**PGCB Clock Gating**

**Micro-Architecture Specification (MAS)**

**Revision 1.20**

**Last Update: 12 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** |
|  | 0.00  06/14/2013 | Kah Meng | Initial revision and incorporated the feedback from the Pre-Aspec review on ww24.4. |
|  | 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. |
|  | 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. |
|  | 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](https://hsd-ifs.fm.intel.com/hsd/ifs/issue/default.aspx?issue_id=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](https://hsd-ifs.fm.intel.com/hsd/ifs/issue/default.aspx?issue_id=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**](mailto:kah.meng.yeem@intel.com) |

Table of Contents

**[Revision History](#_Toc390267819)** [iii](#_Toc390267819)

**[Table of Initials](#_Toc390267820)** [iv](#_Toc390267820)

**[Table of Figures](#_Toc390267821)** [vii](#_Toc390267821)

**[Table of Tables](#_Toc390267822)** [viii](#_Toc390267822)

[1. PGCB Clock Gating 1](#_Toc390267823)

[1.1 Theory of Operation 1](#_Toc390267824)

[1.2 PGCB Clock Gate Control Block (PCGU) 5](#_Toc390267825)

[1.2.1 Parameters 5](#_Toc390267826)

[1.2.2 Configuration Register Bit/Straps 6](#_Toc390267827)

[1.2.3 Interface Signals 6](#_Toc390267828)

[1.2.4 Implementation Details 9](#_Toc390267829)

[1.2.5 Waveforms 13](#_Toc390267830)

[1.2.6 Glitch Prevention on Reset 24](#_Toc390267831)

[1.2.7 Risks 24](#_Toc390267832)

[1.2.8 Tool Waivers 26](#_Toc390267834)

[1.3 PCGU Async Wake Widget (pcgu\_aww) 27](#_Toc390267835)

[1.3.2 Tool Waivers 32](#_Toc390267836)

[1.4 PCGU Integration Reference Designs 32](#_Toc390267837)

[1.4.1 PCGU Integration (Completely Asynchronous) 33](#_Toc390267838)

[1.4.2 PCGU Integration (Using Always On Clock) 42](#_Toc390267839)

[2. Open Issues 45](#_Toc390267840)

[3. Appendix 46](#_Toc390267841)

[3.1 Synchronous Assertion and De-assertion of the PGCB Clock Request 46](#_Toc390267842)

**Table of Figures**

[Figure 1: PGCB Clock Gating Conceptual Block Diagram 1](#_Toc390267843)

[Figure 2: PGCB Clock Gate Entry and Wake Evaluation 4](#_Toc390267844)

[Figure 3: PGCB Clock Control Block w/ Asynchronous Handling of the PGCB Clock Request 10](#_Toc390267845)

[Figure 4: PCGU IP-Inaccessible/Reset Wake Sequence (DEF\_PWRON==0) 15](#_Toc390267846)

[Figure 5: PCGU IP-Inaccessible/Reset Wake Sequence (DEF\_PWRON==1) 17](#_Toc390267847)

[Figure 6: PCGU IP-Inaccessible/Reset Gate Sequence (DEF\_PWRON==0) 18](#_Toc390267848)

[Figure 7: PCGU IP-Inaccessible/Reset Gate Sequence (DEF\_PWRON==1) 19](#_Toc390267849)

[Figure 8: PCGU IP-Accessible Gate Sequence 21](#_Toc390267850)

[Figure 9: PGCB Clock Wake Sequencing 24](#_Toc390267851)

[Figure 10: PCGU Asynchronous Wake Widget Logic Diagram 29](#_Toc390267852)

[Figure 8: PCGU\_AWW Reset Deassertion, Wake Deasserted 30](#_Toc390267853)

[Figure 9: PCGU\_AWW Reset Deassertion, Wake Asserted 31](#_Toc390267854)

[Figure 10: PCGU\_AWW Wake Assertion with Clock Gated 31](#_Toc390267855)

[Figure 11: PCGU\_AWW Wake Assertion with Clock Running 32](#_Toc390267856)

[Figure 12: Top Level PGCB Clock Gating Block Diagram 40](#_Toc390267857)

[Figure 13: Top Level PGCB Clock Gating Block Diagram 41](#_Toc390267858)

[Figure 15: PGCB Clock Control Block w/ Synchronous Handling of the PGCB Clock Request 46](#_Toc390267859)

**Table of Tables**

[Table 1: PGCB Clock Gate Evaluation (Per PGD domain) 2](#_Toc390267860)

[Table 2: Comparison of PGCB Clock Control Implementation Options 5](#_Toc390267861)

# 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 section 1.4 for an example reference design that can be used to implement the design described here.*



Figure 1: PGCB Clock Gating Conceptual Block Diagram

## 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:
   1. The pg\_wake = 0
   2. All pgcb\_idle = 1, and
   3. 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:
   1. IP-Inaccessible Idle Condition
   2. All shim\_clkreq\*/clkack\* = 0, and
   3. 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:
   1. 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:
   1. Any IP-Inaccessible Wake Condition
   2. Any shim\_clkreq\* = 1
   3. Any ism\_fabric\* not in IDLE state, OR
   4. 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.

Table 1: PGCB Clock Gate Evaluation (Per PGD domain)

|  |  |  |  |
| --- | --- | --- | --- |
| **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

## 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.

### 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’ |

### 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 |

### Interface Signals

#### SOC Interface

##### SOC-Specific Parameter

Nil

##### 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-gating. This allows it to be used as a well-behaved wake event that will not deassert until clkack has asserted. | - | SOC | PCGU |
|  |  |  |  |  |

##### 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 |
|  |  |  |  |  |

#### IP Specific Power Control Glue Logic Interface

##### 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 |
|  |  |  |  |  |

##### 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 enabling the PGCB clock consumer to use the PGCB clock. | PGCB | SIP | PCGU |
|  |  |  |  |  |
|  | **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 |
|  |  |  |  |  |

##### 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 |

#### 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.  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 | PGCB (except as noted) | PCGU | SIP |
|  |  |  |  |  |

### 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:
   1. 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.
   2. 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.
   3. Once the clock wake hysteresis timer is timeout, the CLKREQ sequencer will transition from NOSEL to UNGATECLK state, and assert the sync\_clkvld.
   4. 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.
   5. 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.
   6. 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.
   7. 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:
   1. Reload the timer with the clock wake hysteresis value when the CLKREQ sequencer is transitioning from SELWAKE to NOSEL state
   2. Reload the timer with the clock gate hysteresis value when the CLKREQ sequencer is transitioning from UNGATECLK to GATEPEND state
   3. 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 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.
   1. Additional SCONs are needed to waive the clock cross violations on the async\_wake\_b generation.



#### 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.

### Waveforms

#### 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.
   1. 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 4: PCGU IP-Inaccessible/Reset Wake Sequence (DEF\_PWRON==0)

#### 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.
   1. Note that defon\_flag also causes sync\_clkvld to be asserted out of reset, which will open up the clock gate that it feeds.
   2. 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.
   1. 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)

#### 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).
   1. 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.
   1. 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)
   2. 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)



Figure 6: PCGU IP-Inaccessible/Reset Gate Sequence (DEF\_PWRON==0)



Figure 7: PCGU IP-Inaccessible/Reset Gate Sequence (DEF\_PWRON==1)

#### 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).
   1. 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.



Figure 8: PCGU IP-Accessible Gate Sequence

#### 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.
   1. 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.
   1. 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.
   1. 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.
   1. Note that the async\_wake event is ignored when the CLKREQ sequencer is in UNGATECLK state.
   2. Also note that in UNGATECLK, the acc\_wake\_flop is cleared synchronously.



Figure 9: PGCB Clock Wake Sequencing

### 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.

### 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.
   1. [To Do] Need a waveform to show the race condition.
      1. PGCB Clock Gating Block has de-asserted the pgcb\_clkreq
      2. 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).
      3. 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.
      4. 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).
   2. [To Do] Need to analyze the impacts, especially in the CDC blocks
   3. [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.
   1. [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.
   1. 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.



### Tool Waivers

#### Lintra

Lintra waivers can be found under $MODEL\_ROOT/tools/lint/waivers/pcguunit\_waivers.lwv

#### 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

## 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.

#### Interface Signals

##### 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 |

##### 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 |

##### 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 asynchronously and deassert synchronously at least 6 PGCB clocks after async\_wake\_source\_b deasserts. | - | PCGU\_AWW | PCGU |

#### 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.

#### Waveforms

##### Reset Deassertion, Without Wake



Figure 11: PCGU\_AWW Reset Deassertion, Wake Deasserted

##### Reset Deassertion, With Wake Asserted



Figure 12: PCGU\_AWW Reset Deassertion, Wake Asserted

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



Figure 13: PCGU\_AWW Wake Assertion with Clock Gated

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



Figure 14: PCGU\_AWW Wake Assertion with Clock Running

### Tool Waivers

#### Lintra

Lintra waivers can be found under $MODEL\_ROOT/tools/lint/waivers/pcguunit\_waivers.lwv

#### 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

## 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

### PCGU Integration (Completely Asynchronous)

While using an always running clock to deglitch the wake source(s) (ie Section 1.4.1.3.1) 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**

#### Implementation Details

##### 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.

##### Interface

###### 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 |
|  |  |

###### 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.  The PGCB reset with its rise-edge synchronized to pgcb\_clk. The same signal will feed the input to the PGCB and CDCs. | PGCB | In | PGCB  Glue  Logic |
| 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 |

###### 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 |

###### 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 enabling the PGCB clock to be gated at the trunk-level. | PGCB | In | SIP |
| 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 |

###### 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.  Separate input vectors exist for IOSF CDCs and Non-IOSF CDCs. |  | In | 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.**  Raw (unsynchronized) wake input from the PMC. The synchronized version of this signal feeds the “pwrgate\_pmc\_wake” input of the CDCs. | - | In | SoC |
| 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 |

##### 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** |  |

#### Cautions

##### 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.

#### Diagrams



Figure 15: Top Level PGCB Clock Gating Block Diagram



Figure 16: Top Level PGCB Clock Gating Block Diagram

#### Tool Waivers

##### Lintra

Lintra waivers specific to this reference design are under $MODEL\_ROOT/verif/pcgu\_ref/waivers/pgcbcg\_waivers.lwv

##### 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**

### 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.

# 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** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Appendix

## 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.
   1. Once the IP enable the PGCB clock to be gated by asserting the sync\_gate, the sync\_clkvld will be gated immediately.
   2. The sync\_clkvld will be asserted once the PGCB CLKREQ Sequencer brings back the PGCB clock from SOC.
   3. A full handshake between gate\_aon\_b and wake\_pgcb must be used to prevent any race condition on the sync\_clkvld generation.
      1. 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.
      2. 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:
   1. Reset by pgcb\_rst\_b that synchronized to the always-on clock domain. The default reset value should be 0 (De-Asserted).
   2. When pgcb\_clkack is asserted, de-assert the pgcb\_clkreq when the IP enable the PGCB clock to be gated.
   3. When pgcb\_clkack is de-asserted, assert the pgcb\_clkreq when any of the PGCB wake conditions is detected.