Lab2: Unsigned/Signed Saturating Adder

Objective

The objective of this lab is to introduce the student to Altera LPMs and combinational logic via VHDL.

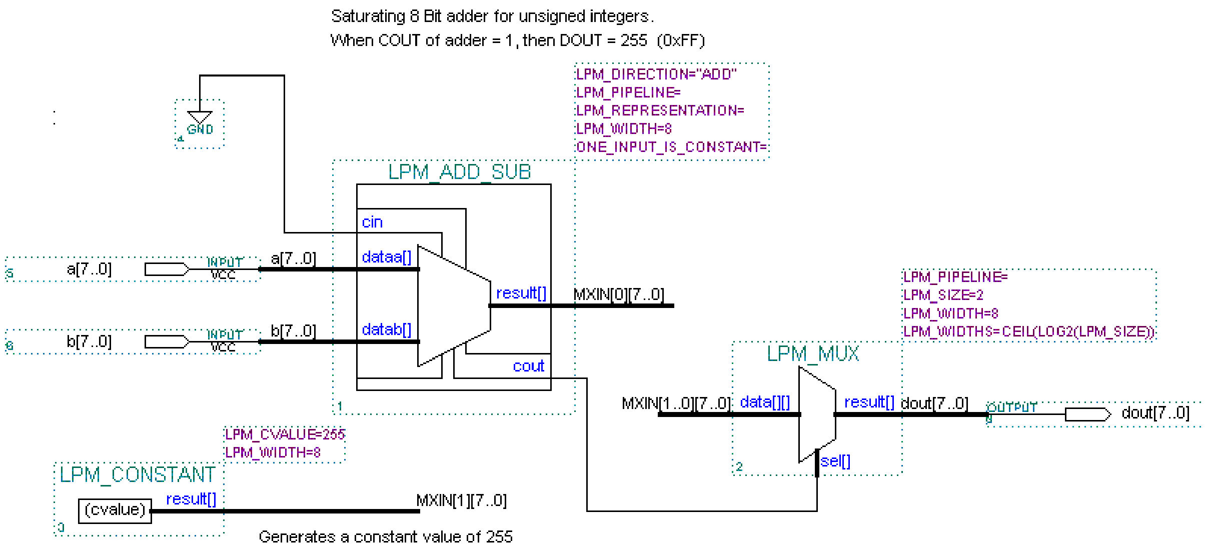
To Do (part 1):

All files referred to in this lab are available in this ZIP archive listed on the WWW page for this lab. To unzip on Unix machines, do '*unzip zipfile.zip*'.

The schematic below (*satadd.gdf*) implements an 8-bit saturating adder for unsigned integers. If the carry out of the adder block is '1' indicating unsigned overflow, then the output will be clamped to 255 (0xFF). Saturating arithmetic is a common feature in 3D graphics and digital signal processing applications.

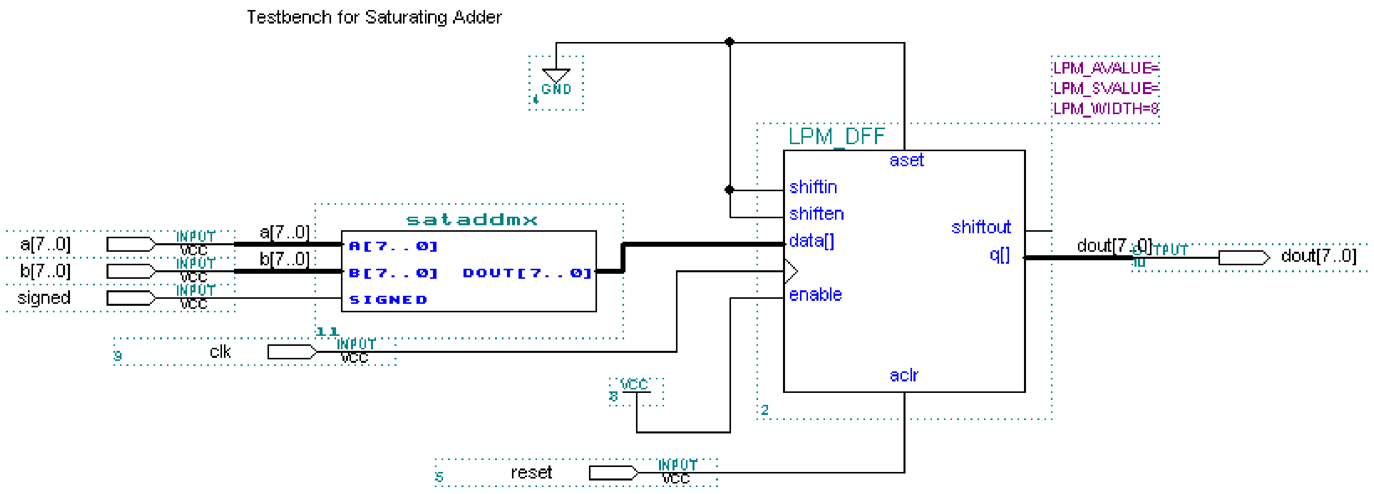
You are to modify this design such that it also supports saturating addition for signed integers (2's complement). An input called *signed* will determine if the addition is for signed integers (*signed* = '1') or unsigned (*signed* = '0'). Look at the lecture on fixed point arithmetic to determine the extra logic that is needed. You must implement at least some of your logic in VHDL. You can implement all of the extra logic in VHDL if you desire. You must call your new design *sataddmx.gdf* (just save the *satadd.gdf* schematic as a new schematic and rename it before editing it).

You can only use ONE adder in your design.



Testing Your Design

The schematic below *(tbsatmx.gdf)* is to be used to test your design.This testbench hooks up an 8-bit wide DFF to the output of *sataddmx* in order to register the output. The waveform file *tbsatmx.scf* can be used to test the design. The waveform file *tbsatmxgold.scf* is the 'golden' file (correct expected results) that can be used to check your results. Use the FILE -> COMPARE command in the waveform editor to compare your *tbsatmx.scf* waveform against the *tbsatmxgold.scf* waveform. When comparing waveforms, red color indicates a match, blue indicates a mismatch. There may be small differences due to combinational delays between your waveform and the golden waveform but the 8-bit value should be the same (the differences will be due to the fact that the programmable device used to produce the golden waveform may not be the same device you use for producing your waveform and thus the output delay of the DFF will be different between the two devices).



TO DO (part 2)

After verifying that your *sataddmx* design works correctly inside of the testbench, map your *sataddmx* design (NOT the *tbsatmx* design) to the Flex 10K family, EPF10K20RC240 device (use any speed grade - the dash numbers indicate speed grade) and the MAX 7000 family, 'auto device', (the 'auto device' will fit your design into the smallest available device from the MAX 7000 family). To assign a particular device, use the 'Assign -> Device' menu while in the schematic editor. Using the report file (*sataddmx.rpt)* that is produced after mapping, give the total number of logic cells (LCs) used in each technology, the %utilization of the device, and the device that the design was mapped to.Use the timing analyzer (Max+plusII -> Timing Analyzer) and report the longest path from input to output for each implementation technology (the pairs of times displayed is the shortest/longest paths for the given pin-to-pin delay). Be sure that you use delays from the *Delay Matrix* (pin to pin delays) - if the timing analyzer does not display 'Delay Matrix' then use the Analysis menu to choose *Delay Matrix*. If the Delay

Matrix has a clock pin, then you are analyzing the delays for the *tbsataddmx.gdf* schematic (testbench), not the *sataddmx.gdf* schematic. Open the *sataddmx.gdf* schematic, use the "File->Set Project to Current File" menu choice, and recompile your design using one of the specified devices. Write a short explanation in the lab report as to which device family is faster (Flex 10K versus MAX 7000) and why.

To Turn In

A printout of your modified schematic and any VHDL files that you wrote, and also the information requested in Part 2.

Check Off

You must DEMONSTRATE you *sataddmx* design to the TA by showing that the waveforms produced using your *sataddmx* design within the testbench matches the golden result.

Questions:

Answer these in your post lab writeup.

1. (2 pts ea)Give the decimal value for the 8-bit binary number “11001000” interpreted as:

    a. Unsigned integer (8.0 format)
    b. Two’s complement integer (8.0) format
    c. Unsigned number, (0.8 format)
    d. Unsigned number (4.4 format)

2. (2 pts ea) Give the result of the following sums (the numbers are in base 16)
    a. 80h + 01h (normal addition)
    b. 80h + 01h (signed saturating addition)
    c. 80h + 01h (unsigned saturating addition)
    d. 7Fh + 01h (normal addition)
    e. 7Fh + 01h (signed saturating addition)
    f. 7Fh + 01h (unsigned saturating addition)
    g. F0h + 20h (normal addition)
    h. F0h + 20h (signed saturating addition)
    i. F0h + 20h (unsigned saturating addition)

3. (5 pts) Why is saturating addition useful?

4. (8 pts) What is the basic programmable element in an Altera FLEX 10K FPGA? Is the FLEX10k volatile or non-volatile?

5. (8 pts)What is the basic programmable element in an Altera Max 7000 device? Is the Max 7000 volatile or non-volatile?

Altera LPMs (Library of Parameterized Modules)

The *satadd.gdf* schematic uses Altera parameterized modules. These modules implement various functions such as add, multiply, muxes, counters, registers, etc. and offer various parameters that allow customization of these components. Most of the LPMs also allow you to select optional input/output pins for the LPMs which add/subtract functionality. To edit the ports/parameters list of an LPM, select the LPM, right click, and choose 'Edit Ports/Parameters'. If you are unclear about what a parameter does, there is a help button in the upper right corner of the 'Edit Ports/Parameters' dialog box that will give you detailed help on the LPM.

LPM_ADD_SUB

The LPM_ADD_SUB module is used to implement an adder or subtractor or a block that does both. In the *satadd.gdf* schematic, the *add_sub* control line is marked as *unused* in the 'Edit Ports/Parameters' list, so this block only does an add function. In addition, the LPM_WIDTH parameter is set to '8' so that it implements an 8-bit addition.

LPM_MUX

The LPM_MUX parameterized mux is straightforward. The parameters control the width of the inputs (LPM_WIDTH, 8 in this case) and the number of inputs (LPM_SIZE, 2 in this case).

Multi-dimensional busses in Maxplus

The parameterized modules make extensive use of multi-dimensional busses. A multi-dimensional bus can be thought of as a group of busses which are all the same width. Suppose I need two 8-bit busses. I could declare two separate busses:

```
A[7..0] B[7..0]
```

or one multi-dimensional bus:

```
DATA[1..0][7..0]
```

I could refer to each of the 8-bit busses in the multi-dimensional bus via the names:

```
DATA[1][7..0]
DATA[0][7..0]
```

In the LPM_MUX module, if you set LPM_WIDTH=8, LPM_SIZE=4 (4 inputs, each 8 bits wide), you will get a symbol whose input bus needs to be labeled as:

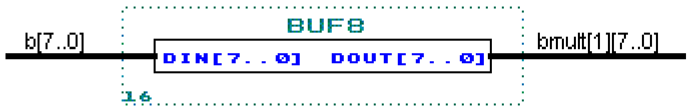
DATA[3..0][7..0]

Connecting Single Dimensional Busses to Multi-Dimensional Busses

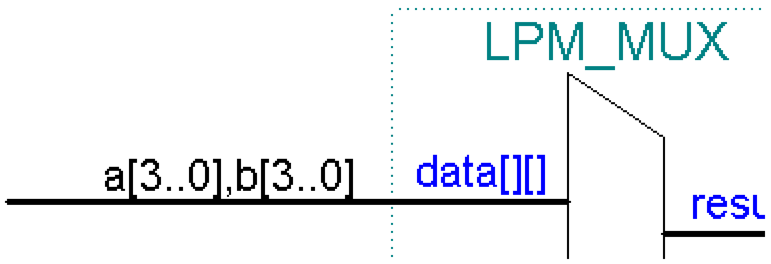
You may often find it necessary to connect a single dimensional bus to a multi-dimensional bus:

A[7..0] => DATA[1][7..0]
B[7..0] => DATA[0][7..0]

One way to do this is by using buffers to connect the two busses. For convenience purposes, I created a VHDL file called buf8 that I use for connecting an 8-bit 1D bus to an 8-bit 2D bus. Example usage of this is shown below:



You can also use labeling to connect two busses to a multi-dimensional bus by putting commas between the bus names on the multidimensional bus:



a[3..0] $\rightarrow$ data[1][3..0]
b[3..0] $\rightarrow$ data[0][3..0]

Common Schematic Editing Problems

Here is a list of some common schematic editing problems. When fixing problems, try to fix the first problem and recompile; multiple error messages may be due to the same problem.

1. "Error: Input pinstub/port *somepin* is unconnected and has no default value" -- To locate the problem component, double click on the error message and the component will be highlighted.You may forgotten to connect the pin, or misspelled the label of the net that is connected to the pin. Sometimes, it will look like a net is connected to a pin but it is not actually connected. Try clicking on the net and moving it around - if it is connected, then the net will 'rubberband' and remain connected.

2. "Error: node missing source :"*pinname*"[ID:*compnum*:*pinname*]". The *pinname* is the name of the pin, the *compnum* is the number of the component (the component number is displayed in the lower left hand corner of the component - the BUF8 picture above has a component number of '16'). This error happens if there is actually a net connected to an input pin, but the net does not connect to an output pin of some other component. The most common cause of this problem is that you have either mislabeled the output input net name or the input pin net name.

3. "Error: Illegal node or pin name :"*pinname*"[ID:*compnum*:*pinname*]". This usually happens if an illegal syntax has been used for a bus name, such as A[7:0] or A[7.0], or A(7..0), etc. A bus name uses two periods to seperate the high/low bus indexes, and brackets around the bus indexes (A[7..0]).

4. "Error: Width mismatch in pinstub :"*pinname*"[ID:*compnum*:*pinname*]". This is due to a mismatch between the label on a bus and the pin that it is connecting to. For example, if a bus has the label A[8..0] (9-bits wide), and the pin has the label A[7..0] (8 bits wide), this error will be generated. This also happens if you use a single dimensional bus label where a multi-dimensional bus label is required, or vice-versa.
5. "Error: Tri-state node must be driven by a TRI buffer, but is driven by a primitive :"*pinname*"[ID:*compnum*:*pinname*]". This happens when you connect mistakenly connect two outputs together by physical connection or by using the same net name (you can only do this if you are using a tri-state buffer, and the FLEX devices we are mapping to do not implement tri-state buffers).

These are not all of the error messages due to schematic errors, but are definitely the most common ones.

Common VHDL Problems

The number of possible syntax problems in VHDL are too numerous to list. The best advice is to try to solve the first problem and then recompiling. Also, when you edit a file via the "Max+plus II -> Text Editor", you can use the "Templates-> VHDL Template" menu to insert various templates for VHDL structures. This can save you from many common syntax problems - I would highly suggest that you use this.

Also, when you save a file, **it must have** a **.vhd** file extension!!!!! The default file extension when you open a file in the Max Plus text editor is not **.vhd**, -- you must explicitly choose this file extension when you save the file.If you use any other file extension, the VHDL compiler will not be used on this file and you will spend an infinite amount of time trying to fix strange error messages that have nothing to do with VHDL!!! Also, the file name must match the entity name! If you use an entity name of 'mymodel', then the file should be saved as 'mymodel.vhd'.