Timing Modeling Issues for Cores

This document contains a collection of timing modeling issues for cores and modeling proposals. The proposals suggest to define natural extensions to existing modeling paradigms in ALF and OLA.

1.0 Cycle-dependent timing arcs

Timing analysis must verify whether data transactions on a bus shared amongst multiple devices occur only within certain time slots reserved for each device according to a well-defined protocol. On a synchronous bus this means that certain clock cycles are reserved for certain devices.

Figure 1 shows an example of a device which receives data during cycle #2 and cycle #5 after a synchronization signal and remains idle during the remaining clock cycles. Therefore timing constraints (setup, hold) apply only during cycle #2 and #5. The timing analysis tool should not check the timing constraints during every clock cycle, since eventual timing violations in idle clock cycles would be erroneous.

FIGURE 1. Cycle-dependent timing constraints

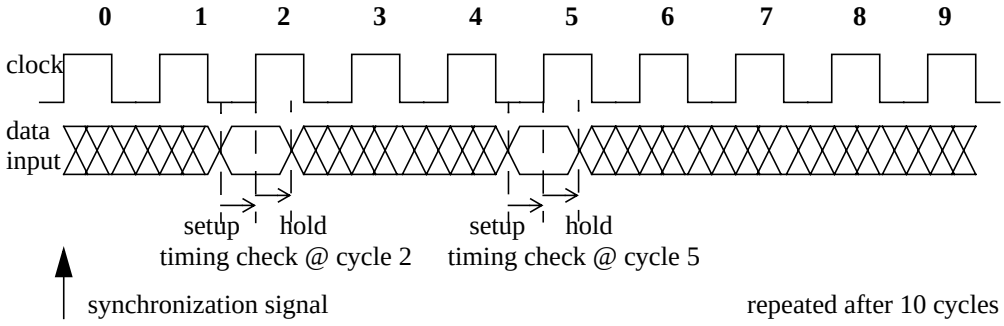
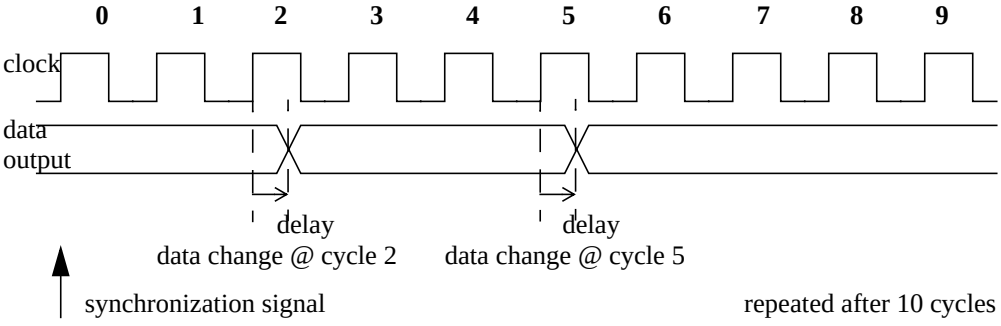


Figure 2 shows an example of a device which emits data during cycle #2 and cycle #5 after a synchronization signal and remains idle during the remaining clock cycles. Therefore the timing data (delay, slewrate) apply only during cycle #2 and #5. The timing analysis tool should not propagate delay and slewrate during every clock cycle, since this erroneous data would eventually generate erroneous timing constraint violations.

FIGURE 2. Cycle dependent delays



The following existing modeling capabilities in ALF and OLA can be used:

- state-dependent timing arcs: a virtual pin is used for the cycle counter.
- signaltype of the involved pins

Example, related to figure 1 and figure 2:

```
CELL my_bus_interface {
   PIN clk { DIRECTION=input; SIGNALTYPE=clock; POLARITY=rising_edge; }
   PIN din { DIRECTION=input; SIGNALTYPE=data; }
   PIN dout { DIRECTION=output; SIGNALTYPE=data; }
   PIN[3:0] cycle_count { VIEW=none; }

   VECTOR (( 01 clk <-> ?! din ) & (cycle_count='d2 | cycle_count='d5)) {
      SETUP {
            FROM { PIN = din; }
            TO { PIN = clk; }
            // fill in the header, table or equation
      }
      HOLD {
            FROM { PIN = din; }
            TO { PIN = clk; }
            // fill in the header, table or equation
      }
   }
   VECTOR (( 01 clk -> ?! dout ) & (cycle_count='d2 | cycle_count='d5)) {
      DELAY {
            FROM { PIN = clk; }
            TO { PIN = dout; }
            // fill in the header, table or equation
      }
      SLEWRATE {
            PIN = dout;
            // fill in the header, table or equation
      }
   }
}
```

This example can be compiled into valid OLA today. The OLA library will have the timing models as well as the associated condition array containing the boolean expression `(cycle_count='d2 | cycle_count='d5).` It will also contain the `POLARITY rising_edge` and `SIGNALTYPE clock` and `data`, respectively, for the involved pins. The boolean expression can also be defined as an `EXISTENCE_CONDITION` for the `VECTOR`, in which case the boolean expression will be an existence condition in OLA as well.

The following information is missing in this representation:

- The fact that `cycle_count` is the output of a counter
- The clock related to the counter
- The counting sequence

This information can be described in ALF today in the FUNCTION, and this description can be mapped into a function graph in OLA.

Example: free running counter, counting from 0 to 9 (% is the modulus operator).

```
FUNCTION {
   BEHAVIOR {
         @ ( 01 clk ) { cycle_count = (cycle_count + 'd1)%'d10; }
   }
}
```

However, a timing analysis tool may not be able to infere the information from the FUNCTION description. Some redundant information should be added to the library to alleviate the need for inference of partial function information.

There is already a precedence in ALF. For instance, the SIGNALTYPE and POLARITY of the pin clk. They are accessible in OLA through dpcmGetPinSignalType and dpcmGetPinPolarity, although they could be infered from the vector_expression (01 clk) in ALF and its corresponding function graph in OLA. Note that a timing analysis tool must consider this information as well, in order to do analysis of clock trees.

The definition of a new value for SIGNALTYPE for identification of the counter would be the most easy and straightforward extension to ALF and OLA.

There is also a precedence in ALF for the definition of related pins: A tristate output pin may have an annotation pointing to the related enable pin. In the same way, an annotation may be defined for a counter pin, pointing to the related clock pin.

On the other hand, there is no precedence for description of the counting sequence other than in the FUNCTION itself. Such a description may be needed for other applications as well. For instance, memory BIST also requires a description of a counting sequence for read and write address generation. Therefore we recommend to collect a more complete set of requirements for core modeling before proposing a new data object and its associated API in ALF and OLA.

2.0 Frequency division and multiplication

Description of frequency division and frequency multiplication is another modeling requirement which relates to counters.

Figure 3 shows an example of a waveform generated by a frequency divider. In timing constraint terminology, the derived signal is not a clock, since it contain more than two edges within the defined range of cycles.

FIGURE 3. Frequency diveiders

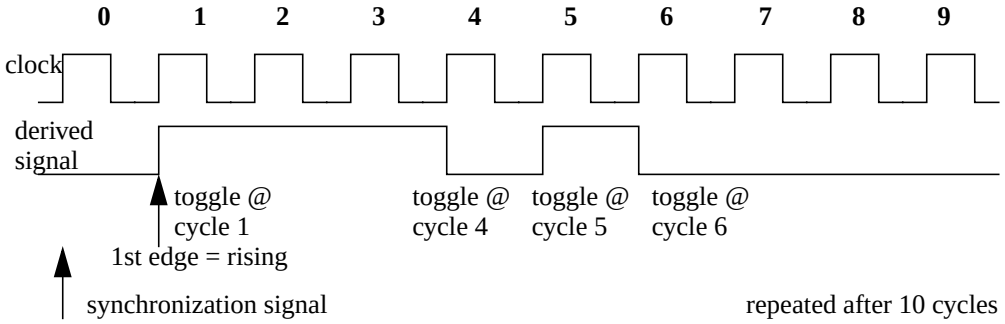
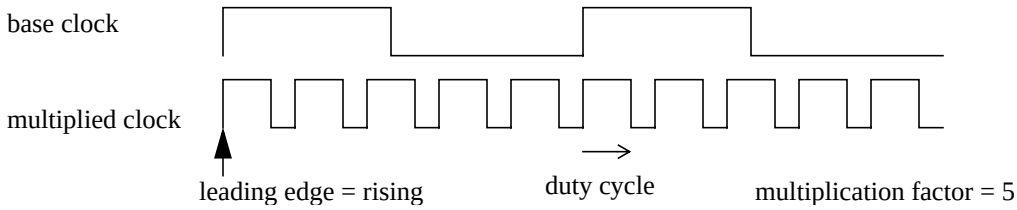


Figure 4 shows the example of a frequency multiplier. The derived signal is a clock in existing timing contraint terminology, since it is completely defined by its frequency and duty cycle. A phase relationship beween the base clock and the multiplied clock can also be described in this terminology.

FIGURE 4. Frequency multipliers



We recommend to review the capabilities and limitations of timing contraint terminology first and then propose extensions in ALF and OLA using a data model which is coherent with that terminology.

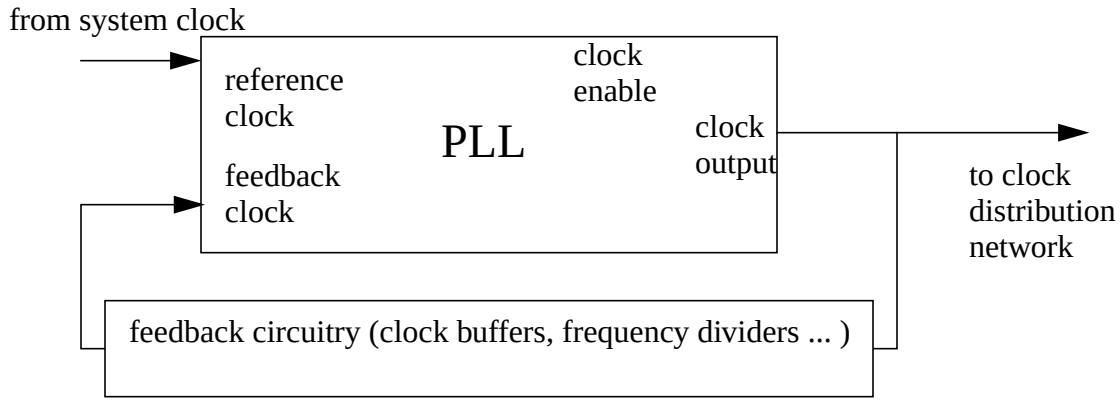
3.0 PLL modeling

Timing analysis of clock distribution networks is difficult, when PLLs are involved. It is impossible to represent a PLL by a logic function, since a PLL is a mixed-signal core with internal analog circuitry and digital I/O.

However, it is possible to define a standard functionality of a basic PLL and define a black box for it. Definition of timing constraints would be much less cumbersome, if a timing analysis tool could understand this functionality.

A basic PLL has 3 inputs and 1 output. The output is a clock, generated by an oscillator. One input pin is an “enable” pin which turns the oscillator on or off. When the oscillator is turned on, the signals on the other two inputs are relevant. They are both edge-sensitive clock inputs, commonly labeled as “reference” and “feedback” clock. The PLL compares the frequency and phase of the two clock signals and keeps changing the oscillator frequency in an attempt to align “reference” and “feedback” clock in both frequency and phase. This can only succeed, if there is an actual feedback between the clock output and the “feedback” input implemented in the design.

FIGURE 5. A PLL with external feedback



The timing analysis tool needs not be concerned how a PLL achieves the phase and frequency alignment. The only information it needs is the fact, that the reference clock and the feedback clock are aligned. From that knowledge, the feedback clock could be propagated backwards through the feedback circuitry to the clock output, and then propagated forward to the clock distribution network.

The general clock propagation capabilities (both forward and backwards) in timing analysis tools also should be enhanced to support clock propagation through frequency dividers, which again can be modeled by counters.

Hence the capabilities for functional description of a counter in a way comprehensible for timing analysis tools has a far reaching potential.

The PLL itself could be defined as an ALF primitive with predefined signaltypes. This corresponds to the definition of a new node primitive in the OLA function graph. Functional models of complex PLLs may instantiate this primitive and describe the additional control logic with ALF behavioral language.