UNIVERSITY OF BRITISH COLUMBIA

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EECE 479: Introduction to VLSI Systems
Fall 2007

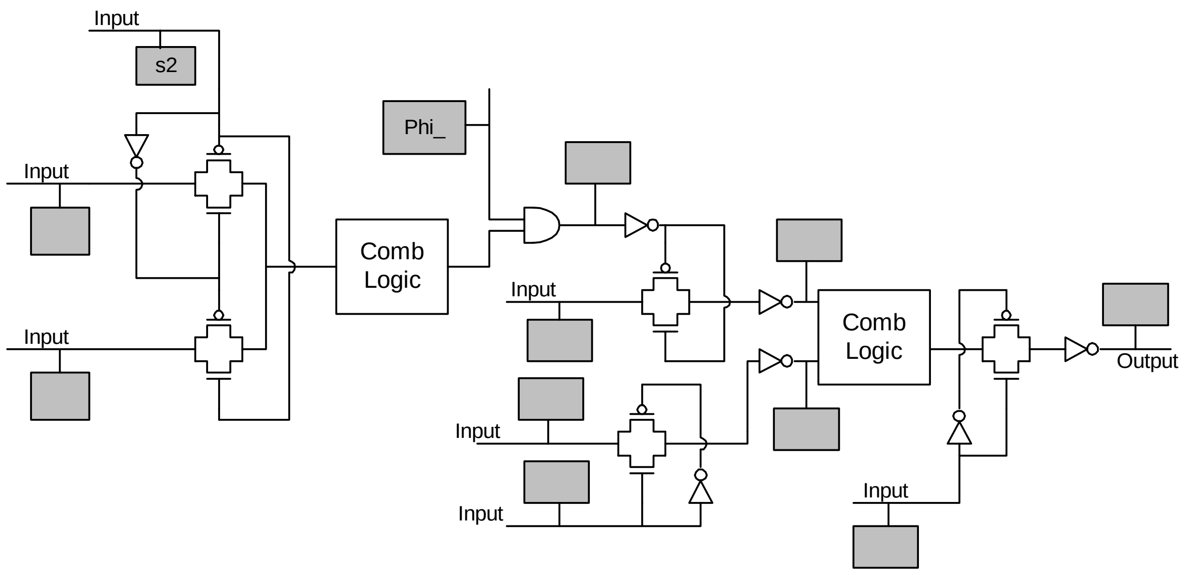
Assignment 4: Two-Phase Clocking and Verilog

Due: October 26, 2007 at 11:59pm

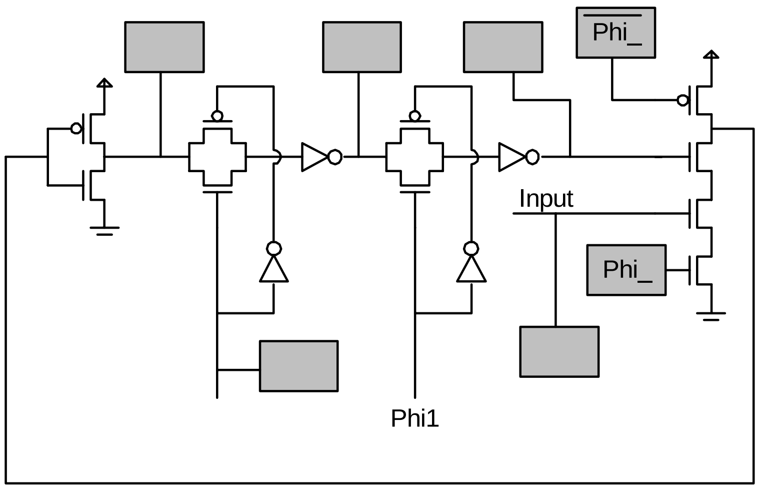
Task 1: Written Exercises.

This task will not be marked, but you should expect questions similar to these on the exam.

1. Fill in the blanks showing the clocking-type for each signal (ie. s1, s2, etc.)



2. Fill in the blanks showing the clocking-type for each signal:



Task 2: Set up your Account for This Assignment

1. Create the directory eece479/ass4 in your home directory (you must give it this name).
2. Copy the files in ~eece479/ass4_files to this new directory:

```
cd  eece479/ass4     -- change to the new directory you just created
cp –r ~eece479/ass4_files .       – copy the files to the current location (. means "here").
```

3. Before running the Verilog simulator, execute the following scripts (this is a different script than in Assignments 2 and 3):

```
source /CMC/cad/bin/ldv.csh
```

To simulate a Verilog specification, use the command:

```
verilog filename.v
```

If your design is in multiple files, you can do something like:

```
verilog counter2.v counter2_test.v
```

In this case, counter2_test.v would contain the testbench to test counter2.v

You can include more that 2 files:

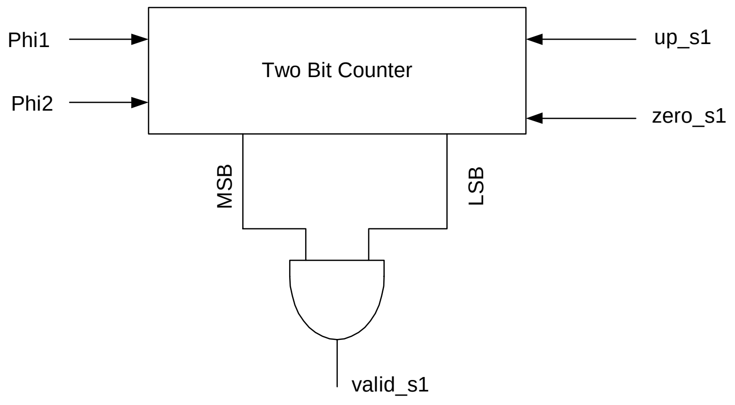
```
verilog counter2.v shift4.v controller.v top.v top_test.v
```

The test files we have created output their results to the standard-output (ie. text).

Task 3: Two-Bit Counter

In Task 5 , you will write a Verilog specification of a serial receiver. In tasks 3 and 4, you will write two of the modules needed for that receiver.

Task 3 is to write and simulate a Verilog description of a two-phased clocked two-bit counter. The inputs and outputs of the counter are as shown below.



There are two clock inputs Phi1 and Phi2 as described in class. The counter has two additional control inputs: up_s1 and zero_s1. Each clock period, the following happens:

1. If zero_s1 is 1, the counter is set to 0 (the counter output is produced by a latch sensitive to Phi2, hence the output is of type _s1. Note that this is a synchronous reset, not an asynchronous reset.
2. If zero_s1 is 0 and up_s1 is 1, the value in the counter register increases by one (rolling over after 3). Again, the counter output is produced by a latch sensitive to Phi2, so the output is of type _s1).
3. If both zero_s1 and up_s1 are 0, the counter holds it's value.

The count value is not an output of the circuit; instead, the circuit contains an AND gate which "and"s the two bits of the counter. When both bits are 1, the output valid_s1 is 1, otherwise it is 0.

You *must* use the following specifications for your design:

Filename: counter2.v
Module Name: counter2
Parameters (in this order):
valid_s1: output
up_s1 : input
zero_s1 : input
phi1, phi2 : clock inputs

You should use the skeleton in counter2.v (which you should have copied to your directory in Task 2). Add all your code for this task to this file.

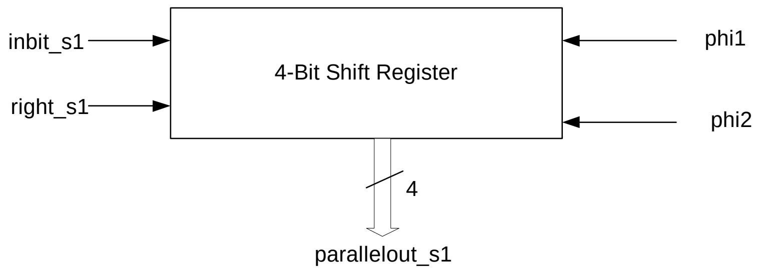
To test your circuit, you can use the testbench in counter2_test.v. This file contains Verilog code which reads test vectors from the file counter2.vec, applies them to the circuit-under-test, and verifies the outputs. Examine this file closely to understand how it works. To simulate with this testbench, use the command:

```
verilog counter2.v counter2_test.v
```

If there are any errors, correct them before going on.

Task 4: Shift Register

In this task, you are to design a 4-bit shift register, as shown below



Each cycle, the following happens:

- if right_s1 is 1, the value in the shift register is shifted to the right by 1 bit. The value on inbit_s1 is shifted into the most significant bit of the shift register.
- If right_s1 is 0, the value in the shift register does not change.

In all cases, the value in the shift register is available on output parallelout_s1. Note that this signal is produced by a latch sensitive to Phi2, therefore, this output is of type _s1.

You must follow the following specifications for your design:

> Filename: shift4.v
> Module name: shift4
> Parameters (in this order):
>> parallelout_s1 : output (4 bits)
>> inbit_s1 : input (1 bit)
>> right_s1 : input (1 bit)
>> phi1, phi2 : clock inputs

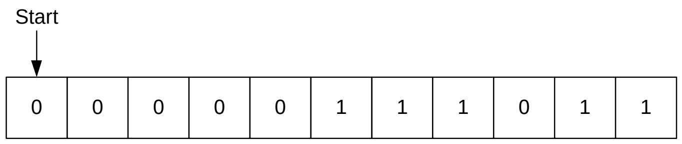
You should use the skeleton available in file shift4.v (which you have copied to your own directory in Task 3). We have not provided a testbench file for this module; you should design one yourself, and simulate the module. Do not proceed until you are sure this module is correct.

<u>Task 5: The last task!</u>

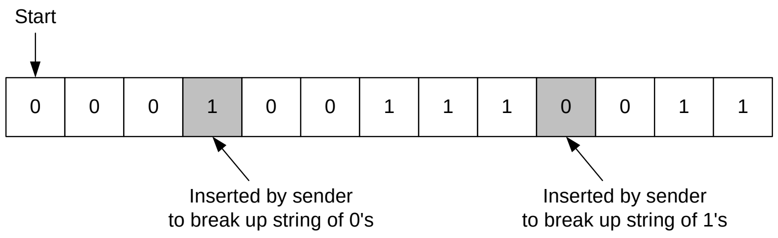
In this task, you are to use the modules from the previous two tasks to build a serial receiver. You will receive bits one bit at a time, and output them in groups of 4.

Serial transmission is usually performed over an optical fiber using some sort of encoding. Often a clock is not sent along with the data; instead, a clock is "recovered" from the input stream, by aligning the clock to changes in the data. This creates a problem if a long string of 0's or a long string of 1's are received; the data doesn't change for a long time, making it hard to lock a clock on the data. Because of this, many serial transmission systems use an encoding called 8B/10B encoding. The primary idea is that a 10-bit code is used for each 8-bit quantity to be sent; each code word is such that there are no long strings of 1's or long strings of 0's. Of course, this results in a 20% transmission-time overhead. If you want more information on 8B/10B encoders do a web search.

We will not implement 8B/10B decoding in this assignment. Instead, we will use a simpler alternative. Whenever the sender detects that three consecutive 0's have been sent, it inserts a 1 into the bitstream, delaying the remaining data by one clock cycle. If the sender detects that three consecutive 1's have been sent, it inserts a 0 into the bitstream. Thus, if the following is to be sent:



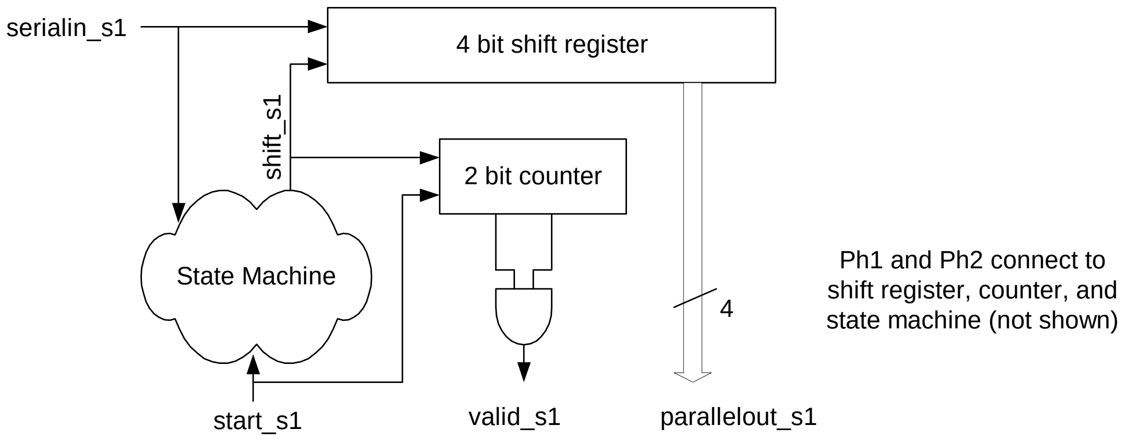
then what would really be sent is:



Note that a 1 has been inserted after the third 0 to break up the string of 0's. A 0 was inserted after three 1's to break up the string of 1's. Note that this 0 was not actually necessary, since the following bit is a 0, however, the sender doesn't look at future bits; it always inserts a 0 after three 1's and a 1 after three 0's.

In this task, you are to design a *receiver* that receives a string with these extra bits, removes these extra bits, and groups the remaining bits into groups of 4.

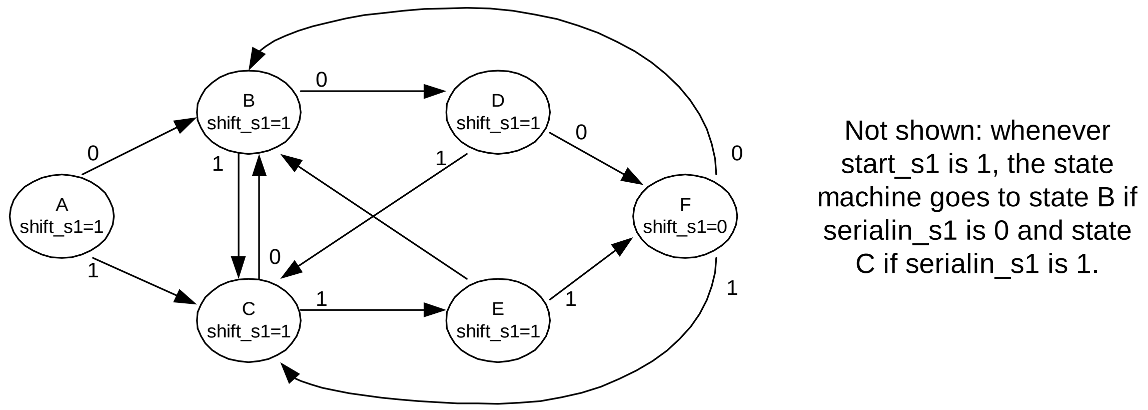
A circuit to do this is shown below:



The shift register and counter are as in Tasks 2 and 3. The state machine is new. The state machine monitors the incoming data stream, and, as long as most recent 3 bits are not all the same, it asserts shift_s1, causing the 4-bit shift register to shift in the incoming data. If the state machine does notice three 0's or three 1's, it de-asserts shift_s1 *during the next cycle*, causing the shift register to ignore one incoming bit (this would be the bit that was inserted to break up the string of 0's or 1's).

The two-bit counter is used to count the number of shifts. When four bits have been shifted, it asserts valid_s1, which indicates that the four bits on the parallelout_s1 output are valid. Note that the counter is only incremented when a new piece of data is shifted into the shift register. If a bit of data is ignored (in which case shift_s1 would be 0), the counter does not increment.

The first thing you need to do for this task is to design the state machine. The following state machine should give you an idea what to do (note: there are other ways to do it, other than implement the following state machine; these other ways are ok, as long as the behaviour is the same).


A shift_s1=1
B shift_s1=1
C shift_s1=1
D shift_s1=1
E shift_s1=1
F shift_s1=0
0
1
Not shown: whenever start_s1 is 1, the state machine goes to state B if serialin_s1 is 0 and state C if serialin_s1 is 1.

You should first specify this state machine using Verilog. You can find a sample state machine in the file sample_fsm.v (you copied this file to your home directory in Task 2). The following specifications must be followed:

> Filename: controller.v
> Module Name: controller
> Parameter List: (in this order)
> > shift_s1 : output
> > serialin_s1 : input
> > start_s1 : input
> > phi1, phi2 : clock inputs

A skeleton file has been given to you in controller.v. Some of the code is already there; you can either use it or replace it.

Once the state machine is done and tested, you can put the pieces together. Your top-level module should be implemented in the file top.v, following these specifications:

> Filename: top.v
> Module Name: top
> Parameter List: (in this order)
> > parallelout_s1 : output (4 bits)
> > valid_s1 : output
> > serialin_s1 : input
> > start_s1 : input
> > phi1, phi2 : clock inputs

Simulate your top-level design using the testbench in top_test.v. This file reads in test vectors from the file top.vec. It is important that your design pass all tests.

What to hand in:

Task 1 should not be handed in (but answers will be handed out).

Tasks 3 to 5 should be handed in using the handin program. We will do much of the marking electronically. For this to work, you must follow the following instructions exactly (if you don't, and we have to mark your assignment by hand, we will be REALLY REALLY TOUGH MARKERS).

1. In Task 2, when you copied your files, you will have created sub-directories for each task. All filenames must be as specified in Tasks 3 to 5 and be in the appropriate sub-directories (there are stubs there, all you really need to do is fill out the stubs).

2. Create another file in your ass4 directory called **info.txt**. This file should contain three lines. The first line contains your name and the second line contains your student number. The third line contains your email address.

3. Enter the following to hand in the file:

```
handin eece479 ass4
```