Controller FSM Design Issues: Correctness and Efficiency

Lecture Notes # 10

CAD Based Logic Design

ECE 368
For a major operation (e.g., fetch & decode), trace the datapath through the computer system (processor, memory, I/O, etc.) needed for executing this operation. For a major operation (e.g., fetch & decode), trace the datapath through the computer system (processor, memory, I/O, etc.) needed for executing this operation.

state(s).

Control signals to implement a subpath are set in the corresponding state(s).

Each subpath is dependent on the data produced as a register write in the previous subpath. This data dependency is reflected by their corresponding states in the case of multi-cc operations (that are disjoint from states in-...
CU FSM Design Strategies: Design Correctness (cont.)

Need to perform data-dependent operations sequentially one after the other (as dictated by the dependency) in disjoint sets of states.
nation register should only be valid in the last cc of the operation.

- In multiple-cc operations, the write control signal for writing to the dest-
  for all cc of the operation inputs to FU need to be asserted.
- For multi-cc operation, selective connection to input of FU.
  (c) 1 cc operation with input of FU

(b) 1 cc operation

(3 cc operation) selective connection to 1 cc operation with

In multiple-cc operations, the write control signal for writing to the desti-

For multi-cc operation, need to be maintained for the entire period of the required

Connections to the input of the FU (functional unit, i.e., some combin-

CU FSM Design Strategies: Design Correctness (contd.)
Feedback connection from a register back to itself via one or more FUs and without other intermediate registers is OK if registers are edge-triggered or operate on a 2-phase non-overlapping clocking system—there is no race condition in such a situation.

For operations that can be completed within a clock cycle, the reading from a register and writing back to it (via one or more FUs) can be ac-

For operations that can be completed within a clock cycle without a race condition problem, feedback connection from a register back to itself via one or more FUs is OK if registers are edge-triggered or operate on a 2-phase non-overlapping clocking system—there is no race condition in such a situation.
While subpaths of one operation need to be implemented in separate states, subpaths of 2 or more operations (at most one subpath from each operation) may be combined into one state if: (a) there is no data dependency between them (data produced by one is not needed by the other), and (b) there are no resource conflicts—the same resource is not needed for more than one purpose in any clock cycle. Resources include registers, FUs, buses and other connections, memory.

Implementing subpaths of different operations simultaneously in one state, speeds up the overall speed overall operations and reduces the number of states required in the Control Unit.
(b) Correct but time-inefficient

(c) Concurrent implementation

and no resource conflicts possible, since no data dependency of the 2 operations in 1 cc.

(c) Concurrent implementation
Structural VHDL allows the designer to represent a system in terms of components and their interconnections. This module discusses the constructs available in VHDL to facilitate structural descriptions of designs.

Acknowledgement (this and next 3 slides): http://www.people.vcu.edu/~rhklenke/tutorials/vhdl/modules/m11_23/sld001.htm. The next 3 slides appear in sld037.htm to sld039.htm of this module.
The final example is of an RTL level datapath for an unsigned 8 bit multiplier. It illustrates the use of complex components in a structural description and the use of multiple levels of hierarchy in that the components are themselves structural descriptions of lower level components. This is the entity description for the datapath. The register controls (enable, mode) will go to the control unit when it is added.
This shows the component declarations and binding indications for the datapath.
This is the component instantiations for the datapath. The mapping of individual bits of the D and Q input and output of the shift registers is necessary to reverse the inputs and outputs. Recall that the shift register was a “shift up” type where the scan_in input goes to D(0) and D(0) to D(6) go to D(1) to D(7) when in shift mode. What is needed for the multiplier is a “shift down” type register where scan_in goes to D(7), etc. It should have been possible (we believe) to use the syntax
\[ d \rightarrow \text{multiplier}(0 \text{ to } 7) \]
in the shift_reg8_str PORT MAP to accomplish the same thing, but the QuickVHDL compiler gave a “warning” and the simulator crashed, so we don’t know if it really should work.
Acknowledgement (this and next 7 slides): [http://www.people.vcu.edu/~rhklenke/tutorials/vhdl/modules/m12_23/sld001.htm](http://www.people.vcu.edu/~rhklenke/tutorials/vhdl/modules/m12_23/sld001.htm). The next 7 slides appear in sld050.htm to sld056.htm of this module.
Now we are going to develop a behavioral description of the controller for the unsigned 8 bit multiplier. This is a flow chart of the algorithm that the controller uses.
This is state diagram of the controller state machine. The outputs for each state aren’t shown for clarity. Notice that there is also a “count” variable that must be included in the state machine to count the number of iterations through the loop. The count variable is actually implemented as another state variable.
The state machine actually has two state variables, the current state of the control state machine (e.g., initialize, shift, add), and the present loop count. The loop count is a state variable in that it has a present value and a next value, and it is updated in the clock process. However, the value of count only affects the next control state the machine goes to and doesn’t affect the outputs. The implementation is actually more like two state machines in the same architecture.
This the entity description for the unsigned 8 bit multiplier control unit. It hooks to the datapath via the control signals listed.
This is the beginning of the architecture of the control unit. Note that the constrained subtype of integer is for synthesis - unconstrained integers are hard to synthesize! Also note the state variables are enumerated types. This allows the synthesis tools to encode the state variable using different schemes. Also included here is the clock process. Note that it is edge triggered and that both present_state and present_count are updated on the clock edge. Also note the asynchronous reset signal.
This is the state transition process for the state machine. Note the default assignment of `next_state = present_state` which is only really required for (some) synthesis tools.
This is the output process. Note that the outputs are only dependent on the present_state variable (Moore machine).