Introduction to Digital Logic

CSE/EECS 31L

Midterm Project Design Report

128-bit ALU

Group name: The Bitwits

Kevin Leung (ID: 94196429)

Kevin Li (ID: 58571313)

Hikaru Kasai (ID: 50765935)

Jacky Zhang (ID: 3386625)

October 30, 2016

**128-bit ALU**

The 128-bit arithmetic and logic unit (ALU) takes in a mode selector of 0 or 1 to select between the arithmetic unit or the logic unit. Then, it takes an operation selector (opsel) to select which operation to run in the previously selected unit. The 128-bit ALU is actually 128 1-bit ALUs chained in sequence, which means that each of the units only accepts 1-bit values. The ALU is made up of three main units: the arithmetic unit, logic unit, and the 2 to 1 multiplexer driven by the mode signal.

For the arithmetic unit, we created a full adder to handle all of the arithmetic operations. The b (op2) input is modified inside a 8-to-1 multiplexer (mux), named “newmux8to1”, before getting passed to the full adder. The a (op1) input connects directly to the arithmetic unit without modification since a is constant for all arithmetic operations. For example, when operation “sub with borrowed carry” (a + ~b) is selected, the mux sends the b complement signal to the input of the full adder. A separate carry-in multiplexer adds a carry-in to the full adder if required (ie. the “sub” operation - a + ~b + 1). This carry in mux is attached to the first ALU unit, sending the appropriate signal depending on the operation. The outputs of the arithmetic unit are the result and carry out signals. These signals are wired to 2 to 1 multiplexers that also connect the logic unit outputs. Overall, each 1 bit arithmetic unit takes in inputs of carry in, op1, op2, and a 3 bit selector. It outputs a carry out and result.

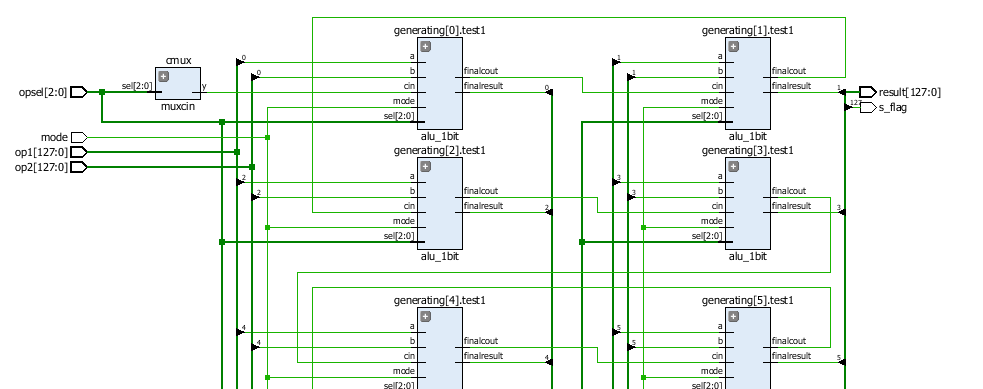
We ran into several issues when making our arithmetic unit. We had trouble with the decrement (a - 1) operation. Decrement is done by adding the two’s complement of 1, or adding 1 to each bit of a (op1). Since we could not determine if this operation was working correctly until generating the 128 bit ALU, we spent a substantial amount of time debugging. In the beginning, we could not add each bit by 1, but could only add 1 to the least significant bit. The remaining bits were added with zeros. Our solution was for b to output 1 for every bit via the newmux8to1. We had many trivial problems such as inconsistent naming for files and variables that sometimes made the coding process confusing. We ran into many warnings such as duplicate source files. For example, we had a problem with the 8 to 1 multiplexer module being illegally re-declared inside the logic unit module, and this caused the port names to overlap with the arithmetic unit multiplexer. Even though our port names were technically correct, Vivado referenced the wrong multiplexer module declaration, causing errors in our 1 bit ALU simulation. Our solution was to rename the illegally declared multiplexer to avoid confusion and the “duplicate file” warning.

For the logic unit, we created an output 8-to-1 multiplexer that chooses between the results of the operations AND, OR, XOR, complement, and the 1-bit logical shift left and returns the result as output. There is a separate carry-out 8-to-1 multiplexer that sets the carry-out to 0 for all of the logic operations other than the 1-bit logical shift left. All of the logic operations aside from the 1-bit logical shift left, were simple enough that they could be written right into the instantiation of the output 8-to-1 multiplexer (for example, AND - A&B, OR - A|B). However, we had a bit of a problem when trying to figure out how to get the output and carry-out of the 1-bit logical shift left into the mux. The solution was to declare two wires: the output was assigned to wire temp\_Y and the carry-out was assigned to wire temp\_cout. Then, we used those wires in the output and carry-out multiplexers to get the right output. Another problem we had was that we did not realize that the selector (opsel) for the 1-bit logical shift left was 101 instead of 100. We programmed the multiplexer to use 100, but our testbench used 101. It took us a while to figure out that we needed to change the multiplexer to use 101 instead.

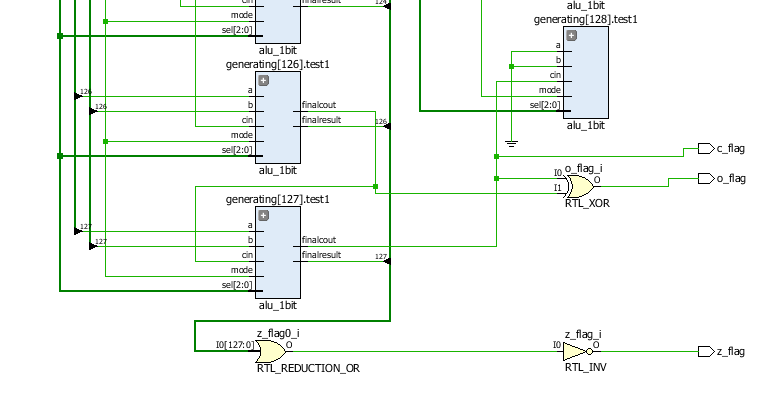
The arithmetic unit and logic unit both have outputs connected to 2 to 1 multiplexers. There are two 2 to 1 multiplexers, each for the result and carry out from the arithmetic and logic units. Both of these multiplexers are driven by the same mode signal that determines whether to send out the arithmetic or logic outputs. When mode is 0, the arithmetic unit outputs are chosen. Likewise, when mode is 1, the logic unit outputs are chosen.

Our ALU generates 4 flags: c\_flag, z\_flag, o\_flag, and s\_flag, which represent carry out, zero, overflow, and sign flags respectively. The c\_flag is raised anytime there is a 1 carried out in the final ALU. This includes anytime there is an overflow, and when there is an addition of two negative numbers, including decrement. Z\_flag is raised every time the computed result is zero. Overflow is raised when final carry bit == 1 XOR second most significant carry bit == 1. S\_flag is raised when the most significant result bit is 1.

**Design Schematic**



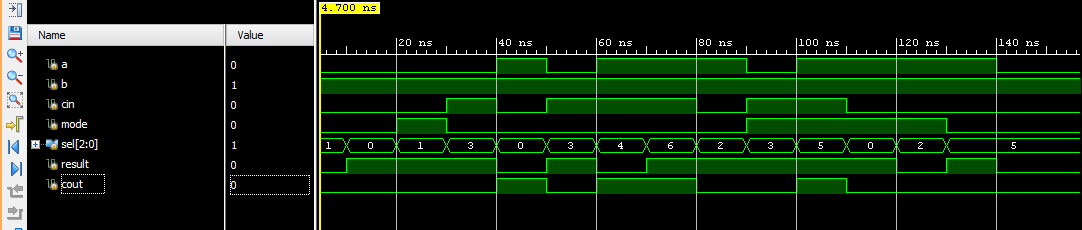
intermediate ALUs omitted



**Port Descriptions**

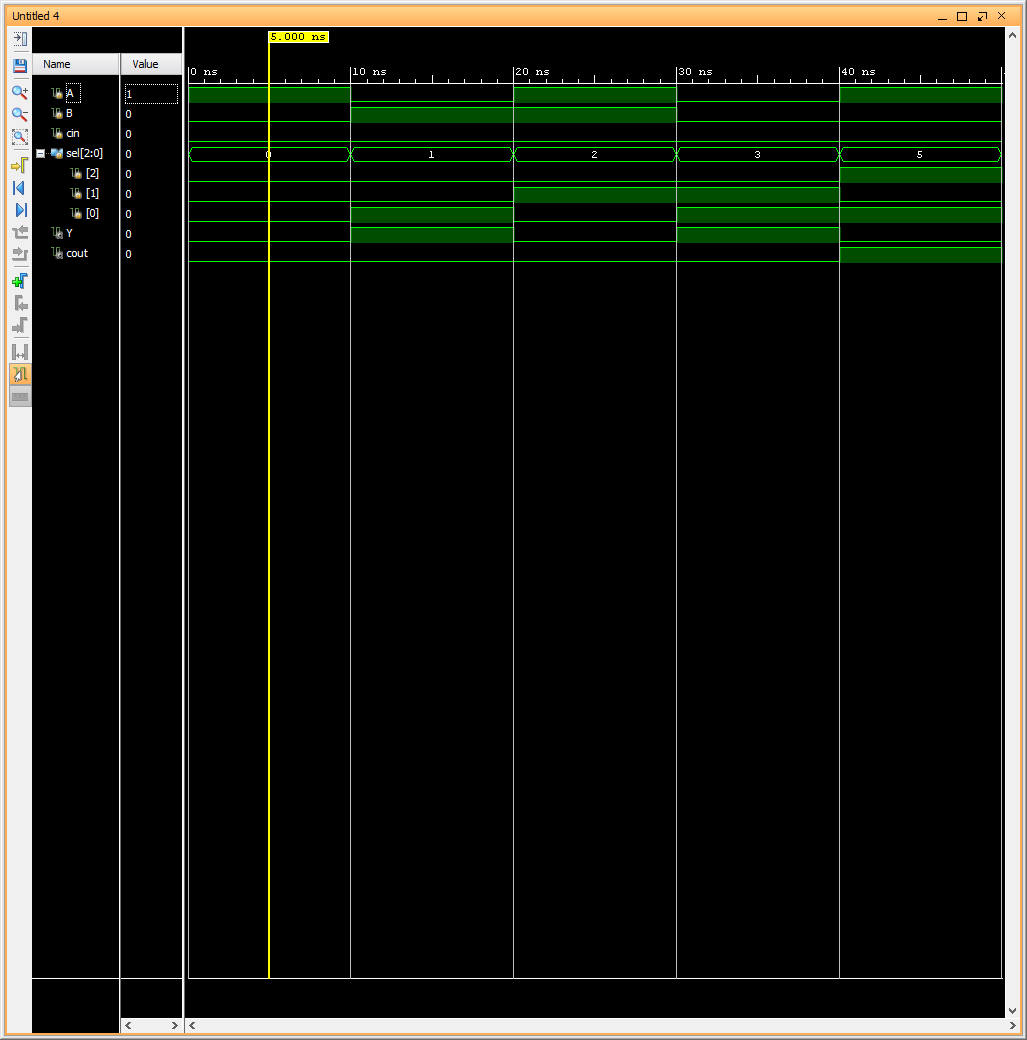
**alu\_1bit - This is the multiplexer that chooses between the arithmetic unit and the logic unit.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| a | 1 | input | This is one of the input ports that will be used to perform micro-operations. For example, A AND B. |
| b | 1 | input | This is another one of the input ports that will be used to perform micro-operations. It will be ignored if complement, move, or 1-bit logical shift left is selected. |
| sel | 3 | input | This is the operation selector (opsel) that will be used to choose between the micro-operations in the arithmetic unit or the logic unit. |
| cin | 1 | input | This is the carry-in input that will be used for some of the operations. |
| mode | 1 | input | This is the selector that will be use to choose between the arithmetic unit or the logic unit. |
| result | 1 | output | This is the output/result of the micro-operation. |
| cout | 1 | output | This is the carry-out bit that will be used for some of the micro-operations. |



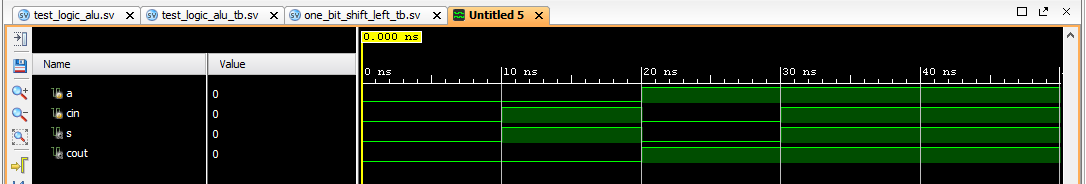
**test\_logic\_alu - This is the logic unit part of the ALU.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| A | 1 | input | This is one of the input ports that will be used to perform micro-operations. For example, A AND B. |
| B | 1 | input | This is another one of the input ports that will be used to perform micro-operations. It will be ignored if complement, move or 1-bit logical shift left is selected. |
| cin | 1 | input | This is the carry-in input that will only be used for the 1-bit logical shift left. |
| sel | 3 | input | This is the operation selector (opsel) that will be used to choose between the 5 micro-operations. |
| Y | 1 | output | This is the output/result of the micro-operation. |
| cout | 1 | output | This is the carry-out bit that will only be used for the 1-bit logical shift left (returns 0 for the other logic micro-operations). |



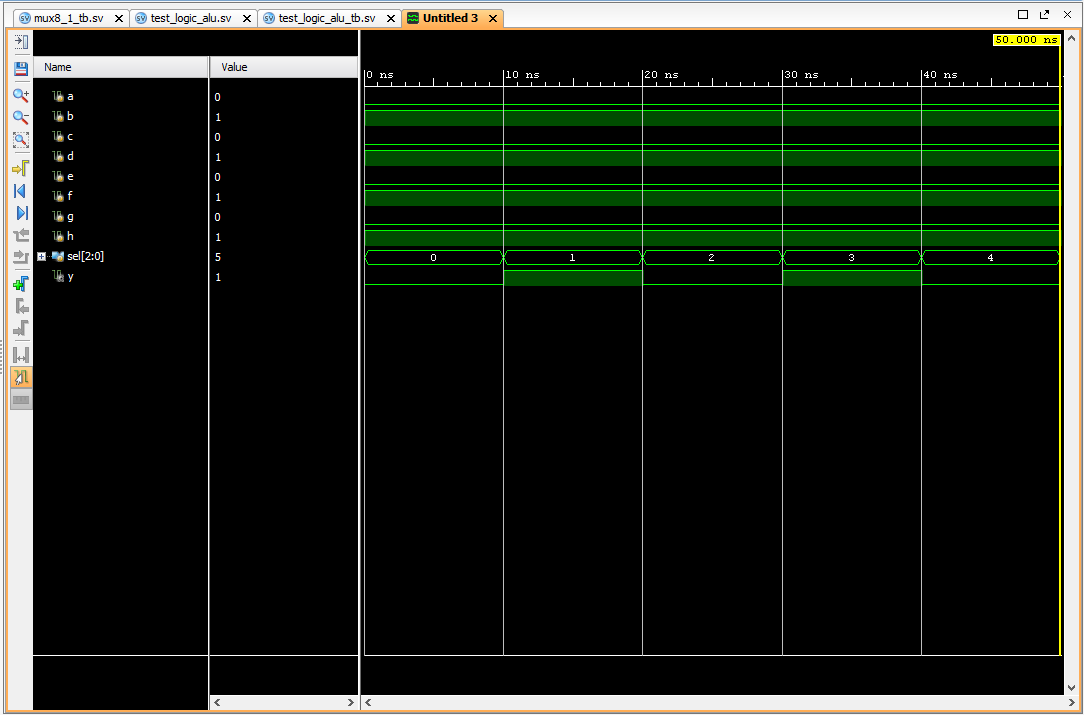
**one\_bit\_shift\_left - This 1-bit shift left module is inside the logic unit.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| a | 1 | input | This is the current bit which will be shifted to the left. |
| cin | 1 | input | This is the carry-in input that will take the place of a in the result. |
| s | 1 | output | This is the current bit after the carry-in has taken the place of a. |
| cout | 1 | output | This is the carry-out bit that will be shifted to the left (serve as the carry-in for the next bit). |



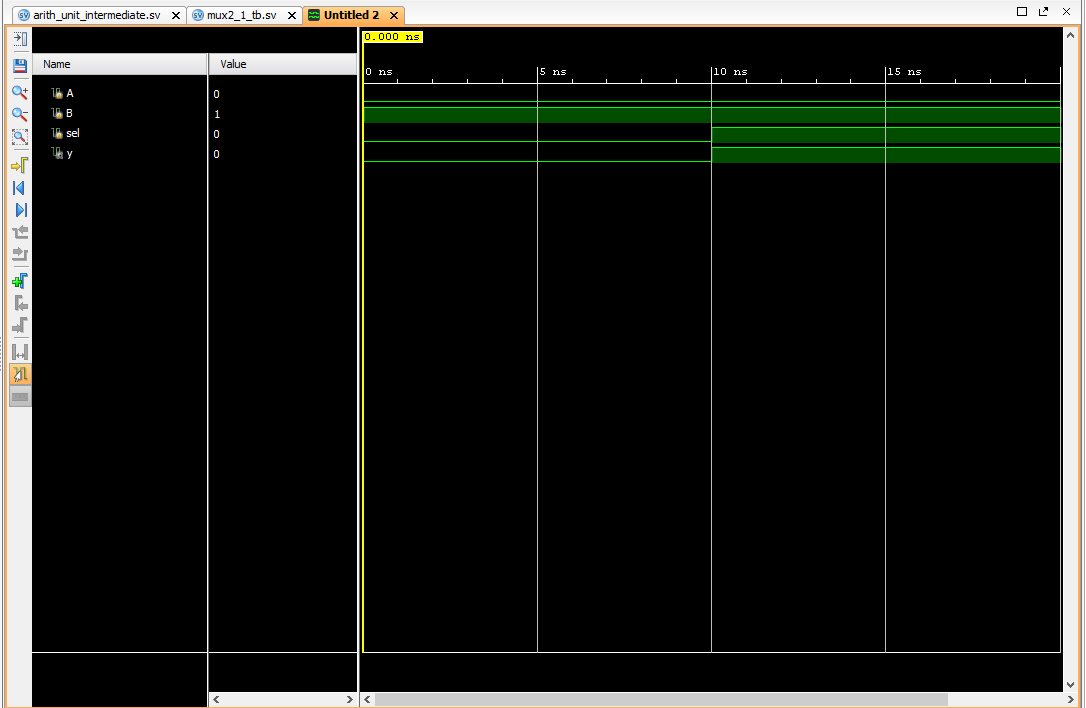
**newlogicmux8to1 - This is an 8-to-1 multiplexer that selects between the 5 micro-operations of the logic unit. It is built out of seven 2-to-1 multiplexers.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| a | 1 | input | This is one of the options that the multiplexer will choose between. |
| b | 1 | input | This is one of the options that the multiplexer will choose between. |
| c | 1 | input | This is one of the options that the multiplexer will choose between. |
| d | 1 | input | This is one of the options that the multiplexer will choose between. |
| e | 1 | input | This is one of the options that the multiplexer will choose between. |
| f | 1 | input | This is one of the options that the multiplexer will choose between. |
| g | 1 | input | This is one of the options that the multiplexer will choose between. |
| h | 1 | input | This is one of the options that the multiplexer will choose between. |
| sel | 3 | input | These three bits serve as the opsel that will select which micro-operation to return as y. |
| y | 1 | output | This is the output of the multiplexer after it has selected which micro-operation to run. |



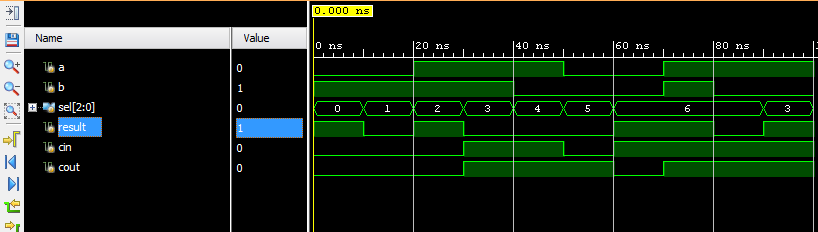
**mux2to1 - This 2-to-1 multiplexer is used to build 8-to-1 multiplexers.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| A | 1 | input | This is one of the values that the multiplexer will choose between. |
| B | 1 | input | This is one of the values that the multiplexer will choose between. |
| sel | 1 | input | This selector bit selects between the two inputs (A and B). |
| y | 1 | output | This is the output after one of the inputs has been selected. |



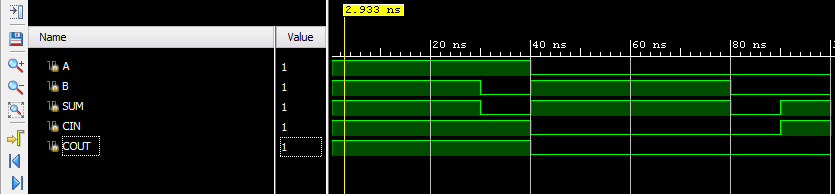
**arith\_unit\_intermediate - This is the arithmetic unit of the ALU**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| **a** | 1 | input | 1 bit of op1 from alu\_1bit |
| **b** | 1 | input | 1 bit of op2 from alu\_1bit |
| **sel** | 3 | input | selects one of the 7 operations |
| **result** | 1 | output | 1 bit result |
| **cin** | 1 | input | Carry out of the previous alu or initial carry in |
| **cout** | 1 | output | Carry out to the next ALU or c\_flag |



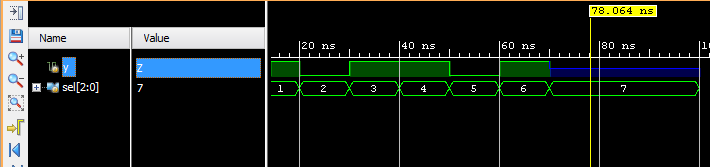
**FullAdder**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| A | 1 | input | This is one of the input values that will be added. |
| B | 1 | input | This is the other input value that will be added to a. |
| CIN | 1 | input | This is the carry-in, which will be added to a and b. |
| SUM | 1 | output | This is the sum of a, b, and cin. |
| COUT | 1 | output | This is the carry-out after adding a, b, and cin. |



**muxcin - This mux only gets gets used for the first alu is initialized. It generates the first cin for the first alu. All subsequent carries are carried in from the previous alu’s carry out.**

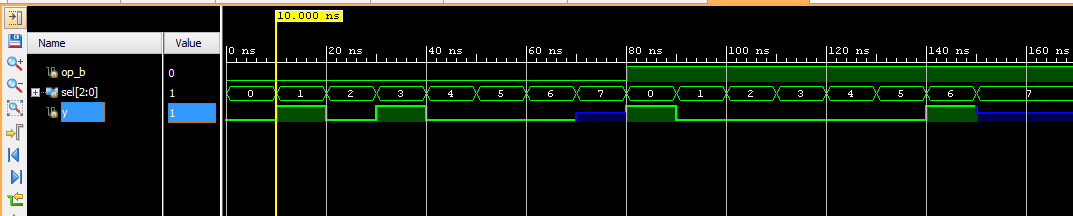
|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| sel | 3 | input | A three bit selector that chooses what should be outputted as the first carry in |
| y | 1 | output | Outputs the carry in for the first alu |



**newmux8to1 - This is an 8-to-1 multiplexer that selects between the 7 micro-operations of the arithmetic unit. It is built out of seven 2-to-1 multiplexers. It takes in a 1 bit b value and outputs a modified value based on the selector**

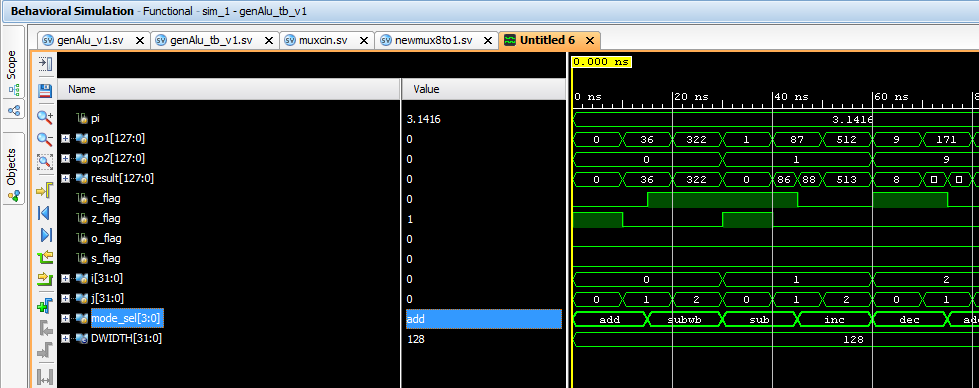
|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| op\_b | 1 | input | This is the 1 bit value of op2 that will be modified by the newmux8to1 |
| sel | 3 | input | A three bit selector that chooses what modified op\_b value should be outputted |
| y | 1 | output | This is the modified op2 bit that will be sent to the full adder (named FullAdder) |

7 in binary is 111 which is not assigned a valid opsel for the alu; therefore, there Z is outputted when sel is 7



**ALU\_128 - The 128 bit ALU**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port Name** | **Port Size** | **Port Type** | **Description** |
| op1 | 128 | input | This is one of the input values |
| op2 | 128 | input | The other input value |
| opsel | 3 | input | Operation selector that chooses which operations to perform in the logic or arithmetic unit |
| mode | 1 | input | The mode determines whether to choose the logic or arithmetic outputs; mode[0] is arithmetic and mode[1] is logic |
| result | 128 | output | The final result, either from the logic or arithmetic unit |
| c\_flag | 1 | output | This is the carry flag where c\_flag = 1 when the 128th ALU produces a carry out of 1. C\_flag = 0 when the 128th ALU produces a carry out of 0. |
| z\_flag | 1 | output | This is the zero flag where z\_flag = 1 if the result is 0. Otherwise z\_flag = 0 |
| s\_flag | 1 | output | This is the sign flag where s\_flag = 1 if the left most bit is 1. If the left most bit is 0, then s\_flag = 0 |
| o\_flag | 1 | output | This is the overflow flag where o\_flag = 1 if the result needs more than 128 bits, a negative result occurs when two positives are added, or when a positive result occurs when two negatives are added. |



**Block Timing**

Total gate delay: 62.516 ns

**Task Assignment**

Although each member contributed to the entire process of making the 128 bit ALU, we had two members focus on the logic unit and and the other focus on the arithmetic unit. Below is a rough breakdown of the task assignments for this project.

|  |  |
| --- | --- |
| **Member** | **Tasks** |
| Hikaru Kasai | Full Adder, arithmetic unit, carry in mux, 1 bit ALU |
| Kevin Leung | Logic unit, logic unit mux, 1 bit ALU |
| Kevin Li | 128 bit ALU, logic unit, generate function |
| Jacky Zhang | 8 to 1 mux, 1 bit ALU, Full Adder, Arithmetic unit, carry in mux, flags |