Abstract

This report describes the design, implementation and evalation of a 32-bit Carry Select Adder (CSA). The design was compared to a 32-bit Carry Ripple Adder (CRA) by compiling an FPGA: both circuits for the ALTERA® Cyclone EP1C20F400C7 target and running timing simulations using ALTERA® Quartus II. The CSA and CRA had a worst case propagation delay of 27.588 ns and 58.125 ns respectively; and required 105 an FPGA: and 64 combinational elements to realise on the target.

1 Circuits and Coding

Carry Ripple Adders (CRA) are an example of iterative logic where the carry out of each full adder comprises the intercell signal. Iterative logic circuits are easy to implement and test, and subsequently the first adders were built this way. A downside of iterative logic is the time for the intercell signal to propagate to the last iterative unit.

Carry Select Adders (CSA) are fast adders which aim to reduce propagation delay by performing much of the computation in parallel. A schematic of a 32-bit CSA is detailed in Figure 1.

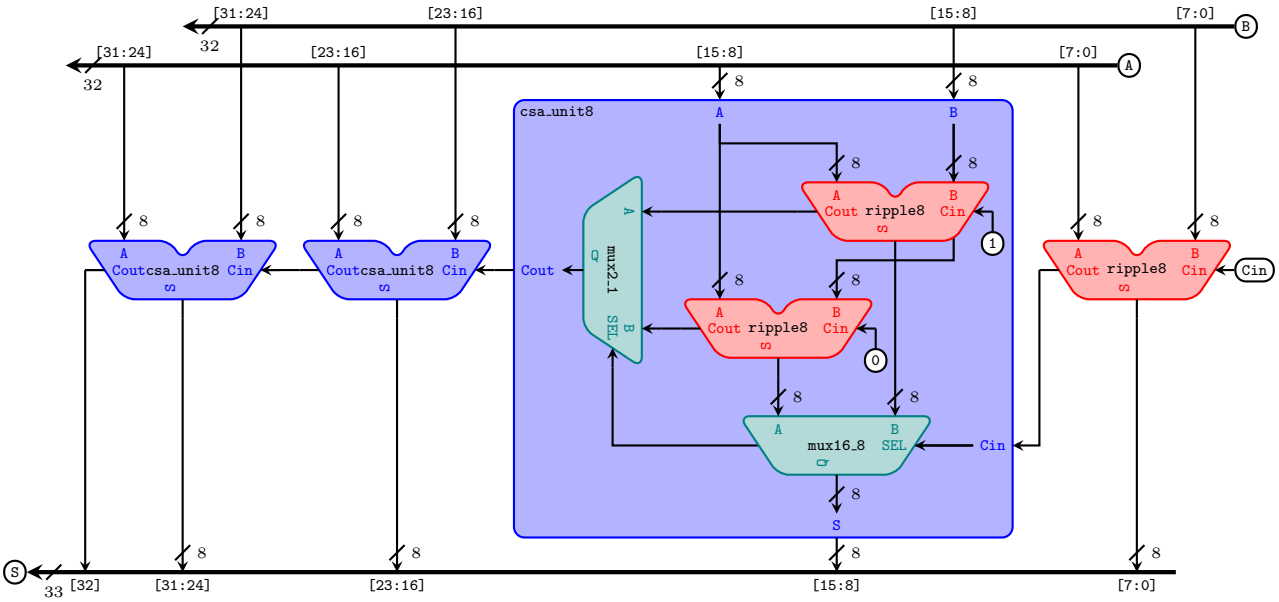


FIGURE 1: Diagram of a 32-bit carry select adder, with 8-bit select units

Figure 2 illustrates the hierarchy of files used to build the 32-bit CSA in Figure 1 and a 32-bit carry ripple adder for comparison. Both The CSA and CRA were implemented in ALTERA® Quartus II and shared as much VHDL as possible to avoid discrepencies in implementation efficiency. Only the top-level files used the Quartus II block schematic file format.

Listings 1 and 2 include the logic expression used to implement the `full_adder` and `mux2_1` design entities.

```
17 ARCHITECTURE full_adder_architecture OF full_adder IS
18 BEGIN
19     Cout <= ((A xor B) and Cin) or (A and B);
20     S <= (A xor B) xor Cin;
21 END;
```

LISTING 1: Architecture declaration in `full_adder.vhd`

```
15 -- architecture
16 ARCHITECTURE mux2_1_architecture OF mux2_1 IS
17 BEGIN
18     Q <= (A and not SEL) or (B and SEL);
19 END mux2_1_architecture;
```

LISTING 2: Architecture declaration in `mux2_1.vhd`

The 16-to-8 multiplexer was constructed using eight parallel `mux2_1` components, mapped to the elements of `std_logic_vector(7 downto 1)` type ports. A similar approach was used to

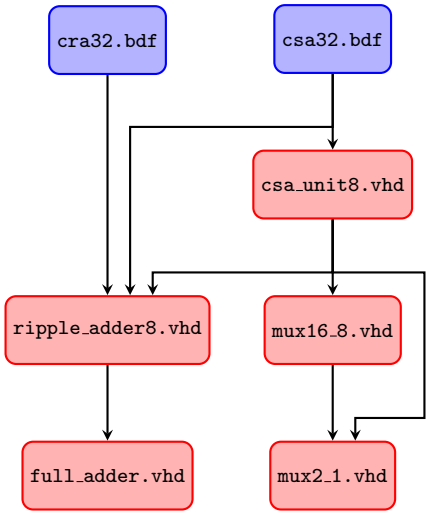
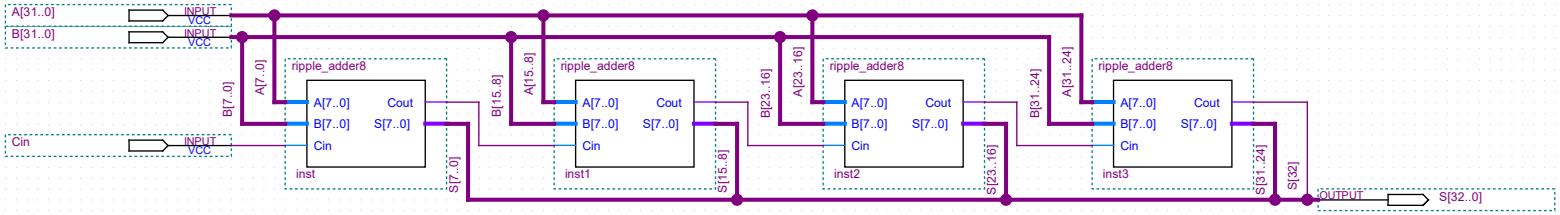


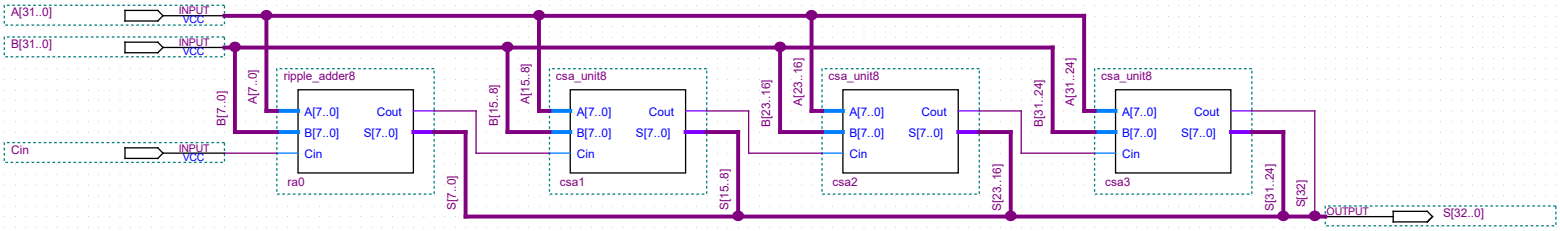
FIGURE 2: Hierarchy of entity declarations

create the 8-bit ripple adder out of `full_adder` components, although the carry in and carry out were mapped to elements of `std_logic_vector(6 downto 0)` type intercell signals. Signals were also used to represent the candidate sum and carry out values from the two 8-bit ripple adders detailed in the `csa_unit8` cells in Figure 1.

The top-level design entities shown in the schematics in Figure 3 were built out of `ripple_adder8` and `csa_unit8` components.



(A) `cra32.bdf`: 32-bit carry ripple adder



(B) `csa32.bdf`: 32-bit carry select adder, 8-bit select units

FIGURE 3: Block diagram schematics for the top-level entities of each project

2 Speed and Operation

The verify the correct operation of the CSA circuit under investigate and CRA circuit for reference, a functional simulation was run on the each project. The circuits were compiled for an FPGA: the ALTERA® Cyclone EP1C20F400C7 target device.

Figure 4 displays the results of the simulation, with the inputs designed to achieve 100 % coverage of 0/1 and 1/0 transitions for all nodes. The circuit outputs and coverage statistics were verified in the simulation report.

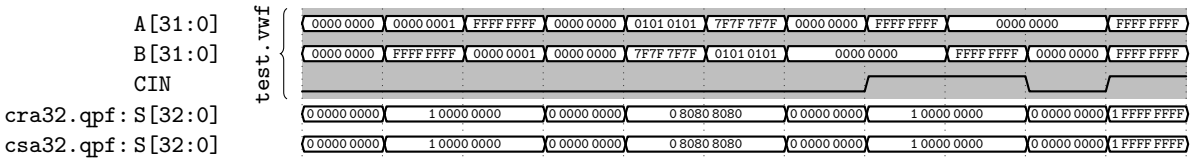
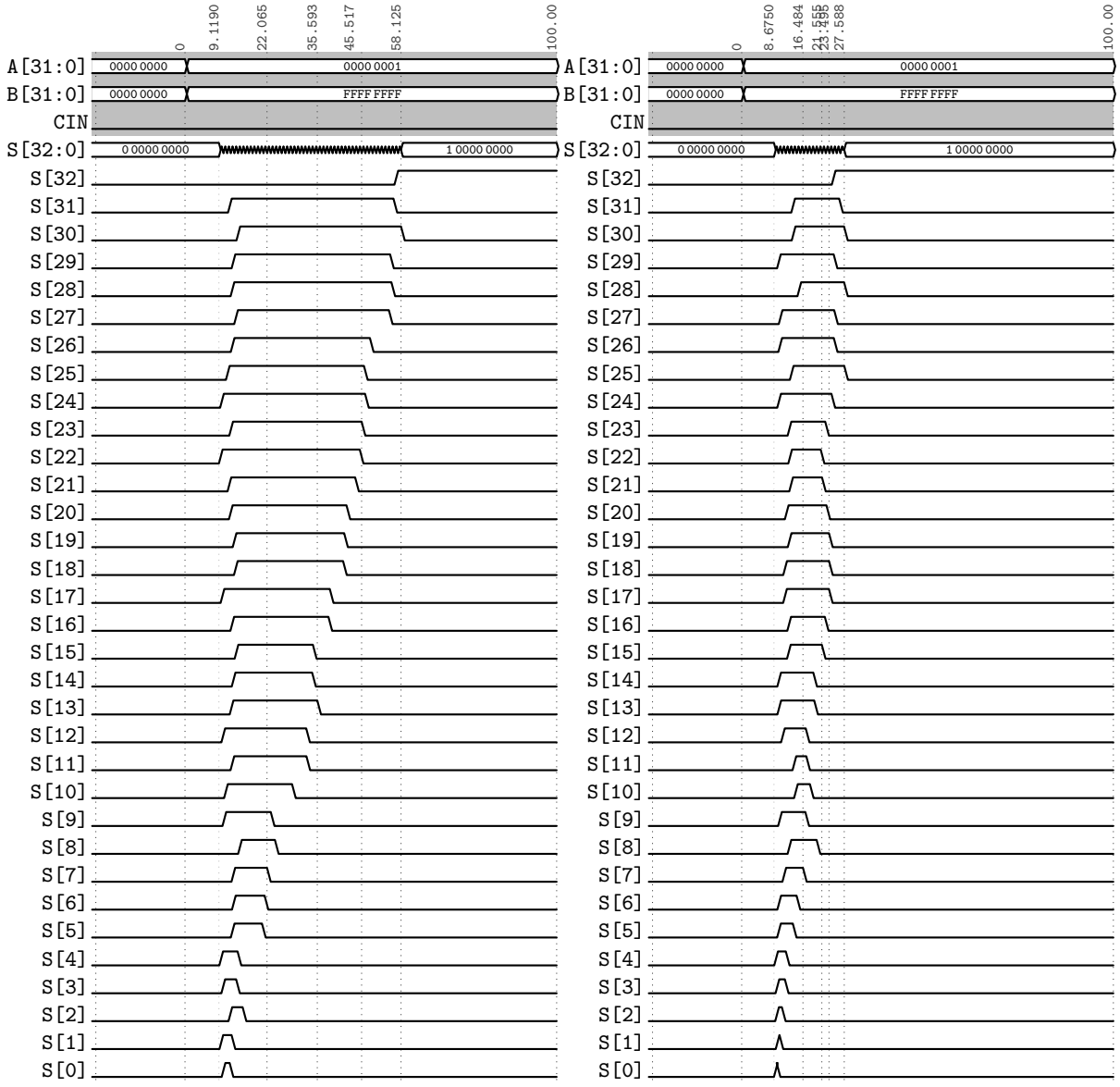


FIGURE 4: Functional simulation output, 100 % coverage

The same test vector was used to compare the speed of the CRA and CSA circuits. A timing simulation was run and the transitions with the longest propagation delay were compared. The results of these simulations can be seen in Figure 5.



(A) 32-bit carry ripple adder (B) 32-bit carry select adder, 8-bit select units

FIGURE 5: Worst case timing simulation output, with the propagation delay (ns) for each 8-bit sum

The propagation delay for the CRA was 58.125 ns, whilst the propagation delay for the CSA was just 27.588 ns. The contamination delays were comparable: 9.119 ns and 8.6750 ns respectively.

Figure 5a reveals the namesake ripple effect through the CRA, as the carry generated by the

the first full adder propagated through the remaining full adders. The sum bits evaluated to 1 when `Cin` = 0, but 0 when `Cin` = 1 once the carry had propagated to the respective full adder. The mismatch in delays from different inputs resulted in erroneous glitches in the simulation output.

In the CSA, the 32-bit addition is split into 8-bit units which are executed in parallel. In each carry select unit, short ripple adders compute sum candidates for both possible carry in values. Simultaneously, the first intercell carry signal is computed by the inital ripple adder, and the carry is then propagated through multiplexors which select the relevent 8-bit sum candidates.

In Figure 5b, the propagation delay for the least significant unit is comparable to propagation delay for the 8-bit units in the CRA, as the construction of the first unit is indentical. The delays for the subsequent units are shorter, as blocks of eight pre-calculated sum bits are selected at a time.

3 FPGA Implementation

Observing Figure 1, apparent drawbacks of the CSA compared to the CRA include the increased complexity, size and power requirements due to duplicating most of the full adders and including multiplexors.

Indeed, Table 1 lists the number of combinational elements required to build each project, alongside the achieved performance, using data extracted from the analysis & synthesis report.

Table 1: FPGA implementation complexity and performance

	`cra32.qpf`	`csa32.qpf`
I/O Pins	98	98
Combinational Elements	64	105
4-input LUT	0	30
3-input LUT	64	52
2-input LUT	0	23
Contamination Delay (ns)	9.119	8.675
Worst Propagation Delay (ns)	58.125	27.588
Maximum Operating Frequency (MHz)	17.205	36.248

For the 211 % increase in performance, 164 % more combinational elements elements were required and the complexity of these logic elements was also more varied. Using more logic elements leads to greater power requirements because more nodes are switched, draining current.

Some complexity and power can be mitigated by the deploying the circuit on a custom ASIC rather than an FPGA. An ASIC can be more efficient by using specific gates rather than generic LUTs, and reducing the length of interconnects between combinational elements. The shorter interconnects in an ASIC also lead to shorter propagation times, resulting in better peformance overall.

However, the time to market to develop, test and manufacture an ASIC is greater than an FPGA implementation. The time and smaller production volume means the unit cost to deploy an ASIC reflect the performance boosts, so ASICs are reserved for the most demanding applications.