A Verilog Designer's Experience Designing In Esterel

Dan Downs
Esterel EDA Technologies
679 av. Dr. J. Lefebvre
06270 Villeneuve-Loubet, France

www.esterel-eda.com

March 12, 2008

This document presents some of the real world coding benefits of using Esterel that might not be obvious in a discussion about Esterel as a design or Electronics Systems Level (ESL) language. In particular, instead of being an Esterel Language tutorial or presenting Verilog versus Esterel *final* code side by side, this document walks through the process of designing a small control circuit in the two languages. It is about the process of *writing* of the code and about how Esterel allows one to remain focused on the behaviour of the circuit to be designed. It gives an illustration of what Esterel Technologies is talking about when we say that Esterel code is easier to read and understand and maintain.

1 The Function to be Implemented

Imagine a device used to control someone's exercise routine. Mainly the exercise is running, so we will call this device a "runner controller", or *runner* for short. This could be, for example, a handheld microelectronic Personal Trainer.

The exercise routine is simple. *Every morning do 4 laps of: 100 Meters of slow running, 15 seconds of simultaneous jumping and breathing at each step, then finish the lap running fast.*

This example was actually taken from documents about the Esterel language where the point of the example was to present the Multiform nature of time representation in Esterel. It is known as "runner" and it has a long history in Esterel documentation, appearing first in about 1985. I simply wanted to see how this would look in Verilog, and this paper presents some of my observations.

2 Beginning the Implementation

As a hardware engineer, when I think about coding this in Verilog (or VHDL) I immediately note the primary inputs and outputs and I see a state machine and three event counters: one for meters, one for seconds, and one for laps. However when I think about coding this in Esterel, I write pretty much what the specification says. In the box below

are the primary inputs and outputs. But for brevity, in the remainder of the document I will leave off the port list and input/output declarations for both Verilog and Esterel:

```
input  reset, clk ;
input  Morning, Second, Meter, Step, Lap ;
output RunSlowly, Jump, Breathe, RunFast ;
```

Here is the code in Esterel:

```
every Morning do
    abort // 4 laps
        loop
            abort
                sustain RunSlowly
            when 100 Meter ;

            abort
                every Step do
                    emit {Jump, Breathe}
                end every
            when 15 Second ;

            sustain RunFast
        each Lap
    when 4 Lap
end every
```

That's it for the Esterel code, it is complete and done. And my manager is happy because it only took 20 minutes to code.

Now before looking at the entire Verilog code, let us look at just the two lines of Esterel code which create a lap counter:

```
    abort // 4 laps
    when 4 Lap
```

This is what needs to be done in Verilog to capture the same thing:

```
reg  [1:0] lap_count ;   // Max   4
reg  DisableLapCounter ;
-----------------------------------------------------------------
// 4 lap counter from !DisableLapCounter
always @ (posedge clk) begin
    if(reset || (DisableLapCounter == 1))begin
        lap_count <= 0 ;
    end
    else if (Lap) begin
        lap_count <= lap_count + 1 ;
    end
end
-----------------------------------------------------------------
      RESET_STATE : begin
                DisableLapCounter <= 1 ;
...
      RUN_SLOWLY : begin
                DisableLapCounter <= 0 ;
...
      RUN_FAST : begin
        else if (Lap) begin
            if (lap_count >= 4) begin
                STATE <= AWAIT_MORNING ;
                RunFast           <= 0 ;
            end
-----------------------------------------------------------------
parameter RESET_STATE   = 0 ;
parameter AWAIT_MORNING = 1 ;
parameter RUN_SLOWLY    = 2 ;
parameter JUMP_BREATHE  = 3 ;
parameter RUN_FAST      = 4 ;

reg  [2:0] STATE ;
```

So I have had to do the following:

- Explicitly create a counter with the right number of bits,
- Create a signal for controlling the counter,
- Create a state machine for controlling that signal,

The first two items have to be done again for each counter in the design.

My first observation is that right away I am putting distance between the function I am trying to implement and what I am actually doing at the keyboard. The real task is not *about* creating counters and state machines, it is about capturing a given behaviour. So in an important way the *explicit coding* of these counter and state machine functional units is a distraction from the task at hand. While I am taking the time to create these functional blocks I might as well be working on any one of thousands of other designs since they are not unique to the function to be implemented.

Earlier I said, "let us look at just the two lines of Esterel code which create a lap counter". But in fact there is much more than that going on in the Esterel code. The

structure of the Esterel code around and within those two lines implicitly describes a state machine; the details of which I do not have to concern myself with.

I can parameterize the three counters in this design but I cannot parameterize the STATE register. It is only after I have sketched a bubble diagram that I can know how many bits the STATE register has. Again, in the Esterel implementation I never need to care about this detail.

After making the sketch and knowing the states however, and thinking about how and when I will start and stop the three counters, I have written the following Verilog code:

```
parameter RESET_STATE   = 0 ;
parameter AWAIT_MORNING = 1 ;
parameter RUN_SLOWLY    = 2 ;
parameter JUMP_BREATHE  = 3 ;
parameter RUN_FAST      = 4 ;

reg  [2:0] STATE ;

reg  [7:0] meter_count ; // Max 100
reg  [3:0] JB_time ;     // Max  15
reg  [1:0] lap_count ;   // Max   4

reg  RunSlowly ;
reg  Enable_JB_timer ;
reg  DisableLapCounter ;

//=============================================
// Timers and counters

// 15 second timer from Enable_JB_timer
always @ (posedge clk) begin
    if (reset || (Enable_JB_timer == 0)) begin
        JB_time <= 0 ;
    end
    else if (Second) begin
        JB_time <= JB_time + 1 ;
    end
end

// 100 meter counter from RunSlowly
always @ (posedge clk) begin
    if (reset || (RunSlowly == 0)) begin
        meter_count <= 0 ;
    end
    else if (Meter) begin
        meter_count <= meter_count + 1 ;
    end
end

// 4 lap counter from !DisableLapCounter
always @ (posedge clk) begin
    if(reset || (DisableLapCounter == 1))begin
        lap_count <= 0 ;
    end
    else if (Lap) begin
        lap_count <= lap_count + 1 ;
    end
end
```

Notice that I have not even started to code the actual state machine yet. This sort of preparation work is not necessary in the Esterel version as all the counter and timer "code" above, plus the state machine, is implicitly described.

3 On to the State Machine

Now for the actual Verilog state machine controller for *runner*. Notice that I have added a comment in the Verilog code as documentation describing the function to be implemented.

```
//===========================================
// Main controller State Machine
//
/* Function to be implemented
every Morning do
    abort // 4 laps
        loop
            abort
                sustain RunSlowly
            when 100 Meter ;

            abort
                every Step do
                    emit {Jump, Breathe}
                end every
            when 15 Second ;

            sustain RunFast
        each Lap
    when 4 Lap
end every
*/
//===========================================
//
always @ (posedge clk) begin // Main SM
  if (reset) begin
                STATE <= RESET_STATE ;
  end
  else
    case (STATE)
      //-------------------------------------
      RESET_STATE : begin
                DisableLapCounter <= 1 ;
                Enable_JB_timer   <= 0 ;

                RunSlowly         <= 0 ;
                RunFast           <= 0 ;
                Jump              <= 0 ;
                Breathe           <= 0 ;

                STATE <= AWAIT_MORNING ;
      end

      //-------------------------------------
      AWAIT_MORNING : begin
                DisableLapCounter <= 1 ;
                Enable_JB_timer   <= 0 ;

        if      (Morning) begin
```

```
            STATE <= RUN_SLOWLY ;
            RunSlowly          <= 1 ;
      end
      else begin
            STATE <= AWAIT_MORNING ;
      end
    end

    //--------------------------------------
    RUN_SLOWLY : begin
            DisableLapCounter <= 0 ;
            Enable_JB_timer   <= 0 ;

      if    (Morning) begin
            STATE <= RUN_SLOWLY ;
      end
      else if (meter_count >= 100) begin
            STATE <= JUMP_BREATHE ;
            RunSlowly          <= 0 ;
            Jump               <= 1 ;
            Breathe            <= 1 ;
      end
      else begin
            STATE <= RUN_SLOWLY ;
      end
    end

    //--------------------------------------
    JUMP_BREATHE : begin
            DisableLapCounter <= 0 ;
            Enable_JB_timer   <= 1 ;

      if    (Morning) begin
            STATE <= RUN_SLOWLY ;
            Jump               <= 0 ;
            Breathe            <= 0 ;
      end
      else if (JB_time == 15) begin
            STATE <= RUN_FAST ;
            RunFast            <= 1 ;
            Jump               <= 0 ;
            Breathe            <= 0 ;
      end
      else if (Step && (JB_time < 15)) begin
            STATE <= JUMP_BREATHE ;
            Jump               <= 1 ;
            Breathe            <= 1 ;
      end
      else begin
            STATE <= JUMP_BREATHE ;
            Jump               <= 0 ;
            Breathe            <= 0 ;
      end
    end
```

```
      //--------------------------------------
      RUN_FAST : begin
                 DisableLapCounter <= 0 ;
                 Enable_JB_timer   <= 0 ;

        if       (Morning) begin
                 STATE <= RUN_SLOWLY ;
        end
        else if (Lap) begin
            if (lap_count >= 4) begin
                 STATE <= AWAIT_MORNING ;
                 RunFast           <= 0 ;
            end
            else begin
                 STATE <= RUN_SLOWLY ;
                 RunSlowly         <= 1 ;
                 RunFast           <= 0 ;
            end
        end
        else begin
                 STATE <= RUN_FAST ;
        end
      end

      //--------------------------------------
      default : begin
                 DisableLapCounter <= 0 ;
                 Enable_JB_timer   <= 0 ;

        if       (Morning) begin
                 STATE <= RUN_SLOWLY ;
                 RunSlowly         <= 1 ;
        end
        else begin
                 STATE <= RESET_STATE ;
        end
      end

  endcase
end // Main SM
```

So now I have my state machine and the timers and controllers and I can attempt a compile and simulation. But first, look at what has been done. Creating a state machine like this is, again, off the main line of what I am actually trying to accomplish. I spent most of my time observing Verilog syntactic rules and code structuring and copying-and-pasting and scrolling ... in short, editing and typing. I feel like a typist that knows Verilog, not a hardware designer. I am beginning to wonder if there is a better, more direct way to code this.

Even if you have perl scripts or Emacs macros or other methods to lessen the pain of creating state machines and other regularly used functional blocks in Verilog, the fact remains that these kind of implementation details do not even have to be considered in the Esterel implementation. I do not want to make the "compare the Esterel and Verilog

code lengths" argument too strongly here, but in addition to the "fewer lines means fewer errors" and "fewer lines are easier to understand" arguments, we could add that it was tedious to code that in Verilog. Most of it had little to do with capturing the particular behaviour of the function of interest and much to do with building generic scaffolding onto which we could hang the particular behaviour.

Pausing briefly, I will note that the Esterel code:

- directly captures the behaviour desired
- is mostly self commenting
- is shorter
- does not require explicit state machine creation
- creates implied counters automatically

4 The Legal Department Arrives

It is a good thing I paused. I have just been informed of a specification change. The Legal department says that if the runner controller does not include a cool-down phase then we are setting ourselves up for a lawsuit. The new specification is *Every morning do 4 laps of: (100 Meters of slow running, 15 seconds of simultaneous jumping and breathing at each step, then finish the lap running fast) and after the 4 laps walk for 1 lap then stretch for 5 minutes.*

Ok I can do that quickly in Esterel. A side-by-side diff of the first version of the Esterel code versus the second version gives me this:

```
every Morning do                          every Morning do
  abort // 4 laps                           abort // 4 laps
    loop                                      loop
      abort                                     abort
        sustain RunSlowly                         sustain RunSlowly
      when 100 Meter ;                          when 100 Meter ;

      abort                                     abort
        every Step do                             every Step do
          emit {Jump,Breathe}                       emit {Jump,Breathe}
        end every                                 end every
      when 15 Second ;                          when 15 Second ;

      sustain RunFast                           sustain RunFast
    each Lap                                  each Lap
  when 4 Lap                                when 4 Lap

                                          >   abort
                                          >   sustain Walk
                                          >   when 1 Lap ;
                                          >
                                          >   abort
                                          >   sustain Stretch
                                          >   when 300 Second ;
                                          >
end every                                 end every
```

The changes are not complicated as they follow in sequence after the exercise routine.

Now in Verilog I first add a WALK and a STRETCH state. That makes 7 states total so I can leave STATE at 3 bits. I don't need to add a lap counter, but I'll add a another seconds counter. I do need to modify the control of the lap counter however. I copy and paste twice the last state in the state machine, do some editing, and end up with a new version. Below is the side by side diff between the two Verilog versions. To keep the main part of this document shorter however, only the changed lines are shown:

```
                                    >  parameter WALK          = 5 ;
                                    >  parameter STRETCH       = 6 ;
                                    >  reg  [9:0] stretch_time ;// Max 300
-------------------------------------------------------------------------------------
                                    >  reg  En_Stretch_timer ;
                                    >  reg  Walk ;
                                    >  reg  Stretch ;
-------------------------------------------------------------------------------------
// 4 lap counter from !DisableLapCounter|  // 5 lap counter from !DisableLapCounter
                                    >  // 300 seconds from En_Stretch_timer
                                    >  always @ (posedge clk) begin
                                    >      if (reset || (En_Stretch_timer == 0))
                                    >                                      begin
                                    >          stretch_time <= 0 ;
                                    >      end
                                    >      else if (Second) begin
                                    >          stretch_time <= stretch_time + 1;
                                    >      end
                                    >  end
-------------------------------------------------------------------------------------
                                    >                  En_Stretch_timer  <= 0 ;
                                    >                  Walk              <= 0 ;
                                    >                  Stretch           <= 0 ;
                                    >                  En_Stretch_timer  <= 0 ;
              STATE <= AWAIT_MORNING ;|                STATE <= WALK ;
                                    >                  Walk              <= 0 ;
-------------------------------------------------------------------------------------
                                    >        WALK : begin
                                    >          if      (Morning) begin
                                    >                  STATE <= RUN_SLOWLY ;
                                    >          end
                                    >          else if (lap_count == 5) begin
                                    >                  STATE <= STRETCH ;
                                    >                  En_Stretch_timer  <= 1 ;
                                    >                  Walk              <= 0 ;
                                    >                  Stretch           <= 0 ;
                                    >          end
                                    >          else begin
                                    >                  STATE <= WALK ;
                                    >          end
                                    >        end
                                    >        //-------------------------------
                                    >        STRETCH : begin
                                    >                  En_Stretch_timer  <= 1 ;
                                    >          if      (Morning) begin
                                    >                  STATE <= RUN_SLOWLY ;
                                    >          end
                                    >          else if (stretch_time >= 300)
                                    >                                      begin
                                    >                  STATE <= AWAIT_MORNING ;
                                    >                  Stretch           <= 0 ;
                                    >          end
                                    >          else begin
                                    >                  STATE <= STRETCH ;
                                    >          end
                                    >        end
```

There seems to be far more to keep track of in the Verilog code than in the Esterel. Adding two states required modification all through the code. And this is a very simple modification to a very simple module.

At this point I can add to my list of observations that in the Esterel code:

- The changes required to update the code follow closely the specification updates.

The reason that the changes to the code were localized and not spread around is that

the Esterel code captured the behaviour whereas the Verilog codes an implementation. With Esterel you have the architecture and the implementation in one.

I am ready to press on and try a compile, but ...

5 Oh No. Here Comes That Marketing Guy Again

It looks like he's been reading *Crosstraining Weekly* again. This means another specification change I just know it.

And today's specification is: *Every morning do 4 laps of:*

```
(100 Meters of slow running,
(
    Odd laps:  (15 seconds of simultaneous jumping and breathing at each step),
    Even laps: (25 seconds of star jumps),
)
then finish the lap running fast) and after the 4 laps walk for 1 lap then
stretch for 5 minutes
```

I have pointed out that with the Esterel code I have not had to explicitly create the counters or timers. But that was because it was not necessary in order to capture the behaviour of the specification. Looking at the kind of specification change that just came in I have two options. I can make a lap toggle bit to determine odd or even laps, or I can predict that tomorrow there will be another change related to the lap count. I'll assume the latter, and so now I will explicitly create a lap counter and change the Esterel code accordingly:

```
                                    > /*
                                    > Upper limit to the Lap count is
                                    >   0 seconds to run 100 meters (faker!)
                                    >   (15 + 25)/2 seconds/lap (make them wait)
                                    >   0 seconds to finish lap (exercise cheat!)
                                    >   Seconds/day 60*60*24 = 86400 (next Morning)
                                    >   is 86400 / ((15+25)/2) = 4320
                                    > */

                                    > signal
                                    > LapCount    : value unsigned<4320> init 1
                                    > in

every Morning do                      every Morning do
                                    >   sustain {
                                    >     if (Morning and not pre(Morning)) then
                                    >       ?LapCount <= 1
                                    >     else
                                    >       ?LapCount <= (pre(?LapCount) + 1) if Lap
                                    >     end if
                                    >   }
                                    >   ||

  abort // 4 laps                     abort // 4 laps
    loop                                loop
      abort                               abort
          sustain RunSlowly                   sustain RunSlowly
      when 100 Meter ;                    when 100 Meter ;

                                    >         if ((?LapCount mod 2) = 1) then
          abort                               abort
            every Step do                          every Step do
                emit {Jump, Breathe}                    emit {Jump, Breathe}
            end every                           end every
          when 15 Second ;                    when 15 Second ;

                                    >         else
                                    >           abort
                                    >             every Step do
                                    >                 emit Starjump
                                    >             end every
                                    >           when 25 Second ;
                                    >         end if;
                                    >
      sustain RunFast                       sustain RunFast
    each Lap                              each Lap
  when 4 Lap ;                          when 4 Lap ;

  abort                                 abort
    sustain Walk                          sustain Walk
  when 1 Lap ;                          when 1 Lap ;

  abort                                 abort
    sustain Stretch                       sustain Stretch
  when 300 Second ;                     when 300 Second ;
end every                             end every
                                    > end signal
```

Some explanation of Esterel seems necessary at this point. Please refer to the added code in the box above (just the parts on the right with the '>' markers).

The added part in the box below declares a counter, and sets its scope between the *signal* and *end signal* keywords. I think of the first lap as lap number 1, not zero, so I set it to 1 initially.

```
signal
LapCount     : value unsigned<4320> init 1
in
...
end signal
```

This part creates a design unit in parallel (using the || operator) with the existing main code.

```
  sustain {
  }
  ||
```

This part is the counter control. The two *pre* operators give the value of a signal in the previous cycle; so that's just a convenient way to detect the rising edge of Morning and reset the counter in the first case, and it is necessary in the second case to prohibit reading and writing to LapCount in the same cycle in the second case. The ? means "the value of" and is there to distinguish the times when you want to know a signal's value as opposed to its status (if it is true or false):

```
    // Detect only new Mornings
    if (Morning and not pre(Morning)) then
      ?LapCount <= 1
    else
      // LapCount=(previous cycle's LapCount)+1
      ?LapCount <= (pre(?LapCount)+1) if Lap
    end if
```

This is the logic for checking the even/oddness of the lap. Tomorrow when the new specification comes in I'll change it to "if ?LapCount ¡= NewBehaviourX" (or whatever):

```
        if ((?LapCount mod 2) = 1) then
```

And this is the new requested behaviour:

```
        else
          abort
            every Step do
                emit Starjump
            end every
          when 25 Second ;
        end if;
```

The comparable Verilog changes are not so straightforward. The Verilog code already has a Lap counter and since I have direct access to lap_count bit 0, I can use that bit to test for even/oddness. But it is the state machine, again, that I am not looking forward to editing. I've got to tear it up a bit, attending to all the states that transitioned *to* the JUMP_BREATHE state and add the if test to the new state STARJUMP and etc. It is not much, but this is the sort of place where bugs creep in. Copying and pasting and editing to manage the multiple state machine transitions where I need to turn on and off signals.

Ok so why am I complaining. Isn't this is part and parcel of the normal ordinary workday of an HDL designer? Maybe, but it is not part of the ordinary workday of a

designer writing in Esterel. In the Esterel code I squeezed in the new behaviour just where I wanted it without ever thinking about the state machine that will implement it. In Esterel I focused on capturing the behaviour. For the act of coding in Verilog however, that editing becomes the main task. In Verilog I instead focused on the construction of a state machine which captures the behaviour.

Here are the changes required in the Verilog version:

```
parameter RESET_STATE   = 0 ;
parameter AWAIT_MORNING = 1 ;
parameter RUN_SLOWLY    = 2 ;
parameter JUMP_BREATHE  = 3 ;
parameter RUN_FAST      = 4 ;
parameter WALK          = 5 ;
parameter STRETCH       = 6 ;


reg  [2:0] STATE ;

reg  [7:0] meter_count ; // Max 100
reg  [3:0] JB_time ;     // Max  15
reg  [2:0] lap_count ;   // Max   5
reg  [9:0] stretch_time ;// Max 300


reg  RunSlowly ;
reg  Jump ;
reg  Breathe ;
reg  RunFast ;
reg  Enable_JB_timer ;
reg  DisableLapCounter ;
reg  En_Stretch_timer ;
reg  Walk ;
reg  Stretch ;

//=====================================
// Timers and counters
                          --------- NO CHANGE -----------
//=====================================
//        MAIN Main main
//
always @ (posedge clk) begin // Main SM
  if (reset) begin
               STATE <= RESET_STATE ;
  end
  else
    case (STATE)
    //--------------------------------
    RESET_STATE : begin
              DisableLapCounter <= 1 ;
              Enable_JB_timer   <= 0 ;
              En_Stretch_timer  <= 0 ;

              RunSlowly         <= 0 ;
              RunFast           <= 0 ;
              Jump              <= 0 ;
              Breathe           <= 0 ;
              Walk              <= 0 ;
              Stretch           <= 0 ;

              STATE <= AWAIT_MORNING ;
    end

    //--------------------------------
    AWAIT_MORNING : begin
                          --------- NO CHANGE -----------
    //--------------------------------
    RUN_SLOWLY : begin
              DisableLapCounter <= 0 ;

      if      (Morning) begin
              STATE <= RUN_SLOWLY ;
      end
```

```
parameter RESET_STATE   = 0 ;
parameter AWAIT_MORNING = 1 ;
parameter RUN_SLOWLY    = 2 ;
parameter JUMP_BREATHE  = 3 ;
parameter RUN_FAST      = 4 ;
parameter WALK          = 5 ;
parameter STRETCH       = 6 ;
> parameter STARJUMP     = 7 ;


reg  [2:0] STATE ;

reg  [7:0] meter_count ; // Max 100
| reg  [4:0] JB_time ;     // Max  25
reg  [2:0] lap_count ;   // Max   5
reg  [9:0] stretch_time ;// Max 300


reg  RunSlowly ;
reg  Jump ;
reg  Breathe ;
reg  RunFast ;
reg  Enable_JB_timer ;
reg  DisableLapCounter ;
reg  En_Stretch_timer ;
reg  Walk ;
reg  Stretch ;
> reg  Starjump ;

//=====================================
// Timers and counters
                          --------- NO CHANGE -----------
//=====================================
//        MAIN Main main
//
always @ (posedge clk) begin // Main SM
  if (reset) begin
               STATE <= RESET_STATE ;
  end
  else
    case (STATE)
    //--------------------------------
    RESET_STATE : begin
              DisableLapCounter <= 1 ;
              Enable_JB_timer   <= 0 ;
              En_Stretch_timer  <= 0 ;

              RunSlowly         <= 0 ;
              RunFast           <= 0 ;
              Jump              <= 0 ;
              Breathe           <= 0 ;
              Walk              <= 0 ;
              Stretch           <= 0 ;
>             Starjump          <= 0 ;

              STATE <= AWAIT_MORNING ;
    end

    //--------------------------------
    AWAIT_MORNING : begin
                          --------- NO CHANGE -----------
    //--------------------------------
    RUN_SLOWLY : begin
              DisableLapCounter <= 0 ;

      if      (Morning) begin
              STATE <= RUN_SLOWLY ;
      end
```

```
      else if (meter_count >= 100) begin
                                        >
              STATE <= JUMP_BREATHE ;
              RunSlowly         <= 0 ;
              Jump              <= 1 ;
              Breathe           <= 1 ;
                                        >
                                        >
                                        >
                                        >
                                        >
      end
      else begin
              STATE <= RUN_SLOWLY ;
      end
    end

    //---------------------------------
    JUMP_BREATHE : begin
                           --------- NO CHANGE -----------
    //---------------------------------
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
                                        >
    RUN_FAST : begin
                           --------- NO CHANGE -----------
    //---------------------------------
    WALK : begin
                           --------- NO CHANGE -----------
    //---------------------------------
    STRETCH : begin
                           --------- NO CHANGE -----------
    //---------------------------------
    default : begin
                           --------- NO CHANGE -----------
  endcase
end // Main SM
```

```
      else if (meter_count >= 100) begin
          if (lap_count[0] == 0) begin
              STATE <= JUMP_BREATHE ;
              RunSlowly         <= 0 ;
              Jump              <= 1 ;
              Breathe           <= 1 ;
          end
          else begin
              STATE <= STARJUMP ;
              RunSlowly         <= 0 ;
              Starjump          <= 1 ;
          end
      end
      else begin
              STATE <= RUN_SLOWLY ;
      end
    end

    //---------------------------------
    JUMP_BREATHE : begin
    //---------------------------------
    STARJUMP : begin
              Enable_JB_timer   <= 1 ;

      if      (Morning) begin
              STATE <= RUN_SLOWLY ;
              Jump              <= 0 ;
              Breathe           <= 0 ;
      end
      else if (Lap) begin
              STATE <= RUN_SLOWLY ;
              Enable_JB_timer   <= 0 ;
              Starjump          <= 0 ;
      end
      else if (JB_time >= 25) begin
              STATE <= RUN_FAST ;
              RunFast           <= 1 ;
              Starjump          <= 0 ;
      end
      else if (Step) begin
              STATE <= STARJUMP ;
              Starjump          <= 1 ;
      end
      else begin
              STATE <= STARJUMP ;
              Starjump          <= 0 ;
      end
    end

    //---------------------------------
    RUN_FAST : begin
    //---------------------------------
    WALK : begin
    //---------------------------------
    STRETCH : begin
    //---------------------------------
    default : begin
  endcase
end // Main SM
```

6 Tomorrow Indeed Brings A New Specification Change

From what I have presented above there may not seem much to complain about. But this is a very, very simple design which has no interaction with other control logic as it would in a real product design. All the arguments for creating unambiguous, clear and easy to understand code become increasingly stronger when the design becomes increasingly complex.

And as I predicted, today the Personal Trainer device specification changed. Now it is going to be used on a par course. The original middle part of the runner's routine, the Jump and Breathe section, will have any one of 19 different routines of various sorts and times. This means that in the Verilog version I will have a greatly complicated state machine, or more sensibly, I will create a separate state machines for those routines. I will have to add control logic to coordinate the two state machines also. I am not going to do this exercise. But I will show the one line Esterel solution to manage the overhead of state machine to state machine coordination:

```
every Morning do
  abort // 4 laps
    loop
      abort
        sustain RunSlowly
      when 100 Meter ;

      abort
          run ParRoutine   // This line here.
      when ParRoutineComplete ;

      sustain RunFast
    each Lap
 when 4 Lap

 abort
 sustain Walk
 when 1 Lap ;

 abort
 sustain Stretch
 when 300 Second ;

end every
```

In a separate file I can describe the par course routines. The module name will be ParRoutine. That `run ParRoutine` line will effectively embed the ParRoutine module controller inside the runner. I will not worry, as with an HDL construction, about creating specific signals to enable and/or start or stop the `ParRoutine` state machine or in any other way coordinate its activities with the main state machine.

7 In Conclusion

When writing the code in Esterel I let the desired behaviour drive the coding. In Verilog, I let the Verilog HDL itself drive the coding. So I've decided what I'll do for future

designs. Since I always draw out bubble diagrams and/or write pseudocode in an effort to turn the paper specification into something implementable, now I will always rewrite the specification in Esterel as a first step. If it is a datapath dominated design, or if the design has trivial control requirements, I may choose to implement that design, or parts of it, in Verilog. But if it is a control dominated design, or has complex control requirements, then I'll do the implementation in Esterel. The benefit of the latter case is, of course, that the rewritten specification *is* the implementation. So then I'm done and I can move on to the testing phase.