Laboratory Assignments on FPGA based hardware realization

Sessional Practice #4

Objectives:

1. Construction and debugging of digital hardware
2. Role and operation of state machines in digital design.
3. Use of the Altera FPGA hardware and design tools.

Problem: To design a simple processor

Description:

Consider a simple processor that contains a number of 4 bit registers, a multiplexer, an adder/subtractor unit, and a control unit (finite state machine). Data is input to this system via the 4 bit DIN input. This data can be loaded through the 4 bit wide multiplexer into the various registers, such as R0….R7 and A. The multiplexer also allows data to be transferred from one register to another. The multiplexer's output wires are called a bus in the figure because this term is often used for wiring that allows data to be transferred from one location in a system to another.

Addition or subtraction is performed by using the multiplexer to first place one 4 bit number onto the bus wires and loading this number into register A. Once this is done, a second 4 bit number is placed onto the bus, the adder/subtractor unit performs the required operation, and the result is loaded into register G. The data in G can then be transferred to one of the other registers as required.

The system can perform different operations in each clock cycle, as governed by the control unit. This unit determines when particular data is placed onto the bus wires and it controls which of the registers is to be loaded with this data. For example, if the control unit asserts the signals R0out and Ain, then the multiplexer will place the contents of register R0 onto the bus and this data will be loaded by the next active clock edge into register A. A system like this is often called a processor. It executes operations specified in the form of instructions.

Table 1 lists the instructions that the processor has to support for this exercise. The left column shows the name of an instruction and its operand. The meaning of the syntax RX←[RY] is that the contents of register RY are loaded into register RX. The *mv* (move) instruction allows data to be copied from one register to another. For the *mvi* (move immediate) instruction the expression RX ←D indicates that the 16-bit constant D is loaded into register RX.

Table 1: Instructions performed in the processor

Operation	Function
mv Rx, Ry	Rx ←[Ry]
mvi Rx, #D	Rx ←D
add Rx, Ry	Rx ←[Rx]+[Ry]
sub Rx, Ry	Rx ←[Rx]-[Ry]
and Rx, Ry	Rx ←[Rx]Λ[Ry]
or Rx, Ry	Rx ←[Rx]V[Ry]
xor Rx, Ry	Rx ←[Rx] ⊕ [Ry]
not Rx	Rx ←¬[Rx]

Each instruction can be encoded and stored in the IR register using the 9-bit format IIIXXXYYY, where III represents the instruction, XXX gives the RX register, and YYY gives the RY register. Although only two bits are needed to encode our four instructions, we are using three bits because other instructions will be added to the processor in later parts of this exercise. Hence IR has to be connected to nine bits of the 4 bit DIN input. For the mvi instruction the YYY field has no meaning and the immediate data #D has to be supplied on the 4-bit DIN input after the mvi instruction word is stored into IR.

Some instructions, such as an addition or subtraction, take more than one clock cycle to complete, because multiple transfers have to be performed across the bus. The finite state machine in the control unit "steps through" such instructions, asserting the control signals needed in successive clock cycles until the instruction has completed. The processor starts executing the instruction on the DIN input when the Run signal is asserted and the processor asserts the Done output when the instruction is finished.

List of things to do:

1. Design and model the processor in VHDL.
2. Verify the functional correctness of your model through timing simulations.
3. Implement the processor on a Cyclone II FPGA board. Interface the processor with LED and switches.
4. Test the correctness of your implementation