SCE-API Version 1.1
Revision 1.0.8

5.0 Formal Specification

5.1 General

5.2 Hardware side interface macros

5.2.1 Dual-ready protocol

5.2.2 SceMiMessageInPort macro

5.2.3 SceMiMessageOutPort macro

5.2.4 SceMiClockPort macro

5.2.5 SceMiClockControl macro

[...]

FIGURE 14. SceMiClockControl macro

[...]

5.2.5.1 Parameters

5.2.5.2 Signals

Uclock

[...]

Ureset

[...]

ReadyForCclock

This input to the macro indicates to the SCE-MI infrastructure that a transactor is willing to allow its associated DUT clock to advance. One of the most useful applications of this feature is to perform complex algorithmic operations on the data content of a transaction before presenting it to the DUT.

If this input to one of the `SceMiClockControl` instances associated with a given controlled clock is deasserted, the next posedge of that cclock will be disabled. In reacting to a `ReadyForCclock` of a slower clock, the infrastructure must not prematurely disable active edges of other faster clocks that occur prior to the last possible uclock preceding the edge to be disabled. In other words, that edge is disabled *just in time* so as to allow faster clock activity to proceed until the last moment possible. Once the edge is finally disabled, all active edges of all controlled clocks are also disabled. This is referred to as *just in time* clock control semantics.

Note: It may sometimes be desired for a transactor to stop all clocks in the system immediately. This is referred to as *emergency brake* clock control semantics. This can simply be done by instantiating a

`SceMiClockControl` associated with the fastest clock int the system and applying normal clock control to it.

CclockEnabled
This macro output signals the transactor, that on the next posedge of uclock, there is a posedge of the controlled clock. The transactor can thus sample this signal to know if a DUT clock posedge occurs. It can also use this signal as a qualifier that says it is okay to sample DUT output data. Transactors shall only sample DUT outputs on valid controlled clock edges. The SCE-MI infrastructure looks at the `ReadyForCclock` inputs from all the transactors and asserts `CclockEnabled` only if they are all asserted. This means any transactor can stop all the clocks in the system by simply de-asserting `ReadyForCclock`.

NOTE—For a *negedge active don't care duty cycle* (see 5.2.4.3), since the user does not care about the posedge, the `CclockEnabled` shall always be 0.

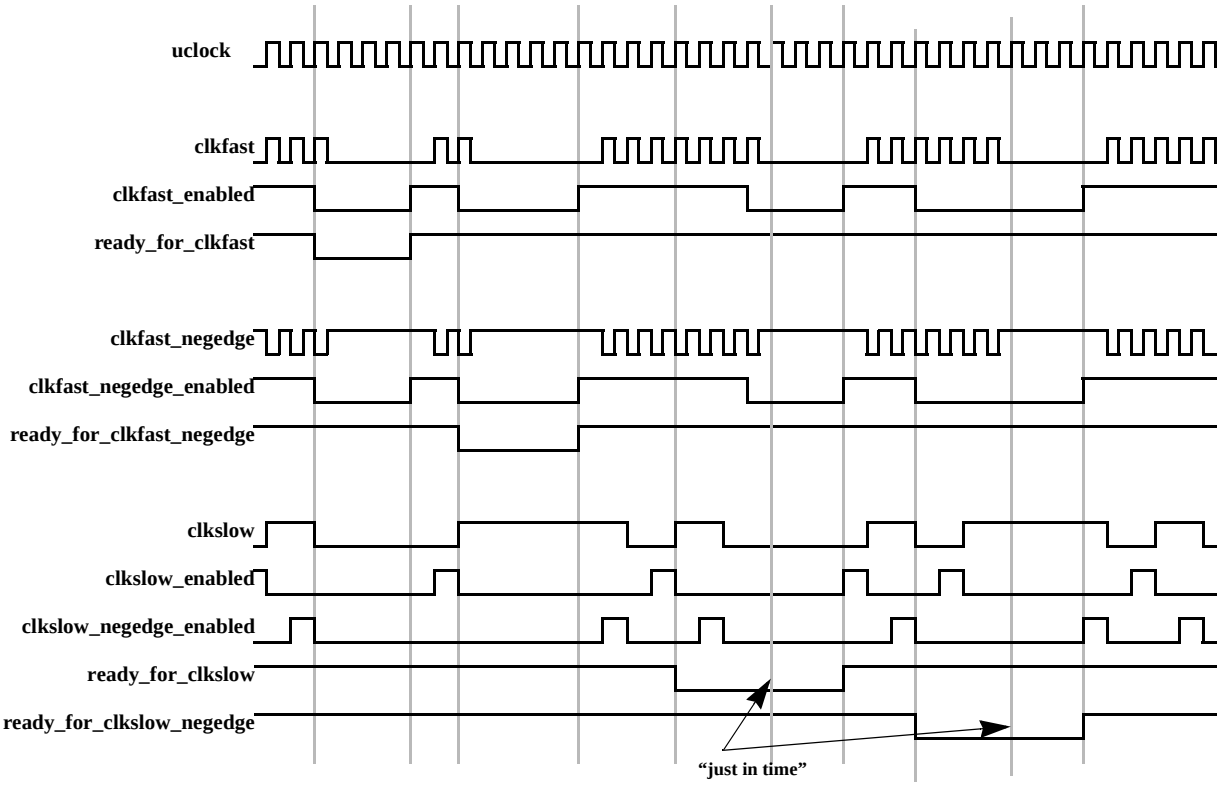
ReadyForCclockNegEdge
Similarly, for negedge control, if this input to one of the `SceMiClockControl` instances that are associated with a given controlled clock is deasserted, the next negedge of that clock will be disabled. In reacting to a `ReadyForCclockNegEdge` of a slower clock, the infrastructure must not prematurely disable active edges of other faster clocks that occur prior to the last possible uclock preceding the edge to be disabled. In other words, that edge is disabled *just in time* so as to allow faster clock activity to proceed until the last moment possible. Once the edge is finally disabled, all active edges of all controlled clocks are also disabled. This is referred to as *just in time* clock control semantics.

NOTE- Support for explicit negedge control is needed for transactors that use the negedge of a controlled clock as an active edge. Transactors that do not care about controlling negedges (such as the one shown in Figure A.1) need to tie this signal high.

CclockNegEdgeEnabled
This signal works like `CclockEnabled`, except it indicates if the negedge of a controlled clock occurs on the next posedge of the uclock. This can be useful for transactors that control double pumped DUTs. Transactors that do not care about negedge control can ignore this signal.

NOTE—For a *posedge active don't care duty cycle* (see 5.2.4.3), since the user does not care about the posedge, the `CclockNegEdgeEnabled` shall always be 0.

FIGURE 15. Clock Control Semantics



5.2.5.3 Clock Control Semantics

Figure 15 shows an example of clock control for two fast clocks (`clkfast`, `clkfast_negedge`) that use *don't care duty cycle* semantics and one slow clock (`clkslow`) that uses a 50/50 duty cycle. `clkfast` uses *posedge active* don't care duty cycle and `clkfast_negedge` uses *negedge active* don't care duty cycle.

The effect of the 4 respective clock control signals `ready_for_clkfast`, `ready_for_clkfast_negedge`, `ready_for_clkslow`, and `ready_for_clkslow_negedge` can be seen.

Deassertion of `ready_for_clkfast` prevents subsequent posedges of `clkfast`, negedges of `clkfast_negedge`, and all edges of `clkslow` from occurring on subsequent posedges of `uclock`. Once re-asserted, all these edges are allowed to occur on the subsequent `uclock` posedges where relevant.

Deassertion of `ready_for_clkfast_negedge` prevents subsequent negedges of `clkfast_negedge`, posedges of `clkfast`, and all edges of `clkslow` from occurring on subsequent posedges of `uclock`. Once re-asserted, all these edges are allowed to occur on the subsequent `uclock` posedges where relevant.

Deassertion of `ready_for_clkslow` prevents subsequent posedges of `clkslow`. But notice that this happens *just in time* for the next scheduled posedge `clkslow`. Prior to this, edges of faster clocks or the negedge of the same clock are allowed to occur. Once the edge is finally disabled, all edges of other

clocks are disabled as well. Once re-asserted, all these edges are allowed to occur on the subsequent `uclock` posedges where relevant.

Deassertion of `ready_for_clkslow_negedge` prevents subsequent negedges of `clkslow`. But notice that this happens *just in time* for the next scheduled negedge `clkslow`. Prior to this, edges of faster clocks or the posedge of the same clock are allowed to occur. Once the edge is finally disabled, all edges of other clocks are disabled as well. Once re-asserted, all these edges are allowed to occur on the subsequent `uclock` posedges where relevant.

Note, that all of the clock enabled signals, `clkfast_enabled`, `clkfast_negedge_enabled`, `clkslow_enabled`, and `clkslow_negedge_enabled` are shown to transition on `uclock` posedges. The implementation can also choose to transition them on negedges. The only hard requirement is that their values can be sampled on the `uclock` posedge at which the associated controlled clock edge will occur.