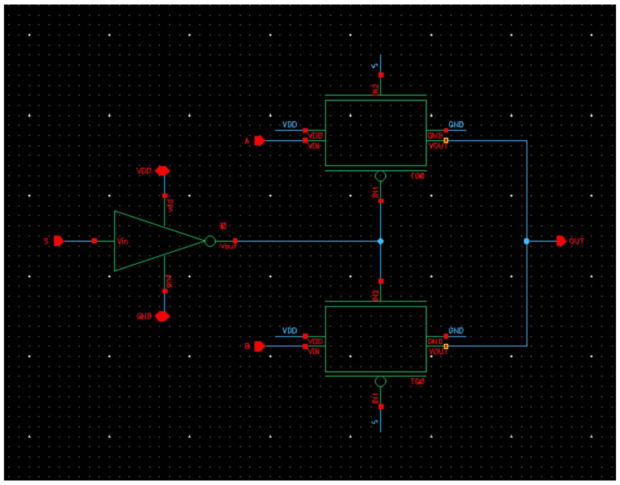

Figure 5: TG-Based 2:1 Multiplexor

IV. SIMULATION RESULTS

Figures 6-15 are the generated waveforms via simulation of the pipelined CSLA executed in Cadence Virtuoso. Due to pipelining at the primary inputs and outputs, and delays from clk2q, setup, and combinational time, the output SUM is delayed. However the CSLA functions as intended and thanks to pipelining, our CSLA can run at higher clock frequencies.

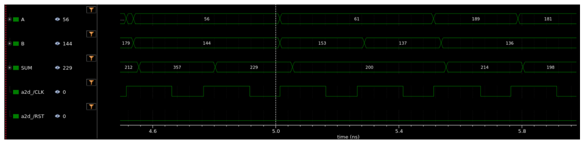

Figure 6: Simulation Time Stamp at 5ns

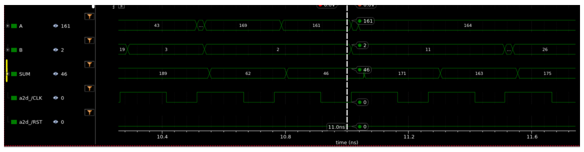

Figure 7: Simulation Time Stamp at 11ns

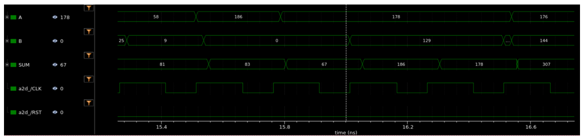

Figure 8: Simulation Time Stamp at 16ns

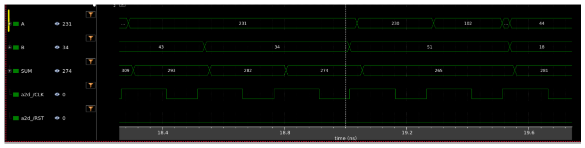

Figure 9: Simulation Time Stamp at 19ns

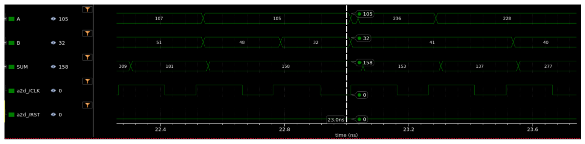

Figure 10: Simulation Time Stamp at 23ns

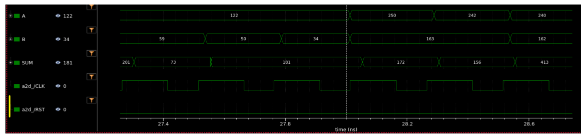

Figure 11: Simulation Time Stamp at 28ns

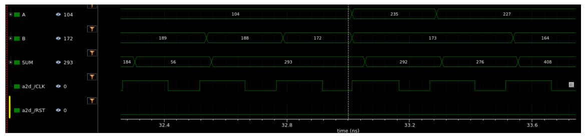

Figure 12: Simulation Time Stamp at 33ns

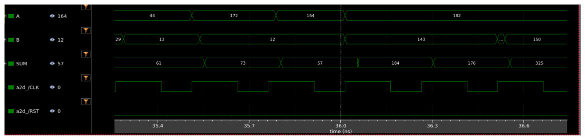

Figure 13: Simulation Time Stamp at 36ns

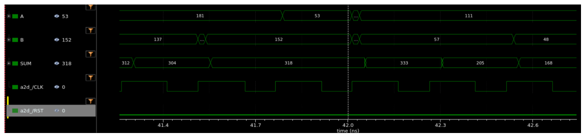

Figure 14: Simulation Time Stamp at 42ns

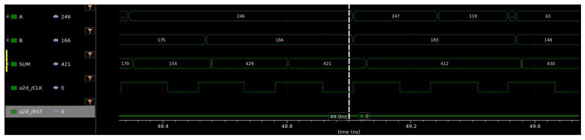

Figure 15: Simulation Time Stamp at 49ns

As shown in each figure, the actual output Sum is correct in each instance, but is simply delayed by approximately 2 clock cycles.

V. DISCUSSION

Pipelining this design allows our CSLA to run at faster frequencies. However, we are still limited by dependencies such as setup and hold time, propagation delay, and combinational delay. Although easy to miss, the design begins to have tiny glitches when it runs at 8GHz.

Clock Frequency (GHz)	Average Power (mW)
4	0.96
6	1.279
8	1.601

Figure 16: Clock Frequency vs Avg. Power

It is interesting to note that the ratio of clock frequency to power increases as we increase the speed of our circuit. Overall, thanks to the use of transmission gate based designs and the use of a mirror full adder, we were able to limit the transistors and area used allowing for less average power consumption.

VI. CONCLUSION

In this paper, we explored the implementation and evaluation of a pipelined synchronous 8-bit CSLA in 45nm CMOS technology. The design is validated for functionality using Cadence Virtuoso for waveform simulation, and our design works with a high operating frequency of 4GHz and a minimal power of 0.96mW. However, this design can be further improved upon. The redundant adder paths in the carry select structure can be reduced by replacing selected stages with carry bypass logic, allowing the carry to skip blocks when all bits propagate. We can also internally pipeline our circuit further, but it is important to balance it so that we do not overuse resources which would result in a larger, unnecessary hardware overhead. We could also optimize the transistor sizing which would improve speed on critical paths while reducing power and area on non-critical paths.