64 Bit Adder with/without Multimedia Extension (MMX)

Melvin Ifeanyichukwu Eze  
CSE 477 Course Project  
Fall 2004

The block diagram shows the basic idea for a 64-bit Carry Select Adder where two partial sums are calculated while the carry signal is charging up the select line. Once that is complete the correct sum and carry output are places on the output lines.

Overview

The aim of this project is to design, implement two architecturally different 64-bit Adder functional units. One of the units will perform simple 64-bit Add operations and the other will also include various bit width, mode-selected parallel additions. The latter, termed Adder64-MMX, will support a subset of the well-known Intel Multimedia Extension (MMX) Add operations. In multiples of 8, parallel additions will be supported for the following data types: byte, half-word, full-word and double-word wide operands.

The details of the implementation are shown below and include the functional design, testing and verification, block design and layout, timing simulations, and preliminary power, area and delay numbers.

The design approach

The initial mindset in designing the 64-bit adder was to build completely separate and custom blocks from logical design to MAX layout. This led to exhausting layout, testing and simulations that became difficult to manage. The decision was made to proceed with a more regular structure still based on the Carry-Select Methodology. The new approach involved implementing a 8 block, global ripple-carry adder, with each of the 8 blocks
implementing a fast Carry Select Addition on 8-bits. This has a number of advantages for the project as a whole: first it allows modularity in the design and it focuses optimization capital on a single critical block. Second, it allows for block reuse in the Adder64-MMX implementation quite seamlessly and finally the design is quite scalable (via some nifty techniques; highly scalable) to larger bit widths for either straight addition or more MMX modes.

64-bit Adder (Adder64)

The design of the 64-bit Adder was structured with speed, power and area as the main concerns. The decision to base the implementation on a fast, small and efficient 8-bit CSA adder is a direct result of these concerns. The block diagram below shows the 64-bit, globally rippled, and locally CSA design.

64-bit Adder with MMX support (Adder64-MMX)

Also based on the byte adder CSA block, the main, functionally different feature of this Adder is in the handling of the Carry-chain under the control of a data type mode selection logic. The detection and handling of overflows for operations with size of a word or less is also an important distinction. However, the latter feature is not currently supported in the design. Structurally, the decoder and selection logic offer additional but not significant area increases. In terms of power consumption and speed, those issues will be covered shortly. The block structure below shows the current approach to the design of the Adder64-MMX.

Similar to the Adder64 but for the carry control logic

8-Bit CSA block

As the heart of the design this block leverages the inherent speed of the CSA architecture by implementing a hierarchical 8-bit Carry Select Adder. However, the prime concern was correct functionality. Specifically, the 8-bit CSA is implemented using two 4-bit CSA blocks linked in a ripple fashion. The 4-bit CSA blocks are in turn built by using two 2-bit CSA blocks and so on. Note that the ripple in necessary in order to grow the size of the CSA block from 1 to 2 to 64 bits.
The fundamental block in this design is the Carry-Select Full Adder cell. This is the engine for all bit pair, full-add operations. The rest of the circuitry can be collectively seen as a sum, carry routing network controlled by multiplexers and carry-in signal. However, a decision was made to verify the correct operation of the Ripple-Select Approach — in line with the functionality first rule, ignoring the redundancies in the signal lines, basic blocks and area. There are a wealth of possible optimizations to the basic design that is discussed here and details will be included in the final report.

**Figure 1** This shows the SUE block implementation of the 8-bit CSA cell. Note that the cell contains two 8-bit RS modules that generate the partial sums.

### 8-bit Ripple Select

In line with the modular design applied throughout this project, this block is based on two 4-bit Carry Select cells linked in a ripple fashion. A pair of ripple select modules becomes the partial sum generators for the 8-bit CSA cell. Similar relationships exist in the design for the 4 bit CSA, and 2-bit CSA and shown below.
4-bit Carry Select

This functional block as previously described, is made up of two 4-bit Ripple Select Adder blocks. The 4-bit RS are the partial sum generators in this CSA. See block diagram below.
4-bit RS

This is shown below and contains two 2-bit CSA cells connected in ripple fashion. The diagram also shows the internals of the block.

A more detailed block design is shown below. The hierarchical design approach is obvious.

![Diagram of 4-bit RS block](image)

**Figure 4** The figure above shows two 4-bit Ripple Select (RS) blocks with their carry inputs hardwired. The RS blocks generate partial sums corresponding to the hardwired carry – in signals. The actual result and carry – out are selected once the actual carry – input arrives. The green oval highlights the hardwiring of the RS blocks, the red dotted line highlights the two RS Blocks, and the RS blocks are in turn implemented using 2-bit CSA Cells –the bold line indicates the Ripple Chain between the two 2-bit CSA cells (blue dotted line) that make up each of the 4-bit RS blocks that are later used to implement the 4 bit CSA cell.

2-bit CSA

This cell is made up of two 2-bit Ripple Select Blocks connected in a CSA configuration. Partial sums and carry bits are generated out of the two RS blocks is and passed to the inputs of multiplexers. The carry-input signal is used to select the correct sum and carry-out signals.

2-bit Ripple Select

This is finally where the last level of the hierarchy becomes visible. This block is designed using the 1-bit Carry Select full Adder.
1-bit CSA

The main feature of this full-adder design is its simplicity. The Table below justifies the functionality of the simple design. Also the two 2 by 1 multiplexers are clearly necessary to ensure correct operation. The Carry Select Adder normally operates by generating two partial sums; one by assuming the carry-in is a logic zero and the other assuming the opposite. The partial sums are passed to the input of a 2 to 1 multiplexer corresponding to the assumption made in generating that value.

<table>
<thead>
<tr>
<th>Carry-in</th>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry-out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For a 1-bit CSA_FA cell, two classic full-adder blocks are used with one of the input lines on each tied to a constant GND or VDD. These circuits are used to generating the partial sums and can be combined to form the circuit shown. Note the XOR/XNOR produces both sum outputs but also one of the multiplexer select signals.

The gate-level implementation of the 1-bit CSA involves transmission gate, pass-transistors and static-CMOS logic. A hybrid static CMOS – Pass Transistor logic is used to design a functional and fast sum generation circuit. The figure below shows the 6-transistor XOR gate at the center of the sum generator. The internal multiplexers to the adder are implemented using pass-transistor logic and static CMOS inverters. The plot below show correct functionality and the table shows operating characteristics of the block.
**Figure 6** This is a gate-level schematic of the CSA full Adder used throughout the design

**Table 1** spice simulation of the extracted layout with 50fF load at cout

<table>
<thead>
<tr>
<th>Operating Characteristics</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_{\text{LH}}$ (s)</td>
<td>$t_{\text{LH}}$ (s)</td>
<td>$t_{\text{PD}}$</td>
<td>Average Power (W)</td>
<td>Peak power (W)</td>
</tr>
<tr>
<td></td>
<td>1.2e-10</td>
<td>3.23e-11</td>
<td>3.35 e-05</td>
<td>3.35 e-05</td>
<td>1.95e-03</td>
</tr>
</tbody>
</table>

(See MAX layout for area and other structural details)

**2 by 1 multiplexer**

This is the most widely used component in the design. At all levels of the CSA hierarchy, 2 by 1 multiplexers are used to select between corresponding bits from the 0-carry_in and 1-carry_in partial sum generators. They also used on the carry_out path; the critical path
of the entire 64-bit design. As a result speed and size are important in the design of this block.

Therefore, the multiplexer was implemented using transmission gates. This decision was made for speed, power and voltage swing reasons but the extra capacitive load especially on the critical path may slow the circuit significantly. As a result driver buffers will be needed to regain performance loss.

The SUE circuit schematic and NST plot above show the functional behavior of the MUX. The MAX Layout was completed and with a load of 50fF the following was obtained from SPICE simulations:

<table>
<thead>
<tr>
<th>Operating Characteristics</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{rLH}$ (s)</td>
<td>$t_{rH}$ (s)</td>
<td>$t_{PD}$</td>
<td>Average Power (W)</td>
<td>Peak power (W)</td>
<td></td>
</tr>
<tr>
<td>2.79e-11</td>
<td>2.81e-11</td>
<td>2.80e-11</td>
<td>1.75 e-05</td>
<td>1.17e-03</td>
<td></td>
</tr>
</tbody>
</table>

(See MAX layout for area and other structural details)

Max Area = 4μm x 4.5μm

**Drive buffers.**

Drive buffers will be needed to select the massive capacitance seen by the carry output of an 8-bit block feeding the carry input. The Exact sizing for optimal performance was calculated using the estimated carry input capacitance of 50fF (approx 47.8fF from spice files)

\[ \ln(50/2.5) - 1 = 2 \text{ approx} \] 
(also from the .sp file)

The scaling factor $f = (\sqrt{20}) = 4.4$ approx.
However considering the need for a non-inverting output and the small improvement in performance as drive stages exceeded two, a [2, 4] drive buffer was implemented and used throughout the design.

<table>
<thead>
<tr>
<th>Operating Characteristics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{pd,H} ) (s)</td>
<td>( t_{pd,H} ) (s)</td>
</tr>
<tr>
<td>4.5e-11</td>
<td>3.95e-11</td>
</tr>
</tbody>
</table>

The MMX Decoder/Selection logic

The logic needed to implement parallel additions for multiple data types is shown below. The basic function implemented here is a pass/no-pass on the carry signal rippling through the 8 bit blocks. Note that ALL 8 BLOCKS START AND COMPLETE INPUT ADDITIONS (PARTIAL SUMS) ABSOLUTELY IN PARALLEL. This simply means that in MMX mode the availability of the results from the computations approach the speed of 8-bit block as the data width in decreased. The carry input signal is only a selector that allows the correct partial sum and the correct carry output signal to pass on as the sum output. The mode decoder allows a two bit encoding of the possible data type operations. The selection logic is used physically to allow or block carry propagation between any two 8-bit blocks.

![Figure 1 Mode Control and Selection logic](image)

A pass vector is used to engage or disable the carry propagation between blocks for any selected mode. For the 8 blocks, there are 7 carry bridges that need pass /no pass signals. If the signals are labeled S \([s0, s1, \ldots, s6]\) and the mode select signals for the four possible modes are m1 and m0, then the table below will follow:
<table>
<thead>
<tr>
<th>M1</th>
<th>M0</th>
<th>s6</th>
<th>s5</th>
<th>s4</th>
<th>s3</th>
<th>s2</th>
<th>s1</th>
<th>s0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

64 bit mode
32-bit mode
16-bit mode
8-bit mode

The bits of the S vectors repeat regularly so by inspection the following is clear:

- \( s_0 = s_2 = s_4 = s_6 \Rightarrow \text{NAND}(m_1, m_0) \)
- \( s_1 = s_5 \Rightarrow \sim m_1 \)
- \( s_3 = \Rightarrow \sim \text{NOR}(m_1, m_0) \)

The circuit was designed in sue, a MAX layout was also completed all in CMOS logic. See the figures above and below.

*Figure 2 The Adder64-MMX fully integrated with its design blocks.*
The MAX Layouts

The full MAX layout for the basic blocks above is shown below. A few preliminary layouts for the aggregate blocks: 2-bit Ripple Select, 2-bit CSA Cell are also included. Some of the major blocks have also been completed although not fully tested.

1 bit Carry Select Cell (CSA_FA)  
(Area: 9.47 x 4.7)

This is the MAX layout of the 1 bit CSA cell. The Area measures at 9.47 by 4.7

<table>
<thead>
<tr>
<th>$t_{pd,H}$ (s)</th>
<th>$t_{pd,L}$ (s)</th>
<th>$t_{pp}$ (s)</th>
<th>Average Power (W)</th>
<th>Peak power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2e-10</td>
<td>3.23e-11</td>
<td>3.35 e-05</td>
<td>1.95e-03</td>
<td></td>
</tr>
</tbody>
</table>
Figure 3 Simulation of the extracted .sp file for the CSA_FA with 50fF load at carry output
2 bit Carry Select Cell
(Area: 18.3 x 14.6)

Figure 4 Shows the 2-bit CSA Cell made up of four
4 Bit Ripple Select Block
(Area: 36.4 x 15.9)
4 bit Carry Select Cell
(Area: 71.7 x 18.47)
8 bit Ripple Select Block
(Area: 142.5 x 18.50)
The 8-bit Carry Select Adder Cell

(Area: 143.40 x 43.8)
Logical functionality.

Correct functionality of the overall design and its clear verification is crucial to progress on the project. The theoretical results remain tentative until simulations can verify that the design meets specifications. At which time assembly of the basic components into blocks, and final layout can fully proceed.

The logical functionality of the Adder64 and Adder64-MMX designs were investigated using the RTL descriptions of the functional units and compiling and simulating the circuit operation in MODELSIM. Links to all the RTL code are provided on the website.

Synthesis was performed using the OKI standard cell library and the results shown below. The figures are the synthesized blocks from Synopsis Design Analyzer.

A Verilog test bench was used to verify functionality with test cases, corner cases and reasonable coverage. Both implementations were instantiated inside a custom test
bench and the both passed this initial test thereby verifying the correct-to-spec functionality of the design.

*The results from the OKI based synthesis are only good for functional Verification. See the Verilog files links for details.*