Blue Pearl Multi-cycle Path Detection

A path is a sequence of logic elements through which data can propagate, bounded by either ports or registers. Most paths bounded by registers associated with a single clock must take no more than one cycle to settle, and synthesis tools can spend considerable effort to optimize long paths that do not meet this specification. Synthesis tools allow the user to declare a path as a multi-cycle path. This allows for much less optimization of the combinational logic on the path, since it has more than one cycle to settle. Less optimization in turn leads to lower run times.

While it is possible to arbitrarily declare a long path as multi-cycle, designers should ensure that the surrounding logic is immune to data-dependent or silicon-dependent variations in settling time which could lead to unpredictable results. The Blue Pearl Software Suite is an RTL analysis tool that will find functional multi-cycle paths that are difficult to find by other means.

For our purposes, the general definition of a multi-cycle path is a register-to-register path with data retention at each end, where there is a detectable minimum number of cycles greater than one between the two ends.

There are various ways to control the data retention at each end of the path, and there are a number of variations in which data is not strictly retained at a path end. The tool needs to account for enough of these variations to cover most real-life design practices. The tool must also be able to account for or ignore situations that may appear to require single-cycle optimization but that the designer does not care about, such as initialization sequences.

The Blue Pearl Software Suite's Create tool will identify multi-cycle paths with data retention controlled by adders, counters, and other cyclic signals (those that follow a fixed repeating pattern) as well as those with data retention controlled by FSMs. It also recognizes any of these wherein the control is pipelined to generate a multi-cycle path of more than two cycles, as well as situations wherein the end of the path is shared by two multi-cycle paths.

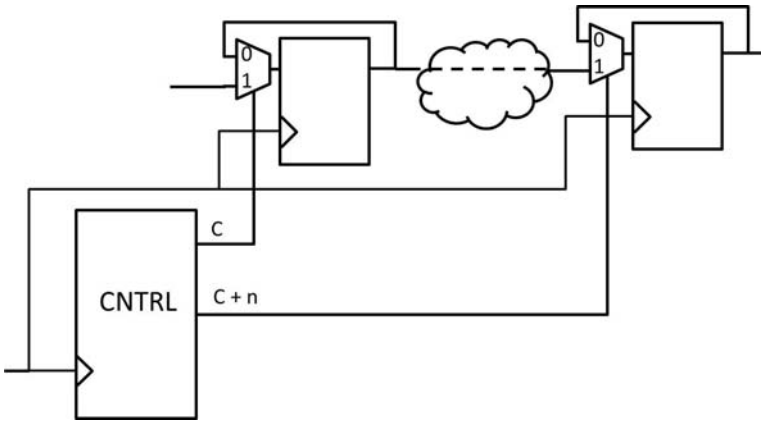

Figure 1

C is decoded from a counter, adder, or FSM and is a pulse one cycle wide.

C+n is offset from C by two or more cycles and is created by a pipeline, or a separate decode of a FSM or counter, or by a different adder with a greater constant. It is also one cycle wide. In the case of a cyclic signal that pulses once every *n* cycles, C and C+n could be the same signal.

Examples:

The following are examples of multi-cycle paths detected by the Blue Pearl Software Suite.

1. Counter by addition

Possibly the simplest example from a Verilog coding point of view is where the CNTRL function from *Figure 1* is implemented as a counter of the form:

```
always @ (posedge clk or negedge rst_n)
        if(rst_n == 1'b0)
                cntrl <= 2'b00;
        else
                cntrl <= cntrl + 1;

assign c = (cntrl == 2'b11) ? 1'b1 : 1'b0;
```

Here, C and C+n from *Figure 1* are the same signal, pulsing once every fourth cycle.

2. Gate-level counter

A slightly different implementation of the same idea utilizes a gate-level implementation of a counter.

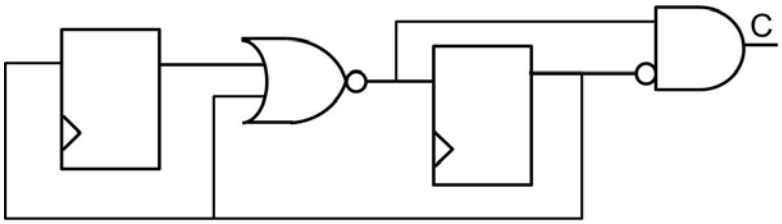

Figure 2

Once again, C and C+n are the same signal, this time pulsing every third cycle.

3. Finite state machines (FSMs)

The Blue Pearl Software Suite will also recognize multi-cycle paths controlled by FSMs. A simple state machine can be implemented by a case statement.

```
case (state)
        s0: begin
                nextstate <= s1;
        end
        s1: begin
                nextstate <= s2
        end
…


        s7: begin
                nextstate <= s0
        end
```

In one implementation, the FSM is treated as quasi-cyclic and the same state is used for both C and C+n.

```
assign c = (state == s4) ? 1'b1 : 1'b0;
```

In another implementation, two states separated by at least one other state are used as C and C+n, respectively.

```
assign c  = (state == s3) ? 1'b1 : 1'b0;
assign cn = (state == s6) ? 1'b1 : 1'b0;
```

4. Pulse generator

A simple example of a non-cyclic multi-cycle path involves a synchronous pulse generator and a shift register.

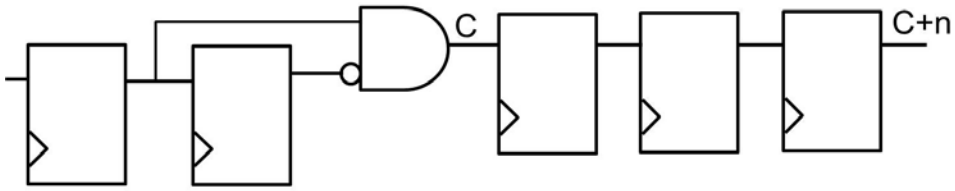

Figure 3

This circuit generates a one-cycle pulse from an edge to create C, and then delays it by three cycles to get C+n.

Detection of multi-cycle paths with pulse generators requires a non-default option in the tool, specifically the *Design Contains Pulse Generators* option found in the *Analysis Settings -> False and Multi-cycle Path Analysis Options* menu for GUI users or the `analysis pulse_generators` option in the **bluepearl.cfg** file for command line users.

5. Distributed control and multiple control

Other variations that the Blue Pearl Software Suite takes into account involve the presence of a more complex data retention scheme with more than one means of control.

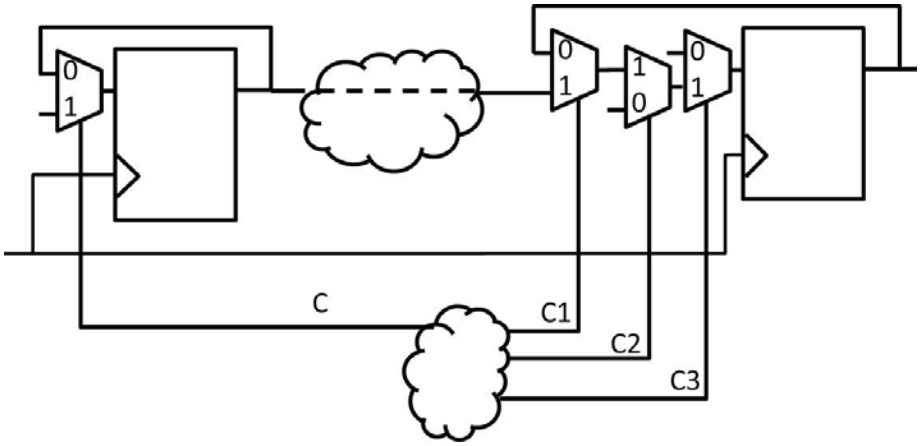

Figure 4

In cases where C, C1, C2, and C3 can be traced to a common source, use *Analysis Settings -> False and Multi-cycle Path Analysis Options -> Handle distributed control signals* from the GUI or insert `analysis distributed_control` into the **bluepearl.cfg** file in command line mode.

In cases where they come from cyclic sources that have a timing relationship, use *Analysis Settings -> False and Multi-cycle Path Analysis Options -> Report multi-cycle paths dependent on multiple aligned cyclic-signals* from the GUI or insert the `analysis distributed_control` option into the **bluepearl.cfg** file in command line mode.

6. Pingpong

There is at least one circumstance where it is not necessary to have data retention at the end of the path. This is when a pair of two cycle multi-cycle paths take turns using the same path endpoint. For obvious reasons this case has become known as the *pingpong*.

Detection of this type of multi-cycle path requires the use of four non-default options in the tool. In the GUI, use all of the following options found in the *Analysis Settings -> False and Multi-cycle Path Analysis Options* menu: *Handle distributed control signals*, *Allow multi-cycle paths without explicit state retention logic at endpoints*, *Report multi-cycle paths dependent on multiple aligned cyclic-signals*, and (not yet in GUI, one of two paths missed without this) *Assume reset only at initialization*. Command line users add the following options to the **bluepearl.cfg** file: `analysis distributed_control`, `analysis no_state_retention_requirement_on_end_points`, `analysis multiply_controlled_endpoints` and `analysis assume_reset_only_at_initialization`.

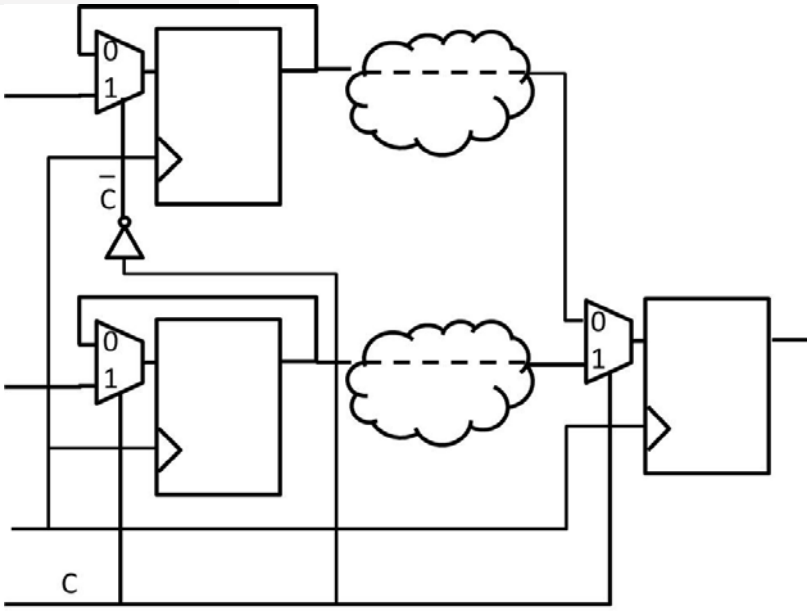

Figure 5

Conclusion:

Correctly identifying multi-cycle paths in your system has a number of benefits. First and foremost, it reduces run times for subsequent synthesis and timing analysis tools. Second, there is an old truism in digital design that bigger is faster. In most cases, synthesis tools use optimization techniques that end up increasing system size, from simply increasing drive strength to utilizing more sophisticated architectures for subsystems such as counters and adders. When the need for optimization is reduced, the overall system is smaller, and the end system saves both die size and power.

The Blue Pearl Software Suite quickly and easily identifies multi-cycle paths and creates SDC constraints for use with a number of different tools and technologies.