ECE 465
Controller or “Action” FSMs

Shantanu Dutt
UIC
Motivating Example 1

- Computation needed: $r_4 = (a+b)(c+d)/x$
- Computation needs to be pipelined
- Functional units with different delays
- No sharing of FUs between operations (simpler situation than sharing)
- Pipeline can move at the rate of 3 cc’s due to Stage 3 having the max delay of 3 cc’s. This essentially means that Stage 1 has to wait 2 cc’s and Stage 2 has to wait 1 cc in order to synchronize with Stage 3, so that all stages move in lock step at the rate of 3 cc (this is the only way to do this when there is no sharing)
- We assume here that the module furnishing input data $a, b, c, d, x$ provides a new data set every 3 cc.
- How to synchronize the movement of data and loading of registers across the stages so that the computation is correctly done w/o new data to any FU clobbering earlier data that is still being processed?
- Need a control or action FSM to orchestrate this
Motivating Example 1 (contd.)

O/p labeling convention in Moore control FSM states: signals not specified in a state are inactive)

Register Transfer Language (RTL) specification of state function

Stage 1

Stage 2

Stage 3

NOP (cc counting state)

NOP (cc counting state)

NOP (cc counting state)
Motivating Example 1 (contd.)

No junk data solution

[repeat for ra to rx's;
ra <= a; rb <= b; rc <= c; rd <= d; rx1 <= x; rx2 <= rx1; rx3 <= rx2]

[repeat for ra to rx's;
r1 <= a+b; r2 <= c+d; rx1 <= x; rx2 <= rx1]

RTL specification

[repeat for ra to rx's;
r1 <= a+b; r2 <= c+d; r3 <= r1*r2; r4 <= r3/rx2; rx1 <= x; rx2 <= rx1]
Motivating Example 2

- Computation: \( z = a \times b + (c+d) \)
- Mapping to h/w w/ constraints: use only 1 \((X)\) (2 cc) and 1 \((+)\) (1 cc)
- Need to share the + FU, so need explicit scheduling of operations (start time assignment)
- Non-overlapped pipelining
- Inputs are loaded in this example

\[
\begin{align*}
\text{lda} & \rightarrow a \\
\text{ldb} & \leftarrow \text{ldb} \\
\text{mux1} & \rightarrow \text{mux} \\
\text{mux2} & \leftarrow \text{mux2} \\
\text{ldx} & \rightarrow x \\
\text{ldc} & \rightarrow c \\
\text{ldd} & \rightarrow d \\
\text{ldy} & \rightarrow y \\
\text{ldz} & \leftarrow \ldc \\
\end{align*}
\]

Controller FSM:

- Non-overlapped pipelined scheduling
- Unspecified control signals have either an inactive value, or if such a concept doesn’t exist for the cs, then the don’t-care value

Note: A register is loaded at the +ve/-ve edge (in a +ve/-ve edge triggered system) of the cc after the one in which its load signal is asserted.
Motivating Example 3

- Same problem of example 3 with the change the pipelining is overlapped, i.e., different iterations are occurring simultaneously in the pipelined datapath (this is “regular” pipelining).

ii) Overlapped pipelined scheduling

Controller FSM:

- For 4 iterations, the overlapped schedule takes 9 cc's versus 12 cc's by the non-overlapped sched.
- Overlap. sched: Time for n iterations = 2n+1
  Throughput = n/(2n+1) ~ 0.5 outputs/cc
- Nonoverlap. sched: Time for n iterations = 3n
  Throughput = n/3n ~ 0.33 outputs/cc
  \( \Rightarrow \) ~ 34% throughput improvement using an overlapped schedule
Part I: Control of Datapaths via Basic Components, Control Operations & Connections

Shantanu Dutt
UIC
(a) $2^n:1$ Mux: $\text{sel} = j \Rightarrow o/p = I_j$

(b) $n:2^n$ DCD: $\text{addr} = j \land \text{en} = 1 \Rightarrow \text{sel} = j$
$\text{& sel} = 0, i \neq j \land \text{en} = 0 \Rightarrow \text{sel} = 0$, for all $j$

(c) Tri-state buffer:
$\text{en} = 1 \Rightarrow o/p = I$
$\text{en} = 0 \Rightarrow o/p = Z$
(high impedance, not connected to any signal)

(d) Bus connection via tri-state buffers: exactly one $\text{en} = 1 \Rightarrow \text{Bus} = I_j$
$\text{en} = 0$, for all $j \Rightarrow \text{Bus} = Z$
(a) +ve edge-trigg. D-FF w/o enable

(b) Timing diagram for +ve edge triggered D-FF (w/o enable)

(c) +ve edge-trigg. D-FF w/ enable

(d) 4-bit register with load/write and reset control
A) Basic Operations:

1) Load a register (individual/separate register, e.g., the Program Counter [PC] in a processor, or one or more registers in a register file)

E.g., $R_1 <- BusA \# Load reg. R_1 \text{ w/ BusA content+1}$

- To perform $R_1 <- BusA$, the "load-$R_i$" signals needs to be "activated" (generally, set to '1') by a controller FSM

- If the FSM sets load-$R_i = 1$ in clock cycle $i$, then $R_1 <- BusA$ (i.e. loading of $R_1$) is accomplished at the $+ve$ edge (assuming a $+ve$ edge system/register) of clock cycle $i+1$. 

CLK

---

---
2) Connecting data to a bus from one of multiple sources

- Example: Write_Bus \leftarrow \text{Adder-output} \text{ if one of many possible FUs}
  or Read_Bus \rightarrow \text{register}_{2.5} \text{ of 32-register Register file.}

- One of multiple sources is selected to connect to a bus via either a multiplexer or tri-state buffer, by activating the right control signals.

- In the above e.g. to accomplish Write_Bus \leftarrow \text{Adder-output}, "alu-set" control signal is set to "00" by a controller, first operand is accomplished in the same clock cycle in which control signal is activated.

C.L.K. \ldots \quad \text{alu-set} = \{0,0\} \quad \text{Write} = \text{Adder-output}
Other combinations of the above 2 operations can be used for varied types of connections and regular loadings.

E.g.: Loading R1 from one of multiple sources

\[ R1 \leftarrow B_{5A} \]  # B_{5A} is one source out of 5 sources

\[ R_1 \leftarrow B_{5A} \]

- \( R1 \leftarrow B_{5A} \) is now accomplished by setting values for 2 control signals \( \text{RI-input-set} \) and \( \text{Load-RI} \) to 010 & 7, respectively.

\[ \text{RI-input-set} = 010 \]

\[ \text{Load-RI} = 7 \]
B) Secondary Operations: 1) Register w/ various shift/rotate/load options

- \( \text{oper} = 00 \) means load-reg from D-bus (I/P Bus)
- \( \text{oper} = 01 \) means LSR
- \( \text{oper} = 10 \) means ASR
- \( \text{oper} = 11 \) means ROR (rotate right)

Controller FSM:
- \( \text{oper} = 10 \)  
  - \( \text{en}_{\text{reg}_1} = 1 \)
  - \([R1 \leftarrow \text{ASR}(R1)] \)

- \( \text{oper} = 00 \)  
  - \( \text{en}_{\text{reg}_1} = 1 \)
  - \([R1 \leftarrow \text{D-Bus}] \)

- \( \text{reset}_{\text{reg}_1} = 1 \)
  - \([R1 \leftarrow 0] \)
Similarly, other symmetric operations can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplished by a combination of the above. Let's consider the basic operations of the 8 bit A, B, C combination. The combination of the above can be accomplis
Part II: Controller FSM Design Issues—Correctness and Efficiency

Shantanu Dutt
UIC
CU FSM Design Strategies: Design Correctness

- 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.

- Break up the datapath into subpaths s.t. each subpath starts from a register read (explicit or implicit), has an optional operation, and, except possibly the last subpath, ends with a register write.

- Each subpath $P_i$ needs to be implemented by a 1 or more states (> 1 states in the case of multi-cc operations) that are disjoint from states implementing subpaths on which $P_i$ is data/control dependent, or those that are data/control dependent on $P_i$.

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

- A subpath is dependent on the data produced as a register write in the previous subpath. This data dependency is reflected by their corresponding states following each other in the same order as the subpaths.
CU FSM Design Strategies: Design Correctness (contd.)

- Need to perform data-dependent operations sequentially one after the other (as dictated by the dependency) in disjoint sets of states.

![Diagram of Datapath](image)

(a) Datapath

- One or more states

- Separate/disjoint set of one or more states (disjointness needed because of data dependency)

- ld_r1, ld_r2, ld_r3, ld_r4, ld_r5

(b) Incorrect assertion of control signals

- [r5 = f2(r3, r4)]
  (disjointness needed)

- Hence, disjoint set of one or more states

(c) Correct assertion of control signals

- (default value is inactive or 0)

(d) FSM when f2 delay is 3ccs

- Counting States (can also do using a multipurpose counter)

© Shantanu Dutt, UIC
CU FSM Design Strategies: Design Correctness (contd.)

- Connections to the input of the FU (functional unit, i.e., some combinational logic) need to be maintained for the entire period of the required operation.

\[ r_4 \leftarrow f_1(r_1, r_3) \]

\[ ld_r4=1, \]

\( (1 \text{ cc operation}) \)

(a) Datapath

(b) 1cc operation with selective connection to input of FU

(c) For multi-cc operation, i/ps to FU need to be asserted for all ccs of the operation

- In multiple-cc operations, the write control signal for writing to the destination register should only be valid in the last cc of the operation.
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-overlapped 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 accomplished in 1cc without a race condition problem.

(a) Datapath with feedback

(a) Feedback writing to r2 in 2nd state is OK for edge-triggered or 2-phase (non-overlap clocking) clocked registers
CU FSM Design Strategies: Time Efficiency–Concurrency

- 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 over all operations and reduces the number of states required in the Control Unit.
CU FSM Design Strategies: Time Efficiency–Concurrency (contd.)

(a) Datapath

(b) Correct but time-inefficient

(c) Concurrent implementation of the 2 operations in 1 cc; possible, since no data dependency and no resource conflicts

(d) FSM when f2 delay is 3ccs; in the 1st state f1 and f2 start concurrently/simultaneously
MULTIPLICATION

Serial Multiplication

Add-and-shift (A&S) multiplication:
Manual Example:

\[\begin{array}{c}
1001 = 9 \\
1011 = 11 \\
\hline
00011011 = P \\
\hline
0000 = P_0 \\
+ 1001 \\
\hline
00001001 = P_1 \\
+ 1001 \\
\hline
00011011 = P_2 \\
+ 0000 \\
\hline
00011011 = P_3 \\
+ 1001 \\
\hline
01100011 = P_4 = P = 99
\end{array}\]
If the additions are done one at a time, we obtain a sequence of partial products $P_0, P_1, \ldots, P_n$.

Each partial product $P_{i+1}$ is obtained as

\[
P_0 = 0
\]

\[
P_{i+1} = P_i + 2^i (x_i \cdot Y)
\]

Thus

\[
P_n = \sum_{i=0}^{n-1} 2^i (x_i \cdot Y) = Y \cdot \sum_{i=0}^{n-1} 2^i x_i = Y \cdot X
\]

where $X$ is the multiplier and $Y$ the multiplicand.
A&S multiplication (contd.):

The same effect as shifting the multiplicand left \((2^i(x_i.Y))\) can be achieved by keeping the multiplicand fixed at the left-most position and shifting the partial product \(P_i\) right.

Example:

\[
\begin{array}{c}
1001 = 9 \\
1011 = 11 \\
\hline
00000000 = P_0 \\
1001 \\
\hline
10010000
\end{array}
\]

\[
\begin{array}{c}
\text{SHR: } 01001000 = P_1 \\
1001 \\
\hline
11011000
\end{array}
\]

\[
\begin{array}{c}
\text{SHR: } 01101100 = P_2 \\
0000 \\
\hline
01101100
\end{array}
\]

\[
\begin{array}{c}
\text{SHR: } 00110110 = P_3 \\
1001 \\
\hline
11000110
\end{array}
\]

\[
\begin{array}{c}
\text{SHR: } 01100111 = P_4 = P = 99
\end{array}
\]

In this case, the partial products obtained are:

\[
P_0 = 0, \quad P_{i+1} = 2^{-1}(P_i + 2^n(x_i.Y))
\]

However, \(P_n\) is the same:

\[
P_n = \sum_{i=0}^{n-1} 2^i(x_i.Y)
\]
A&S multiplication (contd.):  

Hardware:

Algorithm:
Initialize $AC = 0$; $Q =$ Multiplier; $M =$ Multiplicand.
Do the following steps $n$ times
{
If LSB of $Q$ is 1 then $AC = AC + M$ else $AC = AC$;
Shift $C_{out}$-$AC$-$Q$ register combination right by 1 bit
}
Final product is in $AC$-$Q$ register
NOTE: Overflows are tolerated in the additions—$C_{out}$ is fed to the MSB of $AC$ when right shifting
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](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.
Behavioral VHDL
RASSP Education & Facilitation
Module 12
Version 2.01

Copyright © 1995-1998 RASSP E&F

All rights reserved. This information is copyrighted by the RASSP E&F Program and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the RASSP E&F Program is prohibited. All information contained herein may be duplicated for non-commercial educational use provided this copyright notice is included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

FEEDBACK:
The RASSP E&F Program welcomes and encourages any feedback that you may have including any changes that you may make to improve or update the material. You can contact us at feedback@rassp.scca.org or http://rassp.scca.org/module-request/FEEDBACK/feedback-on-modules.html

Acknowledgement (this and next 7 slides): 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.
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.
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.
Appendix: VHDL Codes for Datapath (Structural) and Controller FSM (Behavioral) for Unsigned 8-bit Multiplication
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 => multiplier(0 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.
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).