EE 476 Final Project: 16-bit Accumulator with BIST

Kendrick Tang
School of Electrical Engineering
University of Washington
Email: tangkend@u.washington.edu

Cameron Forbis
School of Electrical Engineering
University of Washington
Email: forbisc@u.washington.edu

Sung Kim
School of Electrical Engineering
University of Washington
Email: sungk9@u.washington.edu

***Abstract*—An accumulator is used to save a running sum of inputs, which can be used in multiplication or signal processing. Due to the complexity of the design, it is important that an accumulator be designed with a method for verification. This paper presents a design for an accumulator which accumulates 16-bit sign-magnitude words and outputs a 22-bit 2's complement word. The accumulator is implemented with a 22-bit carry lookahead adder and a built-in-self-test (BIST), consisting of a 16-bit Galois configured LFSR and a 22-bit signature analyzer.**

I. Introduction

The high-level function of an accumulator is to sum a series of inputs. The input specified for this design is a 16-bit word in sign-magnitude format. The output specified for this design is a 22-bit word in 2's complement format. As a result, the accumulator must be capable of converting from sign-magnitude to 2's complement. As opposed to indefinitely keeping a running sum of the inputs, the accumulator is only sums every 64 inputs, and only changes its output at the end of every 64 adds. For the purpose, a 6-bit adder is used to trigger the change in the output every 64 cycles. To enable state-saving behavior, two registers are used to save the running sum and the output.

To verify this functionality, a built-in-self-test (BIST) is integrated into the design of the accumulator. The BIST includes a Galois configured 16-bit LFSR and a signature analyzer. To be able to switch between user-defined inputs and LFSR-defined inputs, as well as be able to switch between monitoring the signature analyzer and monitoring the accumulated sum, a set of MUXs is used at the input and at the output.

The high-level functionality is summarized in Fig. 1, and the layout is presented in Fig. 4.

II. Functionality and Implementation

The accumulator supports regular operation, or 'test mode' with BIST. Regular operation begins with a reset signal, after which the accumulator consumes a 16-bit word from the user input every clock cycle. The assumption is made that the user supplies data in sign-magnitude format, with the accumulator performing sign conversion to 2's complement. At every 64th clock cycle, the 22-bit running sum is clocked to the output pins by a register after which the running sum is reset. In test mode, a MUX that selects the input stream to the accumulator switches from user input to a pseudo-random stream of data generated by a 16-bit LFSR. The output of the accumulator is mixed into a signature analyzer every cycle.

A. Adder Implementation

For the carry-lookahead (CLA) design, the delay from the propagation of the carry signal is mitigated by pre-computing the propagate-generate (PG) terms in parallel without knowledge of the group carry-in. This can be done because the group generate and propagate terms are determined only by the input summands. Hence, we implement a compound PG gate, based on the PG equations. Let a_i and b_i be input bits to the carry-lookahead logic, then

$$\begin{aligned} p_i &= a + b, && \text{bit propagate} \\ P_i &= p_i p_{i-1} p_{i-2} ... p_0, && \text{group propagate} \\ g_i &= ab, && \text{bit generate} \\ G_i &= g_i + g_{i-1} P_i && \text{group generate.} \end{aligned}$$

The group carry-out c_i at any stage i can be defined as $c_i = G_i + P_i \cdot c_{\text{in}}$ where c_{in} is the group carry-in. A simple AOI is used to implement this function. Due to the ability to pre-compute the group propagates and generates, and the speed of computing group carry-outs with an AOI, the CLA design is much faster than the simple ripple-carry design.

For the sake of efficient and scaleable layout, we used a 3-bit ripple-carry adder for each adder stage. We implemented the p_i's and g_i's as small 2-input NANDs and NORs, in groups of 3-bits. These were used as inputs to larger gates that computed

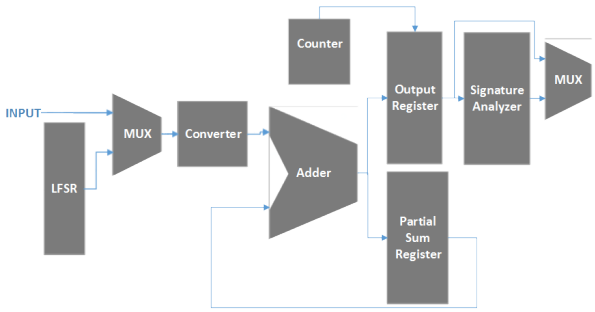

Fig. 1: High-level functionality of 16-to-22 bit accumulator with BIST

G_i and P_i for the 3-bit group. The 22nd bit was summed with a single full adder.

B. Counter Implementation

A 6-bit ripple down counter is used to trigger a change in the register storing the 64-cycle accumulated sum. Since we only care about the trigger state (zero), and the transition to the trigger state, we decided to use an asynchronous counter; the fact that the counter does not follow the clock doesn't really matter, as long as the trigger state is synchronous. To ensure the trigger state is synchronous, the system clock just can't be so fast that the asynchronous count is completely off at the 64th count, which isn't too strict of a requirement. We decided to count down because we want the trigger to occur very quickly. Up counting would mean the trigger state would follow the 63rd state, which is the slowest transition in the ripple counter. Down counting allows us to trigger on a zero state following the first state, which is the fastest transition.

There was an off-by-one issue involving the initial state of the counter that was solved by having a non-zero reset state. We chose to initialize the counter at 1, so that it immediately goes into the trigger state after the initial rising clock edge. With this implementation, the counter successfully triggers the output register to save the correct sum.

III. Results

A. Functional Verification

The BIST was used to verify the functionality of the accumulator. Using an LFSR in a Galois configuration, pseudo-randomized input was summed by the accumulator. The primitive polynomial used for the 16-bit LFSR is:

$$x^{16} + x^{14} + x^{13} + x^{12} + 1.$$

The LFSR provided an input sequence that looks random, but is completely deterministic. Using the deterministic input sequence, MATLAB was used to simulate the expected circuit behavior. Specifically, the output of the signature analyzer was simulated and a signature was recorded at every $n \times 64$-th clock cycle, for small n. Using HSpice and Silicon Explorer, the signature produced by the Spice-simulated accumulator schematic was compared to the MATLAB simulated signature, and the schematic was verified.

Bin's script was also used, and after some modification for timing, our design was able to match the expected output provided.

B. Post Layout Analysis

After verifying the schematic and passing DRC and LVS in Cadence Virtuoso, the layout was analyzed for timing and energy consumption.

When planning the layout, we decided, regretfully, to have the data flow perpendicular to the power tracks. The intention was to have each bit line up with every stage to help us save on metal, but this was not the case. The flip flops were a lot wider than we had expected, and the adder turned out to be much more compact. As a result, we ended up using lots of metal tracks parallel to the power tracks to get signals to line up with their target. Despite our extensive use of metal two, the effects of parasitics and non-idealities only increased the delay by 350 ps. This was much less than we had expected. Section III-C discusses the worst case in more detail.

Post-layout, the average power consumption per cycle was measured in Silicon Explorer to be 1.85 mW (Fig. 2).

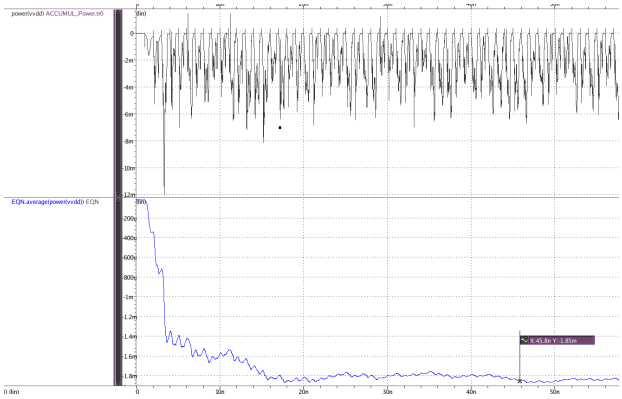

Fig. 2: Power consumption analysis

C. Critical Path

Like most other adders, our adder's worst case involves adding negative one in 2's complement to zero. This is actually implemented by adding negative one in 1's complement to zero with a carry-in. The carry-in has to ripple through all six AOIs and the last adder, which is the critical path.

In addition to the adder's worst case, there is also an asynchronous reset that occurs which will slow down the system. After the rising edge of 64th clock cycle, the zero state of the counter is detected and is used to zero out the feedback path of the adder. This means the add is invalid until after the input to the adder is fully zeroed out. As a result, we have decided that the upper bound for our clock time should be the sum of the reset time and the longest path of the adder. In Silicon Explorer, we measured the delay from clock to the reset of the input as 328 ps (Fig. 3). The same figure, Fig. 3 shows the worst case delay from the a *change* in the input of the adder to a valid output. The critical path of the accumulator really involves the asynchronous reset time and the worst case delay from a *valid* input of the adder to a valid output. This turns out to be close to 840 ps.

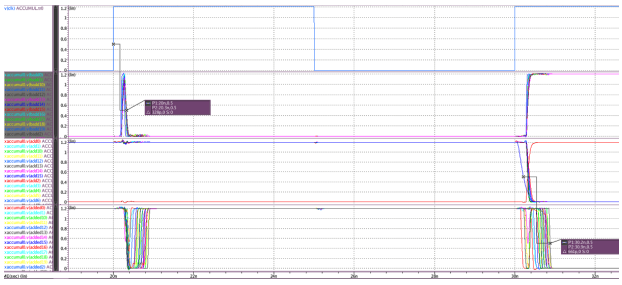

Fig. 3: Timing analysis

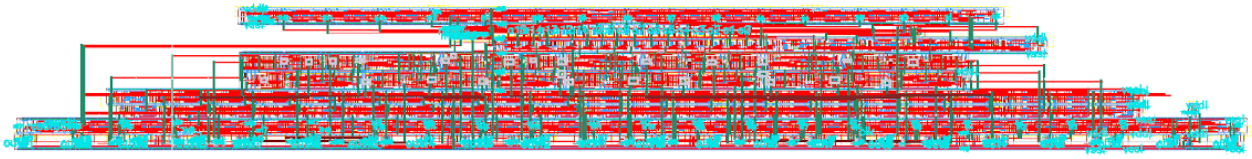

Fig. 4: Accumulator Layout

IV. Conclusion

In this paper we describe the design, implementation and performance of a 16-bit accumulator with built-in self test (BIST). The accumulator operates either in 'normal' mode, taking arbitrary 16-bit user input, or in test mode, where a Galois linear feedback shift register (LFSR) feeds the accumulator a stream of pseudo-random data. During test mode, data generated by the LFSR is fed to a signature analyzer which generates a new signature every 64-cycles. The 22-bit adder architecture included a combination of ripple-carry and carry-lookahead, where the propagate-generate terms were computed in 3-bit groups. After schematic design and layout, the design was functionally verified by comparing the output produced by the circuit with a golden signature, pre-computed in Matlab.

After timing and power analysis, we reflect on our sins. We emphasize the importance of floor-planning, and having metal routing plans before layout begins. Despite the use of tracks, routing without a high-level flow plan resulted in routing congestion, large spacing between cells, and unnecessary empty spaces in the layout. Improvements on the design could include alternative adder architectures, such as log-based/tree adders. As we have mentioned before, the implementation could be greatly improved by a pre-layout routing and placement plan.