PGCB Clock Gating

Micro-Architecture Specification (MAS)

Revision 1.20

Last Update: 16 June 2014

Kah Meng Yeem / Jared Havican

DO NOT COPY OR DISTRIBUTE

INTEL TOP SECRET

The information in this document is subject to change.

Revision History

	Revision/Date	Owner	Description of Change
1.	0.00 06/14/2013	Kah Meng	Initial revision and incorporated the feedback from the Pre-Aspec review on ww24.4.
2.	0.10 07/01/2013	Kah Meng	Repartition the PGCB Clock Gating solution and move the gate/wake consolidations to the IP-Specific Power Control Glue Logic.
3.	0.20 07/18/2013	Kah Meng	Remove the Wake Registration Logic and the Local Clock Gating feature from the PGCB Clock Gating Block per the review with Mikal/Bill/Hartej on ww27.2. Added the Interface signals for the PGCB Clock Gating Block.
4.	0.30 08/01/2013	Kah Meng	Added the clock gate, clock wake and reset sequencing waveform for PCGU. Added a reference design for the entire PGCB clock gating block.
5.	1.15 10/29/2013	Jared Havican	Added details on pcgu_aww (PCGU asynchronous wake widget) Added reference design that is completely asynchronous (ie does not require an always running clock) Added 0in CDC waivers
6	1.20 6/10/2014	Jared Havican	[2280815] Updated PCGU to prevent potential glitch when pgcb_rst_b asserts: - Added reset state which only allows a pmc_ip_wake to start the clock. - Clkreq is no longer asserted in reset but only asserts after a reset when triggered by a pmc_ip_wake (when DEF_PWRON==0) - Added parameter DEF_PWRON, which when set keeps the clock ungated and the clkreq asserted in reset. PCGU interface changes: - Added DEF_PWRON parameter - Added pgcb_pok input - Added async_pmc_ip_wake input - Removed async_pgcb_rst_b input Updated sync_wake_source_b output of PCGU_AWW to such that it only deasserts when sync_clkvld has asserted and async_wake_source_b has deasserted to be more conservative. Updated Asynchronous Reference Design to remove cfg_pgcb_clkgate_disabled input as it is not desirable to keep the clkreq asserted when in an IP-Inaccessible state. [2266758] Moved the pgcb_clk clock gate cell into the reference design.

Table of Initials

Initials	Name
PCGU	PGCB Clock Gating Block
Tresetmin	Minimum pulse width to asynchronously set/clear a flop
PCGU_AWW	PCGU Asynchronous Wake Widget

Contact Information

Section	Name	Phone	E-Mail
Author	Kah Meng Yeem	916-377-8396	kah.meng.yeem@intel.com

Table of Contents

Table of Figures

Table of Tables

1. PGCB Clock Gating

BXT has the requirement to be able to shut down the PGCB clock regardless of whether the IP has been power gated. This is to reduce the IDLE power when Vnn is down. This document proposes a PGCB clock gating solution to meet the BXT requirement. Figure 1 illustrates a conceptual block diagram of the PGCB Clock Gating solution.   Multiples PGCB clock consumers, such as PGCB and CDC blocks, could share a single PGCB clock control block (PCGU).   The IP-Specific Power Control Glue Logic is responsible to evaluate the clock gate and wake conditions for PGCB clock gating. When the IP meet all the IDLE indications, PCGU will trigger the CLKREQ handshake to enable SOC to gate the PGCB clock at the trunk level.

Please see <u>*section 1.4*</u> *for an example reference design that can be used to implement the design described here.*
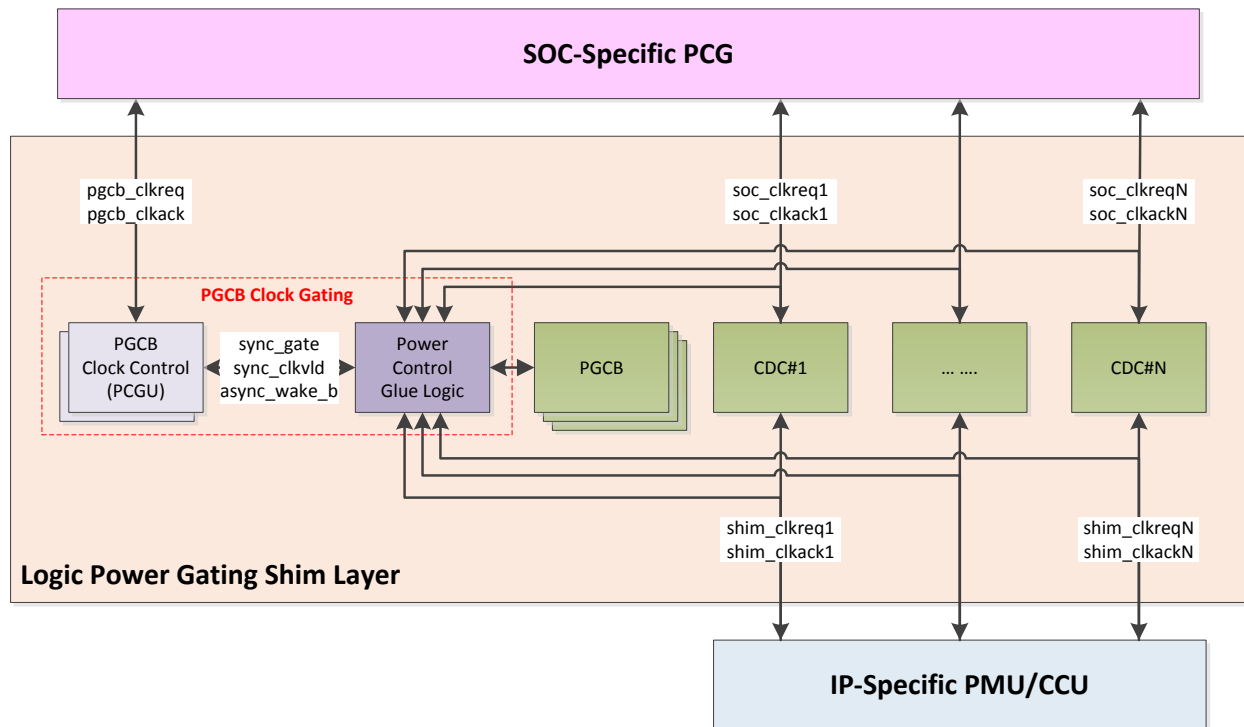
Figure 1: PGCB Clock Gating Conceptual Block Diagram

1.1 Theory of Operation

Since every IP may have their IP-specific requirement on when the PGCB clock could be gated, the IP will be responsible to implement an IP-Specific Power Control Glue logic to determine when the PGCB clock gating could happen. IPs should only allow the PGCB clock to be gated when the entire PGD is in Deep Idle state. This is to make sure that all the PGCB and CDCs are parked at a known state.

In general, the IP is considered in Deep Idle state when the IP is power-gated. For IPs that do not support IP-Accessible power-gating, the IP is considered in Deep Idle when it allows all functional clocks are to be gated at the trunk level (ie. clkreqs/clkacks are deasserted).

Depending on the PG state of the IP, the PGCB clock gating condition for each PGD will be determined differently:
1. IP-Inaccessible Idle Condition. A PGD is considered to be in an IP-Inaccessible state when pgcb_pok = 0. When the IP is in an IP-Inaccessible state, all the following conditions must be met before PGCB clock gating could happen:
    a. The pg_wake = 0
    b. All pgcb_idle = 1, and
    c. All soc_clkreq*/clkack* = 0

2. IP-Accessible Idle Condition. If pgcb_pok = 1, the PGD is considered to be in an IP-Accessible State. When the PGD is in an IP-Accessible state, all the following conditions must be met before PGCB clock gating could happen:
    a. IP-Inaccessible Idle Condition
    b. All shim_clkreq*/clkack* = 0, and
    c. All ism_fabric* = IDLE

Once the PGCB clock is gated, the IP-Specific Power Control Glue Logic should wake up the PGCB clock when it detects any wake condition. Depending on the PG state of the IP, the PGCB clock wake condition for each PGD will be determined differently:
1. IP-Inaccessible Wake Condition. When the PGD is in an IP-Inaccessible state, the following condition will wake up the PGCB clock:
    a. The pg_wake = 1

2. IP-Accessible Wake Condition. When the PGD is in an IP-Accessible state, any of the following conditions will wake up the PGCB clock:
    a. Any IP-Inaccessible Wake Condition
    b. Any shim_clkreq* = 1
    c. Any ism_fabric* not in IDLE state, OR
    d. Any changes in the CDC power gating enables

AON clock domains that do not support trunk level clock gating are exempt from the above requirement. The assumption is that the AON clock domain is only used to detect the wake condition from the Deep Idle State. It will not require the PGCB clock when all functional clocks have been gated at the trunk level.

Table 1 summarizes the PGCB clock gating and wake conditions for each PG domain. The PGCB clock gating will only happen when all the PGDs allows the clock to be gated. The PGCB clock will be ungated if any PGD requires the clock to be running.

<div align="center">Table 1: PGCB Clock Gate Evaluation (Per PGD domain)</div>

IP State		PGCB Clock Gate Evaluation	
State Description	pgcb_pok	Gate Condition	Wake Condition
IP-Inaccessible	0	• pg_wake = 0 • pgcb_idle = 1 • soc_clkreq*/clkack* = 0	• pg_wake = 1 • pgcb_idle = 0 • soc_clkreq* = 1

| IP-Accessible | 1 | • IP-Inaccessible Gate Condition
• shim_clkreq*/clkack* = 0
• ism_fabric* in IDLE state | • IP-Inaccessible Wake Condition
• shim_clkreq* = 1
• ism_fabric* not in IDLE state
• Any changes in the CDC power gating enables |
| | | | |

Figure 2 logically illustrates the PGCB clock gate and wake evaluation logic. Suggestions on how this logic should actually be implemented are provided in section 1.4. The IP could choose to aggregate the gate and wake conditions from multiple PGDs, and eventually share a PGCB clock control block for multiple PGDs. Alternatively, each PGD could have its own PCGU.

The sync_gate control signal must be synchronous to the PGCB clock domain, but the async_wake_b control signal can be asserted and de-asserted in an asynchronous manner. The async_wake_b may contain glitch, but IP is responsible to make sure that the assertion pulse width of async_wake_b is compliant to the process dependent Tresetmin requirement. The behavior is undefined if the assertion pulse width of async_wake_b is less than Tresetmin.

When the sync_clkvld is de-asserted, the PGCB clock consumer is recommended to stall all the activities that are using the PGCB clock. As the PGCB and CDC currently do not comprehend whether or not the PGCB clock is running, the recommended method for stalling them is to gate their clock when sync_clkvld is '0'.
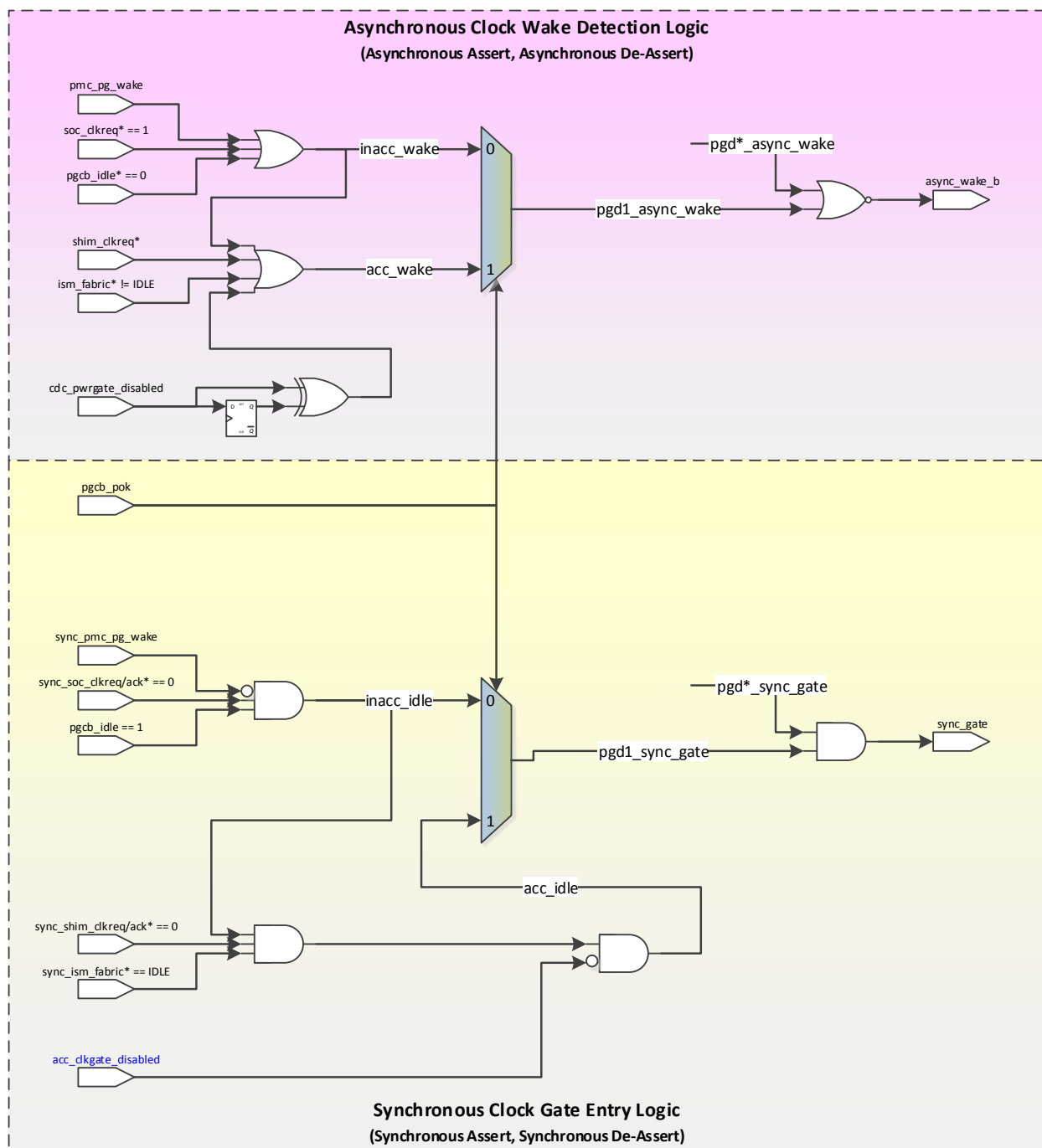
Figure 2: PGCB Clock Gate Entry and Wake Evaluation

1.2 PGCB Clock Gate Control Block (PCGU)

The PGCB Clock Gate Control block (PCGU) is responsible to manage the PGCB CLKREQ handshake with SOC.

There are 2 options to implement the PGCB clock control logic. The comparisons of these 2 options are summarized as follow:

Table 2: Comparison of PGCB Clock Control Implementation Options

	Pro	Con
Option 1: Synchronous Assertion and De-Assertion of the PGCB Clock Request.	• Simplest design as everything is synchronous	• Needs an additional Always-On Clock • Requires additional power to run the synchronizer in Always-On clock domain • Longer wake latency
Option 2: Asynchronous Assertion, but Synchronous De-Assertion on the PGCB Clock Request.	• Does not require additional clock • Lower power • Shorter wake latency	• Need SCONs/Waivers • More complicated design

Option 2 is the preferable option, because it adds more generality and removes dependency on an always on clock source, such as RTC clock.   Furthermore, BXT already required their IPs support an asynchronous handling of the PGCB clock gating.   Hence, Option 2 will be implemented unless we have solid circuit simulation data to prove that the asynchronous handling of the PGCB clock request assertion is fundamentally broken.

PCGU supports a 4-bit hysteresis timer to regulate the PGCB clock gate and wake sequencing. Once the sync_gate is asserted, PCGU will de-assert the sync_clkvld and trigger the PGCB clock gating flow immediately. After triggered the PGCB clock gating flow, PCGU will wait for the clock gate hysteresis timer timeout before de-asserting the pgcb_clkreq to SOC. If the sync_gate is de-asserted before the clock gate hysteresis timer expired, the control block will abort the PGCB clock gating flow and re-assert the sync_clkvld immediately. Once the clock gate hysteresis timer is timeout, PCGU will ignore sync_gate, async_wake_b and async_pmc_ip_wake until it completes the CLKREQ handshake with SOC. Once the clock gating flow with SOC is completed, the PCGU will assert the pgcb_clkreq to trigger the clock wake flow when detected an async_wake_b or async_pmc_ip_wake (depending on the PG state of the IP).   Since the clock wake flow will be handled in an asynchronous manner, a clock wake hysteresis timer is supported to relax the SD routing requirement. PCGU will wait for the clock wake hysteresis timer expired before re-asserting the sync_clkvld.

1.2.1 Parameters

Parameter	Description	Default
DEF_PWRON	Indicates the default power-gated state of the IP. When set to '0', clkreq is deasserted and the clock is gated in reset and will require a pmc_ip_wake event to cause clkreq to assert. When set to '1', clkreq is asserted and the clock is ungated in reset.	'0'

1.2.2 Configuration Register Bit/Straps

The PGCB clock gating block requires the IP-Specific Power Control Glue Logic to consolidate the gate and wake condition for the PGCB clock gating. As a survivability feature, the IP-Specific glue logic is recommended to implement the following straps to configure the PGCB clock gating condition. An IP could choose to use a tie-off value, or dynamically configure it thru an external register unit. The configuration mechanism is beyond the scope of this document.

Signal	Description	Default	CLK	Recommendation
	PGCB Clock Gating Configuration			
acc_clkgate_disabled	**IP-Accessible Clock Gating Disable.** When set, the PGCB clock gating feature will be disabled if the IP is still in IP-Accessible State. This bit is ignored if the pgcb_clkgate_disabled is set.	'1'	PGCB	Register Bit
t_clkgate[3:0]	**PGCB Clock Gating Hysteresis Delay.** Specify the minimum number of delay clocks the PGCB clock gate sequencing should wait, before enabling the PGCB clock to be gated at the trunk-level.	'0'	PGCB	Register Bit
t_clkwake[3:0]	**PGCB Clock Wake Hysteresis Delay.** Specify the minimum number of delay clocks the PGCB clock wake sequencing should wait, before enabling the PGCB clock consumer to use the PGCB clock.	'0'	PGCB	Tie-Off Strap

1.2.3 Interface Signals

1.2.3.1 SOC Interface

1.2.3.1.1 SOC-Specific Parameter
Nil

1.2.3.1.2 Clock and Reset

Signal	Description	CLK	SRC	DEST
	Clock and Reset			
pgcb_clk	**PGCB clock.**	PGCB	SOC	PCGU
pgcb_clkreq	**PGCB Clock CLKREQ**	-	PCGU	SOC
pgcb_clkack	**PGCB Clock CLKACK**	-	SOC	PCGU
async_pmc_ip_wake	**Raw pmc_ip_wake signal directly from PMC** This signal triggers the initial clkreq assertion when in an IP-Inaccessible sate. Note that it is required that PMC keep this signal asserted until the IP has exited power-	-	SOC	PCGU

Signal	Description	CLK	SRC	DEST
	gating. This allows it to be used as a well-behaved wake event that will not deassert until clkack has asserted.			

1.2.3.1.3 DFx

Signal	Description	CLK	SRC	DEST
	DFX			
fscan_byprst_b	**Fabric Scan Bypass Reset.** This signal is a reset input for scan operations that bypasses the internal agent reset logic and applies a reset directly to the agent. The reset override signal group must be implemented for IP-blocks with embedded or derived internal reset signals.	-	SOC	PCGU
fscan_rstbypen	**Fabric Scan Reset Bypass Enable.** This signal will enable the ability for the bypass reset signals to be active. The reset override signal group must be implemented for IP-blocks with embedded or derived internal reset signals.	-	SOC	PCGU

1.2.3.2 IP Specific Power Control Glue Logic Interface

1.2.3.2.1 Reset

Signal	Description	CLK	SRC	DEST
	Clock and Reset			
pgcb_rst_b	**PGCB reset.** This is the PGCB reset that has its deassertion synchronized to the PGCB clock domain.	PGCB	SOC	PCGU

1.2.3.2.2 Configuration Registers

The following registers are defined for survivability purpose.

Signal	Description	CLK	SRC	DEST
	Hysteresis Delay			
t_clkgate[3:0]	**PGCB Clock Gating Hysteresis Delay.** Specify the minimum number of delay clocks the PGCB clock gate sequencing should wait, before enabling the PGCB clock to be gated at the trunk-level.	PGCB	SIP	PCGU
t_clkwake[3:0]	**PGCB Clock Wake Hysteresis Delay.** Specify the minimum number of delay clocks the PGCB clock wake sequencing should wait, before	PGCB	SIP	PCGU

	enabling the PGCB clock consumer to use the PGCB clock.			
	ECO Bits			
scratchpad[7:0]	**Scratchpad.** This field is a generic scratchpad with reserved hardware functional implementation behind it. IP must tie these inputs to all 0s.	PGCB	SIP	PCGU

1.2.3.2.3 PGCB Clock and Wake Interface

This is the control interface with the IP-Specific Power Control Glue logic to trigger the PGCB clock gating operation.

Signal	Description	CLK	SRC	DEST
	Clock Gate and Wake Control			
sync_gate	**PGCB Clock Gate Enable.** When asserted, it indicates that the IP does not require PGCB clocks for a significant time, and allows SOC to gate (optionally shutting down) the PGCB clock source. When de-asserted, it indicates that wake events has been detected in PGCB clock domain and has to ungate PGCB clock source.	PGCB	SIP	PCGU
async_wake_b	**PGCB Clock Wake.** Asserted when there is a PGCB clock wake event. The assertion of this signal must meet the minimum pulse width of reset (i.e. Tresetmin). The mechanism to meet the Tresetmin requirement will be IP-Specific.	-	SIP	PCGU
sync_clkvld	**PGCB Clock Valid.** Asserted when the PGCB clock is ready to be used. De-asserted when the PGCB clock gating block is about or already to enable the PGCB clock to be shut down.	PGCB	PCGU	SIP
pgcb_pok	**PGCB POK.** Asserted when in an IP-Accessible state, deasserted when in IP-Inaccessible.	PGCB	PGCB	PCGU

1.2.3.3 DFx

This is the Visa Nodes for debug purpose.

Signal	Description	CLK	SRC	DEST
	VISA Vector Signals			
pcgu_visa[15:0]	**PCGU Visa Vector.** The latest recommendation is that the integrating IP have the VISA tool automatically insert VISA in the PCGU and that this output be ignored. See the provided .sig files under $MODEL_ROOT/tools/visa.	PGCB (except as noted)	PCGU	SIP

	Note that these should be fed into a VISA ULM in the always on domain. Bit Definitions: ``` [15] - acc_wake_flop #async [14] - defon_flag [13] - pmc_wake_assert #async [12] - acc_wake_assert #async [11] - clkreq_sustain [10] - clkack_syn [9] - async_wake_b [8] - sync_gate [7] - pgcb_clkreq #async [6:3] - tmr [2:0] - clkreqseqsm_ps ```			

1.2.4 Implementation Details

Illustrates the PGCB Clock Control Block with Asynchronous Assertion and Synchronous De-Assertion of the PGCB Clock Request.
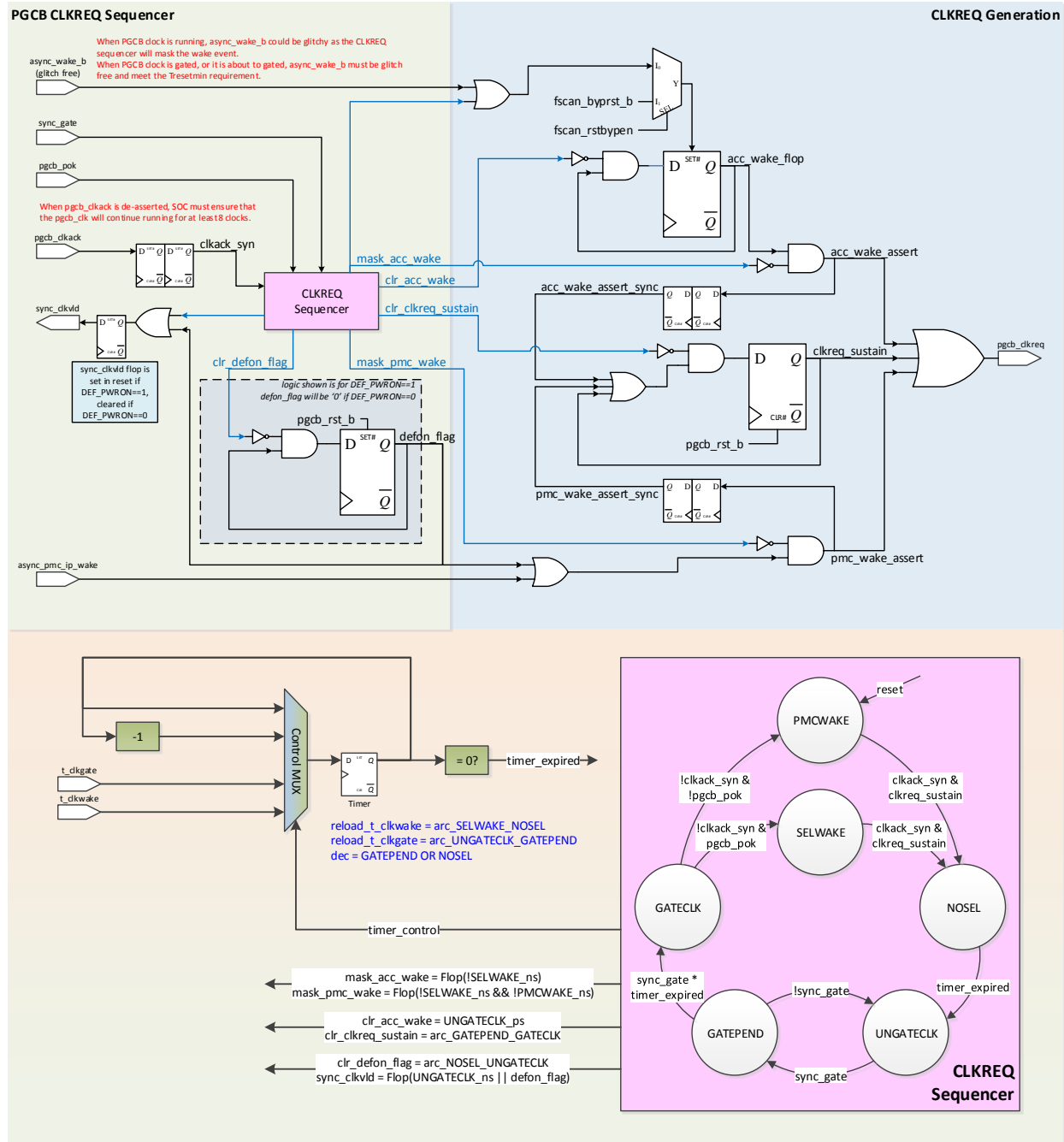
Figure 3: PGCB Clock Control Block w/ Asynchronous Handling of the PGCB Clock Request

The PGCB Clock Control logic is summarized as follow:

1. Use PGCB clock to synchronize the pgcb_clkack.
2. Implement clock request control logic to generate the pgcb_clkreq in the PGCB domain.
    - There are two paths to cause clkreq to assert:
        1. async_pmc_ip_wake – will be the only path that is open during IP-Inaccessible PG. Is masked off when the clkreq is driven synchronously.

2. async_wake_b – IP-Accessible wake path. This path sets a flag (acc_wake_flop) asynchronously which will be cleared and masked off when the clkreq is driven synchronously.

- The wake events are synchronized back into the pgcb_clk domain and set a flop (clkreq_sustain) synchronously to keep the clkreq asserted. When this flop is set, the async wake paths are masked off.
- The clkreq_sustain flop is cleared when gating the clock, which causes clkreq to deassert.
- When clkack is confirmed deasserted, the wake paths are again opened back up to allow clkreq to assert again.

3. Implement a CLKREQ sequencer to perform the PGCG clock gate and wake sequencing:
   a. Since the pgcb_clkreq is not asserted when PGCB reset is asserted, the CLKREQ sequencer will be default to PMCWAKE state to enable the asynchronous pmc_ip_wake path of the clock request control logic. The sync_clkvld will be de-asserted.
   b. Once the SOC indicated that the PGCB clock is running, the CLKREQ sequencer will transition from SELWAKE to NOSEL state and disable the asynchronous wake paths.   Since SD is not going to PV the asynchronous wake path timings, there is a concern on the routing delay of the asynchronous signals. Hence, a clock wake hysteresis timer is supported to ensure that the asynchronous wake paths are completely disabled at the clock request control logic.
   c. Once the clock wake hysteresis timer is timeout, the CLKREQ sequencer will transition from NOSEL to UNGATECLK state, and assert the sync_clkvld.
   d. When the CLKREQ sequencer is in UNGATECLK, it will transition to GATEPEND if the sync_gate is asserted. The sync_clkvld will be de-asserted as the indication to the IP that PGCB clock will be gated soon.
   e. When the CLKREQ sequencer is in GATEPEND, it will activate the clock gate safety-net timer and wait for the timer to expire before de-asserting the pgcb_clkreq. If sync_gate is de-asserted before the timer expires, the CLKREQ sequencer will abort the PGCB clock gating flow, and return to the UNGATECLK state. sync_clkvld will be asserted as the indication to the IP that PGCB clock is again available.
      - **Note**: the timer that is running in GATEPEND cannot truly be called a hysteresis timer as the clock that drives the fanin to sync_gate will likely be gated when sync_clkvld deasserts (if following the provided reference designs). Instead it acts as a safety-net as sync_gate could assert in the same cycle as sync_clkvld deasserts.
   f. If the sync_gate remains asserted when the clock gate timer expires, the CLKREQ sequencer will transition from GATEPEND to GATECLK state and de-assert the pgcb_clkreq.
   g. Once the SOC acknowledges the pgcb_clkreq de-assertion event, the CLKREQ sequencer will transition to either the SELWAKE state or the PMCWAKE state to enable the asynchronous wake paths of the clock request control logic in a glitch free fashion.
      - The design assumes the SOC is compliant to Clock Chassis Gen 2, such that it will only shutdown the clock at least 8 clocks after de-asserting the pgcb_clkack.   This is required for the pgcb_clkack to be synchronized to the PGCB clock domain.

4. Implement a timer for clock gate and wake sequencing:
   a. Reload the timer with the clock wake hysteresis value when the CLKREQ sequencer is transitioning from SELWAKE to NOSEL state
   b. Reload the timer with the clock gate hysteresis value when the CLKREQ sequencer is transitioning from UNGATECLK to GATEPEND state
   c. Decrement the timer when the CLKREQ sequencer is in NOSEL or GATEPEND state.

5. Since the async_wake_b signal is asserted asynchronously, there is a concern that it may introduce some meta-stability issue if the width of the async_wake_b signal is less than the Tresetmin requirement.   If this happens, BXT library team is unable to guarantee there is no meta-stability issue, even though the clock request control logic is clock gated during that time. Hence, PCGU will

PGCB Clock Gating

not attempt to locally clock gate the clock request control logic.   It imposes a restriction to have IP to generate the async_wake_b in a glitch free manner, and able to meet the Tresetmin requirement.
   a.   Additional SCONs are needed to waive the clock cross violations on the async_wake_b generation.

1.2.4.1 Default Power-On (DEF_PWRON==1)

Starting with PCGU 1.20, support has been added for IP's the default to powered on. The implementation assumes that such IP's require their clock to be ungated and clkreq to be asserted during and after reset.

The PCGU accomplishes this, when the DEF_PWRON flag is set to '1', through the "defon_flag" shown above. This flag is set while in reset and causes the pgcb_clkreq to assert through the pmc_wake_assert path. sync_clkvld also resets to '1' (and will stay set until the defon_flag is cleared) to keep the clock-gate open. defon_flag is then cleared when the FSM transitions to the UNGATECLK state and sync_clkvld is kept asserted by the FSM.

Note that if an ip that defaults to powered on is put into an IP-Inaccessible state, it will require a pmc_ip_wake assertion to cause clkreq to assert again, similar to if the DEF_PWRON parameter was set to '0'. However, as of this writing, there is no known usage model for putting an IP that defaults powered on into IP-Inaccessible PG.

1.2.5 Waveforms

1.2.5.1 Reset/IP-Inaccessible Wake Sequence (DEF_PWRON==0)

Figure 4 illustrates the reset exit sequence of the PCGU block with the DEF_PWRON parameter set to '0'. The detailed steps are summarized as follow:
   1. When in reset, the FSM will be in the PMCWAKE state, and the pmc_ip_wake path is open (mask_pmc_wake==0)
   2. When pmc_ip_wake asserts, it will cause clkreq to assert to the SOC. (Note that this can happen before or after pgcb_rst_b deassertion)
   3. At some point when the pgcb_rst_b deasserts and the clock from the SOC starts running, pmc_ip_wake is synchronized and causes clkreq_sustain to be set
   4. The FSM will move to NOSEL when clkreq_sustain is set and clkack_sync goes high. This will mask off the pmc_ip_wake path. (The timer is also loaded with t_clkwake on the transition from PMCWAKE to NOSEL)
   5. When the timer reaches 0, the CLKREQ sequencer will transition to UNGATECLK state.
   6. Once the CLKREQ sequencer is in UNGATECLK, it will assert PGCB clock valid to notify the IP that PGCB clock is now available.
      a.   In UNGATE_CLK the acc_wake_flop is cleared synchronously, on a cold boot its value would have been indeterminate prior to this point.

Intel Top Secret   12
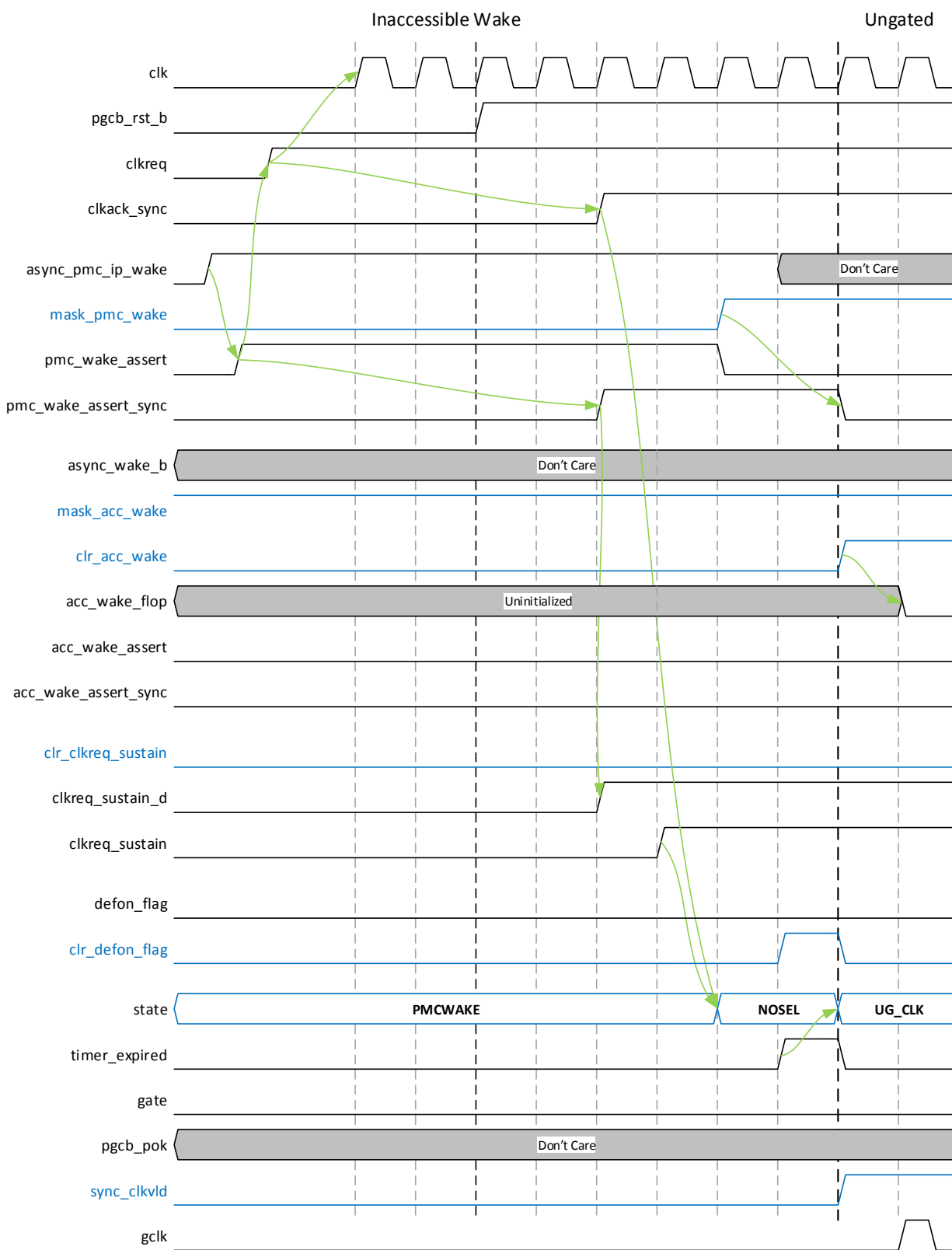
PGCB Clock Gating

Figure 4: PCGU IP-Inaccessible/Reset Wake Sequence (DEF_PWRON==0)

1.2.5.2 Reset/IP-Inaccessible Wake Sequence (DEF_PWRON==1)

Figure 8 illustrates the reset exit sequence of the PCGU block with the DEF_PWRON parameter set to '1'. The detailed steps are summarized as follow:

1. When in reset, the FSM will be in the PMCWAKE state, and the pmc_ip_wake path is open (mask_pmc_wake==0)
2. In reset, the defon_flag is also set, which causes clkreq to assert through the pmc_ip_wake path.
    a. Note that defon_flag also causes sync_clkvld to be asserted out of reset, which will open up the clock gate that it feeds.
    b. Note that if IP entered IP-Inaccessible without pgcb_rst_b asserting, the defon_flag would remain low and a pmc_ip_wake would be required to wake the IP similar to previous waveform.
3. At some point when the pgcb_rst_b deasserts and the clock from the SOC starts running, pmc_ip_wake path (driven by defon_flag) is synchronized and causes clkreq_sustain to be set
4. The FSM will move to NOSEL when clkreq_sustain is set and clkack_sync goes high. This will mask off the pmc_ip_wake path. (The timer is also loaded with t_clkwake on the transition from PMCWAKE to NOSEL)
5. When the timer reaches 0, the CLKREQ sequencer will transition to UNGATECLK state.
6. Once the CLKREQ sequencer is in UNGATECLK, it will assert PGCB clock valid to notify the IP that PGCB clock is now available.
    a. In UNGATE_CLK the acc_wake_flop is cleared synchronously, on a cold boot its value would have been indeterminate prior to this point.
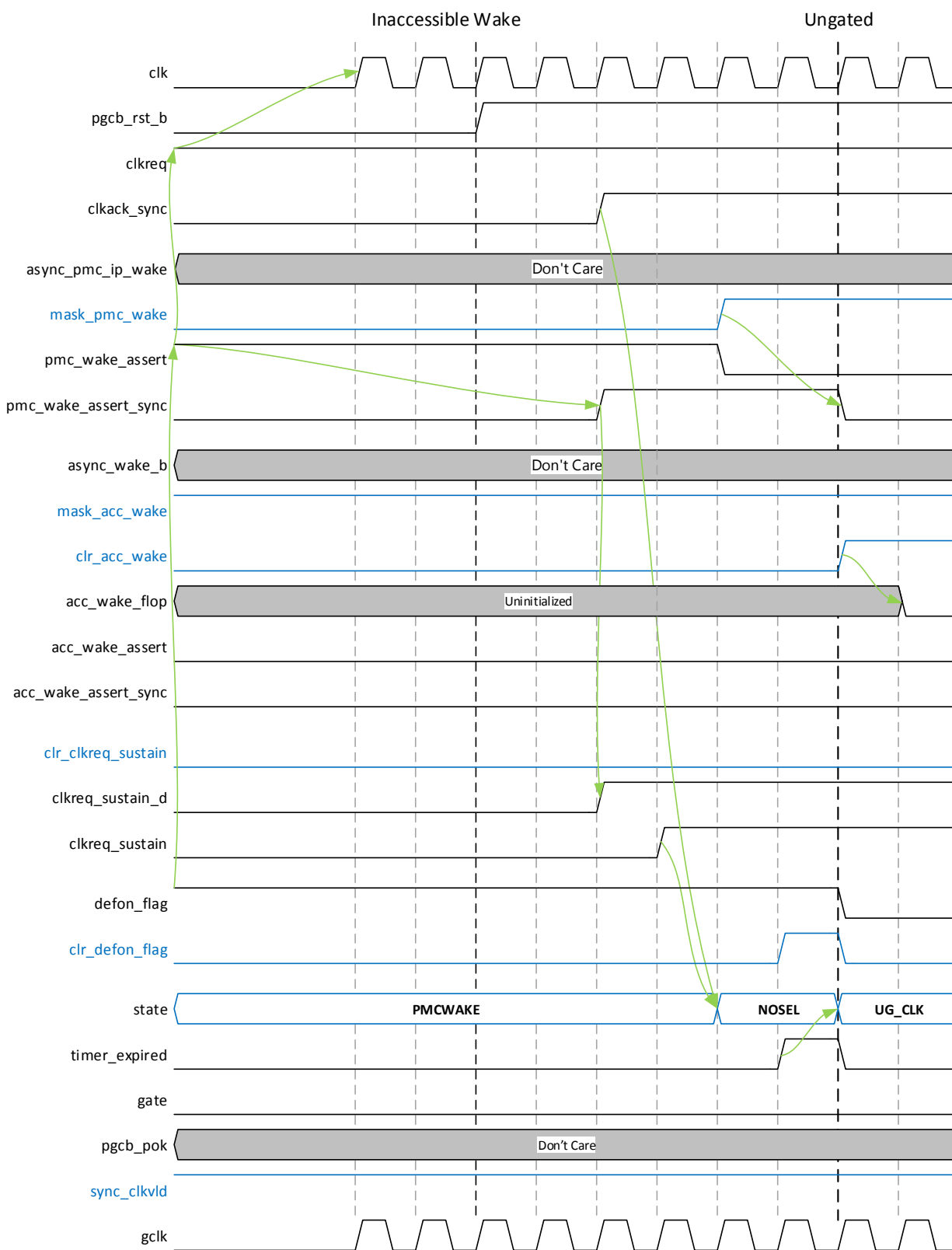
Figure 5: PCGU IP-Inaccessible/Reset Wake Sequence (DEF_PWRON==1)

1.2.5.3 Reset/IP-Inaccessible Gate Sequence

Figure 6 and Figure 7 illustrate the reset/IP-Inaccessible entry flow, after which pgcb_rst_b can assert without the fear of it introducing glitches on pgcb_clkreq. The detailed steps are summarized as follow:

1. The CLKREQ sequencer stays at UNGATECLK state, and waits for IP to enter IP-Inaccessible PG state. The PGCB clock is available and the PGCB clock valid is asserted.
2. Once the IP asserts sync_gate as an indication that it is in IP-Inaccessible PG, the CLKREQ sequencer will transition to GATEPEND state and reload the timer with t_clkgate value.
3. In GATEPEND sync_clkvld is deasserted and the clock to the PGCB/CDC would be gated. When the timer reaches '0' and sync_gate is still asserted, the FSM then moves to GATECLK.
4. In GATECLK the clkreq_sustain flop is cleared and clkreq deasserts. The FSM waits in GATECLK until clkack_sync deasserts and then moves to PMCWAKE (because pgcb_pok is low).
   a. Once the SOC de-assert the PGCB CLKACK, the SOC must guarantee that the PGCB clock will be available for at least 8 clocks. This is to allow the PCGU to synchronize the PGCB CLKACK and complete the remaining clock gating sequencing.
5. In PMCWAKE the pmc_ip_wake path is now opened up and the FSM stays in PMCWAKE and waits for a pmc_ip_wake event or a pgcb_rst_b event.
   a. If DEF_PWRON==0 and pgcb_rst_b asserts, there will be no transition on any of the internal signals and there is no possibility of a glitch. (see Figure 6)
   b. If DEF_PWRON==1 and pgcb_rst_b asserts, clkreq will assert and sync_clkvld will assert because defon_flag is set, note that clkreq assertion would be glitch-free as pmc_ip_wake is glitch free. (see Figure 7)
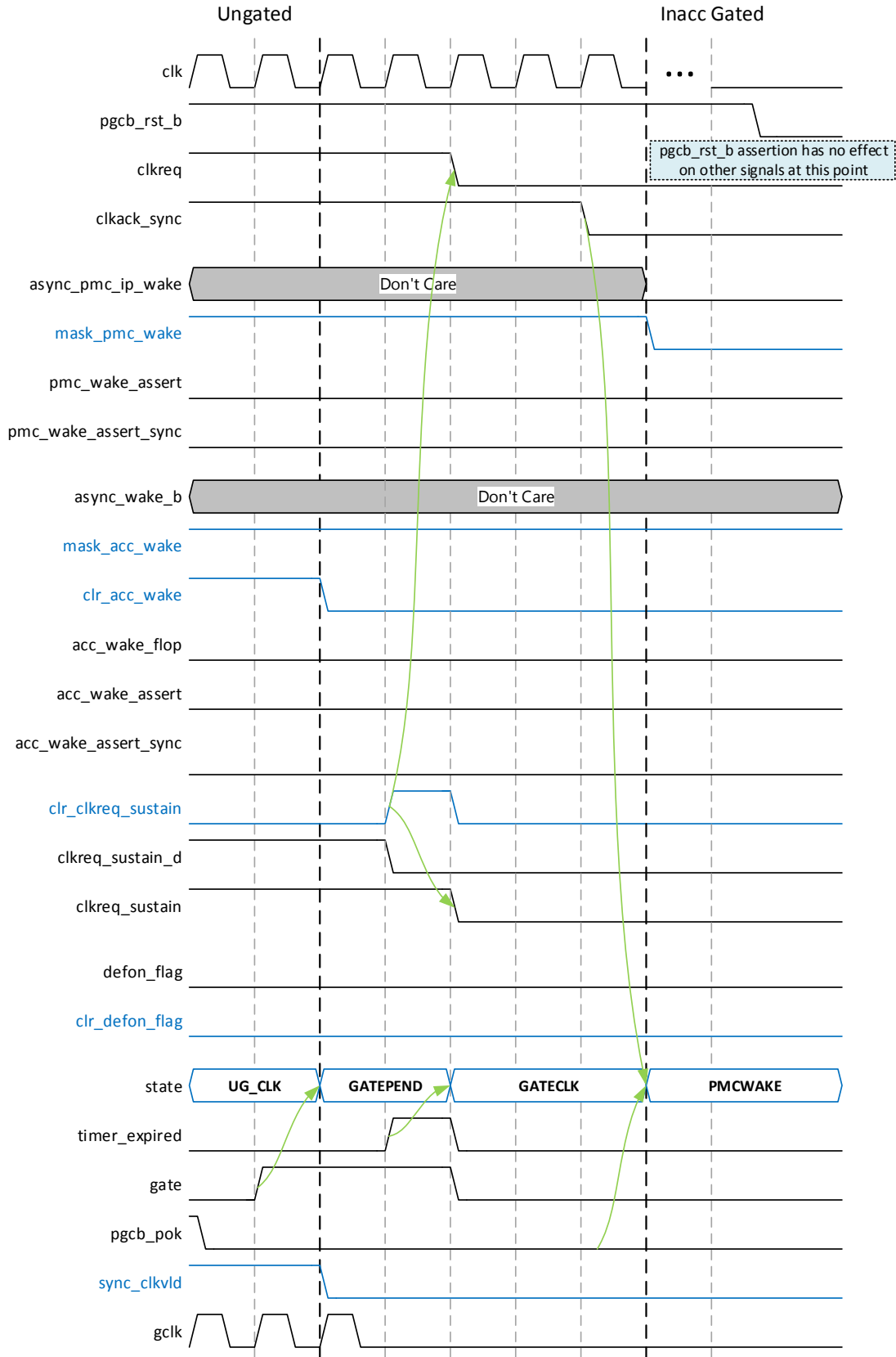
PGCB Clock Gating

Figure 6: PCGU IP-Inaccessible/Reset Gate Sequence (DEF_PWRON==0)
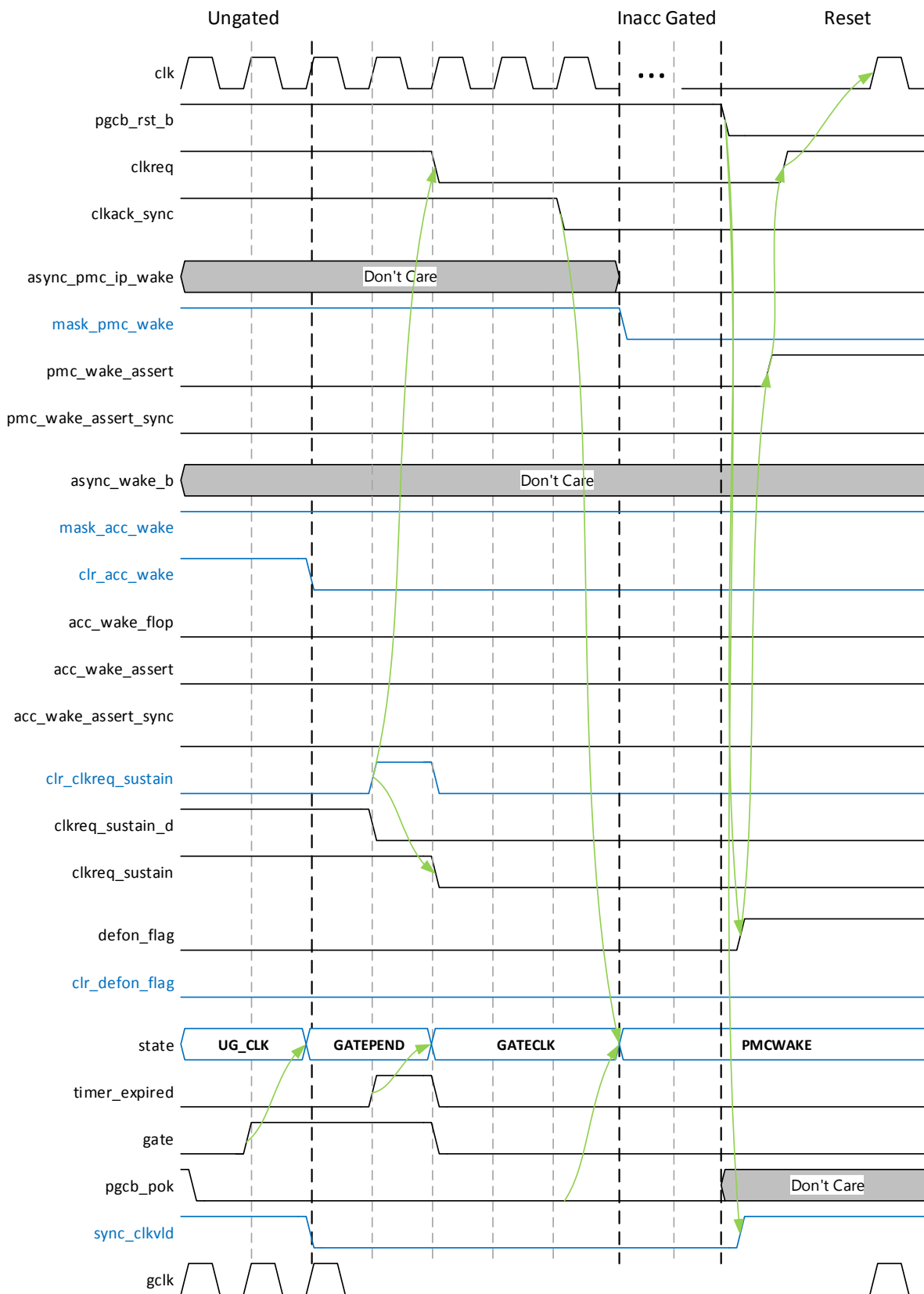
Figure 7: PCGU IP-Inaccessible/Reset Gate Sequence (DEF_PWRON==1)

1.2.5.4 IP-Accessible Gate Sequence

Figure 8 illustrates the IP-Accessible clock gate sequencing of the PCGU block. The detailed steps are summarized as follow:

1. The CLKREQ sequencer stays at UNGATECLK state, and waits for the IP to enter a deep idle IP-Accessible state. The PGCB clock is available and the PGCB clock valid is asserted.
2. Once the IP asserts sync_gate as an indication that it is in a Deep Idle state, the CLKREQ sequencer will transition to GATEPEND state and reload the timer with t_clkgate value.
3. In GATEPEND sync_clkvld is deasserted and the clock to the PGCB/CDC would be gated. When the timer reaches '0' and sync_gate is still asserted, the FSM then moves to GATECLK.
4. In GATECLK the clkreq_sustain flop is cleared and clkreq deasserts. The FSM waits in GATECLK until clkack_sync deasserts and then moves to SELWAKE (because pgcb_pok is high).
   a. Once the SOC de-assert the PGCB CLKACK, the SOC must guarantee that the PGCB clock will be available for at least 8 clocks. This is to allow the PCGU to synchronize the PGCB CLKACK and complete the remaining clock gating sequencing.
5. In SELWAKE both the pmc_ip_wake and the async_wake_b paths are opened up, the FSM stays in SELWAKE and waits for either async_wake_b or pmc_ip_wake to assert to start the wake sequence.
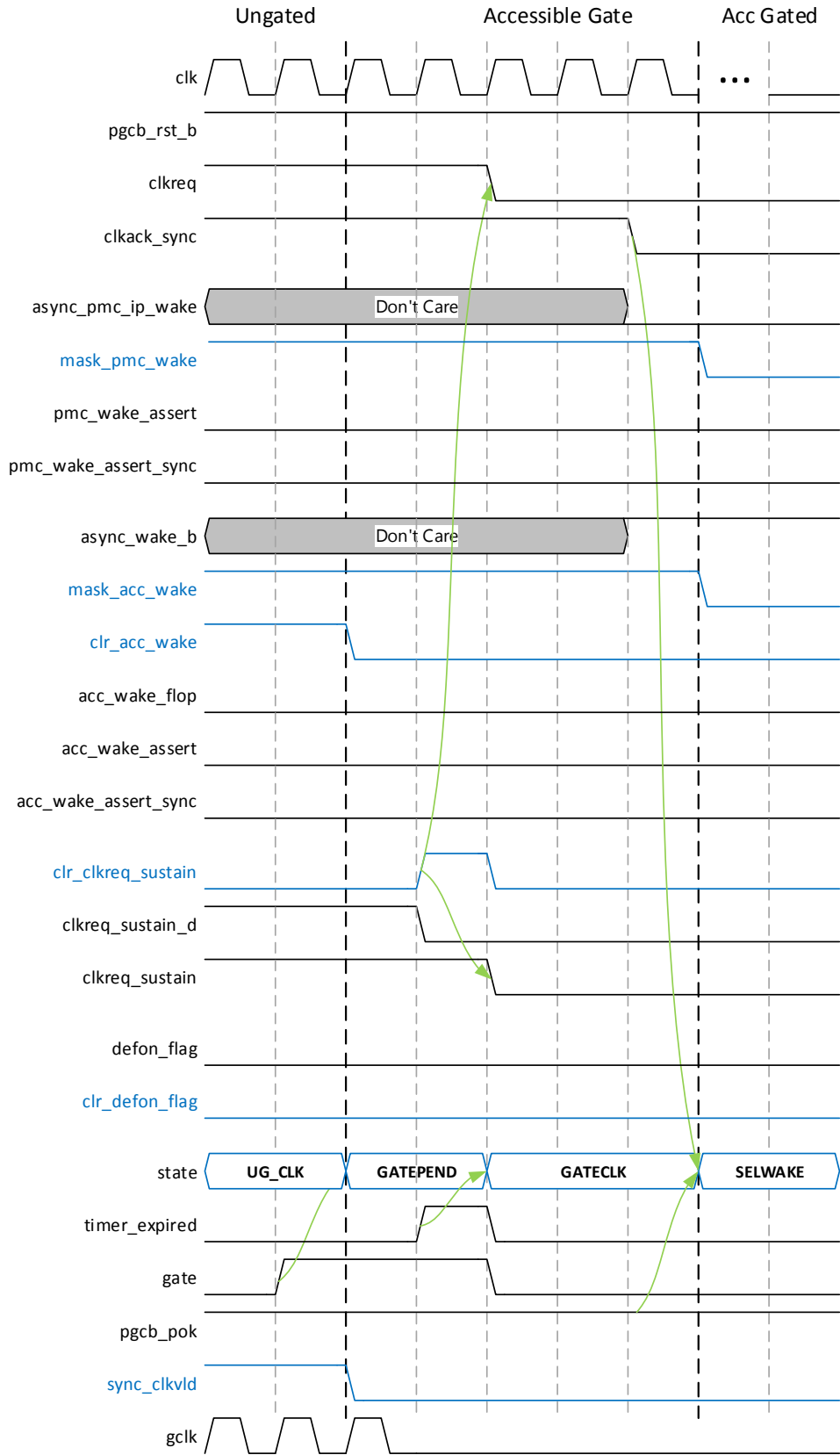
PGCB Clock Gating

Figure 8: PCGU IP-Accessible Gate Sequence

1.2.5.5 IP-Accessible Wake Sequence

Figure 9 illustrates the IP-Accessible clock wake sequencing of the PCGU block. The detailed steps are summarized as follow:

1. The CLKREQ sequencer stays at SELWAKE state, and waits for IP to detect the Deep Idle wake event. The PGCB clock could be gated by SOC.
   a. Note that the sync_gate event is ignored when the CLKREQ sequencer is in SELWAKE state.
2. Once the IP asserts async_wake_b (or the SOC asserts pmc_ip_wake), as an indication that it will like to exit from Deep Idle state, the CLKREQ sequencer will asynchronously assert the PGCB CLKREQ to ask for the PGCB clock.
3. SOC will assert the PGCB CLKACK and make sure that the PGCB clock is running. PCGU could begin to synchronize the PGCB CLKACK to PGCB clock.
   a. The wake event will also be synchronized when the clock starts running which will cause clkreq_sustain to be set synchronously.
4. When the synchronized version of the PGCB CLKCACK is asserted and clkreq_sustain is asserted, the CLKREQ sequencer will transition to NOSEL state and reload the timer with the t_clkwake value.
5. The CLKREQ sequencer will mask both wake paths and start decrementing the timer once it is in NOSEL state.
   a. Note that the sync_gate or async_wake_b events are ignored when the CLKREQ sequencer is in NOSEL state.
6. When the timer reaches 0, the CLKREQ sequencer will transition to UNGATECLK state.
7. Once the CLKREQ sequencer is in UNGATECLK, it will assert PGCB clock valid to notify the IP that PGCB clock is now available. The CLKREQ sequencer stays at UNGATECLK state, and waits for IP to enter the Deep Idle state.
   a. Note that the async_wake event is ignored when the CLKREQ sequencer is in UNGATECLK state.
   b. Also note that in UNGATECLK, the acc_wake_flop is cleared synchronously.
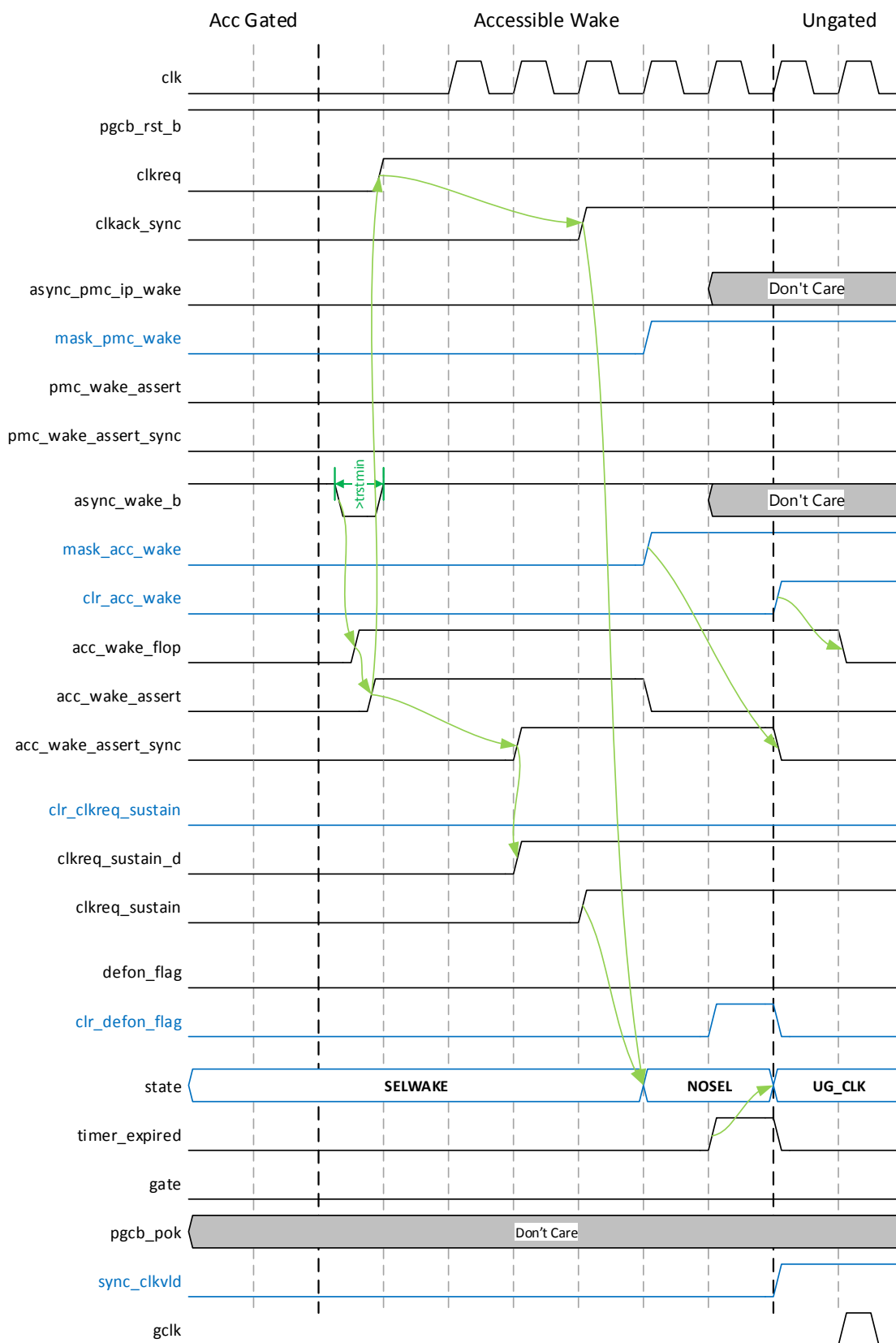
PGCB Clock Gating

Figure 9: PGCB Clock Wake Sequencing

1.2.6 Glitch Prevention on Reset

Starting with PCGU 1.20, the logic has been modified to remove a potential race on the pgcb_rst_b path which could lead to a glitch on the pgcb_clkreq output. As long as the IP/PCGU is gracefully put into an IP-Inaccessible state (ie through ForcePwrgatePOK message, not a reset), an assertion of pgcb_rst_b -- after pok is low and pgcb_clkack has been low for 8 clocks -- will not cause a glitch on pgcb_clkreq.

Note that as of this writing, BXT is not known to gracefully put IPs into an IP-Inaccessible state before turning off VNN, but instead ungracefully asserts pgcb_rst_b. The behavior of pgcb_clkreq is not guaranteed to be glitch-free under such scenarios and such glitches must be able to be tolerated by the SoC.

1.2.7 Risks

1. The PGCB clock gating block does not have a handshake to communicate the clock gating event to the PGCB and CDC. This may incurs a race condition when the CDC/PGCB happens to use the clock when the SOC is about to shut down the PGCB clock.
    a. [To Do] Need a waveform to show the race condition.
        i. PGCB Clock Gating Block has de-asserted the pgcb_clkreq
        ii. While waiting for SOC to de-assert the pgcb_clkack, PGCB/CDC has been triggered to service some wake events (e.g. Force Power Gate POK).
        iii. When PGCB/CDC is in the middle of servicing the wake events, SOC returns the acknowledged and eventually causing the PGCB clock to be shut down at the trunk level.
        iv. Some internal PGCB/CDC states must be communicated to the PGCB Clock Gating block to wake up the PGCB clock (e.g. pgcb_idle and soc_clkreq).
    b. [To Do] Need to analyze the impacts, especially in the CDC blocks
    c. [To Do] Need to check if we need the IP-Specific Power Control Glue logic to implement a local clock gating mechanism to gate the PGCB clock to CDC/PGCB using the sync_clkvld.

2. Regardless of the default power state when the IP is first powered up from reset, the PGCB clock control block will always request for PGCB clock as soon as the PGCB reset de-asserted.   This is to simplify the design. Once the PGCB clock is ungated, it will evaluate the clock gating condition to determine if it should go back to the clock gating state.
    a. [To Do] Check if this HW behavior has violated any chassis/SD requirement.

3. Since the async_wake_b will be generated from multiple asynchronous clock sources using a combinatorial logic, it may introduce some glitches that violate the Tresetmin requirement. This could eventually cause a meta-stability on the PGCB CLKREQ output.
    a. Checked with library team and realized that there is impossible to guarantee that there is no meta-stability issue. The meta-stability may not be resolved immediately, even if the clock is gated.   Hence, the recommendation is to have IP to implement an IP-Specific mechanism to guarantee that the async_wake_b is glitch-free and meet the Tresetmin requirement. This restriction is essential to avoid meta-stability.

1.2.8 Tool Waivers

1.2.8.1 Lintra

Lintra waivers can be found under $MODEL_ROOT/tools/lint/waivers/pcguunit_waivers.lwv

1.2.8.2 Questa/0in CDC

The following waivers may be needed when running CDC:

```
#### PCGU Waivers ####
# pmc_ip_wake goes directly to clkreq output for IP-Inaccessible wake
# per-implementation, can only cause a glitch free transition on clkreq and will stay asserted
until
# clkreq is driven by synchronous logic
cdc report crossing -scheme no_sync -through async_pmc_ip_wake -through pgcb_clkreq -severity
waived -module pcgu

# Only relevant if setting DEF_PWRON to '1':
# defon_flag goes directly to clkreq output for cold boot wake will stay asserted until
# clkreq is driven by synchronous logic to avoid glitches
cdc report crossing -scheme no_sync -tx_clock PGCB_CLK -from defon_flag -through pgcb_clkreq -
severity waived -module pcgu

# The following mask terms will not cause transitions on clkreq as it will be driven from another
# another path when they assert/deassert
cdc report crossing -scheme no_sync -tx_clock PGCB_CLK -from mask_pmc_wake -through pgcb_clkreq -
severity waived -module pcgu
cdc report crossing -scheme no_sync -tx_clock PGCB_CLK -from mask_acc_wake -through pgcb_clkreq -
severity waived -module pcgu

# clkreq_sustain is takes over driving pgcb_clkreq when the clock starts running, and will cause
# synchronous deassertion of clkreq, all of which will be glitch free
cdc report crossing -scheme no_sync -tx_clock PGCB_CLK -from clkreq_sustain -through pgcb_clkreq
-severity waived -module pcgu

# Path from acc_wake_flop to pgcb_clkreq, will not cause glitches on pgcb_clkreq
cdc report crossing -scheme no_sync -tx_clock PGCB_CLK -from acc_wake_flop -through pgcb_clkreq -
severity waived -module pcgu

# Path from async_pmc_ip_wake to doublesync has combi logic, pmc_wake is mostly static and is
well behaved,
# if it asserts it will stay asserted until it has been synchronized and by the time it deasserts
it should be masked off
cdc report crossing -scheme combo_logic -through async_pmc_ip_wake -through
i_pgcb_ctech_doublesync_pmc_wake.d -severity waived -module pcgu
```

1.3 PCGU Async Wake Widget (pcgu_aww)

The PCGU Async Wake Widget (pcgu_aww) is distributed as a reusable component which can be used along with the PCGU. Its purpose is to capture an asynchronous wake event and keep it asserted until sync_clkvld asserts, thus creating a handshake with the PCGU.

The async_wake_extended_b output is asserted asynchronously along with the async_wake_source_b input and deasserts synchronously to the pgcb_clk. The async output can more safely be combined with other asynchronous wake sources to drive the async_wake_b input of the PCGU as its active pulse is at least 6 pgcb_clk periods wide.   Thus when ANDed with other wake events, there will be less risk of reducing the active pulse width and violating the Tresetmin requirement of async_wake_b.

1.3.1.1 Interface Signals

1.3.1.1.1 Clock and Reset

Signal	Description	CLK	SRC	DEST
	Clock and Reset			
pgcb_clk	**PGCB clock.**	PGCB	SOC	PCGU
pgcb_rst_b	**PGCB reset.** This is the PGCB reset that has its deassertion synchronized to the PGCB clock domain.	PGCB	SOC	PCGU

1.3.1.1.2 DFx Interface

Signal	Description	CLK	SRC	DEST
	DFX			
fscan_byprst_b	**Fabric Scan Bypass Reset.** This signal is a reset input for scan operations that bypasses the internal agent reset logic and applies a reset directly to the agent. The reset override signal group must be implemented for IP-blocks with embedded or derived internal reset signals.	-	SOC	PCGU
fscan_rstbypen	**Fabric Scan Reset Bypass Enable.** This signal will enable the ability for the bypass reset signals to be active. The reset override signal group must be implemented for IP-blocks with embedded or derived internal reset signals.	-	SOC	PCGU

1.3.1.1.3 Functional Interface

Signal	Description	CLK	SRC	DEST
	Clock Gate and Wake Control			
sync_clkvld	**PGCB Clock Valid.** Asserted when the PGCB clock is ready to be used. De-asserted when the PGCB clock gating block is about or already to enable the PGCB clock to be shut down.	PGCB	PCGU	PCGU_AWW
async_wake_source_b	**PGCB Clock Wake.** Asserted (active-low) when there is a PGCB clock wake event. The assertion of this signal must meet the minimum pulse width of reset (i.e. Tresetmin).	-	SIP	PCGU
sync_wake_source_b	**Synchronous PGCB Clock Wake.** Synchronized version of async_wake_source_b that can be used as one term feeding the sync_gate input to the PCGU. Stays asserted until sync_clkvld asserts and async_wake_source_b deasserts	PGCB	PCGU_AWW	PCGU
async_wake_extended_b	**Asynchronous Extended PGCB Clock Wake.** Extended version of async_wake_source_b that will assert	-	PCGU_AWW	PCGU

	asynchronously and deassert synchronously at least 6 PGCB clocks after async_wake_source_b deasserts.			

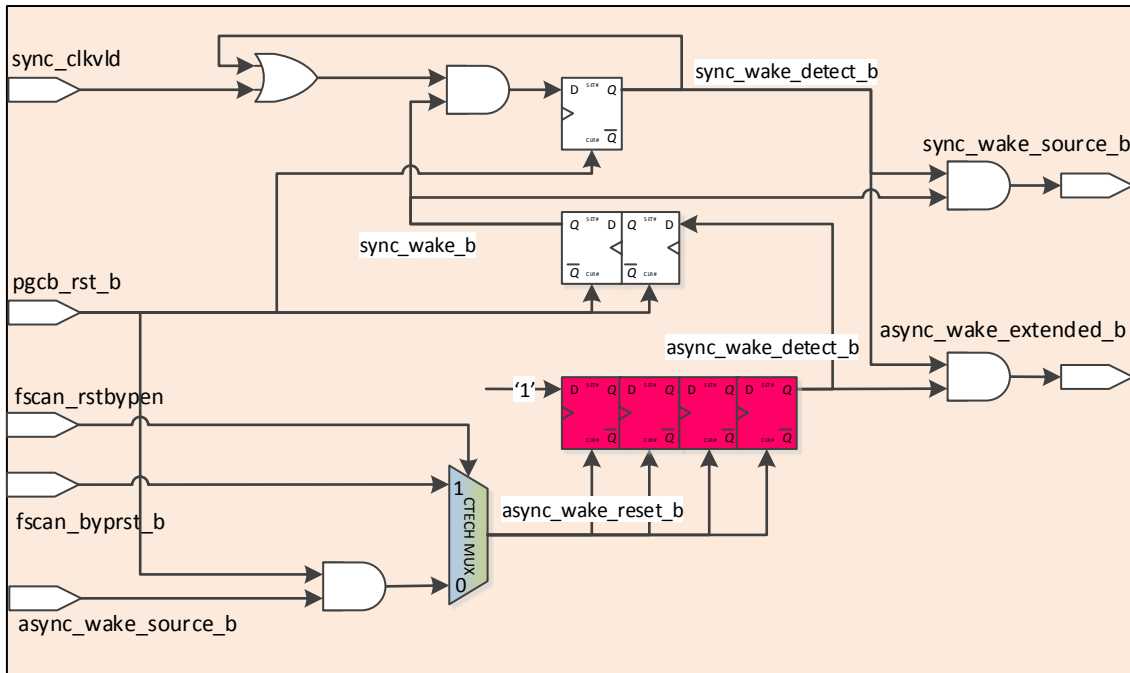## 1.3.1.2 Implementation Details

Figure 10: PCGU Asynchronous Wake Widget Logic Diagram

Reset State: When the pgcb_rst_b is asserted, the outputs of the PCGU_AWW will be asserted. The PCGU defaults to having the async_wake_b path masked so this does not change the final behavior of the pgcb_clkreq.

The PCGU_AWW design consists of the following pieces:
1. **2 Double-Flop Synchronizers (colored red)** – The synchronizers assert/clear asynchronously when async_wake_source_b or pgcb_rst_b assert. The output of the synchronizers deasserts synchronously after async_wake_reset_b has deasserted and 4 positive edges of pgcb_clk have occurred. This guarantees the wake event is asserted long enough to propagate through a normal synchronizer (sync_wake_reset_b) and then captured synchronously in the set-hold-clear flop.
2. **Traditional Double-Flop Synchronizer** (sync_wake_reset_b) – This synchronizer creates a fully synchronous signal out of the async_wake_source_b input to be captured in the set-hold-clear flop.
3. **Set-Hold-Clear Flop** – This flop is cleared (asserted) when the wake event is synchronized to PGCB clock (sync_wake_reset_b asserts). It is then held asserted, even if sync_wake_reset_b deasserts, until sync_clkvld assertes, indicating that the pgcb_clk is running.
4. **Final AND Gate** – The output of the 2 Double-Flop Synchronizers is ANDed with the output of the set-hold-clear flop, resulting in an output that asserts asynchronously with async_wake_source_b, and deasserts synchronously when the pgcb_clk has been ungated and is running.

5. **DFx SCAN Mux** – A SCAN mux is inserted to prevent the 2 Double-Flop Synchronizers from asserting randomly during SCAN mode.
6. **sync_wake_source_b** – a combination of the sync_wake_b and the output of the set-hold-clear flop, to be used as fanin to the sync_gate term to the PCGU. This output will stay asserted until both sync_clkvld has asserted and async_wake_source_b has deasserted.

1.3.1.3 Waveforms

1.3.1.3.1 Reset Deassertion, Without Wake
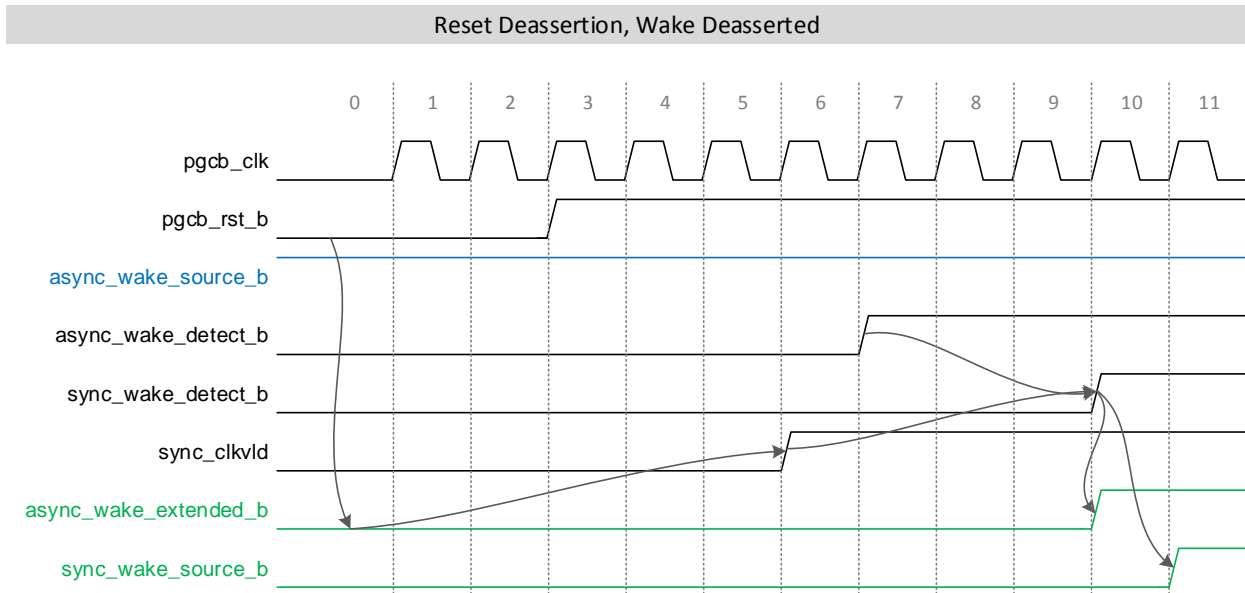
Figure 11: PCGU_AWW Reset Deassertion, Wake Deasserted

1.3.1.3.2 Reset Deassertion, With Wake Asserted

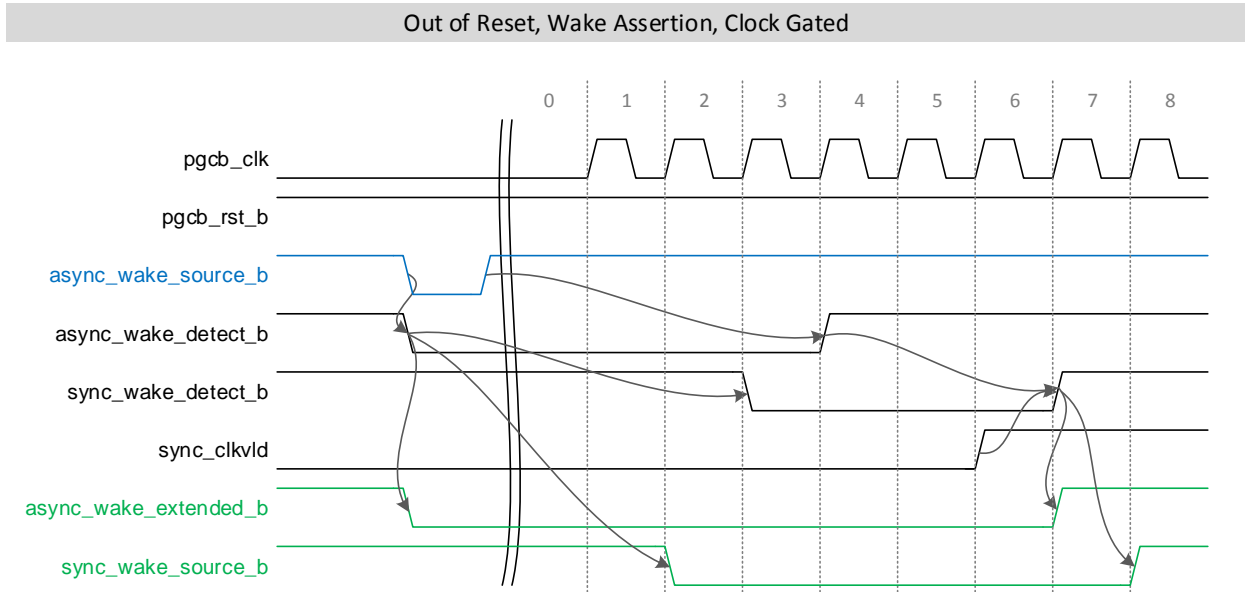### 1.3.1.3.3 Out of Reset, Wake Assertion, Clock Gated

Figure 13: PCGU_AWW Wake Assertion with Clock Gated

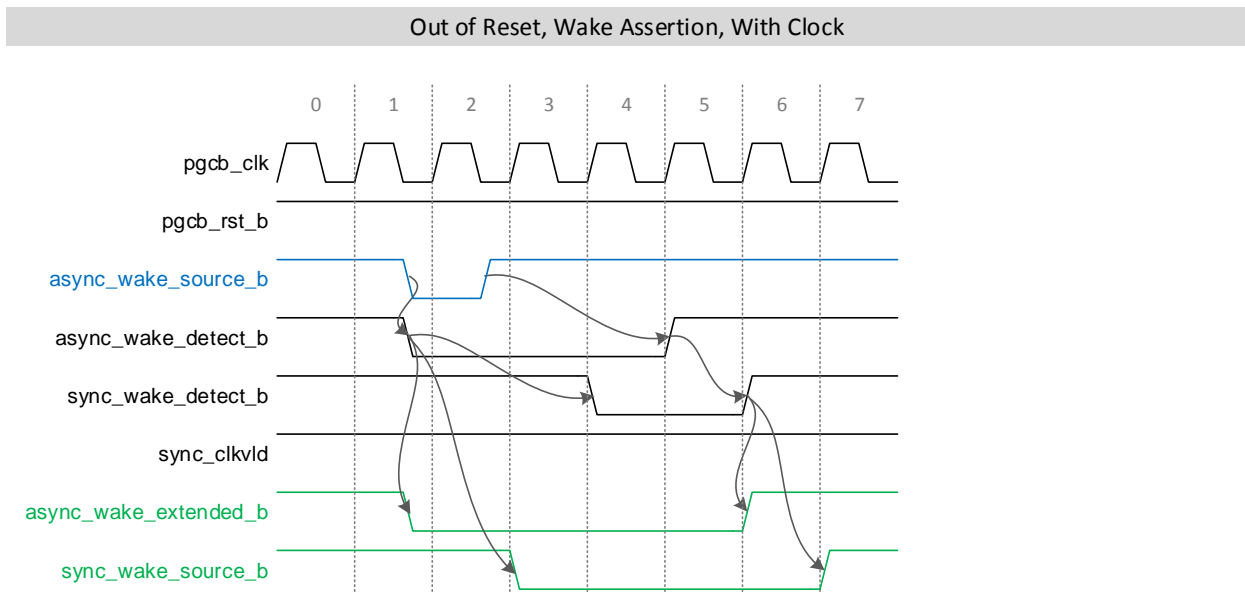### 1.3.1.3.4 Out of Reset, Wake Assertion, Clock Running

Figure 14: PCGU_AWW Wake Assertion with Clock Running

1.3.2 Tool Waivers

1.3.2.1 Lintra

Lintra waivers can be found under $MODEL_ROOT/tools/lint/waivers/pcguunit_waivers.lwv

1.3.2.2 Questa/0in CDC

The following waivers may be needed when running CDC:

```
#### PCGU_AWW Waivers ####
# PCGU async set circuit, combi logic on the clr_b term of the 2 doublesyncs used to set the
flops
cdc report crossing -through async_wake_source_b -through
i_pgcb_ctech_doublesync_async_wake_*.clr_b -rx_clock PGCB_CLK -module pcgu_aww -severity waived
```

1.4 PCGU Integration Reference Designs

The following sections describe methods for integrating the PCGU. They should be taken as reference designs and should not be used as-is without being

1.4.1 PCGU Integration (Completely Asynchronous)

While using an always running clock to deglitch the wake source(s) (ie Section 1.4.1.4.1) is probably the safest way to ensure that the async_wake_b input to the PCGU is glitch-free and meets the Tresetmin requirement, it may not be practical for all IPs. This section demonstrates an alternative approach to generating this wake signal using the PCGU_AWW described in Section 1.2.8.

Figure 15 shows an example of how to integrate this reference design with the PGCB and CDCs.

Figure 16 shows the detailed implementation of the PCGU glue logic reference design.

An example RTL implementation of this reference design is available under:
$MODEL_ROOT/verif/pcgu_ref/rtl/pgcbcg.sv

1.4.1.1 Implementation Details

1.4.1.1.1 Overview

The PCGU glue logic implementation described here shows a typical implementation for an IP using the PGCB and CDCs together. The glue logic uses the pcgu_aww to asynchronously capture wake events until the pgcb_clk is running (sync_clkvld is '1'). The combination of all these captured wake events drives the final "async_wake_b" input to the PCGU.

The pcgu_aww's sync_wake_source_b outputs contribute to the "sync_gate" input to the PCGU, such that additional synchronizers are not needed on these signals.

The PGCB's "pgcb_pok" output is used to mask off IP-Accessible wake/gate terms when in IP-Inaccessible power-gating. Note that "pgcb_pok" toggling could theoretically introduce a glitch on the "async_wake_b" input, however "pgcb_idle" will be '0' whenever "pgcb_pok" is changing, thus causing "async_wake_b" to assert regardless and should force an assertion period of at least Tresetmin.

Wake terms used by this design:
- General:
  - pmc_pg_wake=='1' from PMC (input directly into PCGU)
  - pgcb_idle=='0' from PGCB
  - clkreq=='1' from CDCs
- IP-Accessible:
  - gclock_req_async=='1' from CDCs
  - ism_fabric!='000' from CDCs
  - pwrgate_disabled toggling from CDCs

Gate terms used by this design:
- General:
  - pmc_pg_wake=='0' from PMC
  - pgcb_idle=='1' from PGCB
  - clkreq=='0' from CDCs
  - clkack=='0' from CDCs
- IP-Accessible:
  - gclock_req_async=='0' from CDCs
  - gclock_ack_async=='0' from CDCs
  - ism_fabric=='000' from CDCs
  - pwrgated_disabled stable from CDCs

PGCB Clock Gating can be disabled during IP-Accessible power gating based on the cfg_acc_clkgate_disabled config input.

1.4.1.1.2 Interface

1.4.1.1.2.1 Parameters

Parameter	Description
ICDC	Number of CDCs controlling IOSF clock domains (ex. prim_clk, side_clk)
NCDC	Number of CDCs controlling non-IOSF clock domains
IGCLK_REQ_ASYNC	Total number of gclock_req_async inputs to all the IOSF CDCs
NGCLK_REQ_ASYNC	Total number of gclock_req_async inputs to all the non-IOSF CDCs

1.4.1.1.2.2 Clocks and Resets

Signal	Description	CLK	DIR	SRC/ DEST
pgcb_clk	PGCB Clock.	PGCB	In	SOC
pgcb_clkreq	PGCB Clock CLKREQ	-	Out	SOC
pgcb_clkack	PGCB Clock CLKACK	-	In	SOC
pgcb_rst_b	Synchronous PGCB reset. Note: this must by rising-edge synchronized to the ungated pgcb_clk.	PGCB	In	PGCB Glue Logic

	The PGCB reset with its rise-edge synchronized to pgcb_clk. The same signal will feed the input to the PGCB and CDCs.			
iosf_cdc_clock[ICDC-1:0]	**IOSF Primary/Sideband Clocks.** Vector of raw/ungated IOSF Primary/Sideband Clocks that are controlled by IOSF CDCs. The order of the "iosf_cdc_clock", "iosf_cdc_reset_b" and "iosf_cdc_ism_fabric" vectors should match such that a given index in each vector corresponds to the same CDC/IOSF clock domain.	IOSF	In	SOC
iosf_cdc_reset_b[ICDC-1:0]	**IOSF Primary/Sideband Resets.** Vector of IOSF resets, rise-edge synchronized to the IOSF clock of the same index in "iosf_cdc_clock". These will be taken from the "sync_reset_b" output of the CDCs.	IOSF	In	CDCs

1.4.1.1.2.3 DFx

Signal	Description	CLK	DIR	SRC/ DEST
fscan_byprst_b[8:0]	**Fabric Scan Bypass Reset.** This signal is a reset input for scan operations that bypasses the internal agent reset logic and applies a reset directly to the agent. The reset override signal group must be implemented for IP-blocks with embedded or derived internal reset signals.	-	In	SOC
fscan_rstbypen[8:0]	**Fabric Scan Reset Bypass Enable.** This signal will enable the ability for the bypass reset signals to be active. The reset override signal group must be implemented for IP-blocks with embedded or derived internal reset signals.	-	In	SOC
fscan_clkungate	**Fabric Scan Clock Ungate.** This signal will override the internal clock gating and ungate the clock.	-	In	SOC
visa_bus[31:0]	**PCGU Glue Logic Visa Vector.** The VISA ULM must be in the Always-ON domain.	-	Out	SIP

1.4.1.1.2.4 Configuration Registers

Signal	Description	CLK	DIR	SRC/ DEST
cfg_t_clkgate[3:0]	**PGCB Clock Gating Hysteresis Delay.** Specify the minimum number of delay clocks the PGCB clock gate sequencing should wait, before	PGCB	In	SIP

	enabling the PGCB clock to be gated at the trunk-level.			
cfg_t_clkwake[3:0]	**PGCB Clock Wake Hysteresis Delay.** Specify the minimum number of delay clocks the PGCB clock wake sequencing should wait, before enabling the PGCB clock consumer to use the PGCB clock.	PGCB	In	SIP
cfg_acc_clkgate_disabled	**IP-Accessible PGCB Clock Gating Disable.** When asserted '1', prevents the PGCB clock from gating during IP-Accessible power-gating, will still allow the clock to gate as part of IP-Inaccessible power-gating.	PGCB	In	SIP

1.4.1.1.2.5 Functional Interface

Note: All of the inputs listed below are required to be glitch free.

Signal	Description	CLK	DIR	SRC/ DEST
	IP-Accessible Wake/Gate Indications			
iosf_cdc_ism_fabric[ICDC-1:0][2:0]	**IOSF ISM States.** Vector of 3-bit ISM states from the IOSF Fabrics/Routers corresponding to each IOSF CDC. These will be the same signals that connect to the IOSF CDCs' "ism_fabric" input. Each ISM State in this vector goes through combi-logic to detect if it is in a non-idle state, and then fed through a flop on the corresponding "iosf_cdc_clock" and "iosf_cdc_reset_b" to create a glitch free indication of a fabric wake. The native IOSF clock can be used as it is guaranteed to be running if the Fabric/Router's ISM state changes.	IOSF	In	SoC
iosf_cdc_gclock_req_async [IGCLK_REQ_ASYNC-1:0] / non_iosf_cdc_gclock_req_ async [NGCLK_REQ_ASYNC-1:0]	**IP Clock Requests.** Vector of individual clock requests from IP logic. Taken from CDCs' "gclock_req_async" inputs. Separate input vectors exist for IOSF CDCs and Non-IOSF CDCs.		In	SIP
iosf_cdc_gclock_ack_async [IGCLK_REQ_ASYNC-1:0] / non_iosf_cdc_gclock_ack_ async [NGCLK_REQ_ASYNC-1:0]	**IP Clock Acknowledgments.** Vector of individual clock request acknowledgments from the CDCs to the IP logic. Taken from CDCs' "gclock_ack_async" outputs.		In	CDCs

	Separate input vectors exist for IOSF CDCs and Non-IOSF CDCs.			

async_pwrgate_disabled	**Power Gate Disabled Indication.**
	Indication of whether IP-Accessible power-gating is disabled. This signal will be similar to the "pwrgate_disabled" input on the CDCs, however it is required to be asynchronous.
	
	A change in the value of this signal wakes up the PGCB clock. For this reason, this signal must be able to toggle when the PGCB clock is gated.
	
	In the typical case, the CDC integration guide suggests that "pwrgate_disabled" could consist of the following logic:
	`(PCE.HAE) ||`
	`(PCE.D3HE && PMCSR[1:0]=='11') ||`
	`(PCE.I3E && D0i3C[2]=='1') ||`
	`(PCE.PMCRE && pmc_ip_sw_pg_req_b=='0')`
	
	If the registers contributing to this logic are part of the power-gate domain, then the PGCB clock would be ungated when they are changed due to the corresponding CDC's clkreq being asserted. So the only portion that would be required for this input would be:
	`(PCE.PMCRE && pmc_ip_sw_pg_req_b=='0')`
	And "pmc_ip_sw_pg_req_b" would be the raw/async input from PMC.
	
	Note that some IPs may have other logic contributing to the "pwrgate_disabled" term, and so this logic will be IP specific and should be looked at carefully for each IP.			

	IP-Accessible Wake/Gate Indications			
iosf_cdc_clkreq[ICDC-1:0] / non_iosf_cdc_clkreq[NCDC-1:0]	**CDC Clock Requests.** Vector of final clock request outputs from the CDCs that go to the SoC. Taken from CDCs' "clkreq" outputs. Separate input vectors exist for IOSF CDCs and Non-IOSF CDCs	-	In	CDCs
iosf_cdc_clkack[ICDC-1:0] / non_iosf_cdc_clkack[NCDC-1:0]	**CDC Clock Acknowledgments.** Vector of final clock request acknowledgment inputs to the CDCs from the SoC. Taken from CDCs' "clkack" inputs. Separate input vectors exist for IOSF CDCs and Non-IOSF CDCs	-	In	SoC
pmc_pg_wake	**PMC Power Gate Wake.**	-	In	SoC

	Raw (unsynchronized) wake input from the PMC. The synchronized version of this signal feeds the "pwrgate_pmc_wake" input of the CDCs.			
pgcb_idle	**PGCB Idle Indication.** The "pgcb_idle" output from the PGCB.	PGCB	In	PGCB
	Other Control Signals			
pgcb_pok	**PGCB POK Indication.** The "pgcb_pok" output from the PGCB. This is used to decide whether or not to consider the IP-Accessible wake/gate terms. Note that this signal is used in such a way that it could introduce glitches on the final "async_wake_b" input to the PCGU. However, this reference design uses the assumption that the pgcb_idle indication will be '0' when "pgcb_pok" is changing and thus the "async_wake_b" will always resolve to a good asserted value that meets the Tresetmin width.	PGCB	In	PGCB
sync_clkvld	**PGCB Synchronous Clock Valid Indication.** This is the sync_clkvld output from the PCGU. Due to fact that the CDC/PGCB do not natively comprehend if their clock is available, it is recommended that IPs gate the pgcb_clk to the PGCB/CDCs when this signal is deasserted (0). This signal can feed the "enclk" input of a latch-based clock-gate cell directly. **Note**: the clock-gate cell is not included within the reference design to emphasize that this clock-gate is not required by the clock gating logic to function properly, but rather it is recommended so that SoC doesn't gate the clock when the PGCB/CDC are in a non-idle state, as they may transition even if pgcb_clkreq is low.	PGCB	Out	SIP

1.4.1.1.3 Optimizations

This example shows a pcgu_aww widget used on every wake source. It may be possible to optimize this logic use the wake sources directly without this widget. The following table shows which wake sources may be used directly under certain circumstances.

Note: if the pcgu_aww is not used, a doublesync will be needed to synchronize each term as it feeds into the sync_gate logic.

Signal	Details / Conclusion	Assumptions

pgcb_idle	Synchronous to pgcb_clk. Thus if the clock is able to be gated when pgcb_idle deasserts, will require the clock to be running again in order to assert. This is also used to mask any possible glitches from pgcb_pok changing, however when it is deasserted, will stay deasserted for many clock cycles, so may not need the pcgu_aww to extend it for more cycles. Conclusion: May not need pcgu_aww	-using the PGCB -pgcb_clk is gated when sync_clkvld is '0'
*_cdc_clkreq	Asserts in the PGCB clock domain of the CDC and will stay asserted until all wake sources have been synchronized and seen deasserted. Thus, this signal requires the PGCB clock to be running for both assertion and deassertion and so it is guaranteed to stay asserted until the PGCB clock is running and valid. Conclusion: May not need pcgu_aww	-using the PGCB+CDC -clkreq is driven from the CDC -pgcb_clk is gated when sync_clkvld is '0'
*_cdc_gclock_req_async	When asserted, will require the corresponding gclock_ack_async to assert before it can deassert. gclock_ack_async will only assert when the CDC's clkreq and clkack are asserted. It is theoretically possible for CDC's clkreq to assert on the boundary of the PGCB clock being gated, however the CDC's clkreq will not deassert until the pgcb_clk is running again, thus ensuring the PGCB clock will wake up again. Conclusion: May not need pcgu_aww	-using the PGCB+CDC -clkreq is driven from the CDC -pgcb_clk is gated when sync_clkvld is '0'
*_ism_fabric	ism_fabric can leave IDLE without any relationship to PGCB clock. However, when it leaves IDLE, it will stay non-IDLE until the agent returns to IDLE. Additionally, the agent is required to assert its clkreq before it is allowed to return to IDLE, this clkreq will trigger the PGCB clock to be ungated. Conclusion: May not need pcgu_aww	-using the PGCB+CDC -clkreq is driven from the CDC -pgcb_clk is gated when sync_clkvld is '0'
*pwrgate_disabled	May toggle multiple times while the PGCB clock is gated. The diff between the synced version and the raw version causes a wake, thus if the value changes back to the synced version before the PGCB clock is ungated, the wake would disappear. The pcgu_aww would detect the first diff and keep the wake asserted until the clock is running. Conclusion: Use the pcgu_aww	

1.4.1.2 Cautions

1.4.1.2.1 Synchronous Resets

This implementation assumes that there is no logic that is synchronously reset in the pgcb_clk gated domain (pgcb_gclk) and pgcb_rst_b domain as pgcb_gclk will be gated until the reset deasserts and thus the assertion may not be captured by synchronously reset logic.
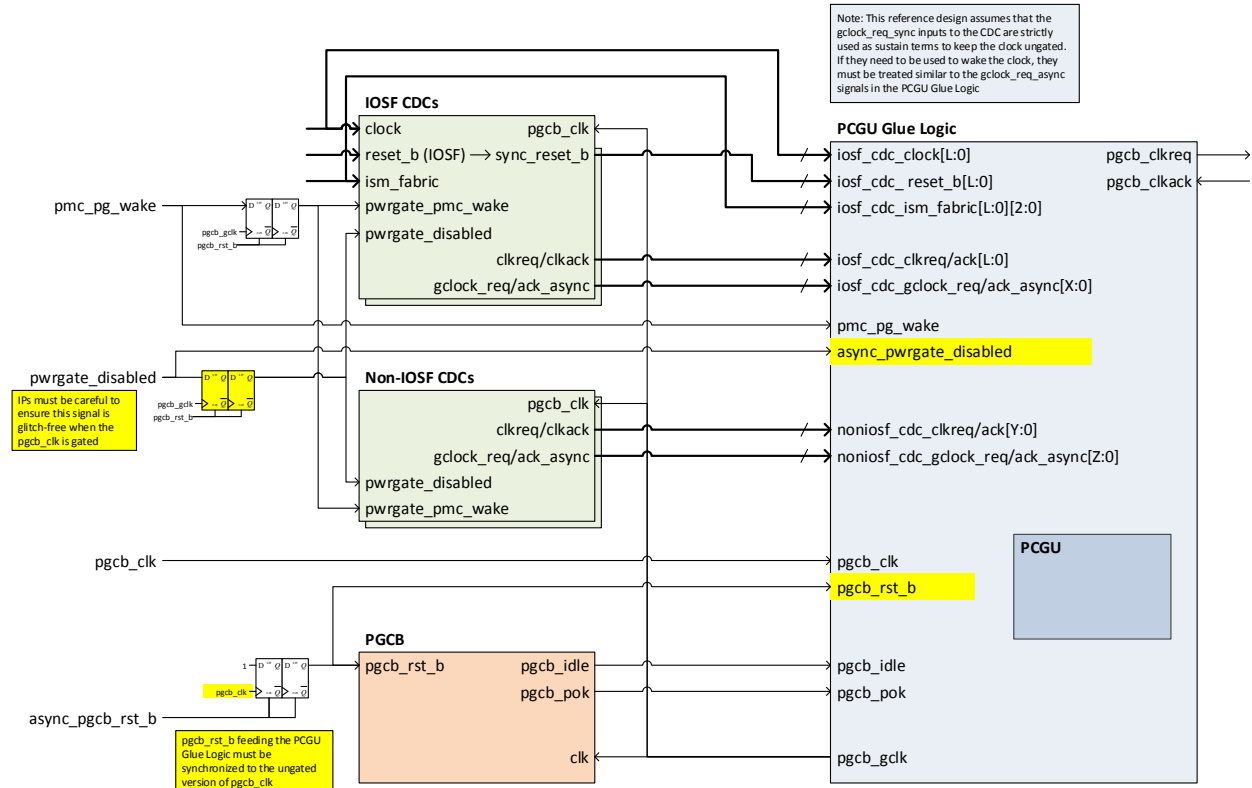
1.4.1.3 Diagrams

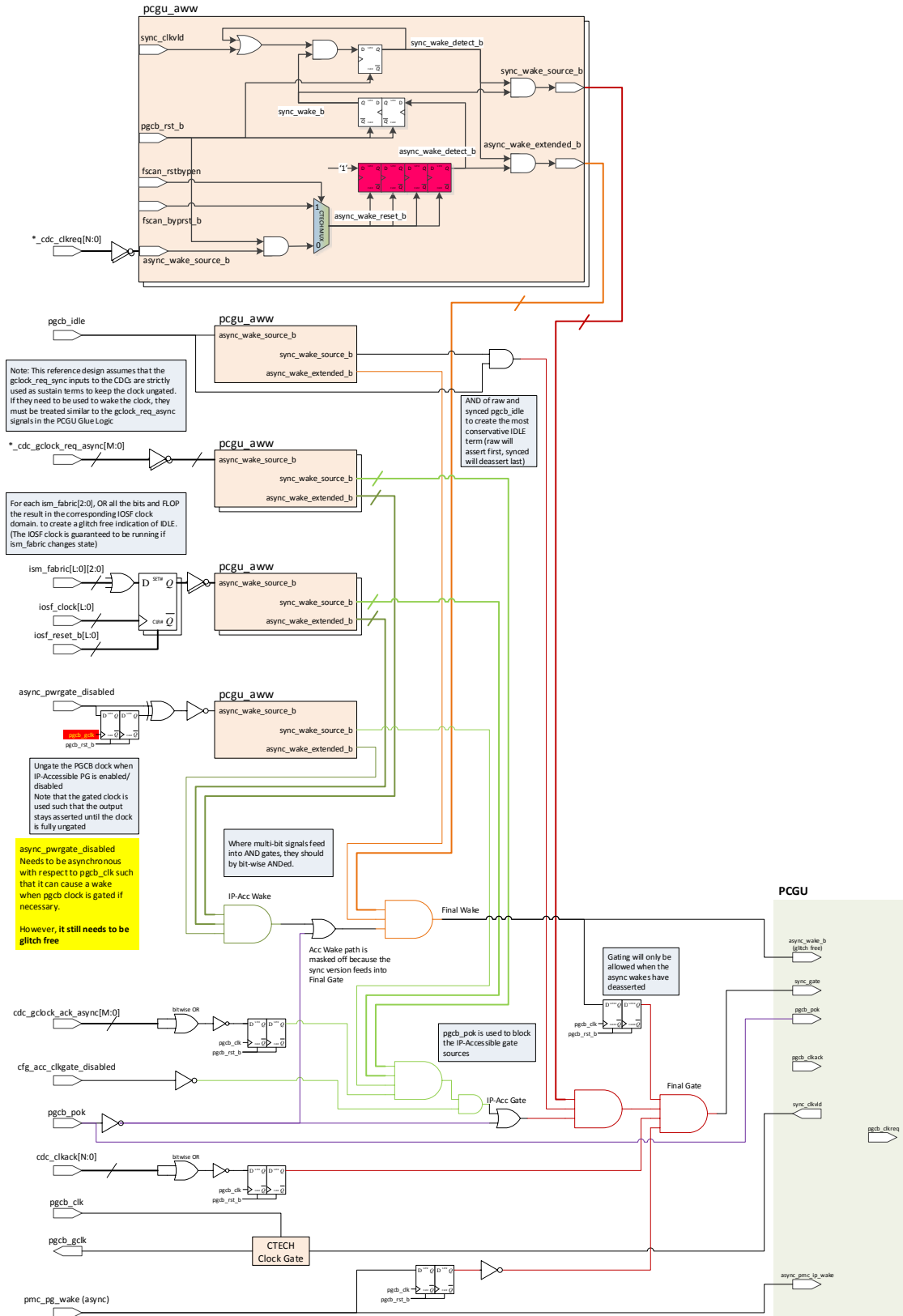Figure 15: Top Level PGCB Clock Gating Block Diagram

Figure 16: Top Level PGCB Clock Gating Block Diagram

1.4.1.4 Tool Waivers

1.4.1.4.1 Lintra

Lintra waivers specific to this reference design are under
$MODEL_ROOT/verif/pcgu_ref/waivers/pgcbcg_waivers.lwv

1.4.1.4.2 Questa/0in CDC

The following waivers may be needed when running CDC (waivers for the pcgu and pcgu_aww are found in
the corresponding sections that describe them)

```
#### PGCBCG Reference Design Waivers ####

# CDC clkacks are combined before feeding double_sync, hysteresis in the PCGU will filter any
    clocks where a '0' is synchronized erroneously
cdc report crossing -scheme combo_logic -through *iosf_cdc_clkack -through
    *i_pgcb_ctech_doublesync_idle_event.d -rx_clock PGCB_CLK -severity waived -module pgcbcg

# CDC gclock_ack's are combined before feeding double_sync, hysteresis in the PCGU will filter
    any clocks where a '0' is synchronized erroneously
cdc report crossing -scheme combo_logic -through *iosf_cdc_gclock_ack_async* -through
    *i_pgcb_ctech_doublesync_idle_event.d -severity waived -module pgcbcg

# Async signals feed into the visa bus, this waiver might be needed depending on the clock of the
    pgcb_visa
cdc report crossing -scheme no_sync -through visa_bus -severity waived -module pgcbcg
```

1.4.2 PCGU Integration (Using Always On Clock)

One of the difficulties with the design described previously is ensuring that async_wake_b is glitch free. If an
IP feels that it is too difficult to guarantee, there is an option to de-glitch that input by synchronizing to an
always-running clock (such as RTC-clock). Details on this option are no longer provided in the integration guide
and will be IP-specific.

2. Open Issues

No	Issue	Description	Resolution	Status
1	PGCB Clock usage in CDC after de-asserting the CLKREQ to SOC	When IP or Fabric initiates a wake event to the CDC, how does CDC propagate event to PGCB clock domain? What will happens if the PGCB clock stop running for a while?		Open
2.	IP-Specific usage of the PGCB Clock in the AON domain	Should we provide a hook to aggregate the PGCB clock request from the IP-Specific AON logic?	[130702] There is no requirement to aggregate the PGCB clock request. OK to expose a dedicated et of CLKREQ/CLKACK for PGCB clock	Closed
3.	Glitchy Preset Pin	Will it cause any meta-stability issue? If so, how to quantify and eliminate the impact?	[130716] Met with Library folks and conclude that there is no best way to quantify the impact of a meta-stability. The recommendation is to have the IP-specific mechanism to avoid meta-stability. There is a tool from Haswell that could be leveraged to check the glitch violation.	Closed
4	Local PGCB Clock Gating	Do we need to locally gate the PGCB clock once it enable PCG to shut down the clock?	[130702] Michael K confirmed that the local PGCB clock gating is not a requirement for BXT.	Closed
			•	

3. Appendix

3.1 Synchronous Assertion and De-assertion of the PGCB Clock Request

Figure 17 illustrates the PGCB Clock Control Block with Synchronous Assertion and De-Assertion of the PGCB Clock Request.
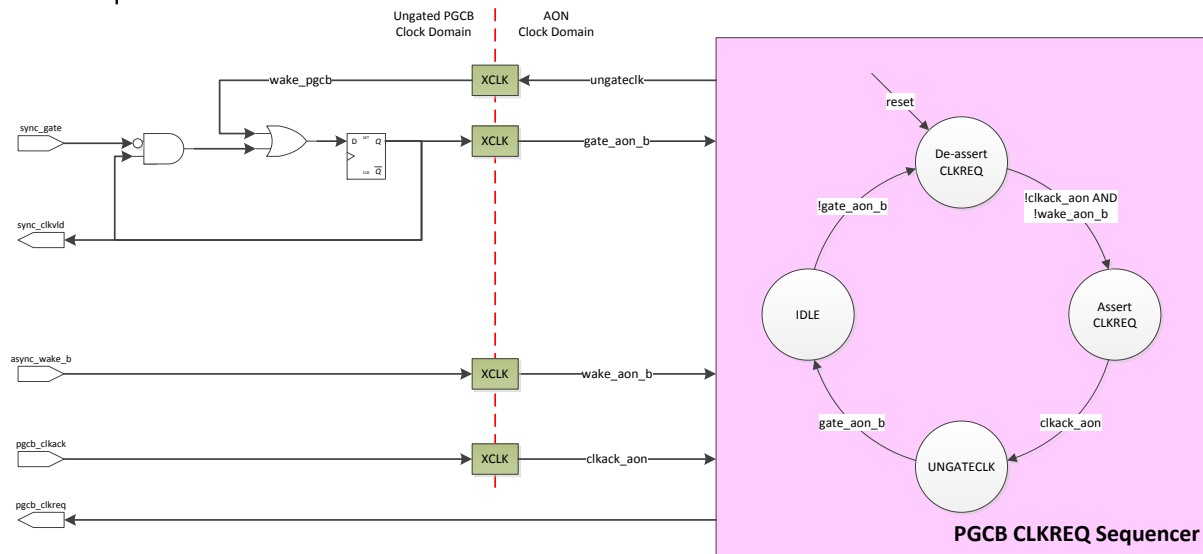

Figure 17: PGCB Clock Control Block w/ Synchronous Handling of the PGCB Clock Request

The implementation of the PGCB Clock Control logic is summarized as follow:
1. Use an always-on clock source (e.g. RTC clock) to synchronize all the sync_clkvld, async_wake_b events, as well as the pgcb_clkack.
2. Implement a PGCB CLKREQ sequencer to synchronously evaluate the PGCB Clock Gate and Wake condition in the always-on-clock domain, and manage the PGCB CLKREQ full handshake with SOC, and the Local PGCB clock gating full handshake within the PGCB clock control block.
3. Generate the sync_clkvld in PGCB clock domain.
    a. Once the IP enable the PGCB clock to be gated by asserting the sync_gate, the sync_clkvld will be gated immediately.
    b. The sync_clkvld will be asserted once the PGCB CLKREQ Sequencer brings back the PGCB clock from SOC.
    c. A full handshake between gate_aon_b and wake_pgcb must be used to prevent any race condition on the sync_clkvld generation.
        i. When sync_clkvld is de-asserted, the gate_aon_b will be asserted in the AON clock domain. Once the gate_aon_b is asserted, it will remains asserted until wake_pgcd is asserted.
        ii. When the wake_pgcb is asserted, the sync_clkvld will be asserted and eventually de-assert the gate_aon_b. Once the wake_pgcb is asserted, it will remains asserted until the gate_aon_b is de-asserted.
4. Implement a clock request control logic to generate the pgcb_clkreq in the always-on-clock domain:
    a. Reset by pgcb_rst_b that synchronized to the always-on clock domain. The default reset value should be 0 (De-Asserted).
    b. When pgcb_clkack is asserted, de-assert the pgcb_clkreq when the IP enable the PGCB clock to be gated.
    c. When pgcb_clkack is de-asserted, assert the pgcb_clkreq when any of the PGCB wake conditions is detected.