Implementation of a 4 bit serial adder

Introduction

In this lab, you are going to design a system to carry out serial addition.

Specifically the system will store two 4-bit binary numbers in registers and perform serial addition on them. Serial arithmetic implies that operations are performed on the operands one bit at a time. Since the input operands are represented using 4 bits, it will take 4 clock cycles to complete each operation.

<table>
<thead>
<tr>
<th>Control bit: S</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LOAD</td>
<td>Load operands: A=Ain, B=Bin</td>
</tr>
<tr>
<td>1</td>
<td>ADD</td>
<td>Add: B = A + B</td>
</tr>
</tbody>
</table>

You must use a one-bit full adder, and not a four-bit adder. You may design your own full adder or use one bit of a four-bit adder IC chip (7483).

It is possible for the operations to produce a result that is larger than 4 bits; therefore, you must also have a single output bit that goes HIGH when this overflow condition is encountered.

The first step will be to serially load (shift) your operands into the serial-in registers. Since the input is serial, it will take 4 clock cycles to do this. Then again for the arithmetic operation, 4 clock cycles will be needed. To keep track of the cycle counts, you can use a 3-bit counter. You should probably use the lower 3 bits of a 4-bit counter.

After the arithmetic operation, the result should be in one of the registers, which can only be read out serially from the least significant bit (LSB) of the shift register. This will require another 4 cycles to read the sum. Since the values of the registers can be read at any time, you do not need an instruction for reading results. The result should be shown using 5 light emitting diodes (LEDs) (4 for the result and one for the overflow).

Inputs and Outputs

Inputs: S: Control Bit
         A_{in}, B_{in}: Operand Inputs, least significant bit first
         CLOCK: Clean clock signal

Outputs: A_{out}, B_{out}: Result digits on LEDs
         OVFL: Bit indicating overflow condition
         C[2..0]: Modulo-8 counter bits (to count the cycles) - Optional

Note that A_{out} is used mostly for checking and debugging.
**Data Path & Control Design**

A good design strategy for such a system (and also for microprocessor design) is to have a data path and a control. The data path performs all the necessary arithmetic operations, and the control signals the data path for what to do, according to the control bits and the current state of the data path. In other words, it controls how the data flows through the data path.

First, design the data path to perform all the required operations and define all the necessary control signals to route the bits appropriately and sequence the operations. Then design the control unit to decode the control code and generates proper values for each control signal with a function table (or a state table).

**The Clean Clock**

Mechanical switches, such as the ones available on the CS152A boards, do not provide a clean switching action. Instead of making or breaking a clean connection, they bounce. Hence, the contact is not a clean ON/OFF connection, but rather a series of brief contact connections that may last several milliseconds.

Note that the circuitry in the FPGA or most TTL ICs operates in nanoseconds, hence great attention must be paid to provide switching signals for your design that have clean ON/OFF characteristics. Such a signal is called a "clean clock".

Since on-bards clock or clocks from 555 timer are relatively high, events triggered by these signals are much too fast to be observed. Hence you must test your designs with clock signals that you operate manually. Therefore, you must then have a clean clock.

There are several ways to provide a clean clock signal. It is suggested that you use a clock signal produced by a flip-flop, whose output is controlled with its asynchronous clear and set pins.
The following schematic shows one of the ways that you can get a clean clock. For the switch, you can use one of four gray push-button switches on the board.

Prelab:

- Design your system and draw out the design.
- Figure out what chips and how many of them you will need.
- Have a “floor plan” ready for your self. A floor plan means an approximate positioning of all of the chips on the breadboard.

Lab Work:

Implement your design using the chips on the breadboard. You can use the “push-buttons” for your clock and control bits (or for your serial inputs).

Wiring can be a tedious task, so make sure that you plan where you are going to put the ICs. Check your circuit as you go along. DO NOT wait until you have wired all the components, to start debugging. It is almost impossible at that stage. You should test each part individually: shift register, adder, etc.

Demo of your design

1. Set the control bit to LOAD, and clock the circuit 4 times, changing the inputs $A_{in}$ and $B_{in}$ appropriately.
2. Change the Control bit to select the desired operation (addition), and then cycle 4 times.
3. Monitor the result on $B_{out}$ by cycling 4 times. Note that these cycles can be done simultaneously with the next LOAD operation, or while performing another operation on the bits in the A and B registers.

Extra Credit: (10 points)
For extra credit, you can implement subtraction along with addition.
Some Part Numbers

7483 - 4-bit full adder (of which you may use only one bit)
74393 - Dual 4-bit binary counter
74157 - 4-bit 2-to-1 multiplexer
D flip-flop, 2:1 mux, and various gates (NOT, AND, OR, NAND, NOR, XOR, etc.)