# N-Bit Saturated Math Carry Look-ahead Combinational Adder Design in Verilog

## N-Bit Saturated Math Carry Look-ahead Combinational Adder Design in Verilog

### Features

The following design topics are covered:

• Methodology and logic behind look ahead carry combinational adders and saturated mathematics
• Using a for loop to scale the logic slices up with a structural combinational approach
• Benchmarking for performance with a Lattice MachXO2-7000HE PLD and the Diamond toolset
• Simulation of the N-Bit Saturated Math Carry Look-ahead Combinational Adder using a Verilog test fixture (testbench)

### Introduction

This Verilog module uses the basic Boolean equations derived from a binary carry look-ahead adder and structurally builds the equations. A Verilog “for” loop and a “genvar” variable are used to scale the slices up to the user’s desired word width.

The saturation part of the adder ensures performance with minimum distortion when used in DSP and control related designs in contrast to an instance of rollover. The adder will saturate the output by identifying an overflow condition and assigning the maximum positive or negative full scale value to the output.

A Lattice **MachXO2** PLD was used to develop the adder, although one is not limited to this particular tool set.

### Background

The carry look-ahead adder works by evaluating the two words being added to identify carry generate and carry propagate bits. These bits will determine all the places where a carry will occur in a combinational fashion. The addition is done at the same time, slice by slice, as each carry bit is available. The saturation is determined by an overflow flag and the most significant bit of the temporary sum.

### Application

#### Building the Circuit

A Verilog “for” loop is the perfect solution for building the Boolean equations/structures for each slice. This allows scaling according to the user’s requirements. A generate variable, “genvar”, is used in the Verilog module to determine the addend word width and the “for” loop builds the structural logic accordingly. The initial carry-in is initialized to ‘0’ and subsequent carries are dependant on the each bit of the two addends from least significant bit to most significant bit. Each slice is pieced together to build the combinational circuit with speed being limited by only gate delays. The basic carry look-ahead adder slice can be seen below in figure 1.
.
.
.
.

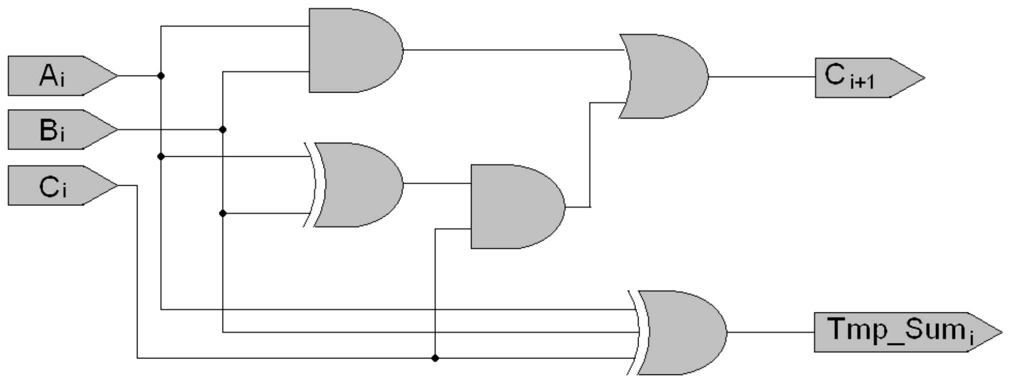


**Figure 1 Basic Carry Look-Ahead Adder Slice**

.
.
.
.

The saturation is determined by an overflow condition. This condition is determined by evaluating the exclusive or of the carry-in and carry-out of the most significant bit, the two most significant bits found in the carry array. If an overflow occurs, the most significant bit of the temporary sum will determine if the final sum should be saturated for the maximum positive value or maximum negative value. A ‘1’ would indicate the overflow caused a positive value to flip negative, so we saturate the final sum value positive. A ‘0’ would indicate the opposite, so we saturate the final sum value negative. A 3-slice implementation of the adder can be seen in figure 2 below. The output sum saturation can be seen in figure 3 below.

.
.
.
.

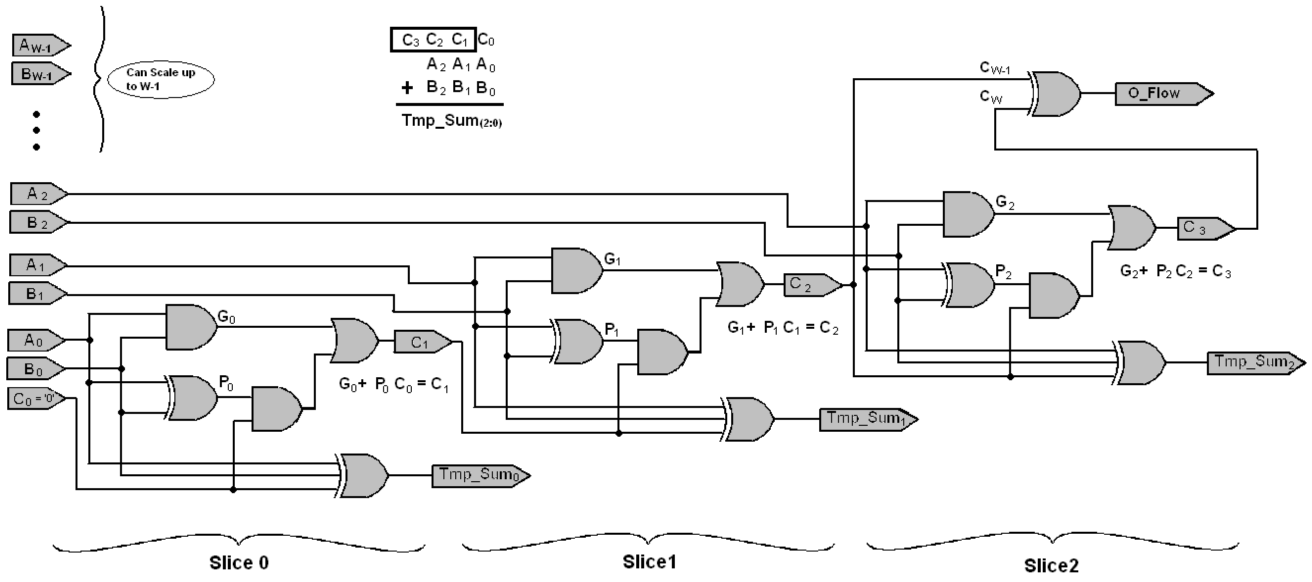


**Figure 2 Three-Slice Implementation with Optional Scaling up to W-bit Addend Width**

.
.
.
.

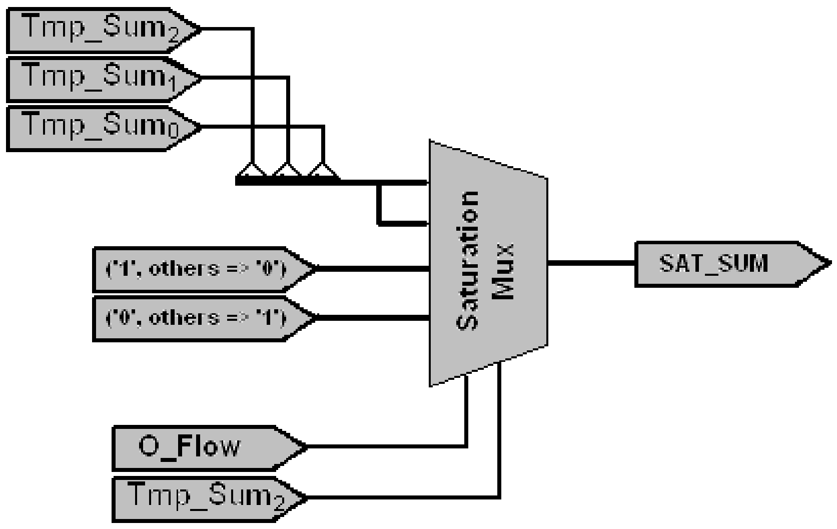


**Figure 3 Three-Slice Implementation with Optional Scaling up to W-bit Addend Width**

.
.
.
.

## Benchmarking

To benchmark the design we can drive the combinational adder between two n-bit clocked flip-flops. This will determine how fast we can run the adder, given a constant addend word width. Given a 64 bit adder, this design was able to run at 114.7 MHz and 188.1 MHz with an 8 bit implementation. These performance numbers were derived from Synplify Pro for Lattice using a MachXO2 device at a -4 speed grade, each result was given in less than 10 nano seconds. The design was auto constrained with no further optimizations selected. One will experience different results given a different FPGA or even the same FPGA at a different speed grade. Optimizations can also be made using a given FPGA supplier's native toolset by properly constraining the design and optimizing for speed over area. A top level diagram for the test schematic can be seen below in figure 4.

.
.
.
.

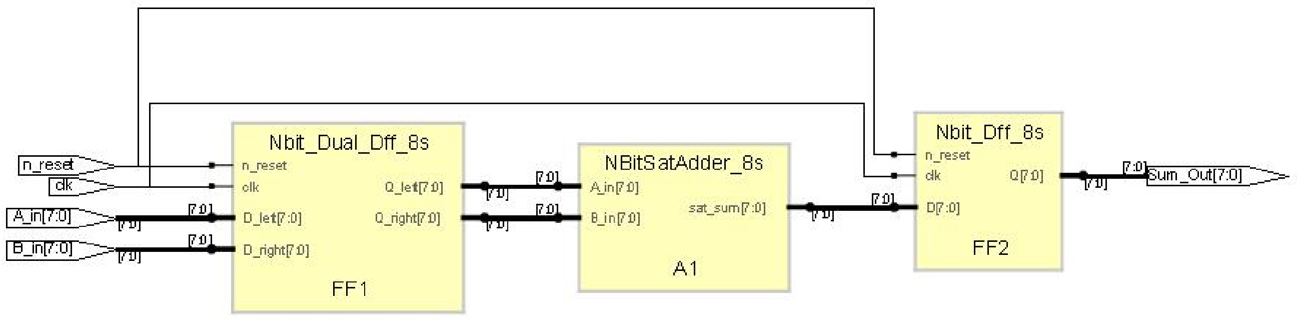


**Figure 4 Top Level Test Schematic for Benchmarking the Combinational Adder**

.
.
.
.

## Simulating the Circuit

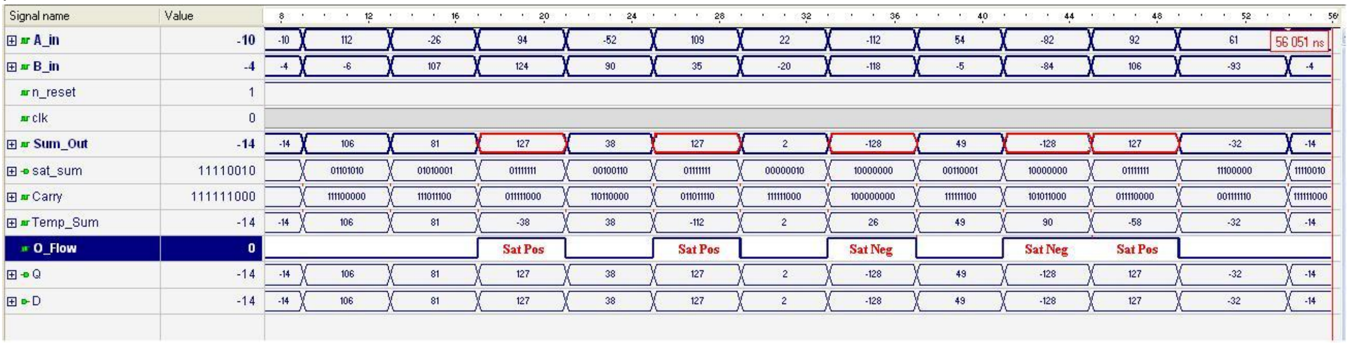
Various HDL simulators, such as Aldec's Active-HDL, can be used in verifying the circuit's output. Active-HDL was used in this particular application. A testbench was created in the Lattice Diamond 2.0.1 development environment. The testbench has two "always" and one "initial" block, for test vector and clock control. The output can be seen in figure 5 below. The red highlights identify where the output has been saturated due to an overflow condition. The testbench for the design is included with the other design files.

.

.

.

.



```
Signal name   Value       8    12    16    20    24    28    32    36    40    44    48    52    56
A_in          -10       -10 | 112 | -26 | 94 | -52 | 109 | 22 | -112 | 54 | -82 | 92 | 61   56 051 ns
B_in          -4        -4 | -6 | 107 | 124 | 90 | 35 | -20 | -118 | -5 | -84 | 106 | -93 | -4
n_reset       1
clk           0
Sum_Out       -14       -14 | 106 | 81 | 127 | 38 | 127 | 2 | -128 | 49 | -128 | 127 | -32 | -14
sat_sum       11110010      | 01101010 | 01010001 | 01111111 | 00100110 | 01111111 | 00000010 | 10000000 | 00110001 | 10000000 | 01111111 | 11100000 | 11110010
Carry         111111000     | 111100000 | 111011100 | 011111000 | 110110000 | 011011110 | 111111000 | 100000000 | 111111100 | 101011000 | 011110000 | 001111110 | 111111000
Temp_Sum      -14       -14 | 106 | 81 | -38 | 38 | -112 | 2 | 26 | 49 | 90 | -58 | -32 | -14
O_Flow        0                     Sat Pos       Sat Pos       Sat Neg       Sat Neg  Sat Pos
Q             -14       -14 | 106 | 81 | 127 | 38 | 127 | 2 | -128 | 49 | -128 | 127 | -32 | -14
D             -14       -14 | 106 | 81 | 127 | 38 | 127 | 2 | -128 | 49 | -128 | 127 | -32 | -14
```

**Figure 5 Active-HDL Testbench Output**

.

.

.

.

## Capabilities, Limitations, and Alterations

This design can be used with any FPGA development platform with varying performance. As the word width gets large, the amount of hardware used also gets quite large. The Lattice Diamond 2.0.1 development environment was used in conjunction with a MachXO2-7000HE breakout board to develop and benchmark this Verilog module. The Aldec Active-HDL simulator was used to simulate and verify the design.

## Conclusion

The NbitSatAdder.v file can be modified and instantiated in any design where speed and output saturation would be of importance. This module works well when used in cascaded adder circuits such as in a parallel FIR filter circuit. When timing is an issue one can implement pipelining and/or a "done" signal for synchronization. In this design, an N-bit, clocked, flip-flop worked well to synchronize the circuit. A multitude of development kits from Xilinx, Microsemi, Lattice, and Altera are available from the Digi-Key Corporation website at the following link: http://search.digikey.com/scripts/dksearch/dksus.dll

## Additional Information

Further design support, product tutorials, application notes, user's guides and other documentation can be found on any of the four major FPGA suppliers' websites:

Lattice at: http://www.latticesemi.com/dynamic/index.cfm?fuseaction=view_category&source=topnav

Microsemi cSOC group's website at: http://www.actel.com/techdocs/default.aspx.

Xilinx applications at: http://www.xilinx.com/applications.index.htm

Altera support at: http://www.altera.com/support/spt-index.html

## Appendix:

The complete NBitSatAdder project files can be downloaded from the Digi-Key, eewiki.net website under the Programmable Logic section

Verilog for Adder Top-Level => NBitSatAdder_TOP.v
Verilog for Adder => NBitSatAdder.v
Verilog for Dual N-bit Flip Flop => Nbit_Dual_Dff.v
Verilog for N-bit Flip Flop => Nbit_Dff.v
Verilog for Testbench = > NBitSatAdder_tf.v

## Feedback for Our Sponsor

Please take a few seconds to help us justify the continued development and expansion of the eewiki.

Click on one of our Digi-Key links on your way to search for or purchase electronic components.

Is the eewiki helpful?  Comments, feedback, and questions can be sent to eewiki@digikey.com.