James Kennedy
Carl Gerisch
Xiwen Wei

Performance Results for Programmable Adder

Refer to Appendix A for a schematic of our speed-optimized programmable adder with transistor sizings. Refer to Appendix B for the results of our speed-optimized programmable adder given the input pattern from Figure 4 of the project specification. Our speed-optimized programmable adder can be clocked at a maximum frequency of 1.852 GHz. The peak power consumption and average power consumption are 3.450 mW and 0.423 mW respectively. The total combined width of all transistors is 2514L where L = 60 nm is the minimum length of a transistor. In other words, the total combined width is 150.84 μm.

Refer to Appendix A for a schematic of our power-optimized programmable adder with transistor sizings. Refer to Appendix C for the results of our power-optimized programmable adder given the input pattern from Figure 4 of the project specification. Our power-optimized programmable adder can be clocked at a maximum frequency of 1.299 GHz. The peak power consumption and average power consumption are 1.167 mW and 0.187 mW respectively. The total combined width of all transistors is 1383L, or 82.98 μm.

Justification for Choice of Logic Family to Meet Design Goals

The design of our full adder primarily consists of pass transistor logic as shown in Appendix A. The necessary logic to generate the sum bit (*S*) and the carry out bit (*Cout*) lends itself extremely well to XOR gates and muxes, both of which can be easily and efficiently implemented using pass transistor logic. Moreover, pass transistor logic possesses several advantages over other logic families.

Compared to CMOS logic, pass transistor logic uses fewer transistors and takes up less area. For example, the baseline mirror full adder presented in lecture uses 24 transistors while our full adder uses only 18 transistors. Moreover, if we size the baseline mirror full adder to match the unit-sized inverter, the combined width of all transistors in the baseline mirror adder is 81L. On the other hand, the combined width of all transistors in our full adder is only 54L in the speed-optimized configuration and 38L in the power-optimized configuration. Compared to dynamic logic, pass transistor logic is easier to implement and consumes less dynamic power. Dynamic logic poses additional design challenges such as charge leakage, charge sharing, and clock feedthrough. These challenges are nonissues for pass transistor logic. Furthermore, dynamic logic requires the strategic placement of capacitive loads, which complicates the design of the full adder and increases dynamic power consumption. By omitting these capacitive loads, pass transistor logic simplifies the design of the full adder and reduces dynamic power consumption. Compared to ratioed logic, pass transistor logic is more robust and consumes less static power. Ratioed logic consumes static power whenever the pull-down network is on because the pull-up network is always on. In contrast, pass transistor logic consumes no static power outside of leakage currents. Additionally, ratioed logic does not permit full swing from rail to rail at the output of a logic gate. On the other hand, pass transistor logic achieves full swing through the use of transmission gates.

Due to the advantages of pass transistor logic outlined above, the design of our full adder, as well as the rest of our programmable adder, primarily consists of pass transistor logic as shown in Appendix A. One exception is our half adder, which uses CMOS logic to implement a NAND gate that generates *Cout**<0>. All inverters are also implemented using CMOS logic. In these cases, CMOS logic is more robust, power-efficient, and area-efficient than the alternatives. On the other hand, all muxes are implemented using pass transistor logic. Our master-slave register consists of both muxes and inverters, which are implemented using pass transistor logic and CMOS logic respectively for the reasons described above. Ultimately, our selection of logic families prioritized a balance of area efficiency, power efficiency, and speed. We felt confident that given our selection of logic families, we had a strong foundation from which we could further steer our design towards either speed or power efficiency by varying transistor sizes.

Description of Logic Simplifications

Our logic simplifications targeted the design of the full adder, which is shown in Appendix A. The design of our full adder is based on the transmission gate full adder presented in lecture. Our full adder uses the

same circuitry to generate the propagate bit (*P*) and *P**. However, our full adder uses different circuitry to generate *S* and *Cout*. In the transmission gate full adder presented in lecture, *S* is generated by a transmission gate mux followed by a CMOS inverter. In our full adder, *S* is instead generated by a XOR gate. As a result, our full adder requires two fewer transistors to generate the sum bit *S*. Moreover, our full adder can generate either *S* or *S** from either the carry in bit (*Cin*) or *Cin** just by swapping the inputs to this XOR gate. In the transmission gate full adder presented in lecture, *Cout* is also generated by a transmission gate mux followed by a CMOS inverter. In our full adder, *Cout* is instead generated solely by a transmission gate mux. As a result, our full adder requires two fewer transistors to generate *Cout*. Moreover, our full adder can generate either *Cout* from *Cin* or *Cout** from *Cin** just by supplying the correct inputs to this transmission gate mux.

Given these observations, we designed two versions of our full adder as shown in Appendix A: an even full adder and an odd full adder. The even full adder accepts *Cin* and generates *Cout*. The odd full adder accepts *Cin** and generates *Cout**. Both the even full adder and the odd full adder could generate either *S* or *S** just by swapping the inputs to the appropriate XOR gate. Because neither the even full adder nor the odd full adder needs to invert a carry bit, our full adder once again requires two fewer transistors than the transmission gate full adder presented in lecture. In this fashion, the design of our full adder afforded us immense flexibility. For example, we could cascade even full adders and odd full adders in any order so long as consecutive full adders of different versions were separated by an inverter. Inserting inverters in between full adders also allowed us to optimize the delay of the critical path, which is discussed in further detail below. Additionally, we could generate either *S*<7:0> or *S**<7:0> to match the number of chained inverters driving the 22 fF capacitive loads at the outputs of the ripple carry adder. For an even number of chained inverters, we could generate *S*<7:0>. For an odd number of chained inverters, we could instead generate *S**<7:0>. Thus, the number of chained inverters was not subject to an arbitrary constraint.

Finally, we simplified the first and last full adders as shown in Appendix A. We replaced the first full adder with a half adder, which benefits from the fact that *Cin* is guaranteed to be a logical 0. As a result, the half adder can generate *Cout** with a simple NAND gate. The half adder also uses simplified logic to generate either *S* or *S**. In a similar fashion, the last adder benefits from the fact that it does not need to generate either *Cout* or *Cout**. As a result, we stripped away some logic to reduce power consumption.

Description of Critical Path

Refer to Appendices D, E, and F to see the critical path and the waveforms of all key circuit nodes on the critical path for both the speed-optimized programmable adder and the power-optimized programmable adder. Both versions of our programmable adder have the same critical path, which originates at the rising edge of *CLK*. The longest CLK-to-Q delay belongs to the register that latches the value of the 1-bit control signal *CTRL*. The output signal of this register, referred to as *CTRL_REG* in Appendices E and F, drives the select line of the mux that chooses one of the inputs to the ripple carry adder. The least-significant bit of the output of this mux, referred to as *MUX_OUT*<0>, is an input to the half adder in our ripple carry adder. This half adder generates *Cout**<0>, which propagates through several full adders in series. The last adder generates *S**<7>, referred to as *SUM_BAR*<7>. This signal propagates through the inverter chain to produce *SUM*<7>, which in turn propagates through the reset logic to produce *S*<7>. Lastly, the value of *S*<7> is latched by a register at the rising edge of *CLK*.

Circuit Techniques to Minimize Delay

We applied several techniques to minimize the worst-case propagation delay of our programmable adder. For example, at one point in our critical path, our half adder generates *Cout**<0>, which then propagates through several transmission gates in series. Transmission gates in series resemble an RC chain. Inverters can be inserted into an RC chain to minimize delay. We tried inserting inverters at different intervals into our ripple carry adder. Ultimately, we inserted an inverter every two full adders to minimize delay as shown in Appendix A. As another example, the design of our master-slave register closely resembles the

mux-based master-slave register presented in lecture. However, we omitted two inverters from our master-slave register as shown in Appendix A, which reduces the setup time and CLK-to-Q delay.

We also observed that each sum bit drives a 22 fF capacitive load. When presented with a large capacitive load, a chain of inverters can be inserted to minimize delay. At first, we inserted a chain of two inverters and configured our full adders to generate *S*<7:0>. We scaled the first inverter to two times the unit-sized inverter because Cadence would not allow us to size a transistor to a width less than 120 nm without the use of variables. Through brute force, we determined that the optimal scaling of the second inverter was approximately 13 times the unit-sized inverter. Based on this information, we estimated that the 22 fF capacitive load had an effective capacitance of approximately 84.5 times the gate input capacitance of the unit-sized inverter. Assuming that a fanout near four was optimal to minimize delay, we concluded that a chain of three inverters was optimal. Given a chain of three inverters, we calculated that the optimal fanout was approximately 3.5. Thus, we ultimately inserted a chain of three inverters that were scaled to two times, seven times, and 25 times the unit-sized inverter respectively to minimize delay. This chain of three inverters is present in the speed-optimized programmable adder as shown in Appendix A. However, this chain is reduced to a single inverter in the power-optimized programmable adder as discussed later in the report. We also configured our full adders to generate *S**<7:0>.

Sizing Transistors to Minimize Delay

In addition to the techniques described above, we sized our transistors appropriately to minimize delay for our speed-optimized programmable adder. At first, we sized all CMOS logic gates to twice the size of the unit-sized inverter. We sized all other transistors to a width of 120 nm. We then increased the widths of our transistors on a component-by-component basis to minimize delay. We continued to increase the widths of our transistors until we observed diminishing returns or worsening delay.

Increasing the sizes of the NAND gate in our half adder and the inverters between our full adders created stronger connections to the power supply and ground, which helped drive the pass transistor logic gates downstream from the CMOS logic gates and reduced delay. Increasing the sizes of the muxes in our combinational logic lowered the effective resistances of these muxes, which reduced delay until the point at which self-loading began to dominate. Unsurprisingly, increasing the sizes of the inverters off the critical path had no effect on delay. Increasing the sizes of the XOR gates in our half adder and full adders increased delay, likely due to self-loading. Increasing the sizes of the muxes and inverters in our master-slave register increased the combined CLK-to-Q delay plus setup time, also likely due to self-loading. In this fashion, we eventually determined the optimal sizes of our transistors to minimize delay for our speed-optimized programmable adder. These sizes are shown in Appendix A.

Circuit Techniques to Minimize Power Consumption

We also applied several techniques to minimize the power consumption of our programmable adder. For example, our pass transistor logic makes heavy use of transmission gates to achieve full swing from rail to rail at the output of each pass transistor logic gate as shown in Appendix A. Not only do pass transistor logic gates consume no static power outside of leakage currents, but full swing guarantees that subsequent CMOS logic gates also consume no static power outside of leakage currents. As another example, our reset logic uses a simplified mux to reduce power consumption. Using pass transistor logic, a standard mux can be implemented with two transmission gates. In our reset logic, we exploit the fact that one of the inputs to our mux is guaranteed to be a logical 0. Thus, we replace one of the transmission gates with a small NMOS transistor as shown in Appendix A, which eliminates some parasitic capacitances and reduces power consumption.

We determined that the majority of the power consumption in our speed-optimized programmable adder could be attributed to the chain of inverters driving the 22 fF capacitive loads at the outputs of the ripple carry adder. The average power consumption of our speed-optimized programmable adder is 0.423 mW.

Of this 0.423 mW, only 0.0266 mW can be attributed to the ripple carry adder itself. 0.306 mW can be attributed to the chain of inverters and capacitive loads, and 0.0904 mW can be attributed to the remainder of the circuit, including the sequential logic. For this reason, we decided to target the chain of inverters to minimize the power consumption of our power-optimized programmable adder. We removed the second and third stages from the chain of inverters in our power-optimized programmable adder as shown in Appendix A, which drastically lowered our power consumption. This result should come as no surprise, seeing as these stages contained some of the largest transistors in our programmable adder.

Sizing Transistors to Minimize Power Consumption

In addition to the techniques described above, we sized our transistors appropriately to further reduce power consumption for our power-optimized programmable adder. At first, we used the same transistor sizings as our speed-optimized programmable adder. After removing two stages from the chain of inverters, we next targeted the master-slave registers. We decreased the sizes of the muxes in our master-slave registers, which reduced our power consumption with little penalty to our maximum clock frequency. We then targeted the largest remaining transistors in our programmable adder. Specifically, we decreased the sizes of the muxes and CMOS logic gates in our ripple carry adder as much as possible while still maintaining a maximum clock frequency of 1.299 GHz. In this fashion, we eventually determined the optimal sizes of our transistors to minimize power consumption for our power-optimized programmable adder. These sizes are shown in Appendix A.

Validation of Programmable Adder

We began the validation of our programmable adder by considering only the ripple carry adder, chain of inverters, and capacitive load in isolation. In other words, we temporarily ignored the accumulation logic, reset logic, and sequential logic. In a single clock cycle, there are 2^{16} possible states of the ripple carry adder. To measure the worst-case propagation delay, it is necessary to consider how the state of the ripple carry adder changes between two consecutive clock cycles. In two consecutive clock cycles, the ripple carry adder transitions from one state to another. Thus, there are 2^{32} possible values for the propagation delay. An exhaustive search of the space of all possible propagation delays is impractical. Therefore, we made several simplifying assumptions about the worst-case propagation delay to limit this search space.

Fundamentally, we assume that the worst-case propagation delay arises when *Cout**<0> propagates through every full adder in the ripple carry adder and (dis)charges all associated parasitic capacitances. We can decompose this fundamental assumption into three constituent assumptions that we state below.

1. For both consecutive clock cycles, bits *A*<6:1> are either all logical 0s or all logical 1s. Similarly, bits *B*<6:1> are also either all logical 0s or all logical 1s. This assumption guarantees that all even full adders generate the same *Cout*, and all odd full adders generate the same *Cout**. Thus, all parasitic capacitances along the critical path are (dis)charged together.
2. In the second of the two consecutive clock cycles, *P* is a logical 1 for all full adders. This assumption guarantees that *Cout**<0> propagates through the entire length of the ripple carry adder. Thus, the reduced search space of possible propagation delays still features the longest combinational logic path.
3. *Cout**<0> transitions from either a logical 0 to a logical 1 or a logical 1 to a logical 0 between the two consecutive clock cycles. In the case in which *P* is a logical 1 for all full adders in both consecutive clock cycles, this assumption guarantees that all parasitic capacitances along the critical path are (dis)charged. Otherwise, this assumption guarantees that at least the parasitic capacitances at *Cout**<0> are (dis)charged in addition to other parasitic capacitances. Thus, the sum of all parasitic capacitances that are (dis)charged is maximized within the constraints imposed by the first two constituent assumptions.

These three constituent assumptions narrowed the search space to 768 possible propagation delays. We measured all 768 possible propagation delays and identified ~40 state transitions that produced some of the worst-case propagation delays. We then compiled these state transitions into a few vector files. In the creation of these vector files, we once again took into consideration the accumulation logic, reset logic, and sequential logic. For each of these state transitions, we set the 1-bit control signal *CTRL* to a logical 0 for both consecutive clock cycles. However, we also observed that some of the state transitions could be reproduced by setting *CTRL* to a logical 0 in the first clock cycle and a logical 1 in the second clock cycle. As a result, we duplicated these state transitions in the vector files. For each of these duplicated state transitions, we set *CTRL* to a logical 0 in the first clock cycle and a logical 1 in the second clock cycle.

Once we created these vector files, we experimentally determined the maximum clock frequency at which the programmable adder produced the correct output when given these vector files as input. The test results of our speed-optimized programmable adder given these vector files and additional vector files are shown in Appendix J. The test results of our power-optimized programmable adder given these vector files and additional vector files are shown in Appendix K. Ultimately, we achieved maximum clock frequencies of 1.852 GHz for the speed-optimized programmable adder and 1.299 GHz for the power-optimized programmable adder. These are the maximum clock frequencies that do not break the functionality of our circuit.

Worst-Case Propagation Delay

After extensive testing, we determined that the worst-case propagation delay for both the speed-optimized programmable adder and the power-optimized programmable adder occurs when *A*<7:0> transitions from 0x81 to 0xFF, *B*<7:0> transitions from 0xFF to 0x7F, and *CTRL* transitions from a logical 0 to a logical 1 between two consecutive clock cycles. In hindsight, this result appears obvious. These transitions satisfy the three constituent assumptions stated above. Moreover, these transitions produce the worst-case behavior in the NAND gate that generates *Cout**<0> and the XOR gate that generates *S**<7>. Specifically, these transitions create the worst-case delay in the NAND gate and leave the output of the XOR gate with the weakest possible connection to the power supply. Lastly, *CTRL* suffers from the longest CLK-to-Q delay in our programmable adder due to the numerous capacitive loads at the output of the corresponding register. The worst-case propagation delay for the speed-optimized programmable adder and the power-optimized programmable adder are shown in Appendices G and H respectively.

Measurement of Power Consumption

To correctly measure the power consumption of our programmable adder, we tied VDD to a single node. All terminals that required a connection to the power supply (e.g., the body terminal of a PMOS transistor, the pull-up network of a CMOS logic gate, etc.) were wired to this node. In a similar fashion, we also tied GND to a single node. All terminals that required a connection to ground (e.g., the body terminal of an NMOS transistor, the pull-down network of a CMOS logic gate, etc.) were wired to this node. Besides the VDD power supply in the top-level schematic of our programmable adder, we did not include any other power supplies in our design. Similarly, besides the instance of GND in the top-level schematic of our programmable adder, we did not include any other instances of GND in our design.

To measure the power consumption of our programmable adder, we supplied our programmable adder with the input pattern shown in Figure 4 of the project description. We then measured the current draw from the VDD power supply. We multiplied the current draw by VDD to obtain the instantaneous power. We then plotted the instantaneous power versus time. We measured the maximum value of this plot to obtain the peak power consumption. We then integrated this plot over the power measurement time shown in Figure 4 of the project description to obtain the energy consumed by our programmable adder. We divided this energy by the power measurement time to obtain the average power consumption of our programmable adder. Our power measurement setup is shown in Appendix I. The results of our power measurements are shown in Appendices B and C.

Appendix A: Schematic with Transistor Widths

Please refer to the schematic of our programmable adder shown below. Text in red represents the name of a component that has its own schematic. Inverters, although not named, also have their own schematic. Text in blue represents a parameter of a component. In the schematics below, only the inverter and the PTL mux are parametrized. Each instance of a parametrized component assigns values to its parameters. Text in green represents one of the transistor widths listed in the table below. These transistor widths may vary between the speed-optimized configuration and the power-optimized configuration of our programmable adder.

Note that not all connections to VDD and GND are explicitly shown in the schematics below. Rest assured, however, that we tied all connections to VDD to the single node shown in the top-level schematic. Similarly, we also tied all connections to GND to the single node shown in the top-level schematic.

Transistor Width Number	Description	Width (Speed-Optimized Configuration)	Width (Power-Optimized Configuration)
W1	8-bit PTL MUX	180 nm	180 nm
W2	1st inverter in chain	120 nm	180 nm
W3	2nd inverter in chain	420 nm	N/A
W4	3rd inverter in chain	1500 nm	N/A
W5	8-bit Slim MUX	240 nm	120 nm
W6	MUX in 1-bit MSR	120 nm	60 nm
W7	PTL XOR Gate	120 nm	120 nm
W8	NAND Gate in Half Adder	720 nm	360 nm
W9)	PTL MUX in Odd Adder	360 nm	120 nm
W10	PTL MUX in Even Adder	360 nm	120 nm
W11	Inverters in Ripple Carry Adder	360 nm	180 nm