Make your Testbenches Run Like Clockwork!

Markus Brosch, Salman Tanvir, Martin Ruhwandl

accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Why do we need a Clocks UVC?
- Motivation
- Advantages
- Architecture
- Conclusions & Recommendations
- Q&A

Why do we need a Clocks UVC?

- Handling Clocks (Driving/Monitoring) is an integral part of any verification environment.
- For a system with a complex clock system, a dedicated component is required.

Motivation

- Propose a UVM Clocks UVC Architecture
  - Genericity: Able to be used in any testbench with any number of clocks
  - Reusability: Efficient reuse across multiple projects
  - Driving a reference clock and a varying number of derived clocks mimicking a real clock tree
    - Configurable timing characteristics
    - Support for integer/fractional division
    - Glitch free clock switching
  - Monitoring of each individual clock
  - Common HDL BFM's that can be integrated with both SystemVerilog & Specman HVL components.

Advantages

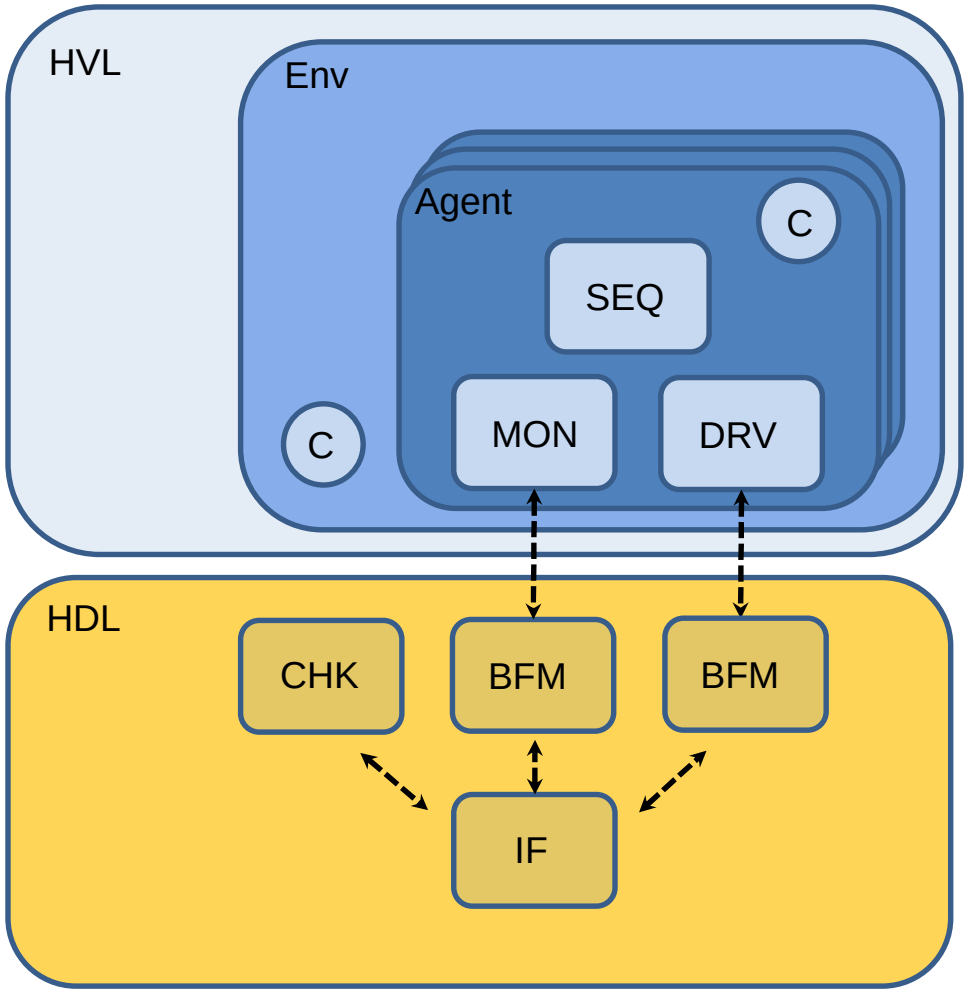
- Unified clocks handling
- Fosters reusability/portability
- Time saving
- High code quality

Architecture

- Split transactors (transaction level/BFM)
- Testbench to DUV connection
  - Abstract class approach
  - Specman DPI-C
- Configuration/Transaction modelling
- Driving
- Monitoring
  - Checks

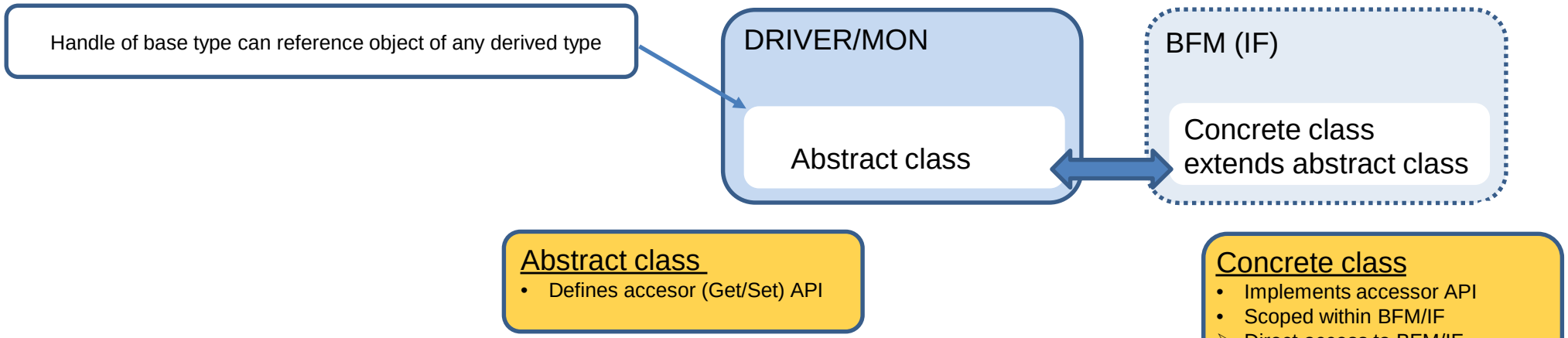
Split Transactors

- The clocks UVC should be usable in SystemVerilog & Specman environments.
- Propose to split the UVC into:
  - Switchable class based component
  - Common interface based BFM part

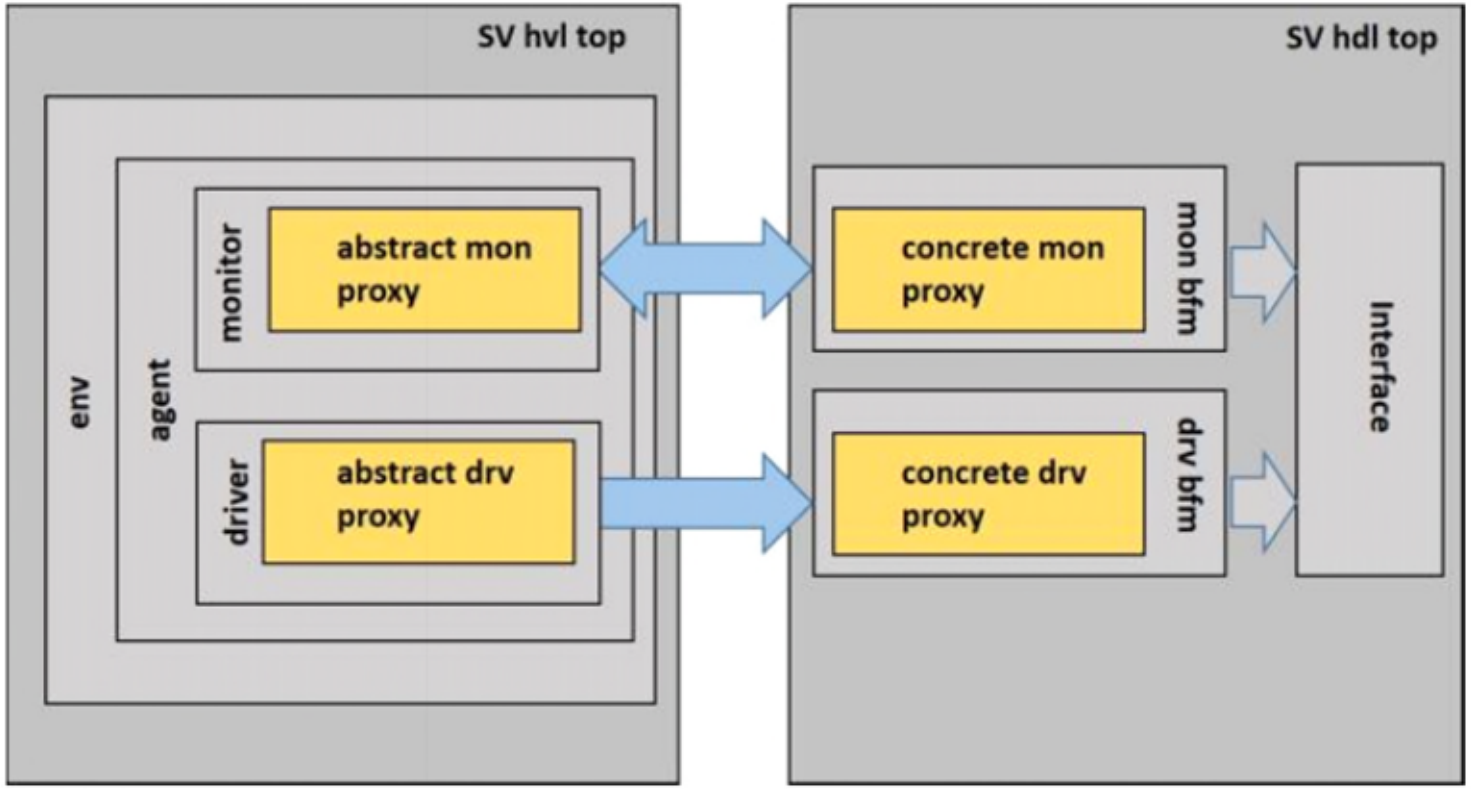


Testbench to DUV Connection (SystemVerilog)

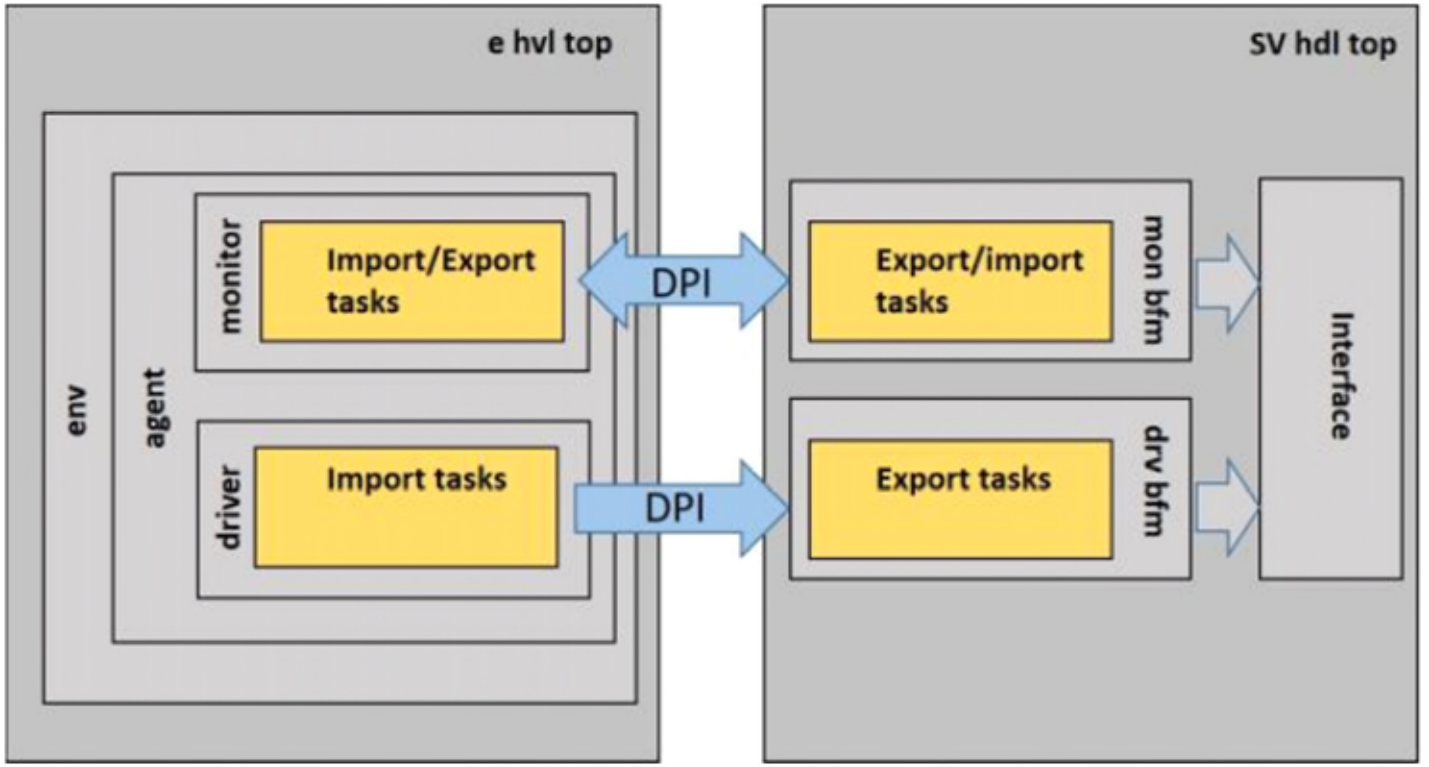
- Virtual interface connections are type specific
  - Different interface types cannot be referenced
  - Each specialization of a parameterized Interface is a new type
- Support of configurable number of clocks is required for a clocks UVC.
- The abstract class construct of SV solves the type specificity problem.



Testbench to DUV Connection (SystemVerilog)



Testbench to DUV Connection (Specman DPI-C)



Transaction Modelling (Configuration)

- Monitor & Driver are configured using class based configuration objects
  - Passed to BFM's via the testbench to DUV connections mechanisms
  - HVL configuration class objects translated into structs before passing to HDL.
  - Needed for DPI-C and to be emulation friendly if required
- The driver contains one configuration for the reference clock and one for each derived clock.
- The monitor uses a separate configuration object for each monitored clock.

Transaction Modelling (Configuration)

```
class ifx_clocks_driver_ref_clock_config extends uvm_object;
  rand bit [31:0]   clk_high_phase_width;
  rand bit [31:0]   clk_low_phase_width;
  rand bit          clk_enable;
  rand bit          jitter_enable;
  rand int unsigned jitter_factor;  //percentage jitter
  string            clk_name;       //clock identifier
  ...


class ifx_clocks_driver_derived_clock_config extends uvm_object;
  string           clk_name;              //clock identifier
  rand logic       clk_startval;          //initial clock signal level
  rand bit [31:0]  clk_high_phase_width;  //high phase time for mode 1
  rand bit [31:0]  phase_shift;           //phase shift from ref clock
  rand bit         clk_enable;            //clock enable signal ('1'=on / '0'=off)
  rand bit         clk_high_z;            //tristate high z clock out
  rand bit [7:0]   pattern_size;          //pattern size, range = 1 to 128
  rand bit [127:0] enable_pattern;        //modes 0,1: sequence pattern, mode 2: edge counts
  rand bit [1:0]   mode;                  //mode 0 = Sequence patterns
                                          //mode 1 = Sequence patterns with configurable high time
                                          //mode 2 = Edge Counter
  ...

class ifx_clocks_monitor_clock_config extends uvm_object;
  string   clk_name;    //clock identifier
  rand bit publish_en;  //control knob to enable/disable publishing

  ...
```

Driving

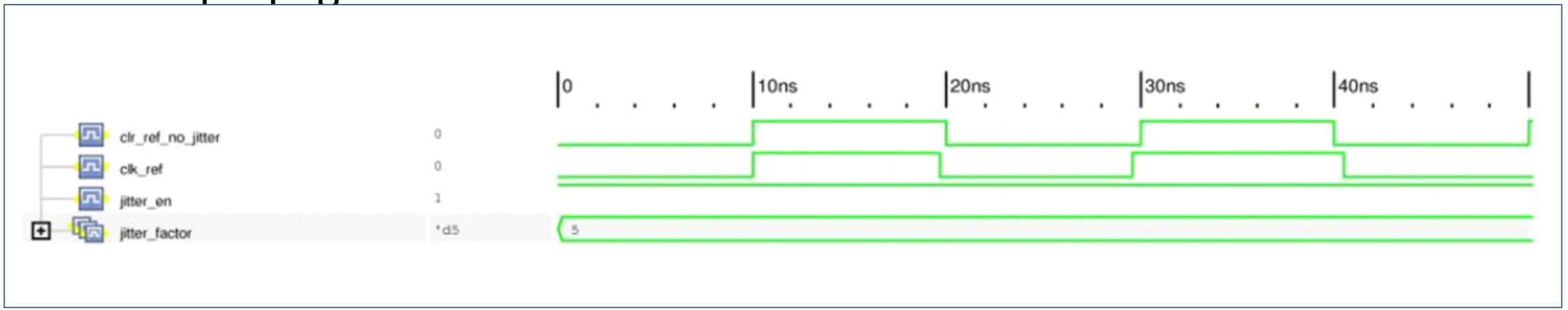
- Parameterized HDL BFM
  - Single reference clock and any number of derived clocks
- Driver BFM consists of dedicated processes for reference and derived clock generation.
- BFM initialized using the HVL agent configuration objects
- Subsequent configuration updates are handled by the sequence layer
  - Sequence item wraps the driver configuration objects and some additional knobs

Driving (Reference Clock)

- The reference clock is generated using the timing characteristics set by the configuration.
- Reference clock can be gated
  - Activated in next low phase
  - All derived clocks are also stopped as a consequence
- All derived clocks are initialized before starting the reference clock generation.

Driving (Jitter)

- Jitter can be enabled with a configurable variation factor.
- Simplistic implementation with variation only applied to clock high phase
- Jitter propagates to derived clocks.

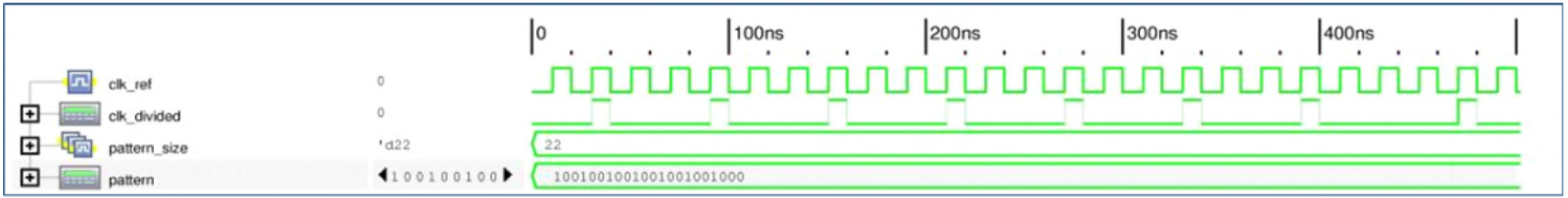


Driving (Derived Clocks)

- Derived clock process is triggered on the reference clock.
  - Single process within a generate for loop construct using the parameterization for the number of derived clocks
- Each clock can be configured to use one of the following modes
  - Sequence patterns
  - Sequence patterns with configurable high time
  - Edge counter

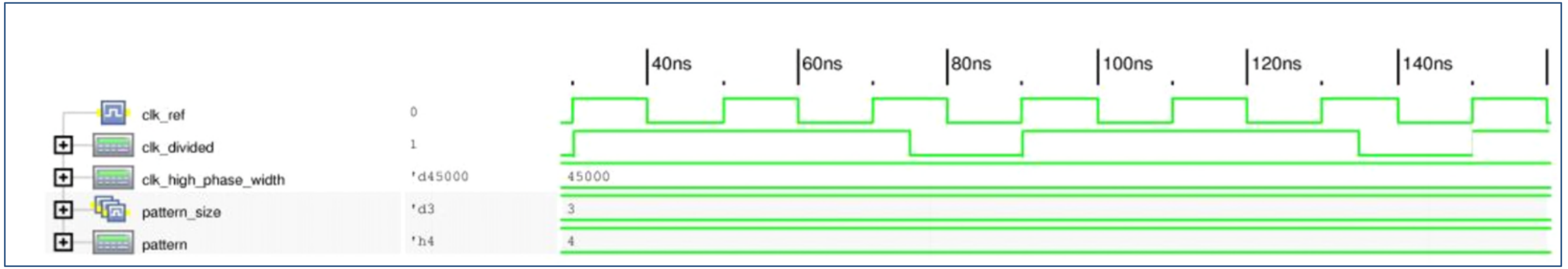
Sequence Patterns

- A bit pattern controls the derived clock generation
  - Pattern size configurable up to 128 bits.
- The clock generation process is triggered on the positive edge of the reference clock.
- On every subsequent trigger, each bit of the pattern is incrementally checked.
  - The reference clock is propagated through in case of a 1 in the pattern
  - The reference clock propagation can optionally be phase shifted by a parameter
- Any arbitrary pulse sequence can be generated which can also be used to achieve fractional division ratios.



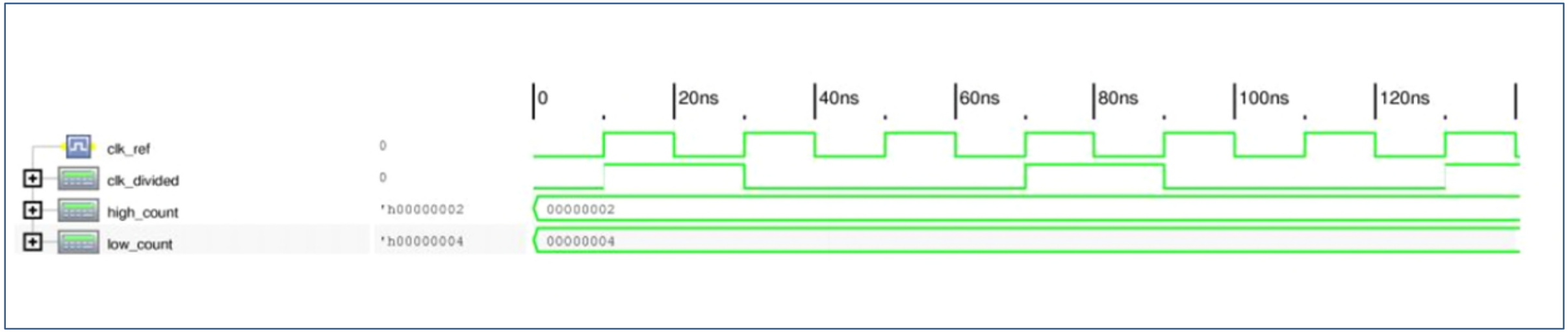
Sequence Patterns (Configurable High Time)

- Identical to the sequence patterns mode with the following exception
  - Reference clock is not propagated through on a 1 but rather the high phase is generated using a configurable parameter



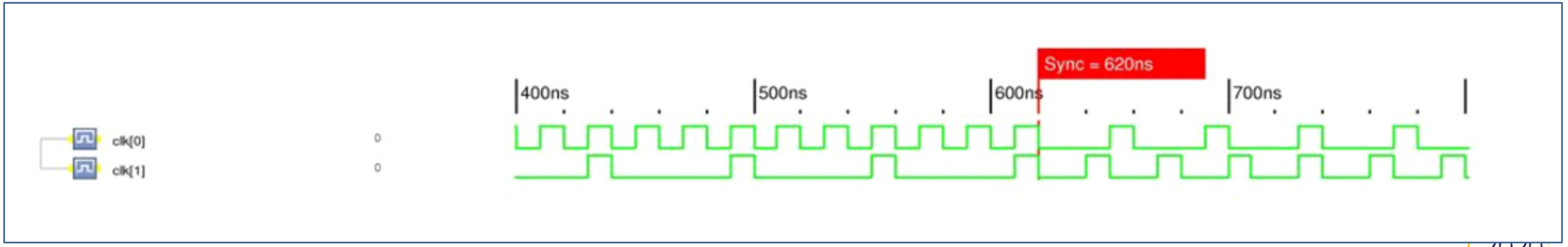
Edge Counter

- High and low phases are configurable in terms of the number of reference clock edges.
- Phase shift is also configurable.



Glitch Free Clock Switching

- Clocks configuration can be updated at any time without producing any intermediate artifacts
  - Implemented using shadow registers to align the configuration update with a synchronization point.
  - Synchronization point can be an edge of the reference clock or when all derived clock periods align.



Monitoring

- Parameterized HDL BFM to monitor any number of clocks
- Each clock is monitored in a dedicated thread
  - Time period and duty cycle is measured for every cycle
  - A monitor transaction is published only when a change is detected
  - Publishing can be disabled which is useful for monitoring clocks with frequently changing period and duty cycle e.g. jittering clocks
- An on demand monitoring task can be started by the user
  - Configurable measuring window (number of cycles)
  - The measured period and duty cycle of each cycle and an average over the window is returned
- A synchronization task is provided which allows the user to align with any edge of any monitored clock.

Monitoring (Checks)

- The monitor BFM implements 2 checking tasks
  - A Clock watchdog which raises an alarm if a monitored clock has not toggled within a configurable timeout.
  - A gated clock check to ensure no activity on a disabled clock.
- Any further checks can be implemented on a higher level using the monitor transactions.

Conclusions & Recommendations

- Fully functional clocks UVC developed
  - Specman integration concept proven but full implementation ongoing.
  - Genericity (Support for any number of clocks)
  - Concise code set (common HDL for SystemVerilog & Specman)
  - Flexible driving capability
    - Pattern sequences, integer/fractional division
    - Glitch free clock switching
  - Monitoring
    - Automatic/On demand measuring tasks
    - Synchronizing utility task
    - Clock checks (stuck, gated clocks)
- Recommendations
  - The jitter implementation could be enhanced.
  - Automatic monitor publishing could be improved to take a tolerance window into account.
  - In built frequency/duty cycle checks against a configurable golden value could be added.
  - Relationship checking of clocks within a tree could be added.

Questions