External power supply design guide for TRAVEO™ T2G family

About this document

Scope and purpose

This application note explains the procedure to design external power supply for TRAVEO™ T2G family CYT3/4 series.

Intended audience

This document is intended for anyone using TRAVEO™ T2G family CYT3/4 series.

Associated part family

CYT3/CYT4 series

Table of contents

Table of contents

1 Introduction

This document describes the procedure to design external power supply for TRAVEO™ T2G MCU family CYT3 and CYT4 Series. CYT3B/4B series are the body controller high family of TRAVEO™ T2G MCUs targeted at automotive systems such as high-end body-control units. CYT3D and CYT4D series are the cluster 2D family of TRAVEO™ T2G MCUs targeted at automotive systems such as instrument clusters.

These MCUs have two built-in regulators (Active Regulator, DeepSleep Regulator). In addition, MCUs of CYT3B/CYT4B series have High Current Regulator Controller (REGHC) and MCUs of CYT3D/CYT4D series have PMIC controller. REGHC or PMIC controller is used to control the external power system. This document explains the design recommendations and restrictions regarding the external power system using REGHC or PMIC controller. The document also describes the handover procedure between internal regulators and external power with various use cases.

See **Table 1** for external power structures supported by each series.

For more details on the device features and relevant settings, see the TRAVEO™ T2G architecture technical reference manual (TRM) **[2]** and the dedicated device datasheet **[1]**.

2 TRAVEO™ T2G power supply system

The following are the features of the TRAVEO™ T2G power supply system:

- V_{DDD} power supply voltage range of 2.7 V to 5.5 V
- Core supply[1] at V_{CCD}
- Multiple on-chip regulators:
  - Active Regulator to power the MCU in Active or Sleep mode in case of low current consumption
  - DeepSleep Regulator to power peripherals in DeepSleep mode. A PMIC may be configured to supply power in DeepSleep power mode, in which case the DeepSleep Regulator can be disabled.
- REGHC, which supports higher load current in Active and Sleep power modes using a pass transistor or by controlling a PMIC.
- PMIC controller, which supports higher load current in Active and Sleep power modes by controlling a PMIC

2.1 Block diagram

TRAVEO™ T2G device has two built-in regulators (Active Regulator, DeepSleep Regulator). In addition, the MCU has REGHC or PMIC controller for external power supply. REGHC or PMIC controller controls V_{CCD} through four terminals: DRV_VOUT and EXT_PS_CTL [0:2]. **Figure 1** shows the power supply system for TRAVEO™ T2G device.

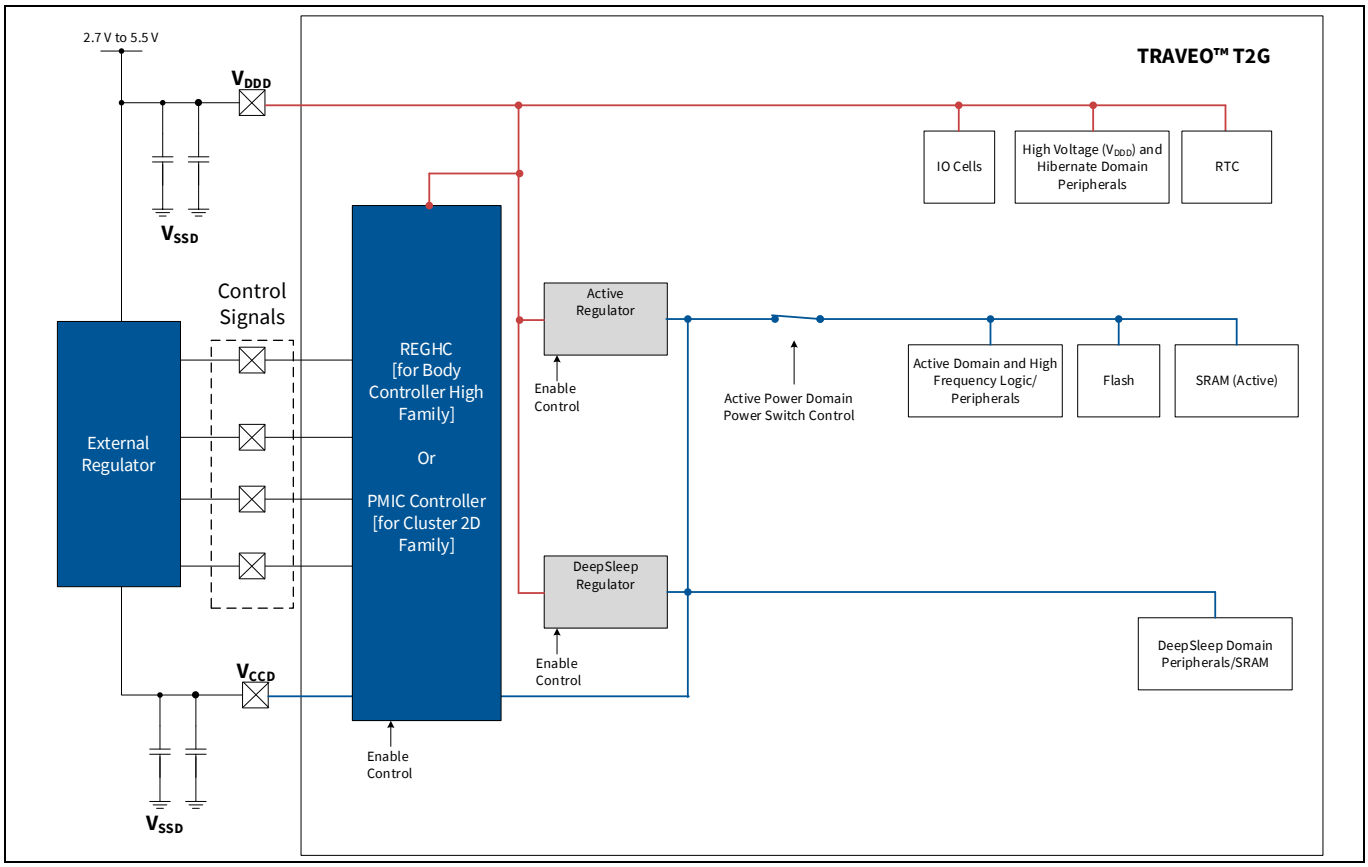

Figure 1 TRAVEO™ T2G power supply system

[1] See the datasheet for specified core supply voltage.

2.1.1 Internal regulators and, REGHC and PMIC controller

V_{CCD} is connected to the core power supply. But when the load current of the Active Regulator exceeds 300 mA, V_{CCD} must be supplied from outside via REGHC or PMIC controller. Note that in Hibernate mode, all regulators are OFF and V_{CCD} is not driven. Hibernate circuit operates from V_{DDD} directly. For details, see the Device Power Modes chapter in the architecture TRM [2]. The device starts with the Active Regulator before the handover to external power supply.

Active regulator:

This is a linear low-dropout (LDO) regulator to power the MCU in Active and Sleep modes. This regulator generates the core voltage (V_{CCD}) from V_{DDD} during Active and Sleep modes. It supports load currents up to 300 mA and is operational during device boot with power ON and High Voltage (HV) resets (XRES, POR, BOD, OVD, OCD, HIBERNATE wakeup) or handover to internal regulator from an external power supply. Active Regulator may support operation of the MCU at low frequencies if load current is lower than 300 mA. This regulator can be enabled or disabled by a register setting when PMIC is enabled. When the Active Regulator is enabled, you can use the Over Current Detection (OCD) feature in the Active Regulator.

DeepSleep regulator:

This is a linear LDO regulator to power the core voltage (V_{CCD}) during DeepSleep mode. When compared to Active Regulator, this regulator has lower drive capability (up to 18 mA) and lower current consumption. A power source in DeepSleep power mode can be selected in the DeepSleep Regulator or the PMIC. PMIC can be enabled in DeepSleep power mode.

REGHC or PMIC controller:

MCU supports higher load currents using REGHC or PMIC controller and external components. REGHC or PMIC controller controls handover between external power supply and internal regulators. Configuration and control pins of REGHC or PMIC controller are initialized only by HV resets (XRES, POR, BOD, OVD, OCD, HIBERNATE wakeup), and not by any Low Voltage (LV) reset. Note that when standard GPIO pins are used for external power control, they may be initialized by an LV reset and need to be reconfigured. For more information, see the architecture TRM [2].

2.2 Difference between CYT3B/4B and CYT3D/4D series

This section shows the differences between the CYT3B/4B (Body Controller High) series and the CYT3D/4D (Cluster 2D) series.

2.2.1 Difference in external power supply configuration

As mentioned **above**, for controlling external power supply, CYT3B/4B series have REGHC, and CYT3D/4D series have PMIC controller. **Table 1** shows the differences between CYT3B/4B series and CYT3D/4D series.

Table 1 Difference external power supply configuration

Item	CYT3B/4B series	CYT3D/4D series
Controller	REGHC	PMIC controller
Supported External Power Structure	**Pass transistor** **PMIC direct connection** **PMIC with load switch**	**PMIC direct connection**
Control pins	See **Table 3**	See **Table 4**

2.2.2 Difference in register name

Register names differ between the CYT3B/4B and CYT3D/4D series. **Table 2** compares the registers for each series. This document uses the CYT3B/4B series name. When using the CYT3D/4D series, read them according to **Table 2**. See the registers TRM **[3]** for register details.

Table 2 Difference in registers

CYT3B/4B series		CYT3D/4D series	
Register name	**Bit field name**	**Register name**	**Bit field name**
PWR_CTL2	DPSLP_REG_DIS	PWR_CTL2	DPSLP_REG_DIS
PWR_REGHC_CTL	REGHC_CONFIGURED	PWR_PMIC_CTL	PMIC_CONFIGURED
	REGHC_PMIC_STATUS_WAIT		PMIC_STATUS_WAIT
	REGHC_PMIC_STATUS_ POLARITY		PMIC_STATUS_POLARITY
	REGHC_PMIC_STATUS_INEN		PMIC_STATUS_INEN
	REGHC_PMIC_CTL_POLARITY		PMIC_CTL_POLARITY
	REGHC_PMIC_CTL_OUTEN		PMIC_CTL_OUTEN
	REGHC_PMIC_RADJ		Not support
	REGHC_PMIC_USE_RADJ		Not support
	REGHC_PMIC_USE_LINREG		PMIC_USE_LINREG
	REGHC_VADJ		PMIC_VADJ
	REGHC_PMIC_DRV_VOUT		PMIC_VREF
	REGHC_MODE		Not support
PWR_REGHC_STATUS	REGHC_SEQ_BUSY	PWR_PMIC_STATUS	PMIC_SEQ_BUSY
	REGHC_PMIC_STATUS_OK		PMIC_STATUS_OK
	REGHC_ENABLED		PMIC_ENABLED
PWR_REGHC_CTL2	REGHC_EN	PWR_PMIC_CTL2	PMIC_EN
PWR_REGHC_CTL4	REGHC_PMIC_DPSLP	PWR_PMIC_CTL4	PMIC_DPSLP
	REGHC_PMIC_VADJ_DIS		PMIC_VADJ_DIS

2.3 Handover between external power supply and internal regulator

This section provides an overview of the handover between external power supply and internal regulator. For more details, see **Pass transistor** and **PMIC (switching regulator)**.

REGHC or PMIC controller is initially disabled and must be enabled by the API. Active Regulator supports the core current until REGHC or PMIC controller is configured, enabled, operational, and ready. The core current must not exceed the limit of the Active Regulator until the handover completes. See **System Call API** for API details.

REGHC has two configurations; one for a pass transistor and another for a PMIC to drive higher load currents. PMIC controller has configuration for PMIC only.

Table 3 lists the REGHC pins for each configuration. Table 4 lists the PMIC controller pins.

Table 3 REGHC pins

Pin Name	Pass transistor		PMIC	
	Direction	Description	Direction	Description
DRV_VOUT	OUT	Base of the pass transistor	Unused	-
EXT_PS_CTL0	IN	Positive terminal of the current sense resistor	IN	Reset out (RO) or Power good (PG) signal input from PMIC
EXT_PS_CTL1	IN	Negative terminal of the current sense resistor	OUT	Enable output for PMIC
EXT_PS_CTL2	Unused	-	OUT (optional)	Reset threshold adjustment for some PMIC

Table 4 PMIC controller pins

Pin Name	PMIC		
	Direction	Description	Correspondence with REGHC
PMIC_STATUS	IN	Reset out (RO) or Power good (PG) signal input from PMIC	EXT_PS_CTL0
PMIC_EN	OUT	Enable output for PMIC	EXT_PS_CTL1

Note: CYT3D/4D series do not have DRV_VOUT and EXT_PS_CTL2 pins.

The handover between the external power supply and internal regulator takes place when:

- User software switches between the internal regulator and the external power supply with dedicated APIs and register.
- REGHC or PMIC controller is disabled by hardware to transition to OFF and XRES states and HIBERNATE. When reset is released, the MCU starts to operate with the Active Regulator. Software enables REGHC or PMIC controller after the device reboots. For OFF and XRES states, see the Device Power Modes chapter in the architecture TRM [2].

2.3.1 Internal regulator configuration using PMIC

For REGHC or PMIC controller to configure PMIC, enable or disable the Active and DeepSleep Regulators.

When PMIC is enabled with the Active Regulator enabled in parallel, you can use the OCD feature of the Active Regulator while supplying power from PMIC. OCD is an integrated part of the Active Regulator. So, the OCD feature is enabled only when the Active Regulator is enabled. In power save modes such as DeepSleep mode, the Active regulator is OFF, therefore the OCD is also OFF. MCU can detect V_{CCD} brown-out, such as the PMIC dropping out of regulation range or being unable to provide a fast load current increase. This is effective when the PMIC does not have an OCD feature. This feature is disabled when the Active Regulator is disabled.

TRAVEO™ T2G power supply system

The DeepSleep Regulator supplies the core supply only in DeepSleep power mode. You can use the register setting to decide whether the PMIC or the DeepSleep Regulator supplies the MCU in DeepSleep power mode.

Therefore, you can select the configurations, listed in **Table 5**, depending on the system.

Table 5 Internal regulator configuration for PMIC case

Use case	Configuration	Active regulator (including integrated OCD)	DeepSleep regulator (including integrated OCD)
Use PMIC without own OCD feature. PMIC is disabled in DeepSleep power mode.	OCD is enabled when supplying power from PMIC. DeepSleep Regulator supplies power in DeepSleep power mode	Enabled	Enabled
Use PMIC with own OCD feature. PMIC is disabled in DeepSleep power mode.	OCD is disabled when supplying power from PMIC. DeepSleep Regulator supplies power in DeepSleep power mode	Disabled	Enabled
Use PMIC with own OCD feature. PMIC is enabled in DeepSleep power mode.	OCD is disabled when supplying power from PMIC. PMIC supplies power in DeepSleep power mode	Disabled	Enabled/Disabled

Operation examples are explained in **PMIC direct connection**.

Note: The use of OCD for over current monitoring of the external power supply is not recommended, as the monitoring is not a direct shunt implementation of the V_{CCD} power rail. It is an integral part of the Active Regulator.

Note: When you set the DeepSleep Regulator to "Disabled" (PWR_CTL2.DPSLP_REG_DIS = "1"), you cannot switch back from the PMIC to the internal regulator. If your system is required to switch back to the internal regulator again after handover to PMIC, DeepSleep Regulator should be enabled. PWR_CTL2.DPSLP_REG_DIS can be set by the `SwitchOverRegulators` API with blocking call or application software.

2.3.2 System Call API

As mentioned **above**, when a handover is performed between the external power supply and internal regulators, a specific System Call API and register are used.

TRAVEO™ T2G device has a supervisory ROM that contains Boot ROM code and SROM APIs. The SROM APIs are designed to be used with Arm® Cortex®-M0+ (CM0+). The SROM APIs are used for specific operations, such as flash programming and lifecycle change. The APIs are also used for the handover between the external power supply and internal regulators. The APIs control REGHC or PMIC controller appropriately by setting the register and handling according to the API parameters, and execute the handover. For details on the APIs, see the Nonvolatile Memory Programming chapter in the architecture TRM **[2]**.

Note: *It is recommended using system calls from CM0+ for performing handover, instead of writing directly to registers.*

Following are the APIs used for the handover:

ConfigureRegulator API:

This API is called initially to configure the only desired regulator, before the handover between the external power supply and internal regulator by the `SwitchOverRegulators` API. This API is required if PWR_REGHC_CTL.REGHC_CONFIGURED = 0. **Table 6** lists the API parameters.

Table 6 Parameters of ConfigureRegulator API

Register/ SRAM_SCRATCH	Name	Bits	Description
IPC_STRUCT.DATA0	SRAM_SCRATCH_ ADDR1	[31:0]	Address of SRAM_SCRATCH1 where the API parameters are stored. Must be a 32-bit aligned address.
IPC_STRUCT.DATA1	SRAM_SCRATCH_ ADDR2	[31:0]	Address of SRAM_SCRATCH2 where the API parameters are stored. Must be a 32-bit aligned address.
SRAM_SCRATCH1	Opcode	[31:24]	Opcode for the `ConfigureRegulator` API 0x15
	RadjValue	[15:13]	This field is used to configure the reset voltage adjustment for PMIC. For details, see PWR_REGHC_CTL.REGHC_PMIC_RADJ in the registers TRM [3]. This field is invalid in CYT3D/4D series.
	VADJ trim value	[12:8]	This field is used to provide the adjust value to obtain the desired Voltage output from REGHC. When operating mode (Bits[1]) is external transistor, this field set to 16 (0x10). For details, see PWR_REGHC_CTL.REGHC_VADJ in the registers TRM [3].
	Vadj	[7]	This bit configures to REGHC_PMIC_VADJ_DIS. This bit should be set to "1" in PMIC configuration because DRV_VOUT pin is not used in TRAVEO™ T2G MCU. For details, see PWR_REGHC_CTL4. REGHC_PMIC_VADJ_DIS in the registers TRM [3].
	UseRadj	[6]	This bit configures the use of reset voltage adjustment for PMIC. 0: No use RADJ. 1: Use RADJ For details, see PWR_REGHC_CTL. REGHC_PMIC_USE_RADJ in the registers TRM [3].

Register/ SRAM_SCRATCH	Name	Bits	Description
			This bit is invalid in CYT3D/4D series.
	UseLinReg	[5]	This bit keeps the Active Regulator enabled in the external PMIC mode. This bit field is valid when operating mode is '1' (External PMIC mode). 0: Internal Active Regulator is disabled after PMIC enabled. 1: Internal Active Regulator kept enabled after PMIC enabled. For details, see PWR_REGHC_CTL.REGHC_PMIC_USE_LINREG in the registers TRM [3]. When the `SwitchOverRegulators` API is called with blocking call, PWR_CTL2.DPSLP_REG_DIS is set by the API according to this bit configuration.
	DeepSleep	[4]	This bit configures PMIC behavior during DeepSleep. This field is valid when operating mode is '1' (External PMIC mode). 0: PMIC is disabled in DeepSleep power mode. 1: PMIC is enabled in DeepSleep power mode. (DeepSleep Regulator is disabled in DeepSleep power mode) For details, see PWR_REGHC_CTL4.REGHC_PMIC_DPSLP in registers TRM [3]. When the `SwitchOverRegulators` API is called with blocking call, PWR_CTL2.DPSLP_REG_DIS is set by the API according to this bit configuration. This bit is invalid in CYT3D/4D series.
	Reset Polarity	[3]	This sets PMIC error condition (PMIC Reset out (RO) asserted level or PMIC PG signal de-asserted level). It is used to trigger a reset action based on the RO or PG signal input. If reset occurs by this factor, it is indicated in the RES_CAUSE.RESET_PMIC. This field is valid when operating mode is set to 1 (External PMIC mode). 0: Trigger reset when RO or PG signal input is "0". (RO or PG signal indicates power good status with "1") 1: Trigger reset when RO or PG signal input is "1" (RO or PG signal indicates power good status with "0") For more details, see PWR_REGHC_CTL.REGHC_PMIC_STATUS_POLARITY

Register/ SRAM_SCRATCH	Name	Bits	Description
			in the registers TRM [3]. Also, this API sets the PWR_REGHC_CTL.REGHC_PMIC_STATUS_INEN bit.
	Enable Polarity	[2]	The polarity is used to enable the PMIC. REGHC uses REGHC_PMIC_CTL_POLARITY to enable the PMIC, and it uses the complement to disable the PMIC. This field is valid when operating mode set to 1. (External PMIC mode) 0: PMIC is enabled by “0”. 1: PMIC is enabled by “1”. For more details, see PWR_REGHC_CTL.REGHC_PMIC_CTL_POLARITY in the registers TRM [3]. Also, this API sets the PWR_REGHC_CTL.REGHC_PMIC_CTL_OUTEN bit.
	Operating Mode	[1]	This bit defines the operating mode (pass transistor or PMIC). 0: External transistor 1: External PMIC For more details, see PWR_REGHC_CTL.REGHC_MODE in registers TRM [3]. This bit must match the Operating mode of the `SwitchOverRegulators` API. For devices without REGHC, the operating mode is ignored.
	Others	-	Not used
SRAM_SCRATCH2	WaitCount	[8:0]	Wait count in steps of 4 µs after PMIC status is OK. This is used by the hardware sequencer to allow additional settling time before disabling the internal regulator. For more details, see PWR_REGHC_CTL. REGHC_PMIC_STATUS_WAIT in registers TRM [3]. Note that `ConfigureRegulator` API supports Wait Count field [8:0], it does not support Wait Count bit [9]. Therefore, if you want the Wait Count to be greater than 0x1FF, user software must set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT[29:20] register after the `ConfigureRegulator` API is successful.

When the WaitCount needs to be greater than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register directly using the application software. **Figure 2** shows the flow when REGHC_PMIC_STATUS_WAIT is set.

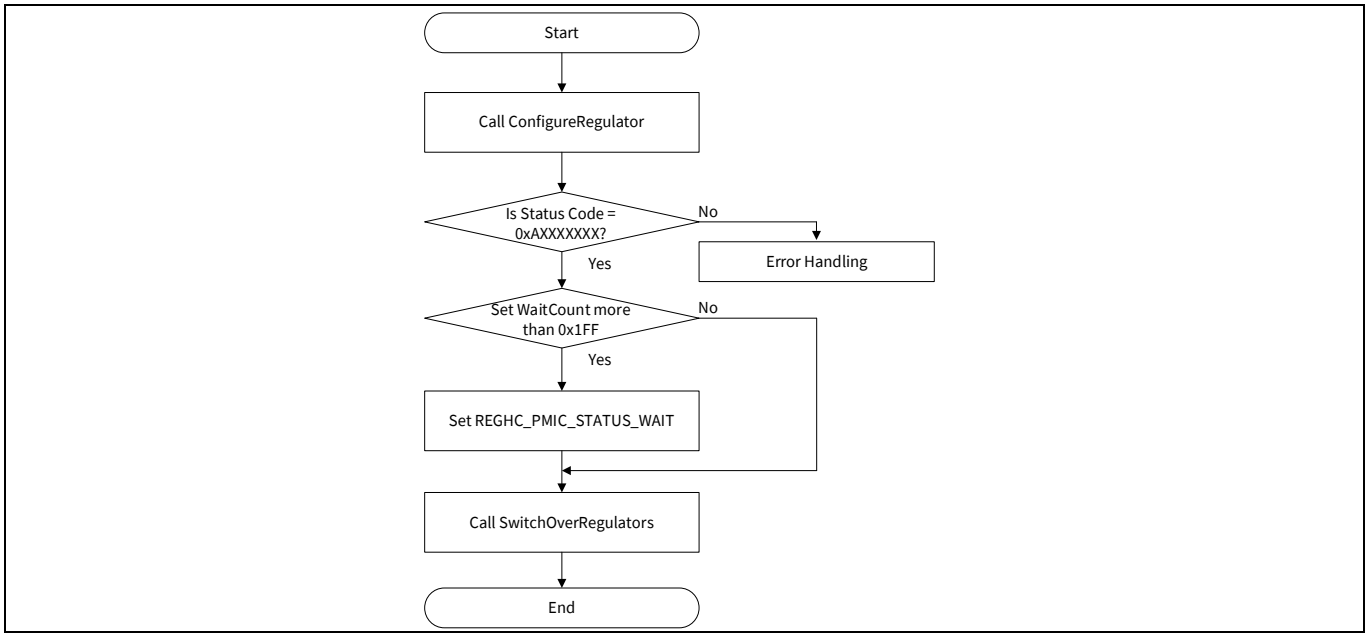

Figure 2 REGHC_PMIC_STATUS_WAIT setting flow

This register must be set after the completion of the `ConfigureRegulator` API, and before calling the `SwitchOverRegulators` API.

Note: When the `ConfigureRegulator` API is called, IPC_STRUCT.DATA1 where the SRAM_SCRATCH_ADDR2 is stored should be set a valid user RAM area, even if the application software sets the WaitCount. If you set an invalid address, a bus error may occur.

SwitchOverRegulators API:

This API is used to select between the external power supply and internal regulator for the core voltage supply V_{CCD} in Active power mode.

When an external supply is selected, this API needs to be called only once and the REGHC or PMIC controller controls the switching between the internal DeepSleep Regulator and the external regulator automatically. This API needs to be called before the activation of the application core(s), because the current consumption might be higher than the internal provisioning of the Active Regulator. This API must be called after the `ConfigureRegulator` API.

Table 7 shows this API parameters.

Table 7 Parameters of the SwitchOverRegulators API

Register/ SRAM_SCRATCH	Name	Bits	Description
IPC_STRUCT.DATA	SRAM_SCRATCH_ ADDR	[31:0]	Address of SRAM where the API parameters are stored. Must be a 32-bit aligned address.
SRAM_SCRATCH	Opcode	[31:24]	Opcode for the `SwitchOverRegulators` API 0x11
	Blocking	[23:16]	This field defines whether this API will be blocking CM0+ core or non-blocking. 0: Non-blocking call

Register/ SRAM_SCRATCH	Name	Bits	Description
			1: Blocking call Blocking calls will come out of the syscall only after the complete handover has happened. Non-blocking calls will return the API status even before waiting for the actual handover to happen. That is, if non-blocking calls are used, user software needs to wait for switching to complete. See the **SwitchOverRegulators API non-blocking call handling** for details When this API is called in the blocking call, internal regulators will be changed to the proper state for handover to external power supply. Note that if the API is called in non-blocking call, the stage will not change. In addition, set PWR_CTL2.DPSLP_REG_DIS to "1" when this API is called in the blocking call and PMIC is enabled in DeepSleep power mode (UseLinReg = "0" and DeepSleep = "1" of `ConfigureRegulator` API).
	Select regulator	[15:8]	This field is used to define handover to external power supply from Active Regulator or to Active Regulator from external power supply. 0: Switch over to external power supply 1: Switch over to Active Regulator
	Operating Mode	[1]	This bit defines the operating mode (pass transistor or PMIC). 0: External transistor 1: External PMIC For more details, see PWR_REGHC_CTL.REGHC_MODE in the registers TRM **[3]**. This bit must match the Operating mode of the `ConfigureRegulator` API. For devices without REGHC, the operating mode is ignored.
	Others	-	Not used

LoadRegulatorTrims API:

This API is used to adapt the output voltage to transition to DeepSleep power mode and for the internal regulators during the handover. **Table 8** and **Table 9** list the parameters of this API. This API must be called every time a load transition requires switching between external and internal regulators, except when using the `SwitchOverRegulators` API with a blocking call.

Table 8 Parameters of the LoadRegulatorTrims API

Register/SRAM_SCRATCH	Name	Bits	Description
IPC_STRUCT.DATA	SRAM_SCRATCH_ADDR	[31:0]	Address of SRAM where the API parameters are stored. Must be a 32-bit aligned address.
SRAM_SCRATCH	Opcode	[31:24]	Opcode for the `LoadRegulatorTrims` API

Register/SRAM_SCRATCH	Name	Bits	Description
			0x16
	Use case	[3:2]	This field defines the case in which the `LoadRegulatorTrims` API is being called. 0: Force trim setting. The syscall will ignore regulator config, as it will set requested trims 1: DeepSleep Entry use case 2: DeepSleep Exit use case 3: Reset Recovery use case See **Table 9** for use case description.
	Operating Mode	[1]	This bit specifies output voltage for the internal regulators. 0: Internal regulators 1: External power supply This field is applicable only when the use case is “0”.
	Others	-	Not used

Table 9 Use case of the LoadRegulatorTrims API

Use case	Description
Force trim setting	This use case requires that you bypass the regulator configuration and set the required output voltage. See **Table 10** for output states.
DeepSleep Entry	In this use case, you need to change the DeepSleep Regulator output voltage before the transition to DeepSleep power mode. However, this use case is not valid, when the DeepSleep Regulator is configured as OFF in DeepSleep power mode.
DeepSleep Exit	This use case describes the system wake up from DeepSleep power mode. DeepSleep Entry must be executed to enter DeepSleep power mode.
Reset Recovery	If a reset interrupt occurs (LV reset) during a handover, it will not affect the power rail transition, but the application is reset. In this situation, you need to use this use case after reset release to verify that the output of the internal regulator complies with the requirements.

Note: The system can transition to Hibernate power mode directly without any prior action.

The `SwitchOverRegulators` and `LoadRegulatorTrims` APIs change the output state of the internal regulators, as listed **Table 10**, according to the handover transition case. The output voltage of the internal regulators is applied according to the output status.

Table 10 Output state of internal regulators after core power supply transition

Handover transition case	Internal regulator output state	Description
Internal Regulators to External Power Supply	External State	Core is supplied by PMIC. PMIC and internal regulators are ON.
	OFF	Core is supplied by PMIC. PMIC is ON and internal regulators are OFF.

Handover transition case	Internal regulator output state	Description
External Power Supply to Internal Regulators	Internal State	Core is supplied by internal regulators. PMIC is OFF and internal regulator are ON.

A status code is returned to the SRAM_SCRATCH address, after the API handling is complete. **Table 11** lists the status codes.

Table 11 Status codes

Status code	Description	Handling example
0xAXXXXXXX	API completed successfully. X: don't care	-
0xF00000E0	Regulator is configured for "Manual" mode	Call the `ConfigureRegulator` API.
0xF00000E1	Regulator is currently in transition. - Wait for completion.	Wait for API completion.
0xF00000E2	Regulator is already enabled.	Additional action is not required.
0xF00000E3	Regulator is not configured with `ConfigureRegulator`.	Perform the `ConfigureRegulator` API.
0xF00000E4	Returned by the `SwitchOverRegulators` API if syscall is called with a different OpMode parameter other than the `ConfigureRegulator` API.	Call API with correct combination.

If these unintended errors reoccur, perform appropriate error handling, such as resetting the entire chip, according to the system design.

2.3.3 Handover sequence overview

This section describes the handover sequence example using APIs and the timing behavior of the REGHC operation.

Figure 3 shows an overview of the handover sequence between the internal regulator and an external power supply. This sequence is used by the `LoadRegulatorTrims`, `ConfigureRegulator`, and `SwitchOverRegulators` APIs with blocking call.

TRAVEO™ T2G power supply system

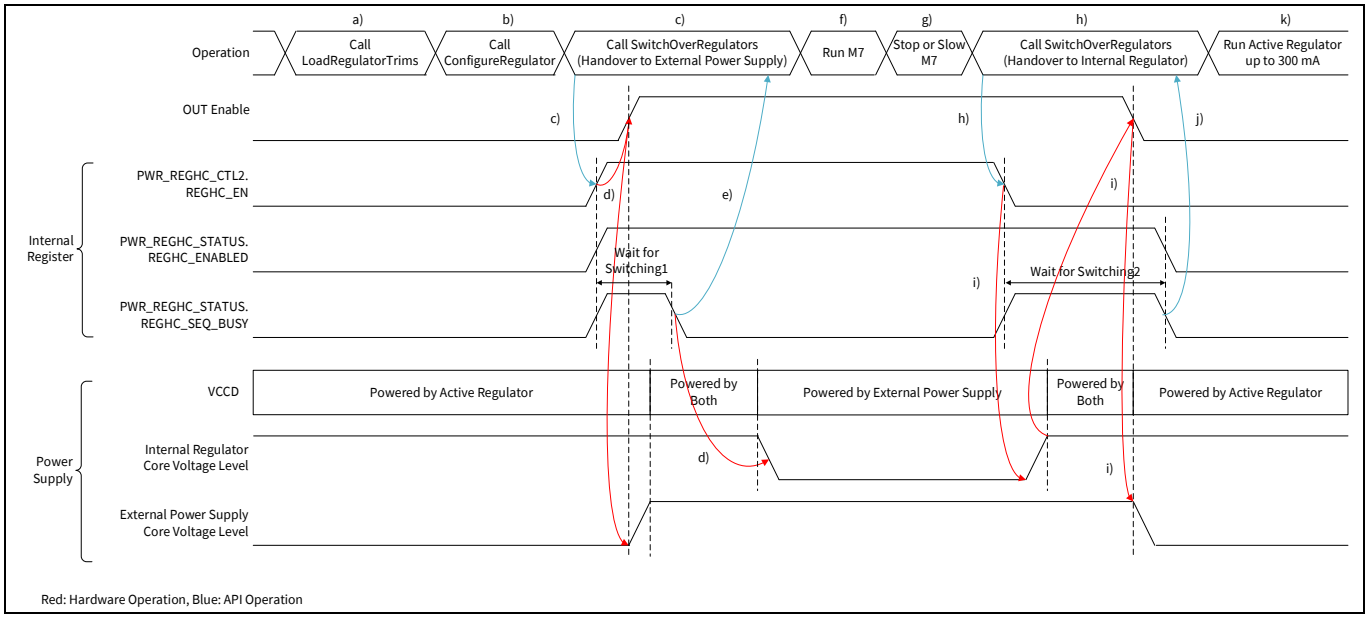

Figure 3 Handover sequence between internal regulator and external power supply

During start up, MCU starts with the internal regulator:

Handover from the internal regulator to an external power supply:

a) SW: Call the `LoadRegulatorTrims` API with reset recovery use case if PWR_REGHC_CTL.REGHC_CONFIGURED is set to “1”. See **Reset recovery** for details.

b) SW: Call the `ConfigureRegulator` API for REGHC with pass transistor or PMIC configuration. Select the operating mode according to the external power supply.

Note: For devices without REGHC, the operating mode is ignored.

c) SW: Call the `SwitchOverRegulators` API. Perform handover. Select the same operating mode that was selected in the `ConfigureRegulator` API. Then, set PWR_REGHC_CTL2.REGHC_EN to ‘1’.

d) HW: The external power supply is enabled, and internal regulators are disabled or set to external state according to the `ConfigureRegulator` API.

e) SW: The `SwitchOverRegulators` API returns the status code. If called with blocking call, the API waits for the handover to complete.

“Wait for switching1” corresponds to the wait time for an external power supply to be fully ready. In pass transistor configuration, wait time completion occurs within 15 us. In PMIC configuration, wait time completion depends on the power good status signal and the value of PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT.

Software can program REGHC_PMIC_STATUS_WAIT. So, the longer the counter, the longer the hardware sequencer keeps the internal Active Regulator enabled. This function can wait for a while before launching PMIC. For more details, see the registers TRM **[3]**.

f) Now, the MCU can run high load cases with the external power supply after successful completion of the `SwitchOverRegulators` API.

Note: The operating clock frequency should be increased step-by-step by the application software to avoid V_{CCD} voltage undershoot.

Handover from external power supply to internal regulator:

g) The MCU must run with clock settings that correspond to the current consumption limits, which are within the specification of the internal regulators before starting the handover.

h) SW: Call the `SwitchOverRegulators` API. Select the same operating mode that was selected in the `ConfigureRegulator` API. Then, set PWR_REGHC_CTL2.REGHC_EN to '0'.

i) HW: Internal regulators are enabled by setting PWR_REGHC_CTL2.REGHC_EN to '0'. When internal regulator starts suppling power, the OUT Enable is de-asserted and the external power supply is disabled.

j) SW: The `SwitchOverRegulators` API returns the status code. If called with Blocking call, the API waits for the handover to complete.

"Wait for switching2" corresponds to the waiting time until the internal regulator is ready. In pass transistor configuration, wait time completion occurs within 10 us. In PMIC configuration, the wait time completion depends on the power good status.

MCU runs with internal regulator after the `SwitchOverRegulators` API indicates a successful completion.

Note: If the transition does not complete within the time specified by "Wait for switching1" and "Wait for switching2" or unintended API error reoccurs, it is recommended to reset the entire chip, including the power system, by using WDT or requesting an external system controller trigger XRES_L.

2.3.4 SwitchOverRegulators API non-blocking call handling

The `SwitchOverRegulators` API has two calling modes: Blocking call and Non-blocking call. When calling the `SwitchOverRegulators` API in blocking call mode, the API waits for the handover to complete and changes the internal regulator to an appropriate state. In addition, the API sets PWR_CTL2.DPSLP_REG_DIS to "1" when PMIC is enabled in DeepSleep power mode (UseLinReg = "0" and DeepSleep = "1" of the `ConfigureRegulator` API).

However, when the call to the `SwitchOverRegulators` API is applied in non-blocking call mode, the API returns the states even before handover operation starts. In addition, the output state of internal regulators will not be changed and PWR_CTL2.DPSLP_REG_DIS is not configured.

This section describes how to handle the `SwitchOverRegulators` API with non-blocking mode. **Table 12** lists the differences between blocking and non-blocking calls.

Table 12 Differences between blocking and non-blocking calls

Operation	Blocking call	Non-blocking call
Wait for Handover Completion	Handled by the `SwitchOverRegulators` API	Handled by Application Software
Internal Regulator Output State	Handled by the `SwitchOverRegulators` API	Need to run the `LoadRegulatorTrims` API
DeepSleep Regulator Configuration	Handled by the `SwitchOverRegulators` API When the `ConfigureRegulator` API sets UseLinReg = "0" and DeepSleep = "1" , the `SwitchOverRegulators` API sets PWR_CTL2.DPSLP_REG_DIS to "1"	Handled by Application Software When the `ConfigureRegulator` API sets UseLinReg to "0" and DeepSleep to "1", the `SwitchOverRegulators` API does not set PWR_CTL2.DPSLP_REG_DIS to "1".

2.3.4.1 Behavior of REGHC_SEQ_BUSY flag

This section describes how to check handover completion by user software. The status of automatic transition between the internal regulator and PMIC for the core voltage is indicated by the REGHC_SEQ_BUSY status flag.

Transition: Internal regulators to external power supply

When the PG signal is still de-asserted after enabling PMIC, the flag is set to "1". Then, when the PG signal is asserted, the flag is set to "0". The flag indicates that the transition sequence completed properly.

Note: *If the PG signal is asserted during **soft-start**, the internal regulator may be turned OFF before PMIC becomes fully operational, and it may cause the V_{CCD} voltage to undershoot. You can configure a delay time to mitigate this problem. The flag is not set to "0" by the programmable delay even after the PG signal is asserted by the PMIC. This delay can be controlled by PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT.*

Figure 4 shows the REGHC_SEQ_BUSY operation when enable to PMIC is active HIGH and power good status from PMIC is active HIGH.

Transition: External power supply to internal regulators

When the PG signal is still asserted after disabling the PMIC, the flag is set to "1". Then, when the PG signal is de-asserted, the flag is set to "0". The flag indicates that the transition sequence completed properly (transition to internal regulators completed).

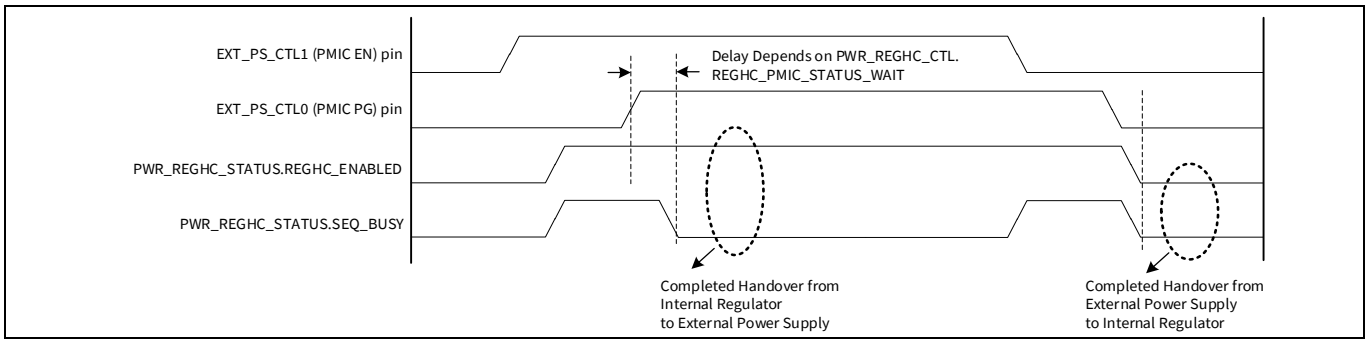

Figure 4 REGHC_SEQ_BUSY operation

- Completion states of handover from internal regulators to external power supply

  PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0, PWR_REGHC_STATUS.REGHC_ENABLED = 1.

- Completion states of handover from external power supply to internal regulators

  PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0, PWR_REGHC_STATUS.REGHC_ENABLED = 0.

2.3.4.2 DeepSleep regulator configuration

This section describes the differences in the configuration of internal regulations between blocking and non-blocking calls of the `SwitchOverRegulators` API and how to use each.

MCU can set the DeepSleep Regulator to "Disabled" (set PWR_CTL2.DPSLP_REG_DIS to "1") by disabling the Active Regulator and enabling PMIC in DeepSleep power mode.

The `SwitchOverRegulators` API will disable the DeepSleep Regulator when:

- The `ConfigureRegulator` API sets UseLinReg = "0" and DeepSleep = "1"
- The `SwitchOverRegulators` API is called with blocking call

When the `ConfigureRegulator` API is not configured to set UseLinReg to "0" and DeepSleep to "1" or when the `SwitchOverRegulators` API is called with non-blocking call, the DeepSleep regulator is not set to "Disabled".

As mentioned **above**, when DeepSleep Regulator is set to “Disabled” (set PWR_CTL2.DPSLP_REG_DIS to "1"), you cannot run handover from PMIC to the internal regulators using the `SwitchOverRegulators` API. If your system requires to run the handover from PMIC to internal regulators by the `SwitchOverRegulators` API, you need to call the `SwitchOverRegulators` API with non-blocking call when running the handover to PMIC.

2.3.4.3 Example of handover handling in non-blocking call

Figure 5 shows an example on how to run the `SwitchOverRegulators` API with non-blocking call for handover from internal regulators to PMIC.

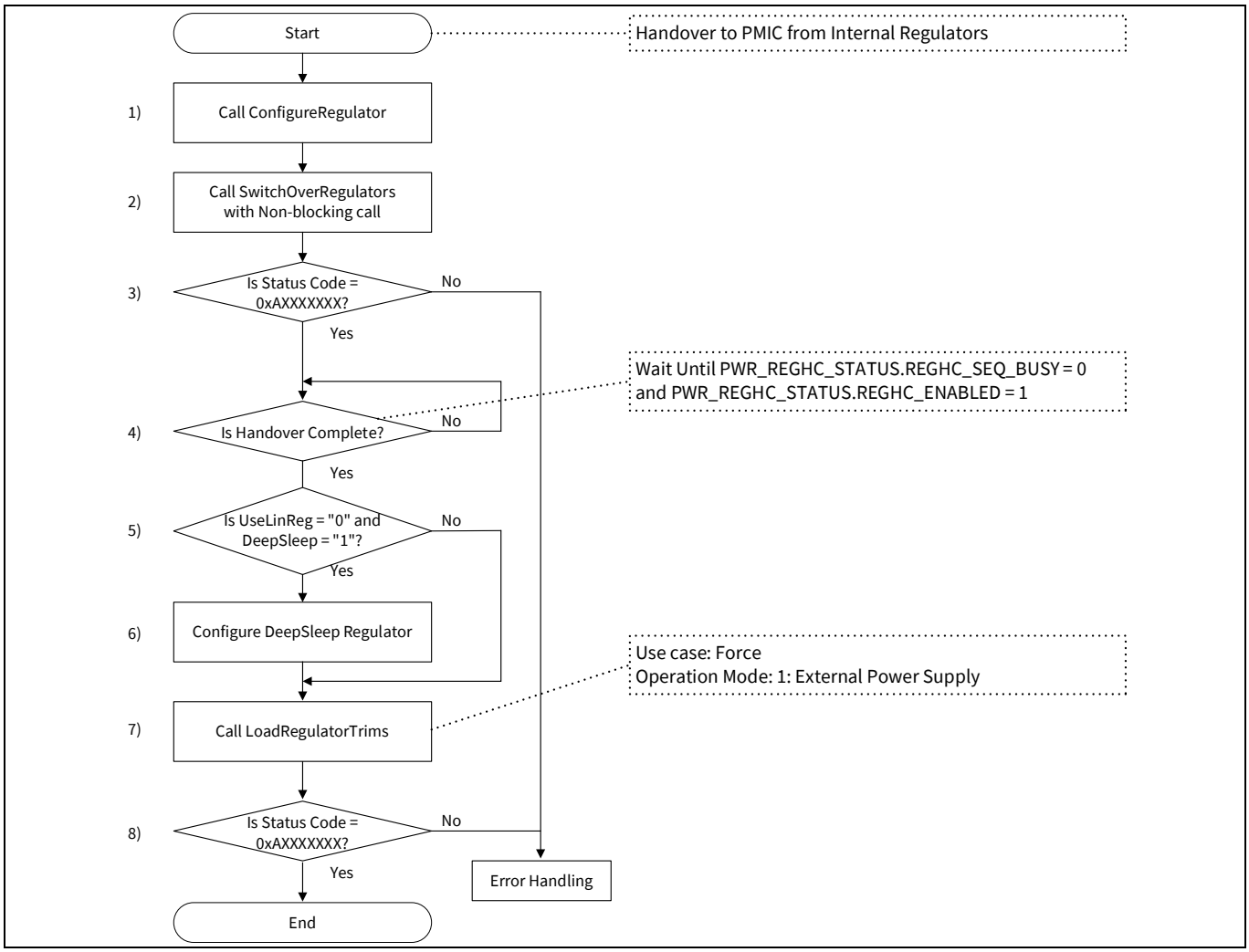

Figure 5 Example of handover from internal regulator to PMIC in non-blocking call

1. Call the `ConfigureRegulator` API for configuring internal regulators.
2. Call the `SwitchOverRegulators` API with non-blocking call.
3. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system. However, the handover is not complete
4. Wait for handover completion until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1
5. Check if PMIC is enabled in DeepSleep power mode.
6. Set PWR_CTL2.DPSLP_REG_DIS to "1". The DeepSleep Regulator is disabled. However, if PWR_CTL2.DPSLP_REG_DIS is set to "1", you cannot run handover from PMIC to the internal regulators by the `SwitchOverRegulators` API again. Therefore, when your system wants to switch back to internal regulators using the API, do not set PWR_CTL2.DPSLP_REG_DIS to "1".
7. Call the `LoadRegulatorTrims` API with Force trim setting use case and External power supply operating mode. The output of internal regulators is changed to external state.

   When the `SwitchOverRegulators` API with non-blocking mode is called, the internal regulators output state may not change to the proper state. In this case, the `LoadRegulatorTrims` API needs to be called to change the output state of internal regulators.

8. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system. However, the handover is not complete.

2.3.5 LoadRegulatorTrims handling for each use case

The `LoadRegulatorTrims` API is used to change the internal regulator output state, and has four types of use case settings. See the **LoadRegulatorTrims** for details.

Force trim setting

This use case will set the internal regulator output state, specified in the operating mode, ignoring the current regulator configuration. In many cases, the internal output state is properly controlled by the `SwitchOverRegulators` API with blocking call and by the `LoadRegulatorTrims` API for other use cases.

DeepSleep entry

This use case will set Internal regulators output state to internal state before the transition to DeepSleep power mode. Therefore, you need to execute the `LoadRegulatorTrims` API with DeepSleep Entry use case before transitioning to DeepSleep power mode. However, this use case is not required when the DeepSleep Regulator is configured as OFF in DeepSleep power mode.

DeepSleep exit

This use case will set the internal regulators output state to external state after the transition from DeepSleep power mode to Active power mode. You need to execute the `LoadRegulatorTrims` API with DeepSleep Exit use case after the system wakes up from DeepSleep power mode and the handover to PMIC is complete. This use case is required only if the `LoadRegulatorTrims` API with DeepSleep Entry use case was executed to the transition to DeepSleep power mode.

Reset recovery

This use case will set the appropriate internal regulator output state depending on the current REGHC or PMIC controller status after reset. REGHC or PMIC controller configuration is maintained after LV reset, but internal regulator output state is initialized. You need to use this use case after reset release, and apply the internal

regulator output state according to current situation. Note that some configuration such as GPIO settings are also initialized.

Figure 6 shows an example flow for the `LoadRegulatorTrims` API with Reset Recovery use case after reset using PMIC.

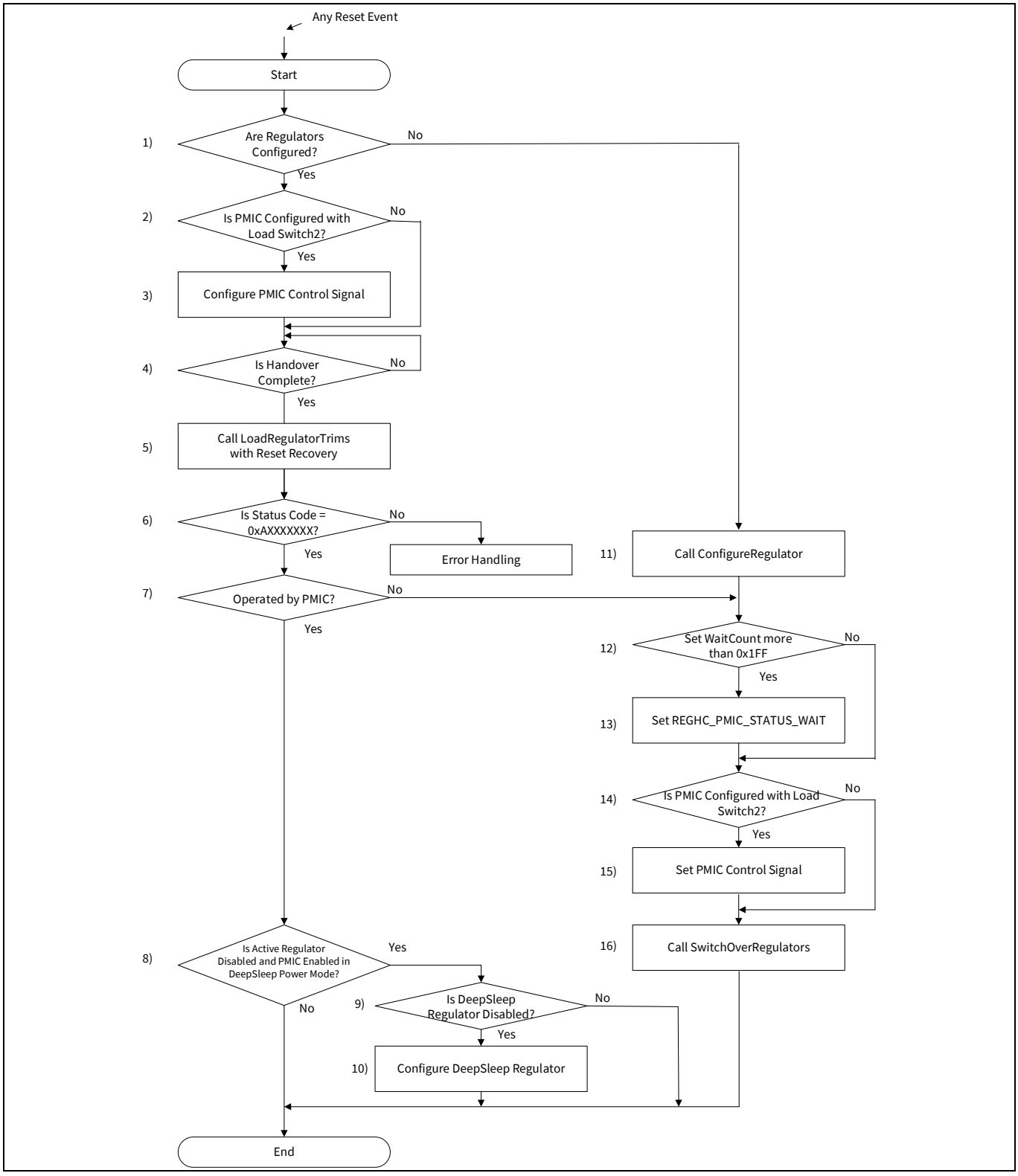

Figure 6 Example of the LoadRegulatorTrims API with reset recovery

TRAVEO™ T2G power supply system

1. Check if REGHC or PMIC controller configuration is already completed.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to “1”, the configuration is already complete and you do not need to call the `ConfigureRegulator` API. When PWR_REGHC_CTL.REGHC_CONFIGURED is set to “0”, the configuration is not complete.

   If REGHC configuration is not completed, go to (11).

2. Check if you have **PMIC with Load Switch2** configuration. If not, go to (4).
3. Configure GPIO for PMIC enable control. In this case, PMIC enable state before LV reset is reconfigured. The following is an example of GPIO reconfiguration:

   a) Write ~ (PWR_REGHC_CTL2.REGHC_EN ^ PWR_REGHC_CTL.REGHC_PMIC_CTL_POLARITY) to GPIO data register.

   b) Configure to GPIO output.

   You need to configure to GPIO output after write GPIO data register. Different procedures cause unintentional output to the PMIC.

   For a different configuration, this operation is not required.

4. Wait for the hardware to complete handover.

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = “0” and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN.

5. Call the `LoadRegulatorTrims` API with Reset Recovery use case.

   Internal regulator output state is set appropriately. If PWR_REGHC_STATUS.REGHC_ENABLED is set to “1”, internal regulator output state is set to external state, and if it set to “0”, internal regulator output state is set to internal state.

6. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
7. Check if MCU operates with PMIC. When PWR_REGHC_STATUS.REGHC_ENABLED is set to “1”, the PMIC is supplying power. When PWR_REGHC_STATUS.REGHC_ENABLED is set to “0”, the internal regulator is supplying power. Go to (12).
8. Check if Active Regulator is disabled and PMIC is configured to enable in DeepSleep power mode (PWR_REGHC_CTL.REGHC_PMIC_USE_LINREG =”0” and PWR_REGHC_CTL4.REGHC_PMIC_DPSLP = “1”). If not, go to End.
9. Check if the DeepSleep Regulator is configured to “Disabled”. If not, go to End.
10. Set PWR_CTL2.DPSLP_REG_DIS to “1”.
11. Call the `ConfigureRegulator` API for regulator configuration.
12. Check if the WaitCount needs to be configured to a value greater than 0x1FF. If not, go to (14).
13. Set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register using the application software. See the `ConfigureRegulator` API in **System Call API** for more details.
14. Check if you have **PMIC with Load Switch2** configuration. If not, go to (16).
15. Set GPIO for PMIC enable control.
16. Call the `SwitchOverRegulators` API for handover to PMIC.

2.4 Configuration of external power supply

REGHC has two external power supply configurations: pass transistor and PMIC.

Table 13 lists the features of external power supply configuration.

Table 13 Feature of external power supply configuration

Type	BOM cost	Conversion efficiency
Pass Transistor	Low	Low
PMIC	High	High

If high efficiency power supply is required, it is better to select a PMIC. However, when using PMIC, take the precautions discussed in **PMIC (switching regulator)**.

See **Table 1** for external power structures supported by each series.

3 Pass transistor

3.1 Hardware configuration

Figure 7 shows an example configuration using a pass transistor. It consists of a pass transistor, a smoothing capacitor at V_{CCD} and V_{DDD}, and a sense resistor. The MCU senses voltages at EXT_PS_CTL0 and EXT_PS_CTL1 inputs and controls the DRV_VOUT output. Only MCUs of CYT3B/4B series support this configuration.

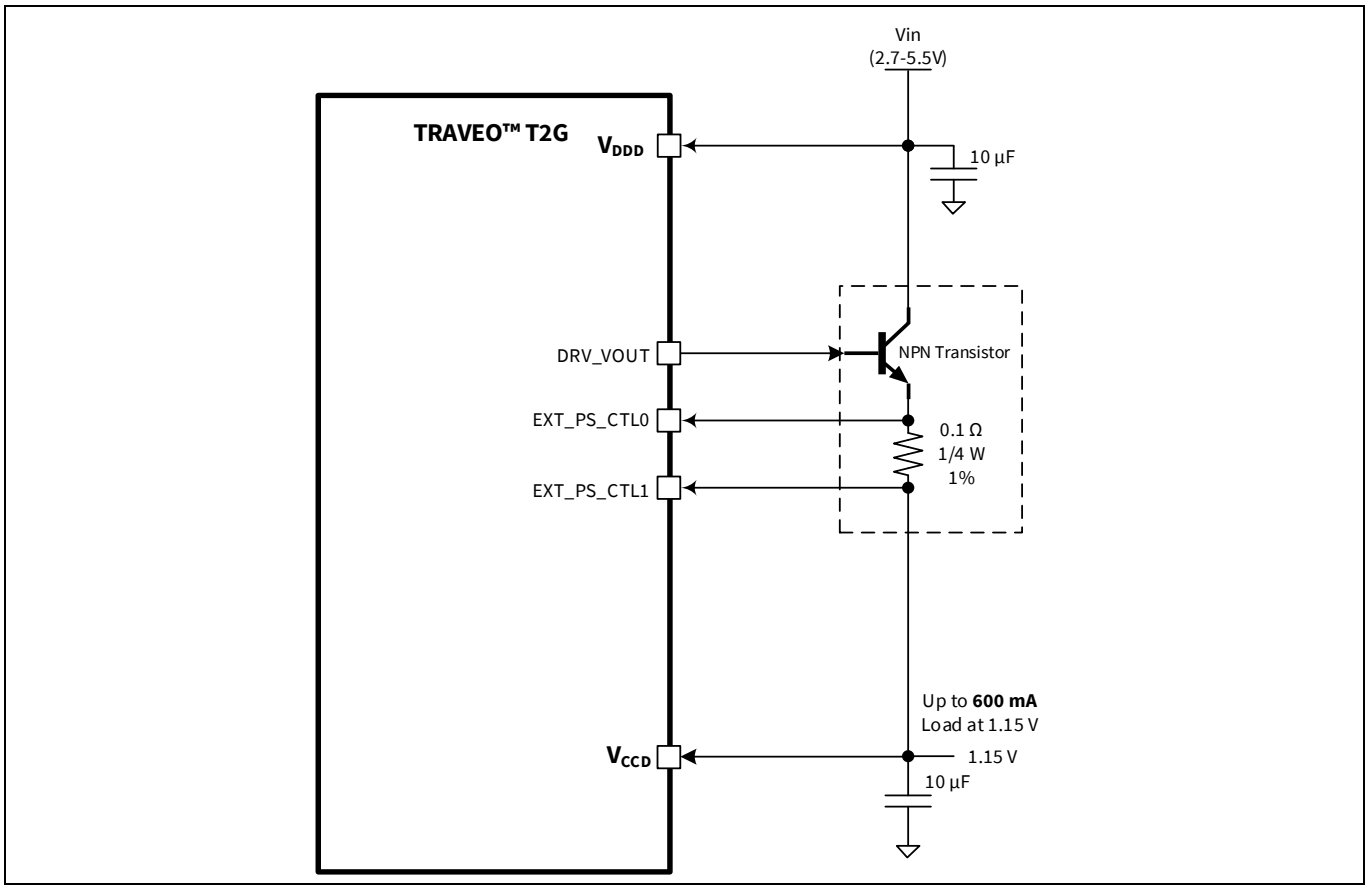

Figure 7 Pass transistor hardware configuration

3.2 Handover timing chart

Figure 8 shows the handover sequence example. This example includes regulator configuration, handover from internal regulators to the pass transistor, transition to DeepSleep power mode, wake up from DeepSleep power mode, and handover from the pass transistor to internal regulators.

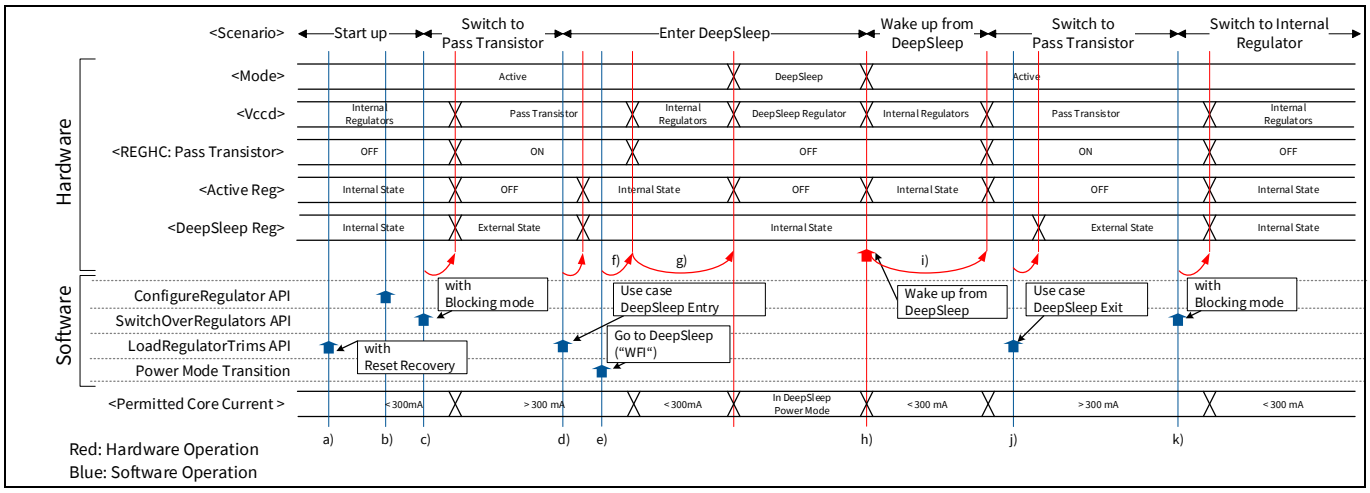

Figure 8 Handover sequence example

a) Call the `LoadRegulatorTrims` API with reset recovery use case, if PWR_REGHC_CTL.REGHC_CONFIGURED is set to "1". See the **reset recovery** for details.

b) Call the `ConfigureRegulator` API for REGHC with the pass transistor configuration, after startup.

c) Call the `SwitchOverRegulators` API with blocking call for handover from internal regulators to pass transistor. Pass transistor is enabled and Active Regulator is disabled. The DeepSleep Regulator changes to external state.

d) Call the `LoadRegulatorTrims` API with DeepSleep Entry use case for changing the regulator internal state.

   The Active and DeepSleep Regulators change to internal state.

e) Reduce the current consumption so that it is within the limit of Active Regulator, which is 300 mA. Then, you can execute the transition to DeepSleep power mode using the "WFI" instruction.

f) When the software executes the transition to DeepSleep power mode, the hardware switches from pass transistor to internal regulators.

g) MCU transitions to DeepSleep power mode after completing the switch to internal regulators.

h) A power mode transition from DeepSleep to Active happens after the wake-up event, and then the Active Regulator starts with internal state.

i) Pass transistor is enabled and the Active Regulator is turned OFF after the completion of Active power mode transition.

j) Call the `LoadRegulatorTrims` API with DeepSleep Exit use case for changing internal regulators to external state

   The DeepSleep Regulators change to external state.

k) Call the `SwitchOverRegulators` API for the handover from pass transistor to internal regulators.

   The pass transistor is disabled. The Active and DeepSleep Regulators change to internal state.

3.3 Software flow

3.3.1 Handover from internal regulator to pass transistor

Figure 9 shows the software flow of the handover from internal regulator to pass transistor.

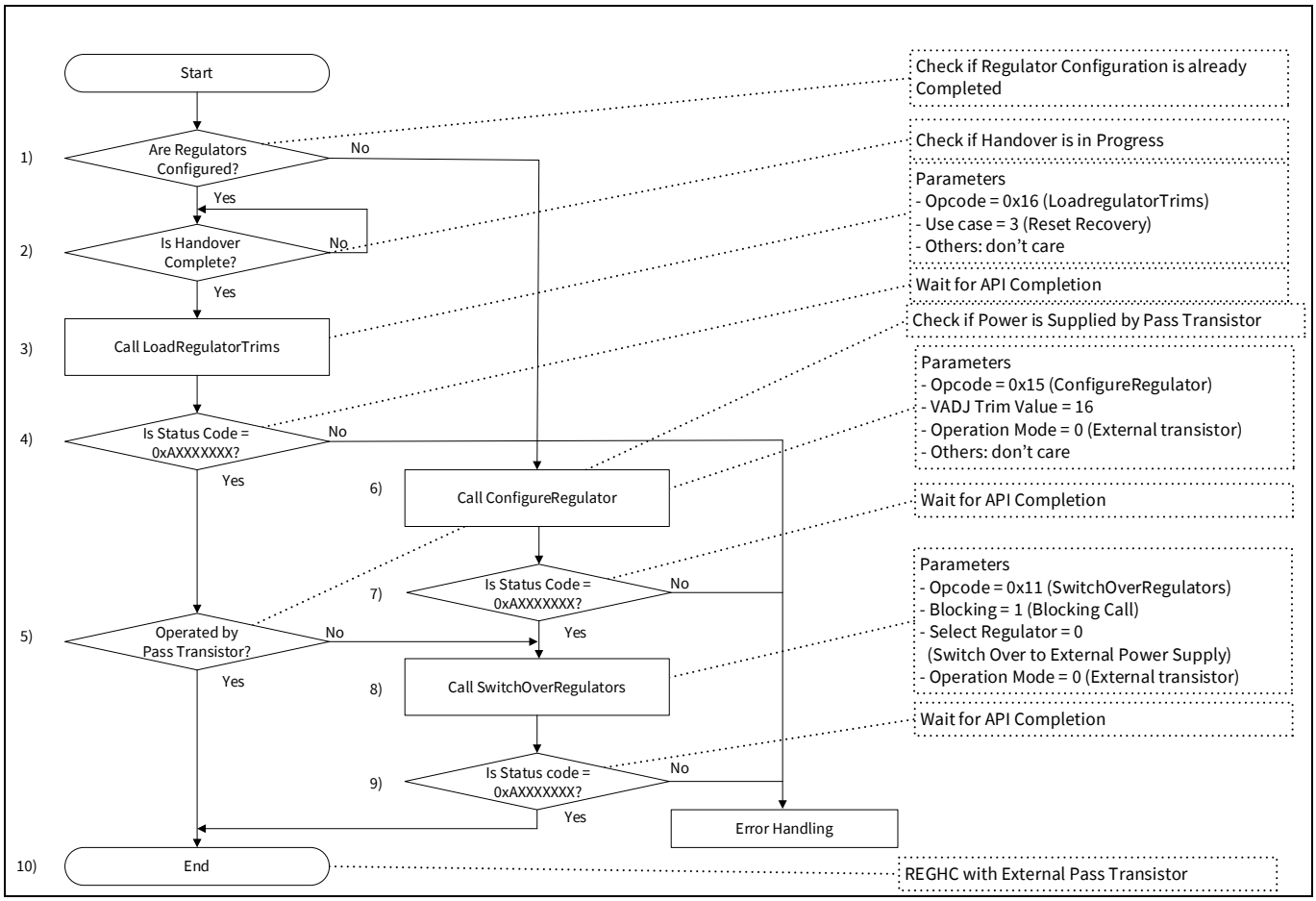

Figure 9 Flow for handover from internal regulator to pass transistor

1. Check if regulator configuration is already completed. Once REGHC is setup, it can be enabled without reconfiguring.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to “0”, the configuration is not complete. Go to (6).

2. Wait for the handover completion.

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = “0” and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN.

3. Call the `LoadRegulatorTrims` API with reset recovery:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 3 (Reset Recovery)
4. Wait for the completion of the `LoadRegulatorTrims` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.
5. Check if MCU operates with pass transistor. When PWR_REGHC_STATUS.REGHC_ENABLED is set to “1”, the pass transistor is supplying power. Go to (10). When PWR_REGHC_STATUS.REGHC_ENABLED is set to “0”, the internal regulator is supplying power. Go to (8).

Pass transistor

6. Call the `ConfigureRegulator` API with the following parameters:
   - Opcode = 0x15 (Call the `ConfigureRegulator` API)
   - VADJ trim value = 16 to output voltage at 1.15 V with adjustment. See the registers TRM **[3]** for more details.
   - Operating Mode = 0 (External transistor)
   - Others = Don't care

Note: *IPC_STRUCT.DATA1 where the SRAM_SCRATCH_ADDR2 is stored should be set a valid user RAM area. If you set an invalid address, a bus error may occur.*

7. Wait for the completion of the `ConfigureRegulator` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.

To enable the REGHC with Pass Transistor, perform these steps:

8. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 0 (Switch over external power supply)
   - Operating Mode = 0 (External transistor)
9. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform the appropriate error handling according to your system.
10. The device is now operating REGHC with pass transistor.

Once REGHC is setup, it can be enabled without reconfiguring.

3.3.2 DeepSleep entry and exit

Figure 10 shows the transition to DeepSleep power mode.

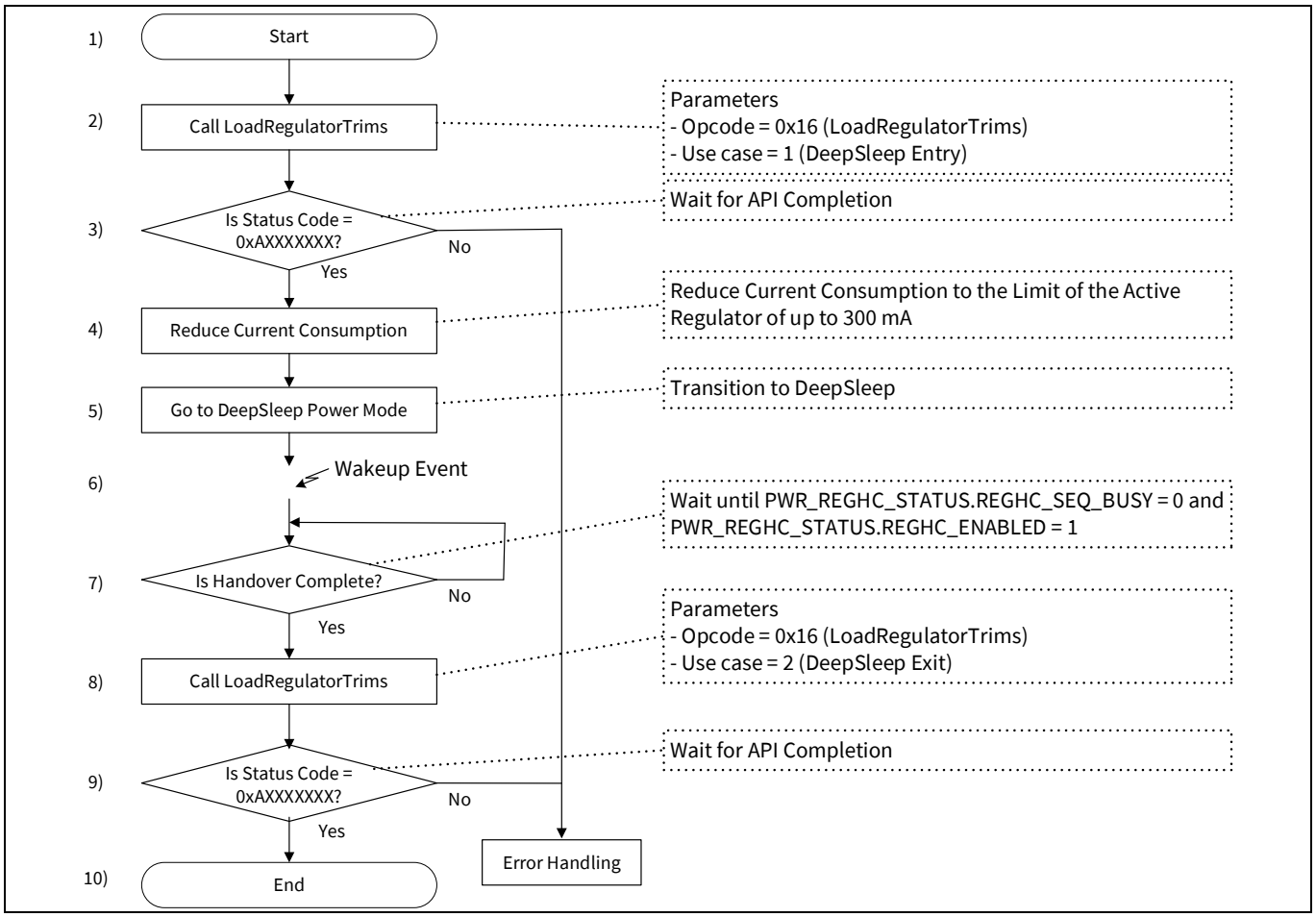

Figure 10 DeepSleep entry and exit flow (Pass transistor)

1. The MCU is powered by pass transistor.
2. Call the `LoadRegulatorTrims` API with the following parameters before the transition to DeepSleep power mode:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 1 (DeepSleep Entry)
3. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. Reduce the current consumption to within the limit of Active Regulator, which is 300 mA.
5. Start the transition to DeepSleep power mode using the “WFI” instruction. The MCU performs the handover to internal regulator mode, and pass transistor is disabled. Then, the Active Regulator is disabled, and DeepSleep Regulator supplies power during DeepSleep power mode.

   The MCU is powered by the DeepSleep Regulator.
6. Due to the wakeup event, the MCU transitions into Active Power mode. Then, pass transistor is enabled by the hardware.
7. Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1

Pass transistor

8. Call the `LoadRegulatorTrims` API with following parameters. Active and DeepSleep Regulators are set to external state.
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 2 (DeepSleep Exit)
9. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
10. The MCU is powered by pass transistor.

3.3.3 Handover from pass transistor to internal regulators

Figure 11 shows the flow of the handover from the pass transistor mode, controlled by the REGHC, to internal regulator.

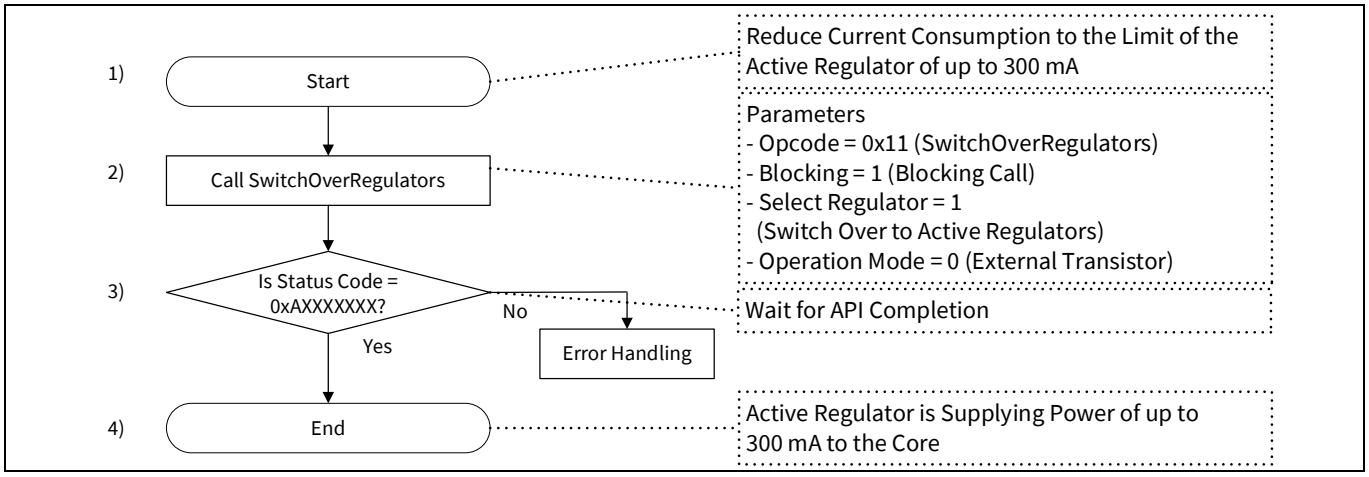

Figure 11 Transition flow from pass transistor to internal regulator

To transition from REGHC with pass transistor to internal regulator:

1. Reduce the current consumption to within the limit of Active Regulator, which is 300 mA.
2. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 1 (switch over to Active Regulator)
   - Operating Mode = 0 (External transistor)
3. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. The device is now operating with Active Regulator.

3.4 Component selection

Following are the components of a pass transistor:

Smoothing capacitor of V_{CCD}:

For capacitance, see the Recommended Operating Conditions for smoothing capacitor in the device datasheet **[1]**. Select a ceramic capacitor that does not have the influence of DC bias at 1.15 V with a temperature characteristic of X7R, +/- 10% tolerance.

Pass transistor

Table 14 lists the recommended smoothing capacitor.

Table 14 Recommended smoothing capacitor of V_{CCD}

Specification	Part number	Quantity	Vendor
10 uF, 6.3 V, X7R, +/-10%	CGA4J1X7R0J106K	1	TDK

Current sense resistor: 0.1 ohm 1/4W +/-1%.

External transistor type: NPN

Table 15 lists the requirements for an external transistor.

Table 15 External transistor requirement specification (Type: NPN)

Parameter	Symbol	Min value	Unit	Conditions
Static forward current transfer ratio	H_{FE}	100	-	I_C = 1 A, V_{CE} = 1 V
Collector-emitter voltage	V_{CEO}	10	V	-
Transition frequency	f_T	100	MHz	-
Collector current	I_C	1	A	-
Collector power dissipation	P_{CD}	2	W	-

Additionally, do not exceed the rated temperature, it is necessary to estimate the junction temperature.

Loss of pass transistor under the operating conditions is calculated by **Equation 1**.

Equation 1

$$P_{Loss} = (V_{DDD\_max} - V_{CCD\_min}) \times I_{VCCD\_max}$$

Where:

- P_{Loss}: Loss of pass transistor (W)
- V_{DDD_max}: Maximum V_{DDD} voltage (V)
- V_{CCD_min}: Minimum V_{CCD} voltage (V)
- I_{VCCD_max}: Maximum V_{CCD} load current (A)

The maximum power dissipation of pass transistor must be greater than or equal to P_{Loss}.

The junction temperature of pass transistor is calculated by **Equation 2**.

Equation 2

$$T_J = P_{Loss} \times \theta_{JA} + T_A$$

Where:

- T_J: Pass transistor junction temperature (°C), which must be below the rated temperature
- T_A: Ambient temperature (°C)
- P_{Loss}: Loss of pass transistor (W)
- θ_{JA}: Thermal resistance from junction to ambient temperature (°C/W)

Pass transistor

Table 16 shows examples of maximum θ_{JA}.

Table 16 Examples of maximum θ_{JA}

$T_{A\,max}$ (°C)	V_{DDD}	$P_{LOSS\,max}$ (W)	θ_{JA} (°C/W)
+125	5 V power rail	2.64	< 9.5
+105			< 17.0
+85			< 24.6
+125	3.3 V power rail	1.50	< 16.7
+105			< 30.0
+85			< 43.3

Note: V_{DDD} maximum voltage = 5.5 V/3.6 V, V_{CCD} minimum voltage = 1.10 V, V_{CCD} maximum load current = 0.6 A.

Loss by pass transistor can be reduced by supplying 3.3 V at V_{DDD}.

3.5 Layout design guidelines

Here are a few PCB layout design guidelines:

- Place the V_{CCD} smoothing capacitor close to the MCU, and place it next to the current sense resistor, pass transistor, and V_{DDD} smoothing capacitor.
- DRV_VOUT is sensitive to noise. Do not route the DRV_VOUT trace close to the source of noise.
- The signal routing between the EXT_PS_CTL0 and EXT_PS_CTL1 terminals and the current sense resistor terminal should be separated from the power wiring and should be wired parallel and close to each other. Since these signals are sensitive to noise, keep away from the source of noise and make them as short as possible.
- Pass transistor generates more heat depending on the operating conditions. It is necessary to lay out the board so that the heat is dissipated from the exposed pad (EP) to the board. The land area is placed to cover the EP on the mounting surface of the pass transistor and the bottom of the PCB. The via holes in the footprint of the EP are electrically connected and thermally coupled to the land area where are placed to cover the EP on each layer of the PCB including the power supply layer. Follow the layout guidelines of NPN transistor from the supplier for other precautions.

4 PMIC (switching regulator)

This section describes how to power V_{CCD} with PMIC.

Figure 12 shows an overview of connections to PMIC.

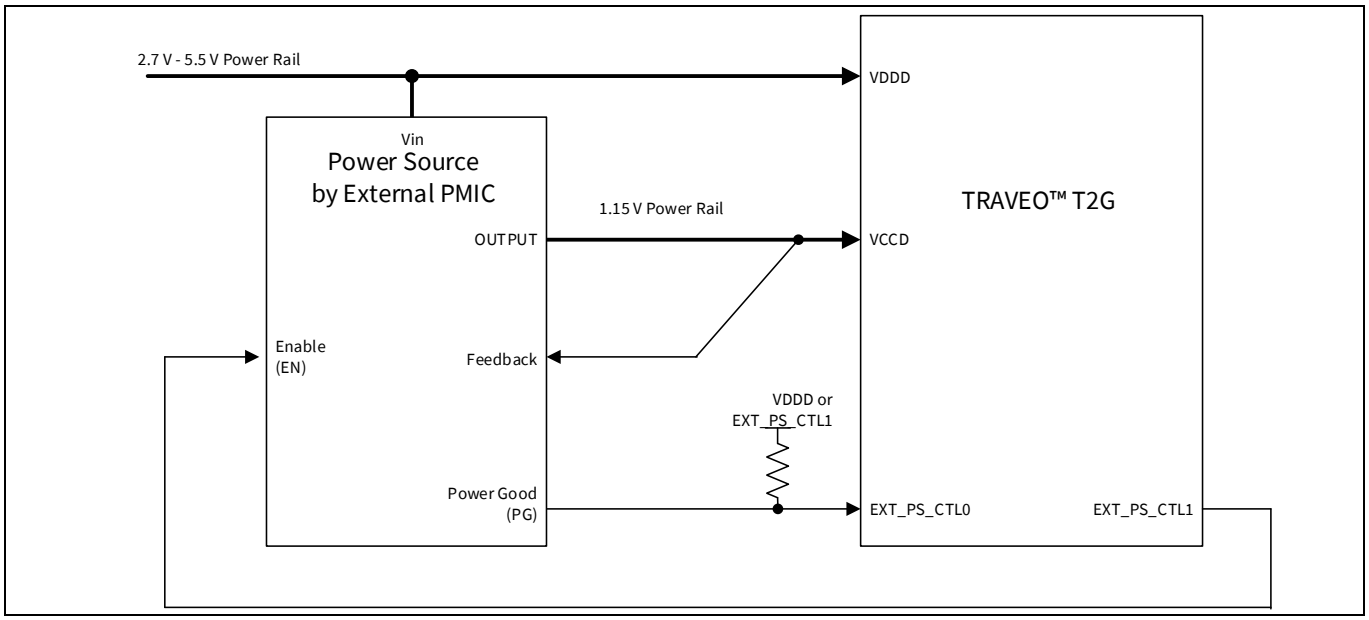

Figure 12 Connections between MCU and PMIC

- OUTPUT of PMIC is connected to V_{CCD} of MCU.
- Feedback of PMIC is connected to V_{CCD} of MCU.
- EN of PMIC is connected to EXT_PS_CTL1 of MCU. A pull-down/up resistor is required if EN voltage is unspecified when EN is opened.
- PG of PMIC is connected to EXT_PS_CTL0 of MCU. A pull-up resistor is required if PG is open drain.

Figure 13 shows basic power handover sequence of the V_{CCD} power rail with the connection.

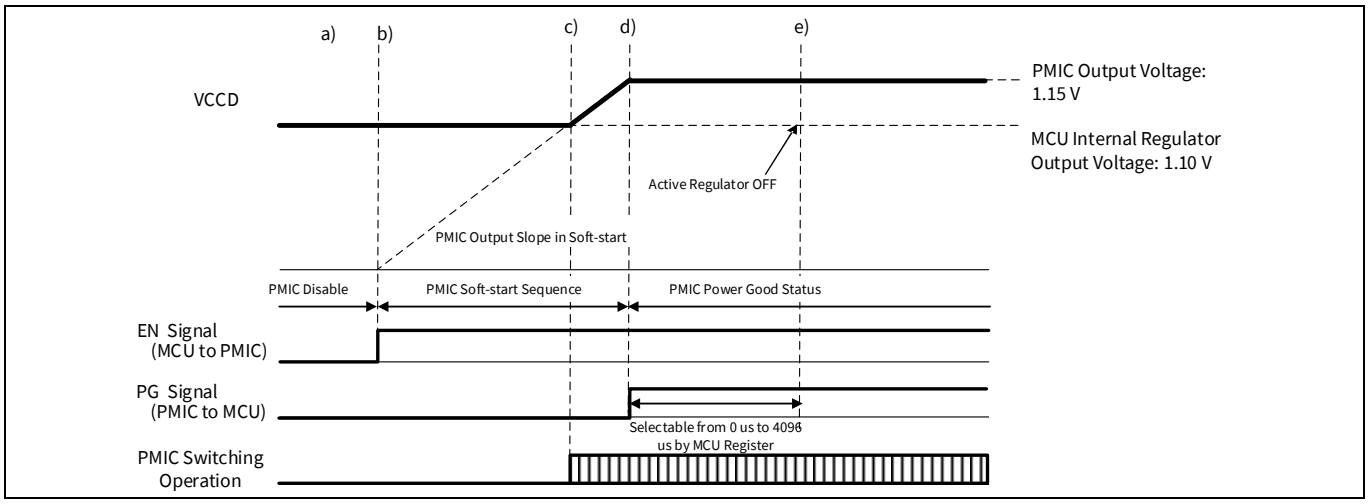

Figure 13 Basic V_{CCD} power handover sequence

PMIC (switching regulator)

a) Active Regulator as internal regulator supplies power to V_{CCD} power rail.

b) MCU enables PMIC. But, PMIC does not start the switching operation because target voltage of PMIC output in soft-start sequence is lower than V_{CCD} voltage output by Active Regulator.

c) PMIC target voltage of soft-start sequence crosses V_{CCD} voltage output by Active Regulator. At that time, PMIC starts switching operation and supplies power to V_{CCD} power rail.

d) VCCD voltage rises to PMIC target voltage for steady state and PMIC soft-start sequence completes, Then, PMIC drops down PG signal.

e) MCU turns OFF Active Regulator when MCU detects that the PG signal is in power good status.

4.1 Required PMIC specification

The MCU controls two power sources: Active Regulator on MCU and PMIC, to supply V_{CCD} power. So, consider the following points while selecting PMIC:

- PMIC must provide correct and stable voltage to MCU.
- If the PMIC has a discharge function, the PMIC sinks current during DeepSleep mode and increases the current consumption.
- Power handover may cause voltage drop and over current due to PMIC sinks current from MCU when the VCCD Power source switches from Active Regulator to PMIC.
- MCU controls PMIC enable with PMIC EN terminal.
- MCU can detect PMIC voltage by using power good function.

Feature of the selected PMIC determines whether a load switch for isolation between PMIC and V_{CCD} terminals is required. **Table 17** shows PMIC feature requirement for power structures, **PMIC direct connection** and **PMIC with load switch**.

Table 17 PMIC requirement

Feature	Mandatory feature for power structure		Comment
	PMIC direct connection	PMIC with load switch	
Output voltage precision/Current ability	✓	✓	For voltage and current ability, see the Recommended Operating Conditions for V_{CCD} in the device datasheet [1].
No discharge function	✓	N/A	PMIC must not operate discharge-function when PMIC is disabled to prevent unintended current consumption.
Pre-Bias soft-start	✓	N/A	PMIC must not sink from the Active Regulator of the MCU.
Enable	✓	✓	Required to control PMIC output
Power good	✓	✓	Required to indicate PMIC output status.

PMIC (switching regulator)

No discharge function

Discharge function discharges the output capacitance by connecting a resistor (for instance, 100-Ω range) between V_{CCD} and ground, when PMIC is disabled. So, when such a PMIC is disabled in the DeepSleep mode, V_{CCD} would be discharged. Some PMICs discharge in case of under voltage lock out (UVLO) also. PMIC must not have the discharge function because the discharge current might cause unintended current consumption.

Pre-bias startup and soft-start

Soft-start is a function that gradually increases the output voltage to prevent rush current when switching starts in a fixed defined ramp up time, independent of the applied voltage level at the starting point of ramp-up (pre-bias startup). There are two types of soft-start operations. Pre-bias soft-start does not sink current from MCU and does not start switching operation until output voltage of PMIC reaches the voltage level of the Active Regulator. The Forced 0V soft-start forcibly starts the output from 0 V by sinking the current from the MCU through V_{CCD} terminal. See the PMIC datasheet or contact PMIC vendor to find out which soft-start is employed. **Figure 14** shows the timing chart of each soft-start.

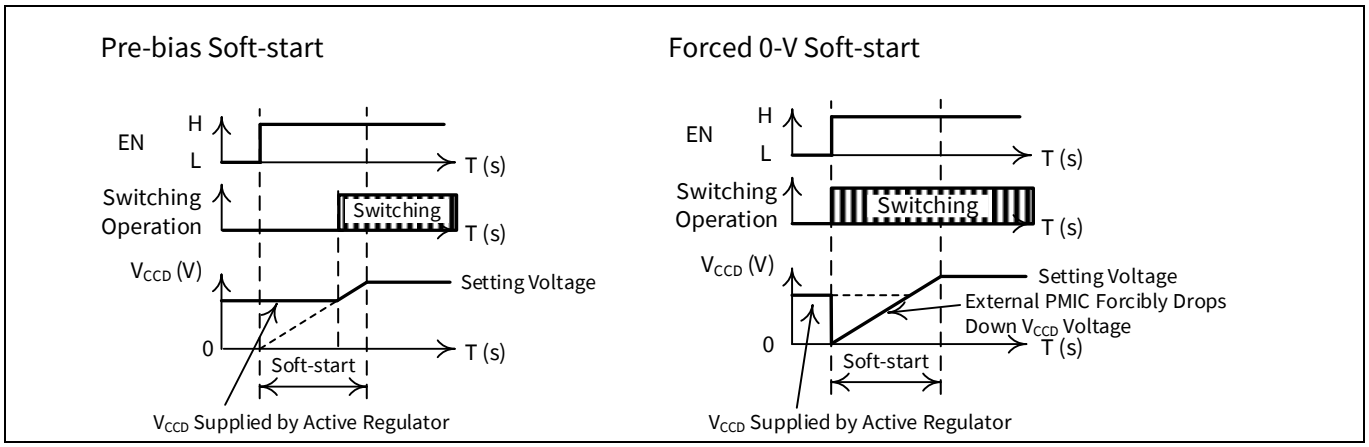

Figure 14 Soft-start timing chart

Note: Certain PMIC activates over voltage detection during soft-start. In this case, PMIC detects over voltage during soft-start and PMIC does not start switching. When it reaches its tolerant range of over voltage detection, PMIC starts switching. Lower limit of this tolerant range results in PMIC actively pulling down V_{CCD} that is supplied by Active Regulator. Such PMIC must be treated in the same manner as forced 0-V soft-start.

Note: It is strongly recommended to check the implementation of the pre-bias startup at the starting point. A minor drop might happen, but the voltage must be kept within the operating range of the internal Active Regulator.

Power good

PG should not indicate power good state during the disabling of PMIC. Furthermore, it is better that the function does not indicate power good status during soft-start function or before reaching 100% of target voltage, because PMIC cannot supply power correctly at that time. Otherwise, the Active Regulator may be turned OFF before PMIC becomes fully operational. If PG indicates power good state during soft-start before reaching 100%, software with the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register can prevent early turn OFF of the Active Regulator. For more detail of PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register, see the “Software Flow” of each power structure.

4.2 Recommended PMIC topology

Table 18 lists the recommended PMIC topology for each use case.

Table 18 PMIC requirement

Use case	Topology	Advantage
PMIC is disabled in DeepSleep mode Case 1, Case 2	Forced PWM mode	Small switching ripple voltage and small voltage fluctuation by load transient response. Fixed switching frequency.
PMIC enabled in DeepSleep mode Case 3, Case 4	Auto PFM/PWM mode	Power saving for V_{CCD} supplied by external PMIC in DeepSleep mode.

Forced PWM mode vs. auto PWM/PFM mode

Forced PWM mode always operates with only PWM operation.

Auto PWM/PFM mode is an operation where PMIC automatically switches between PWM operation and PFM operation depending on the load condition.

This mode works with PWM operation when there is a heavy load current condition, and with PFM operation when there is a light load current condition.

PWM operation works with constant frequency. It regulates output voltage by changing the on duty of the switching FET. Load transient response is better than PFM operation and its output ripple voltage is smaller than PFM operation. Furthermore, EMI noise analysis is easier than PFM operation because of the fixed switching frequency.

PFM regulates the output voltage by changing the switching frequency according to the load current. The advantage of PFM operation is to improve conversion efficiency in light load current conditions.

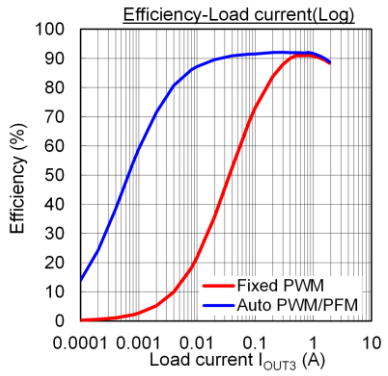

Figure 15 Conversion efficiency example

Note: PMIC does not sink current in PFM operation mode, as only the high-side switch is active. Therefore, the soft-start function behaves similar to pre-bias soft-start if PMIC can operate PFM mode during soft-start also. But make sure, Auto PWM/PFM mode of PMIC operates with the PFM mode during soft-start.

PMIC (switching regulator)

Note: *A PMIC in PFM mode has lower operation accuracy than in PWM mode.*

4.3 PMIC direct connection

This section describes PMIC connection, handover timing, and handover software flow example based on the following use cases depending on the internal regulator configuration for PMIC direct connection:

- **Case 1**: Use of OCD in Active mode with PMIC disabled in DeepSleep power mode
- **Case 2**: No use of OCD in Active mode with PMIC disabled in DeepSleep power mode
- **Case 3**: No use of OCD in Active mode with PMIC enabled in DeepSleep power mode (using SwitchOverRegulators with blocking call)
- **Case 4**: No use of OCD in Active mode with PMIC enabled in DeepSleep power mode (using SwitchOverRegulators with non-blocking call)

For details, see **Internal regulator configuration using PMIC**.

Note: *The use of OCD for over current monitoring of the external power supply is not recommended, as the monitoring is not a direct shunt implementation of the VCCD power rail. It is an integral part of the Active Regulator.*

Note: *The configuration of OCD usage refers only to the MCU operation with external PMIC supply in Active mode. After POR for example, the OCD feature is enabled, as it is part of the internal regulator.*

Note: *In case 3, it is not possible to switch back to the internal regulator after handover to PMIC. If you want to return to the internal regulator, you need to reset the entire chip, including the power system, by using WDT or requesting an external system controller trigger XRES_L.*

4.3.1 Hardware configuration

Figure 16 shows the circuit diagram for PMIC direct connection.

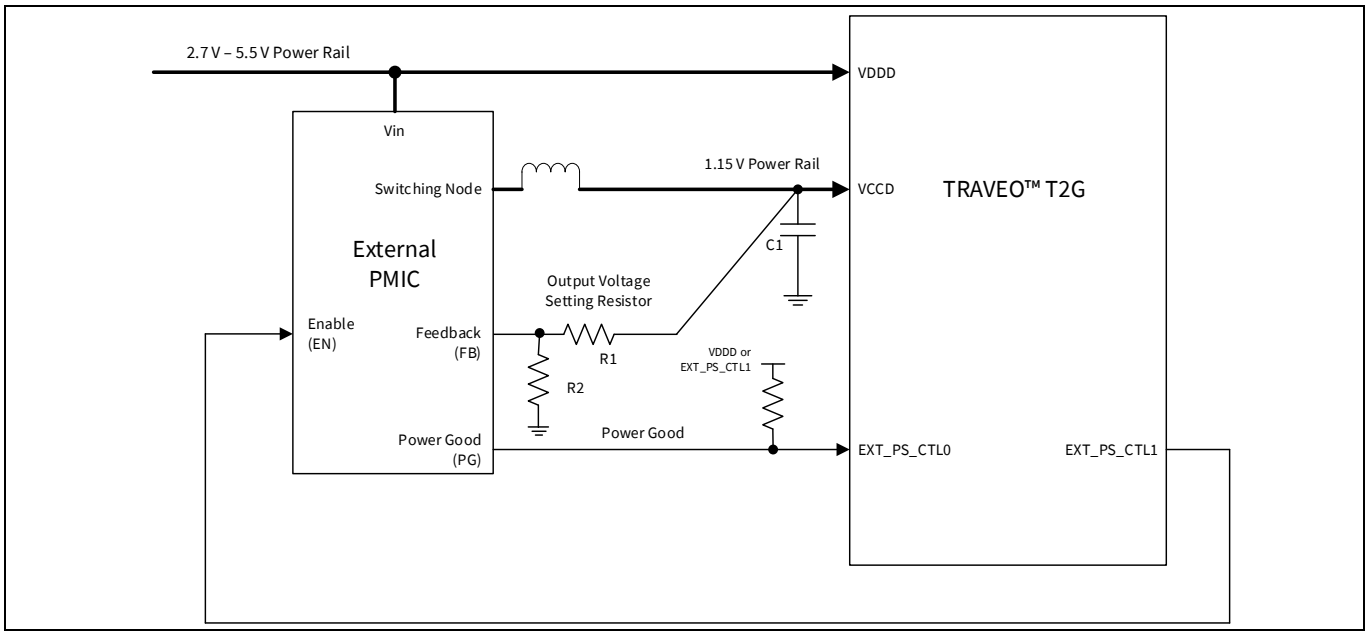

Figure 16 Circuit diagram for PMIC direct connection

- PMIC EN polarity is HIGH for enable. PMIC PG polarity is HIGH for power good.

  RESET_PMIC is generated when PG of PMIC goes LOW. This is HV reset. When HV reset is applied, the MCU including REGHC or PMIC controller, and PMIC is initialized.
- FB terminal of PMIC is connected to V_{CCD}.
- Output voltage setting resistors are needed according to the selected PMIC.
- EN terminal of PMIC is connected to the EXT_PS_CTL1 terminal of MCU. A pull-down resistor on PCB or in PMIC is required to disable PMIC during XRES and Hibernate.
- If EN terminal does not have the internal pull-down resistor, an external pull-down resistor must be placed to keep the PMIC disabled during POR.
- V_{CCD} capacitor (C1) depends on the MCU requirement (see the device datasheet **[1]** for the capacitance).
- Output voltage setting resistors (R1, R2) depend on the selected PMIC.

This section describes the software flow when PMIC enable polarity is “1” and PG status polarity is “1”.

Enable Polarity in the `ConfigureRegulator` API specifies the polarity of the enable signal to PMIC, and Reset Polarity in the `ConfigureRegulator` API specifies the polarity of the PG state from PMIC. Note that the Reset polarity bitfield corresponds the polarity of the de-assertion level of PG signal.

4.3.2 Case 1

This case explains the use of OCD in Active mode, PMIC is disabled in DeepSleep power mode.

Note: *The use of OCD for over current monitoring of the external power supply is not recommended, as the monitoring is not a direct shunt implementation of the VCCD power rail. It is an integral part of the Active Regulator.*

4.3.2.1 Handover timing chart

In Case 1, Active Regulator is enabled (OCD is used), and DeepSleep Regulator is enabled (DeepSleep Regulator is enabled during DeepSleep power mode). **Figure 17** shows an example of the handover sequence. This example includes regulator configuration, handover from internal regulators to PMIC, transition to DeepSleep power mode, wake up from DeepSleep power mode, and handover from PMIC to internal regulators.

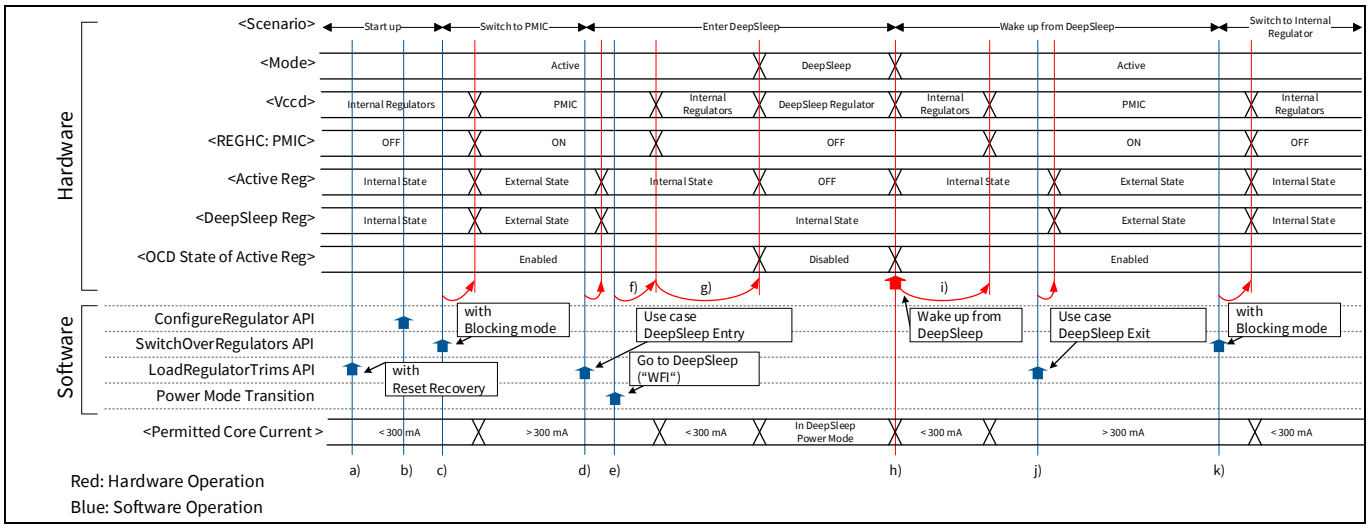

Figure 17 Handover sequence example (case 1)

a) Call the `LoadRegulatorTrims` API with reset recovery use case, if PWR_REGHC_CTL.REGHC_CONFIGURED is set to "1". See the **reset recovery** for details.

b) Call the `ConfigureRegulator` API for REGHC or PMIC controller with PMIC configuration after start up.

c) Call the `SwitchOverRegulators` API for the handover from internal regulators to PMIC.

   PMIC is enabled. Active regulator is still enabled due to the OCD feature, but changes to external state. The DeepSleep Regulator changes to external state.

d) Call the `LoadRegulatorTrims` API with DeepSleep Entry use case for changing the internal state of the regulator.

   The Active and DeepSleep Regulators change to internal state.

e) Reduce the current consumption to be within the limit of the Active Regulator, which is 300 mA. Then, you can execute the transition to DeepSleep power mode using the "WFI" instruction.

f) When the software executes the transition to DeepSleep power mode, the hardware switches from PMIC to internal regulators.

g) MCU transitions to DeepSleep power mode after the switching to internal regulators is complete. The Active Regulator turns OFF.

PMIC (switching regulator)

h) A power mode transition from DeepSleep to Active happens after the wake-up event, and then the Active Regulator starts with internal state.

i) PMIC is enabled after Active power mode transition is complete.

j) Call the `LoadRegulatorTrims` API with DeepSleep Exit use case for changing the internal regulators to external state.

   The Active and DeepSleep Regulators change to external state.

k) Call the `SwitchOverRegulators` API for the handover from PMIC to internal regulators.

   PMIC is disabled. The Active and DeepSleep Regulators change to internal state.

4.3.2.2 Software flow

Handover from internal regulator to PMIC

Figure 18 shows the handover flow from internal regulator to PMIC.

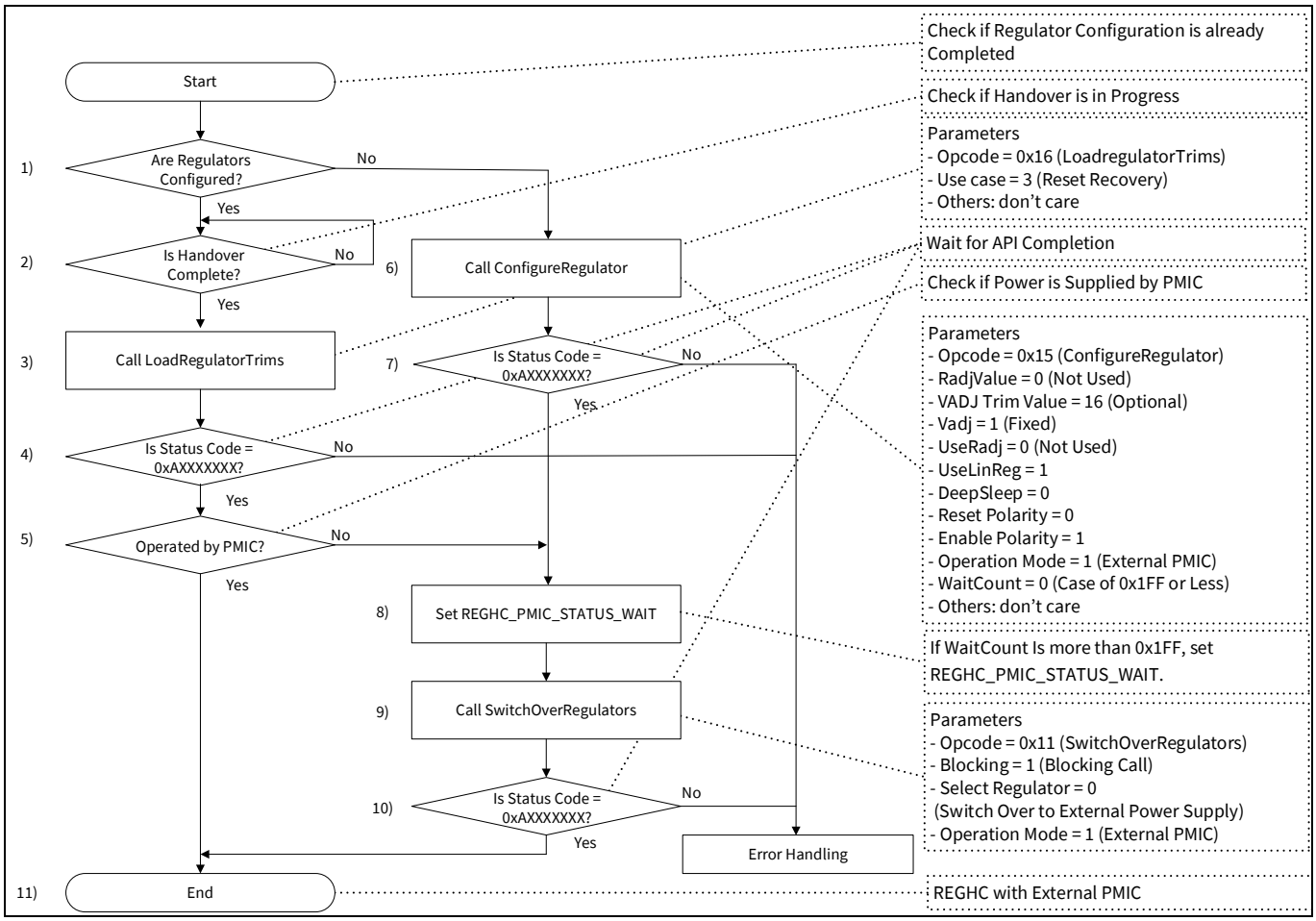

Figure 18 Handover flow from internal regulator to PMIC (case 1)

1. Check if regulator configuration is already completed. Once REGHC or PMIC controller is setup, it can be enabled without reconfiguring.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to "0", the configuration is not complete. Go to (6).

2. Wait for the handover completion.

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = "0" and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN

3. Call the `LoadRegulatorTrims` API with reset recovery:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 3 (Reset Recovery)
4. Wait for the completion of the `LoadRegulatorTrims` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.

PMIC (switching regulator)

5. Check if MCU operates with PMIC. When PWR_REGHC_STATUS.REGHC_ENABLED is set to “1”, the PMIC is supplying power. Go to (11). When PWR_REGHC_STATUS.REGHC_ENABLED is set to “0”, the internal regulator is supplying power. Go to (8).
6. Call the `ConfigureRegulator` API with the following parameters:
   - Opcode = 0x15 (Call the `ConfigureRegulator` API)
   - RadjValue[*1] = 0 (Not used)
   - VADJ trim value = 16
   - Vadj = 1 (Fixed)
   - UseRadj[2] = 0 (Not used)
   - UseLinReg = 1 (Internal Active Regulator is kept as enabled after PMIC is enabled)
   - DeepSleep[2*1] = 0 (PMIC is disabled in DeepSleep power mode)
   - Reset Polarity = 0 (PMIC PG signal indicates power bad status with “0”)
   - Enable Polarity = 1 (PMIC is enabled by “1”)
   - Operating Mode[2] = 1 (External PMIC)
   - WaitCount = 0 (0x1FF or less)
7. Wait for the `ConfigureRegulator` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
8. When you need to set the WaitCount to a value greater than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register. If not, this process is unnecessary because it can be set by the `ConfigurRegulator` API. See the `ConfigureRegulator` API in **System Call API** for more details.

To enable the REGHC or PMIC controller with PMIC, perform these steps:

9. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 0 (Switch over to external power supply)
   - Operating Mode[*1] = 1 (External PMIC) This parameter must match the Operating Mode in the `ConfigureRegulator` API
10. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
11. The device is now operating with PMIC. Active and DeepSleep Regulators operate in external state.

DeepSleep entry and exit

Figure 19 shows transition flow to DeepSleep power mode.

[2] This is invalid in CYT3D/4D series.

PMIC (switching regulator)

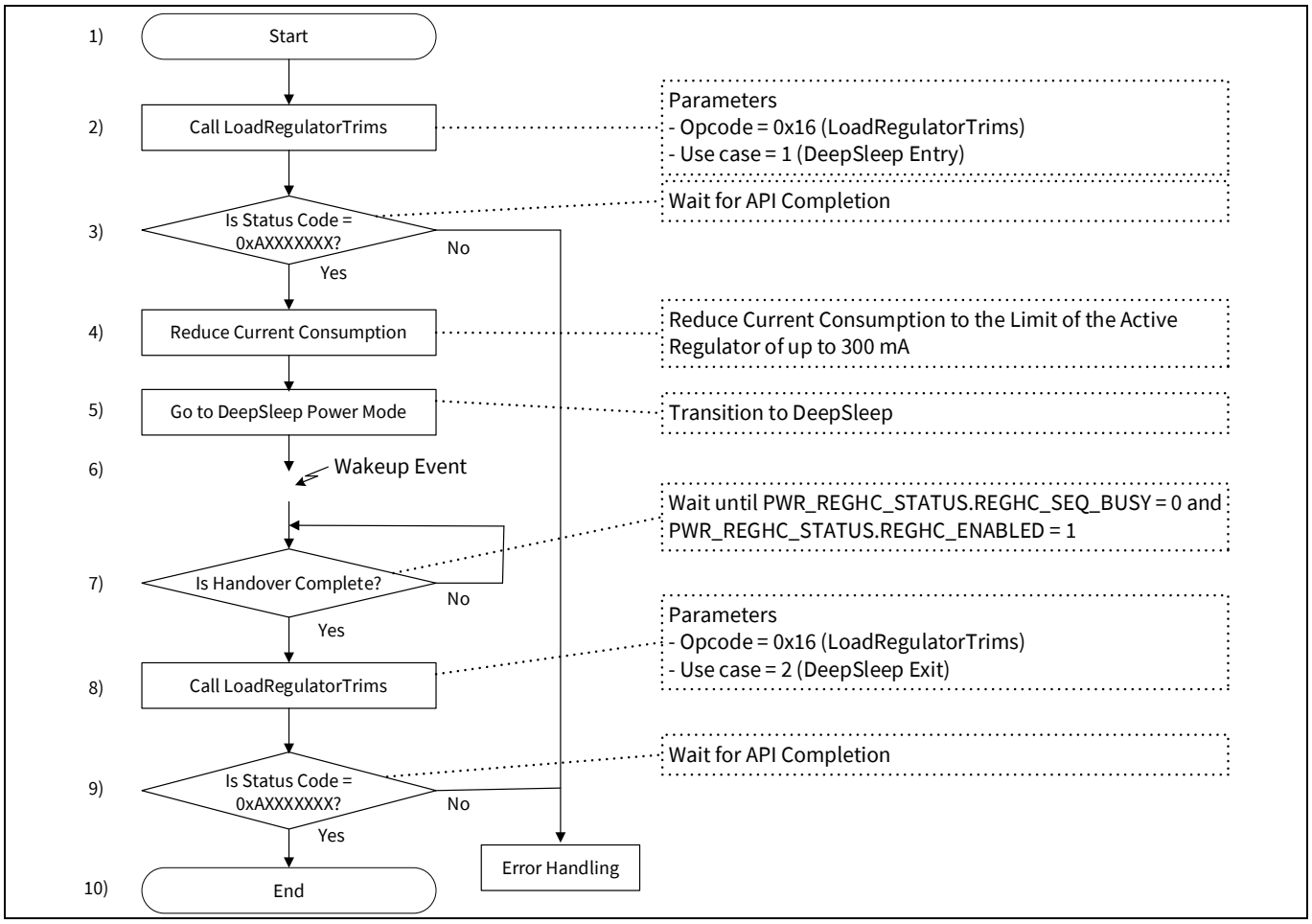

Figure 19 DeepSleep entry and exit flow (case 1)

1. The MCU is powered by PMIC.
2. Call the `LoadRegulatorTrims` API with the following parameters before transitioning to DeepSleep power mode:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 1 (DeepSleep Entry)
3. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. Reduce the current consumption to within the limit of Active Regulator, which is 300 mA.
5. Start the transition to DeepSleep power mode using the “WFI” instruction. Perform handover to internal regulator; and PMIC is disabled. Then, Active Regulator is disabled, and DeepSleep Regulator supplies power during DeepSleep power mode.

   The MCU is powered by the DeepSleep Regulator.
6. Due to wakeup event, the MCU transitions to Active Power mode. Then, the hardware enables PMIC.
7. Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1
8. Call the `LoadRegulatorTrims` API with the following parameters. Active and DeepSleep Regulators are set to external state.
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 2 (DeepSleep Exit)

PMIC (switching regulator)

9. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
10. The MCU is powered by PMIC.

Handover from PMIC to internal regulator

Figure 20 shows the handover flow from PMIC to internal regulator.

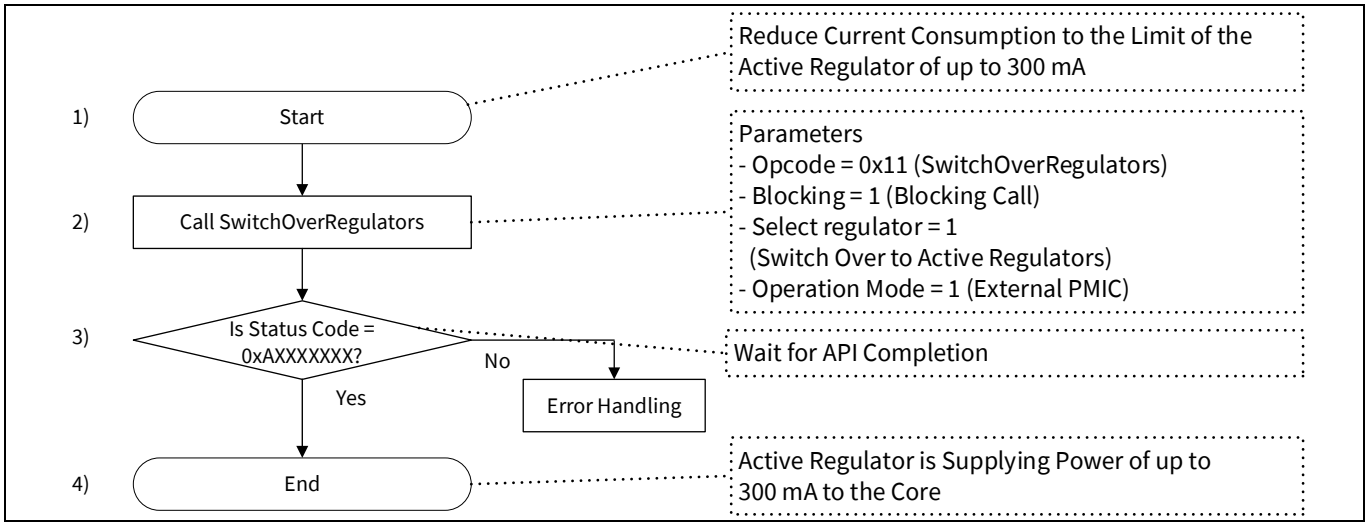

Figure 20 Handover flow from PMIC to internal regulator (case 1)

To transition from PMIC to internal regulator:

1. Reduce the current consumption of the application to the maximum possible current of Active Regulator, which is 300 mA (by configuring the appropriate application such as clock, cores, peripheral).
2. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 1 (Switch over to Active Regulator)
   - Operating Mode[3] = 1 (External PMIC)
3. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. The device is now operating with the internal regulator.

4.3.3 Case 2

In this case, OCD is not used in Active mode, PMIC is disabled in DeepSleep power mode.

4.3.3.1 Handover timing chart

In Case 2, Active Regulator is disabled (OCD is not used) while the PMIC supplies the MCU, and DeepSleep Regulator is enabled in DeepSleep mode (that is PMIC is disabled). **Figure 21** shows the handover sequence example. This example includes regulators configuration, handover from internal regulators to the PMIC, transition to DeepSleep power mode, and wake up from DeepSleep power mode.

[3] This is invalid in CYT3D/4D series.

PMIC (switching regulator)

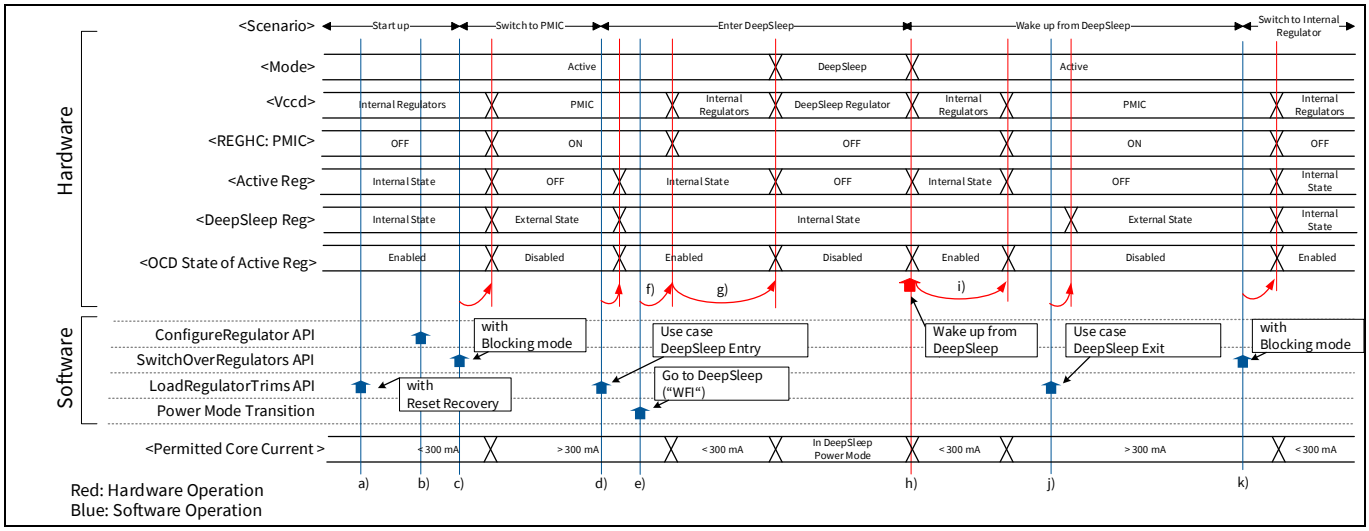

Figure 21 Handover sequence example (case 2)

a) Call the `LoadRegulatorTrims` API with reset recovery use case, if PWR_REGHC_CTL.REGHC_CONFIGURED is set to "1". See the **reset recovery** for details.

b) Call the `ConfigureRegulator` API for REGHC or PMIC controller with PMIC configuration after starting up.

c) Call the `SwitchOverRegulators` API for the handover from internal regulators to PMIC.

   PMIC is enabled. Active Regulator is disabled, because the OCD feature is not requested. The DeepSleep Regulator changes to external state.

d) Call the `LoadRegulatorTrims` API with DeepSleep Entry use case for changing the internal state of the regulator.

   The Active and DeepSleep Regulators change to internal state before transitioning to DeepSleep power mode.

e) Reduce the current consumption to be within the limit of the Active Regulator, which is 300 mA. Then, you execute the transition to DeepSleep power mode using the "WFI" instruction.

f) When the software executes the transition to DeepSleep power mode, the hardware switches from PMIC to internal regulators.

g) MCU transitions to DeepSleep power mode after the switching to internal regulators is complete. The Active Regulator is disabled after transition to DeepSleep power mode.

h) A power mode transition from DeepSleep to Active happens after the wake-up event, and then the Active Regulator starts with internal state.

i) After the Active power mode transition is complete, PMIC is enabled and Active Regulator is disabled.

j) Call the `LoadRegulatorTrims` API with DeepSleep Exit use case for changing the external state of the regulator. The DeepSleep Regulator changes to external state.

k) Call the `SwitchOverRegulators` API for the handover from PMIC to internal regulators.

   PMIC is disabled. The Active and DeepSleep Regulators change to internal state.

4.3.3.2 Software flow

Handover from internal regulator to PMIC

Figure 22 shows the handover flow from internal regulator to PMIC.

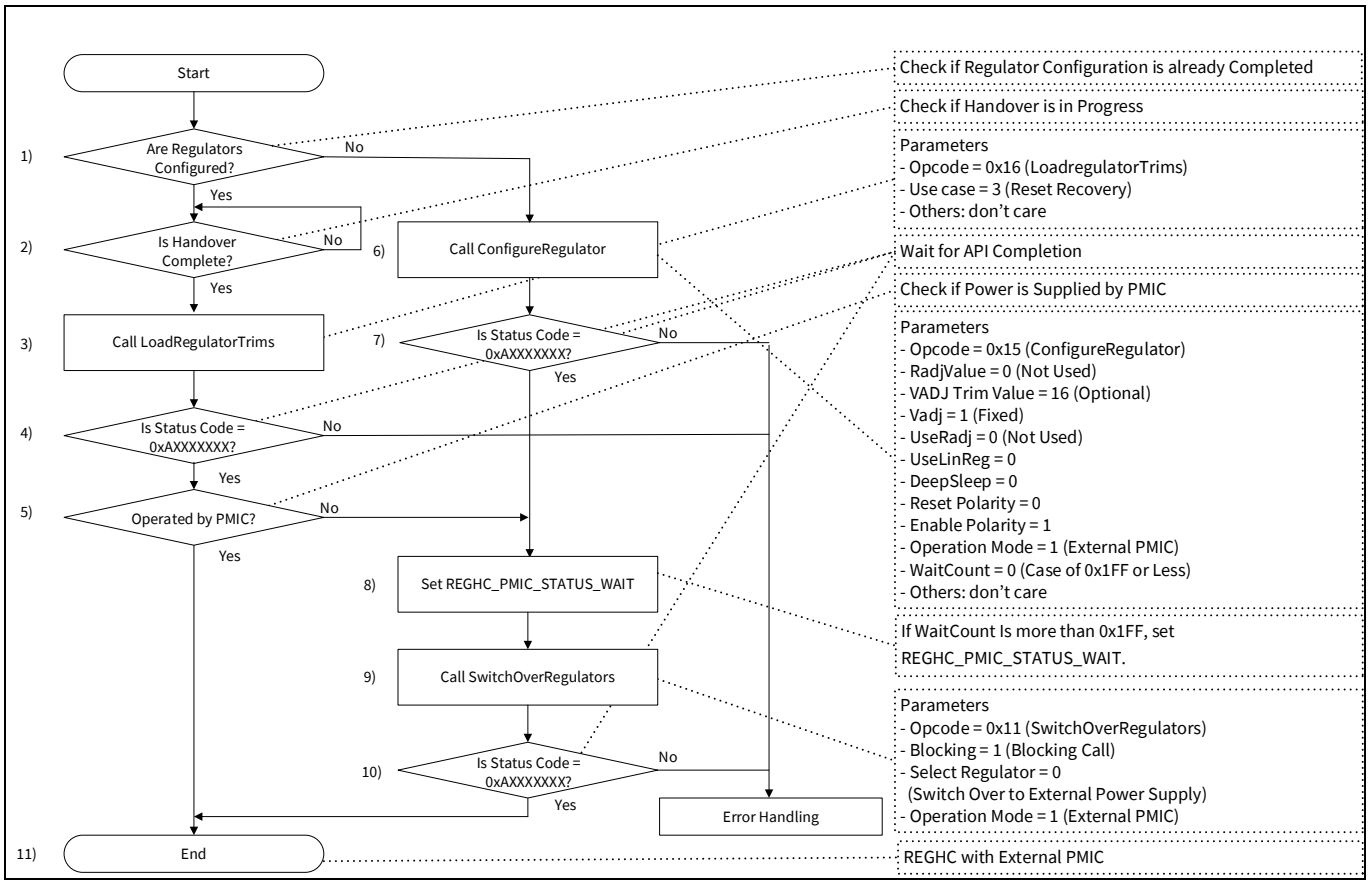

Figure 22 Handover flow from internal regulator to PMIC (case 2)

1. Check if regulator configuration is already completed. Once REGHC or PMIC controller is setup, it can be enabled without reconfiguring.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to "0", the configuration is not complete. Go to (6).

2. Wait for handover completion.

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = "0" and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN

3. Call the `LoadRegulatorTrims` API with reset recovery:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 3 (Reset Recovery)
4. Wait for the completion of the `LoadRegulatorTrims` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.
5. Check if MCU operates with PMIC. When PWR_REGHC_STATUS.REGHC_ENABLED is set to "1", the PMIC is supplying power. Go to (11). When PWR_REGHC_STATUS.REGHC_ENABLED is set to "0", the internal regulator is supplying power. Go to (8).

PMIC (switching regulator)

6. Call the `ConfigureRegulator` API with the following parameters:
   - Opcode = 0x15 (Call the `ConfigureRegulator` API)
   - RadjValue[4] = 0 (Not used)
   - VADJ trim value = 16
   - Vadj = 1 (Fixed)
   - UseRadj[*1] = 0 (Not used)
   - UseLinReg = 0 (Internal Active Regulator is disabled after PMIC is enabled)
   - DeepSleep[4] = 0 (PMIC is disabled in DeepSleep power mode)
   - Reset Polarity = 0 (PMIC PG signal indicates power bad status with “0”)
   - Enable Polarity = 1 (PMIC is enabled by “1”)
   - Operating Mode[4] = 1 (External PMIC)
   - WaitCount = 0 (0x1FF or less)
7. Wait for the `ConfigureRegulator` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
8. When you need to set the WaitCount to a value greater than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register. If not, this process is unnecessary because it can be set by the `ConfigurRegulator` API. See the `ConfigureRegulator` API in **System Call API** for more details.

To enable the REGHC or PMIC controller with PMIC, perform these steps:

9. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 0 (Switch over to external power supply)
   - Operating Mode[*1] = 1 (External PMIC) This parameter must match the Operating Mode in the `ConfigureRegulator` API
10. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
11. The device is now operating with PMIC. Active and DeepSleep Regulators operate in external state.

DeepSleep entry and exit

Figure 23 shows the transition to DeepSleep power mode.

[4] This is invalid in CYT3D/4D series.

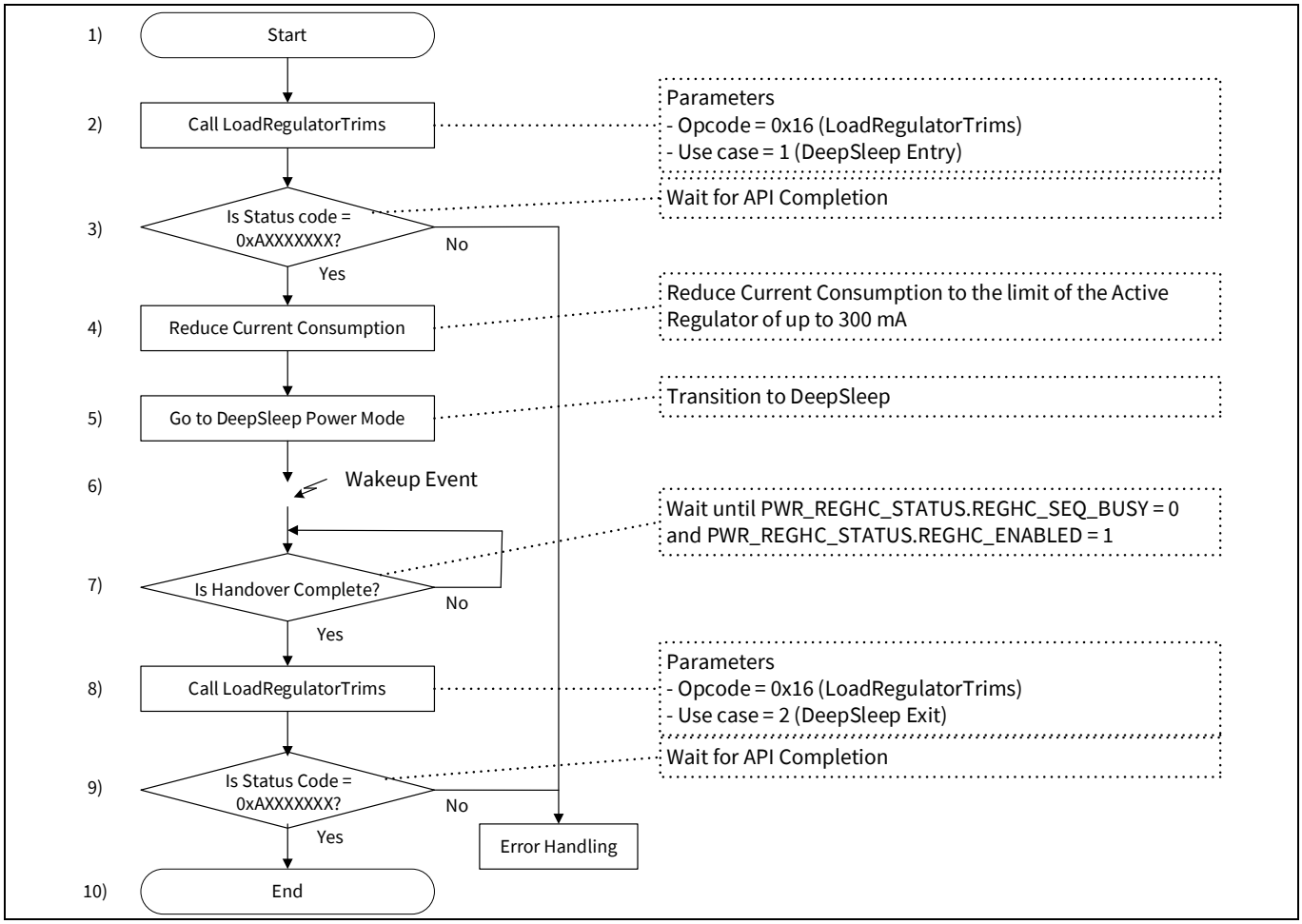

Figure 23 DeepSleep entry and exit flow (case 2)

1. The MCU is powered by PMIC.
2. Call the `LoadRegulatorTrims` API with the following parameters before transitioning to DeepSleep power mode:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 1 (DeepSleep Entry)
3. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. Reduce the current consumption to be within the limit of Active Regulator, which is 300 mA.
5. Start the transition to DeepSleep power mode using the “WFI” instruction. The MCU performs handover to internal regulator, and PMIC is disabled. Then, Active Regulator is disabled, and DeepSleep Regulator supplies power during DeepSleep power mode.

   The MCU is powered by DeepSleep Regulator.
6. Due to the wakeup event, the MCU transitions to Active Power mode. Then, the hardware enables PMIC.
7. Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1
8. Call the `LoadRegulatorTrims` API with the following parameters. Active and DeepSleep Regulators are set to external state:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 2 (DeepSleep Exit)

PMIC (switching regulator)

9. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
10. The MCU is powered by PMIC.

Handover from PMIC to internal regulator

Figure 24 shows the handover flow from PMIC to internal regulator.

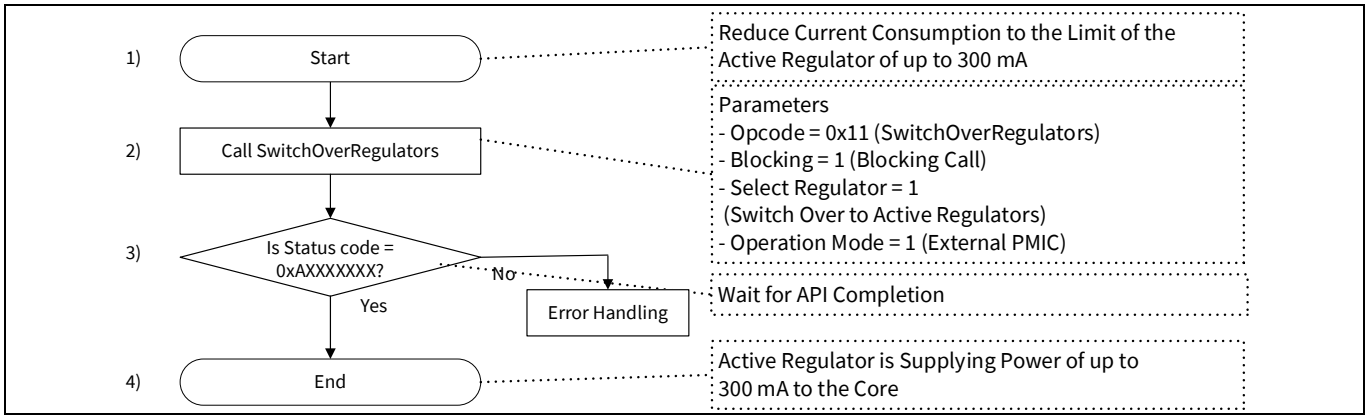

Figure 24 Handover flow from PMIC to internal regulator (case 2)

To transition from PMIC to internal regulator:

1. Reduce the current consumption of the application to the maximum possible current of Active Regulator, which is 300 mA (by configuring the appropriate application such as clock, cores, peripheral).
2. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 1 (Switch over to Active Regulator)
   - Operating Mode[5] = 1 (External PMIC)
3. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. The device is now operating with the internal regulator.

4.3.4 Case 3

In this case, OCD is not used in Active mode, PMIC is enabled in DeepSleep power mode. (Using SwitchOverRegulators with Blocking Call)

4.3.4.1 Handover timing chart

In Case 3, Active Regulator is disabled (OCD is not used), and DeepSleep Regulator is disabled (PMIC supplies power during DeepSleep power mode). **Figure 25** shows an example of the handover sequence. This example includes regulator configuration, handover from internal regulators to the PMIC, transition to DeepSleep power mode, and wake up from DeepSleep power mode. In this configuration, the API cannot be used to switch back

[5] This is invalid in CYT3D/4D series.

from the PMIC to the internal regulator. When operating with the internal regulator, you need to reset the entire chip (HV reset).

If your system requires to run the handover from PMIC to internal regulators using the `SwitchOverRegulators` API, you need to call the `SwitchOverRegulators` API with non-blocking call when running the handover to PMIC. See **Case 4**.

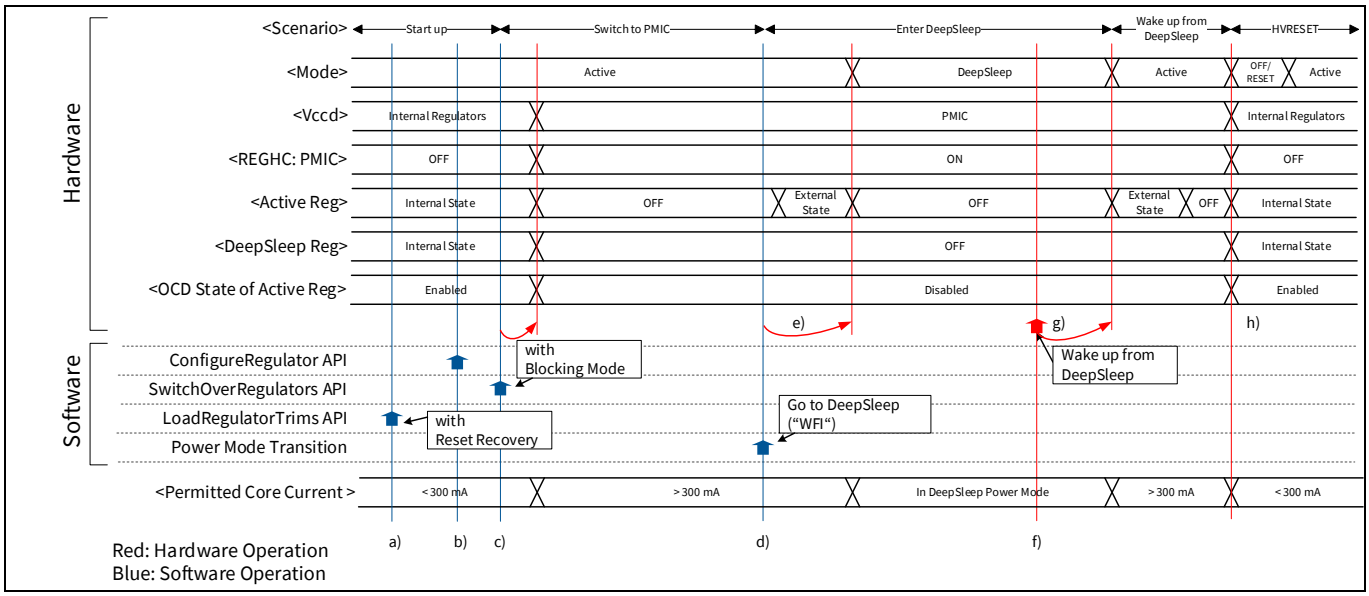

Figure 25 Handover sequence example (case 3)

a) Call the `LoadRegulatorTrims` API with reset recovery use case, if PWR_REGHC_CTL.REGHC_CONFIGURED is set to "1". See the **reset recovery** for details.

b) Call the `ConfigureRegulator` API for REGHC or PMIC controller with PMIC configuration after starting up.

c) Call the `SwitchOverRegulators` API with blocking call for the handover from internal regulators to PMIC

Note: When call the `SwitchOverRegulators` API with blocking call, the API sets the PWR_CTL2.DPSLP_REG_DIS to "1" and you cannot switch back from PMIC to internal regulators.

PMIC is enabled. Active and DeepSleep Regulators are disabled, because OCD is not used and DeepSleep Regulator is disabled in DeepSleep power mode.

d) You execute the transition to DeepSleep power mode using the "WFI" instruction. In this case, the system can transition to DeepSleep power mode directly without reducing current consumption. PMIC will continue to supply power.

e) MCU transitions to DeepSleep power mode.

In this case, Active and DeepSleep Regulators are disabled after calling the `SwitchOverRegulators` API. Power is always supplied from PMIC. Therefore, the `LoadRegulatorTrims` API is unnecessary for the transition to DeepSleep power mode or Active power mode.

f) A power mode transition from DeepSleep to Active happens after the wake-up event. Active and DeepSleep Regulators are disabled.

g) MCU transitions to Active power mode, PMIC will continue to supply power.

h) When HV reset occurs, power system is initialized. Therefore, the MCU starts up with Active Regulator after the reset is released

4.3.4.2 Software flow

Handover from internal regulator to PMIC

Figure 26 shows the handover flow from internal regulator to PMIC.

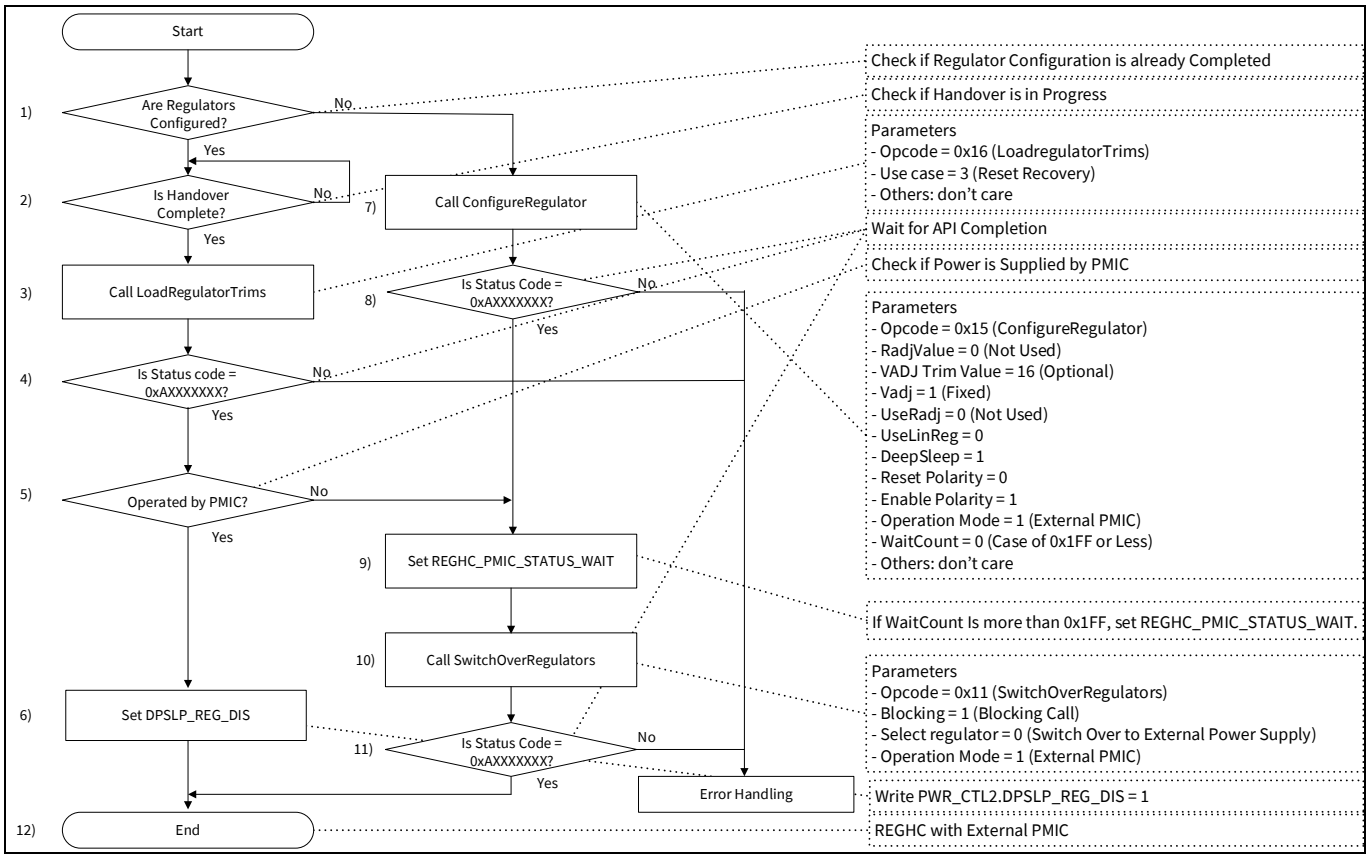

Figure 26 Handover flow from internal regulator to PMIC (case 3)

1. Check if regulator configuration is already completed. Once REGHC or PMIC controller is setup, it can be enabled without reconfiguring.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to "0", the configuration is not complete. Go to (7).

2. Wait for handover completion.

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = "0" and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN

3. Call the `LoadRegulatorTrims` API with reset recovery:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 3 (Reset Recovery)
4. Wait for the completion of the `LoadRegulatorTrims` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.
5. Check if MCU operates with PMIC. When PWR_REGHC_STATUS.REGHC_ENABLED is set to "1", the PMIC is supplying power. Go to (6). When PWR_REGHC_STATUS.REGHC_ENABLED is set to "0", the internal regulator is supplying power. Go to (9).
6. Set PWR_CTL2.DPSLP_REG_DIS to "1"

PMIC (switching regulator)

In case 3, PMIC is enabled in DeepSleep power mode. Therefore, if the handover to the PMIC has already been completed, the user software needs to set PWR_CTL2.DPSLP_REG_DIS to "1". If not, PWR_CTL2.DPSLP_REG_DIS is set by the `SwitchOverRegulators` API with blocking mode.

7. Call the `ConfigureRegulator` API with the following parameters:
   - Opcode = 0x15 (Call the `ConfigureRegulator` API)
   - RadjValue[6] = 0 (Not used)
   - VADJ trim value = 16
   - Vadj = 1 (Fixed)
   - UseRadj[6] = 0 (Not used)
   - UseLinReg = 0 (Internal Active Regulator is disabled after PMIC is enabled)
   - DeepSleep[6] = 1 (PMIC is enabled in DeepSleep power mode)
   - Reset Polarity = 0 (PMIC PG signal indicates power bad status with “0”)
   - Enable Polarity = 1 (PMIC is enabled by “1”)
   - Operating Mode[6] = 1 (External PMIC)
   - WaitCount = 0 (0x1FF or less)
8. Wait for the `ConfigureRegulator` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
9. When you need to set the WaitCount to a value greater than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register. If not, this process is unnecessary because it can be set by the `ConfigurRegulator` API. See the `ConfigureRegulator` API in **System Call API** for more details.

To enable the REGHC or PMIC controller with PMIC, perform these steps:

10. Call the `SwitchOverRegulators` API with the following parameters:
    - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
    - Blocking = 1 (Blocking call)
    - Select regulator = 0 (Switch over to external power supply)
    - Operating Mode[6] = 1 (External PMIC). This parameter must match the Operating Mode in the `ConfigureRegulator` API.

Note: *In this case, the `SwitchOverRegulators` API sets PWR_CTL2.DPSLP_REG_DIS to “1”. and you cannot switch back from PMIC to internal regulators. If your system requires to run the handover from PMIC to internal regulators by the `SwitchOverRegulators` API, you need to call `SwitchOverRegulators` with non-blocking call when running the handover to PMIC. See **SwitchOverRegulators API non-blocking call handling** for details.*

11. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
12. The device is now operating with PMIC. Active and DeepSleep Regulators are disabled.

DeepSleep entry and exit

Figure 27 shows the transition to DeepSleep power mode. In this case, you can transition to DeepSleep power mode without any additional action.

[6] This is invalid in CYT3D/4D series.

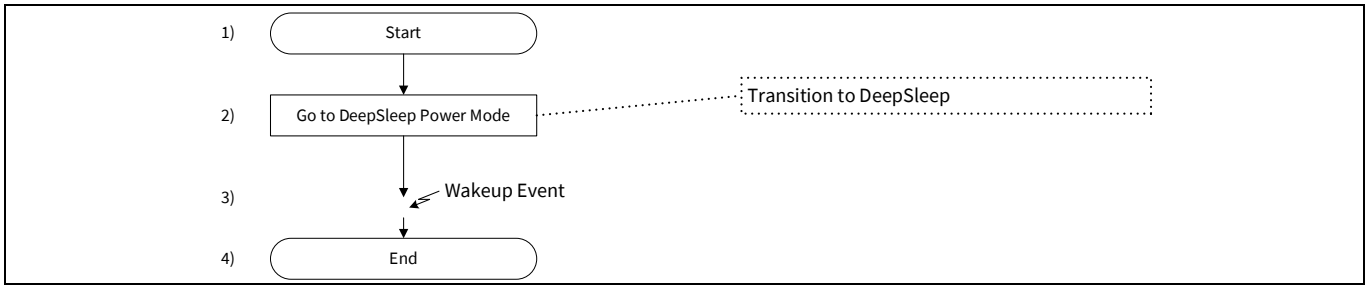

Figure 27 DeepSleep entry and exit flow (case 3)

1. The MCU is powered by PMIC.
2. Start the transition to DeepSleep power mode using the “WFI” instruction. Active Regulator remains disabled, and PMIC continues to supply power.

The MCU is powered by PMIC.

3. Due to wakeup event, the MCU transitions to Active Power mode. Active and DeepSleep Regulators remain disabled.
4. The MCU is powered by PMIC.

4.3.5 Case 4

In this case, OCD is not used in Active mode, PMIC is enabled in DeepSleep power mode (Using SwitchOverRegulators with Non-blocking Call)

4.3.5.1 Handover timing chart

In Case 4, Active Regulator is disabled (OCD is not used), and DeepSleep Regulator is enabled and PMIC supplies power during DeepSleep power mode. **Figure 28** shows an example of the handover sequence. This example includes regulator configuration, handover from internal regulators to the PMIC, and handover from PMIC to internal regulators.

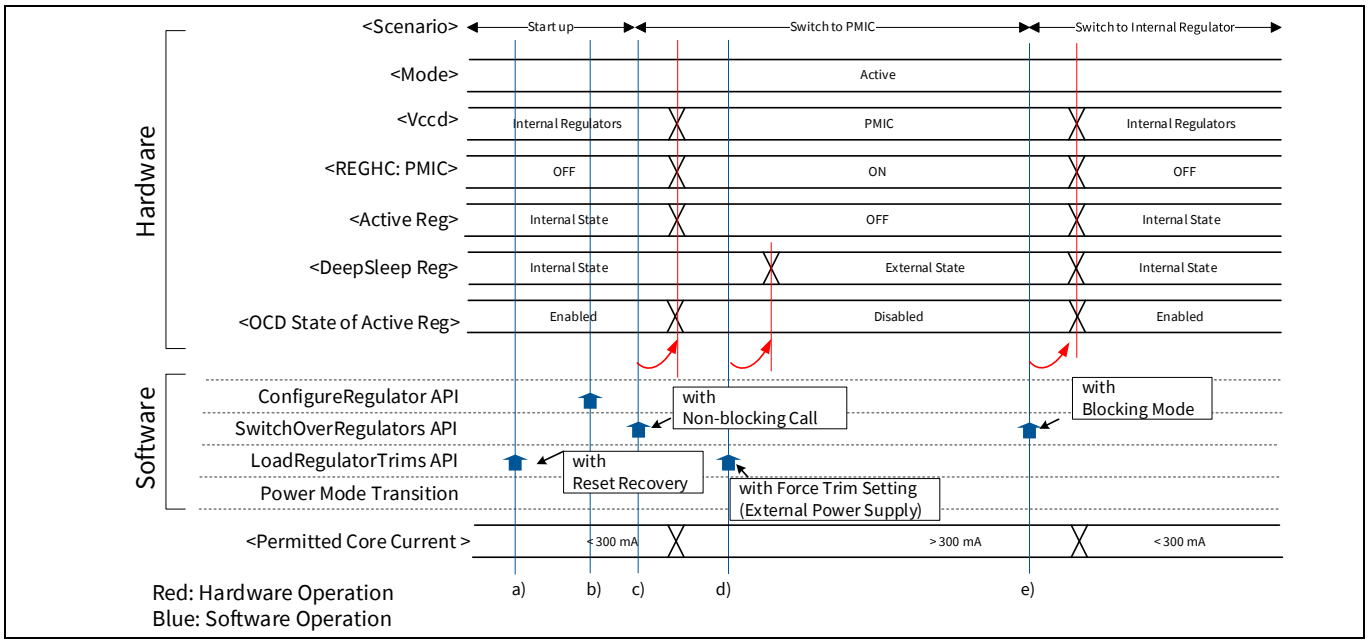

Figure 28 Handover sequence example (case 4)

PMIC (switching regulator)

a) Call the `LoadRegulatorTrims` API with reset recovery use case, if PWR_REGHC_CTL.REGHC_CONFIGURED is set to “1”. See the **reset recovery** for details.

b) Call the `ConfigureRegulator` API for REGHC or PMIC controller with PMIC configuration after start up.

c) Call the `SwitchOverRegulators` API with non-blocking call for the handover from internal regulators to PMIC.

*Note: When calling the `SwitchOverRegulators` API with non-blocking call, the API does not set PWR_CTL2.DPSLP_REG_DIS to “1”. Therefore, you can switch back to internal regulators from PMIC using the `SwitchOverRegulators` API. However, if the application software sets PWR_CTL2.DPSLP_REG_DIS to “1”, you cannot switch back to internal regulators from PMIC. See **SwitchOverRegulators API non-blocking call handling** for details.*

PMIC is enabled. Active Regulator is disabled, because the OCD feature is not requested.

d) Call the `LoadRegulatorTrims` API with Force trim setting use case for changing the external state of the regulator. The DeepSleep Regulator changes to external state.

e) Call the `SwitchOverRegulators` API for the handover from PMIC to internal regulators. PMIC is disabled. The Active and DeepSleep Regulators change to internal state.

4.3.5.2 Software flow

Handover from Internal Regulator to PMIC

Figure 29 shows the handover flow from internal regulator to PMIC.

PMIC (switching regulator)

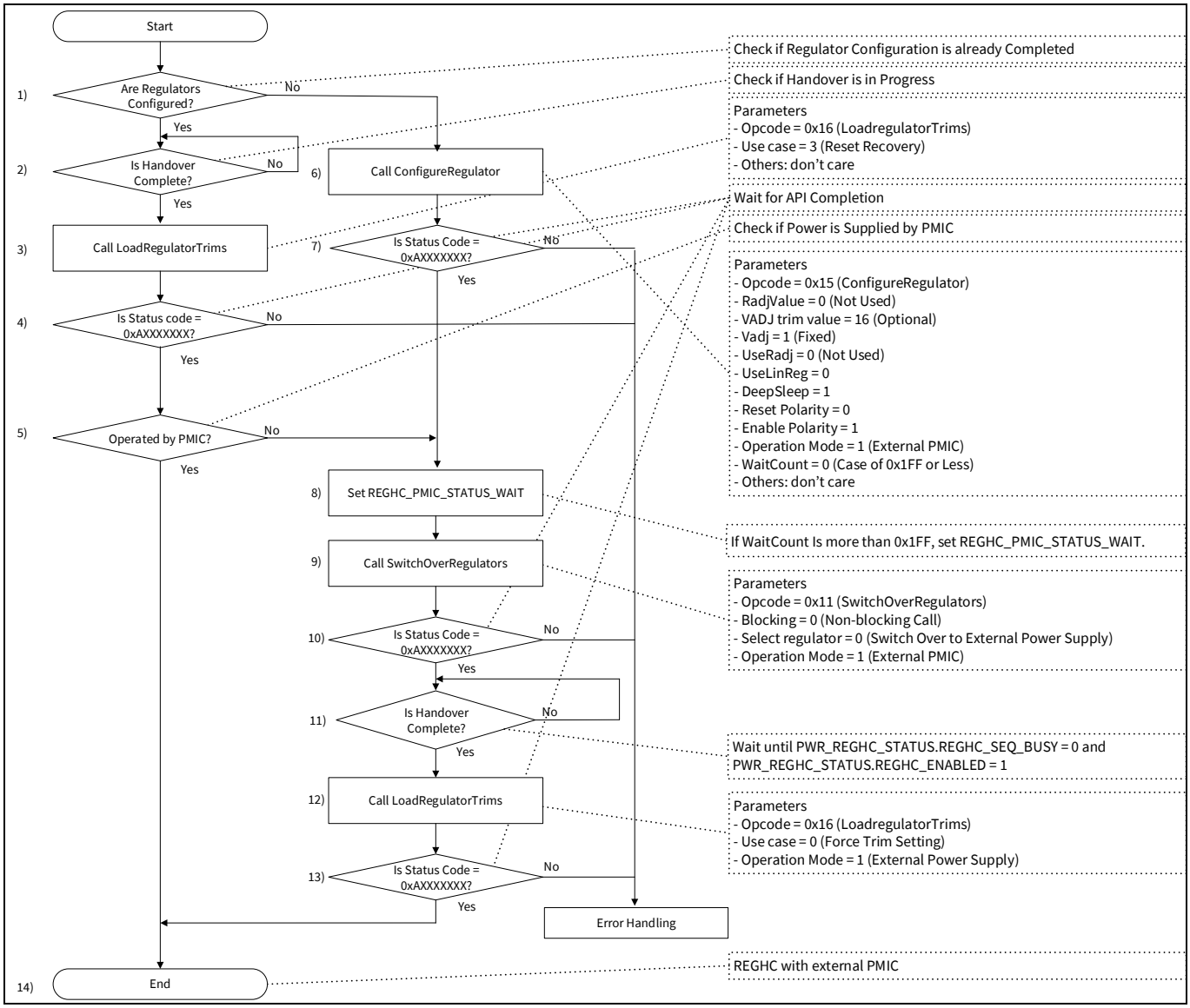

Figure 29 Handover flow from internal regulator to PMIC (case 4)

1. Check if regulator configuration is already completed. Once REGHC or PMIC controller is setup, it can be enabled without reconfiguring.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to "0", the configuration is not complete. Go to (6).

2. Wait for handover completion

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = "0" and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN

3. Call the `LoadRegulatorTrims` API with reset recovery:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 3 (Reset Recovery)
4. Wait for the completion of the `LoadRegulatorTrims` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.

PMIC (switching regulator)

5. Check if MCU operates with PMIC. When PWR_REGHC_STATUS.REGHC_ENABLED is set to “1”, the PMIC is supplying power. Go to (14). When PWR_REGHC_STATUS.REGHC_ENABLED is set to “0”, the internal regulator is supplying power. Go to (8).
6. Call the `ConfigureRegulator` API with the following parameters:
   - Opcode = 0x15 (Call the `ConfigureRegulator` API)
   - RadjValue[7] = 0 (Not used)
   - VADJ trim value = 16
   - Vadj = 1 (Fixed)
   - UseRadj[7] = 0 (Not used)
   - UseLinReg = 0 (Internal Active Regulator is disabled after PMIC is enabled)
   - DeepSleep[7] = 1 (PMIC is enabled in DeepSleep power mode)
   - Reset Polarity = 0 (PMIC PG signal indicates power bad status with “0”)
   - Enable Polarity = 1 (PMIC is enabled by “1”)
   - Operating Mode[7] = 1 (External PMIC)
   - WaitCount = 0 (0x1FF or less)
7. Wait for the `ConfigureRegulator` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
8. When you need to set the WaitCount to a value greater than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register. If not, this process is unnecessary because it can be set by the `ConfigurRegulator` API. See the `ConfigureRegulator` API in **System Call API** for more details.

To enable the REGHC or PMIC controller with PMIC, perform these steps:

9. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 0 (Non-blocking call)
   - Select regulator = 0 (Switch over to external power supply)
   - Operating Mode[7] = 1 (External PMIC) This parameter must match the Operating Mode in the `ConfigureRegulator` API

Note: *The `ConfigureRegulator` API is configured UseLinReg = 0 and DeepSleep =1, but the `SwitchOverRegulators` runs with non-blocking call. Therefore, the `SwitchOverRegulators` API does not set PWR_CTL2.DPSLP_REG_DIS to "1".*

10. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
11. Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1.
12. Call the `LoadRegulatorTrims` API with force trim setting
    - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
    - Use case = 0 (Force trim setting)
    - Operating Mode = 1 (External power supply)

This process changes internal regulator output state to external state.

[7] This is invalid in CYT3D/4D series.

PMIC (switching regulator)

13. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
14. The device is now operating with PMIC. Active Regulator is in OFF state, and DeepSleep Regulator operates in external state.

Handover from PMIC to internal regulator

Figure 30 shows the handover flow from PMIC to internal regulator.

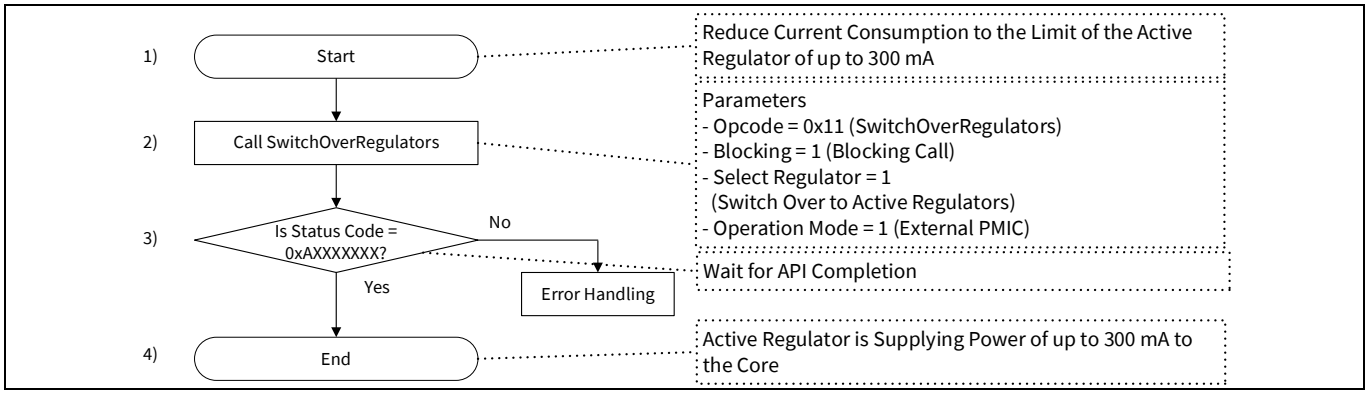

Figure 30 Handover flow from PMIC to internal regulator (case 4)

To transition from PMIC to internal regulator:

1. Reduce the current consumption of the application to the maximum possible current of Active Regulator, which is 300 mA (by configuring the appropriate application such as clock, cores, peripheral).
2. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 1 (Switch over to Active Regulator)
   - Operating Mode[8] = 1 (External PMIC)
3. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. The device is now operating with the internal regulator.

4.4 PMIC with load switch

This section considers the deployment of a load switch for the core supply rail, when decoupling between PMIC output and MCU V_{CCD} is required. See **Required PMIC specification** for the conditions.

In general, variants with load switch are not recommended. The reasons are the circuit complexity to ensure the low voltage drop by the Drain-Source resistance of load switch under all operating conditions, and the risk of sink current by the PMIC when disabling PMIC.

This section describes PMIC connection, handover timing, and handover software flow example based on the following use cases depending on the internal regulator configuration for PMIC with load switch.

Only MCUs of CYT3B/4B series support this configuration.

[8] This is invalid in CYT3D/4D series.

PMIC (switching regulator)

- **Case 1**: Load switch gate is controlled by power good signal of PMIC. OCD is used in Active mode, and PMIC is disabled in DeepSleep power mode
- **Case 2**: Load switch gate is controlled by the REGHC (EXT_PS_CTL1) pin. OCD is not used in Active mode, and PMIC is enabled in DeepSleep power mode (using SwitchOverRegulators with blocking call)
- **Case 3**: Load switch gate is controlled by the REGHC (EXT_PS_CTL1) pin. OCD in used in Active mode, and PMIC is enabled in DeepSleep power mode (using SwitchOverRegulators with non-blocking call)

4.4.1 Case 1

This section describes the solution using load switch. Load switch is required for isolating PMIC from V_{CCD} depending on the choice of PMIC topology. In this solution, the PMIC output must be sufficiently stable, when PMIC indicates a PG status.

In this case, OCD is used in Active mode, PMIC is disabled in DeepSleep power mode. The load switch gate is controlled by power good signal of PMIC.

4.4.1.1 Hardware configuration

Figure 31 shows the circuit diagram for PMIC with load switch 1. In this configuration, load switch gate is connected to the PG signal, which means that the load switch is controlled directly by the PG signal.

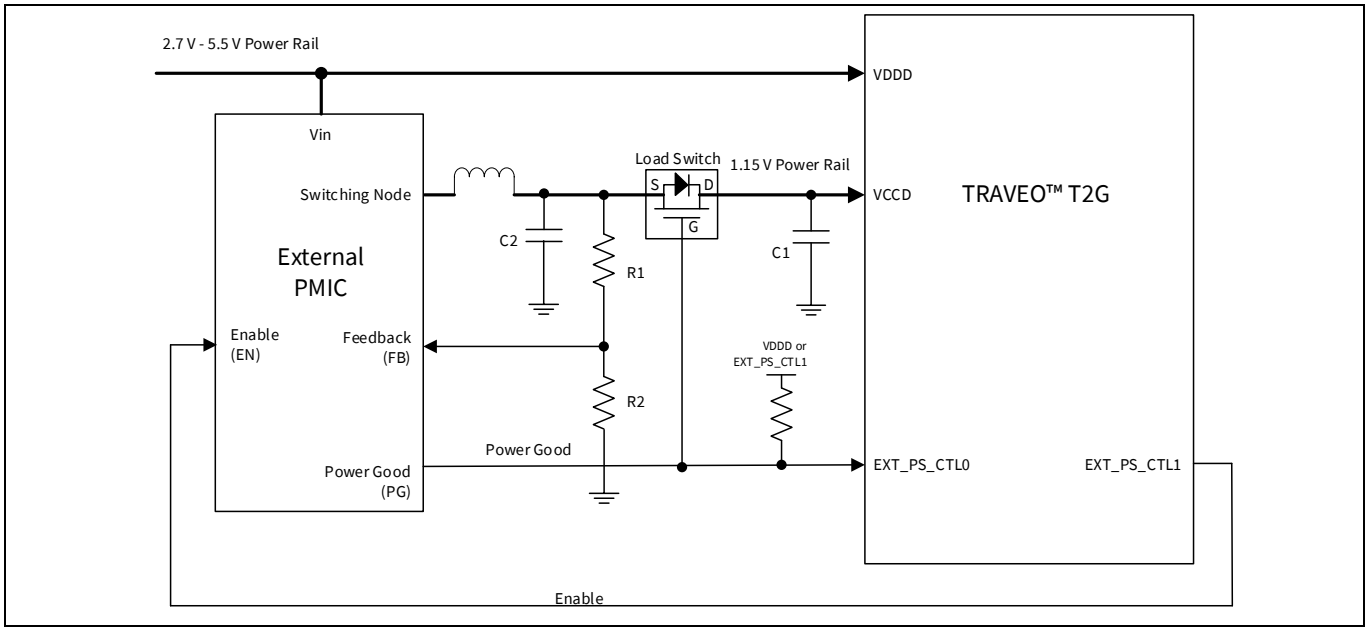

Figure 31 Circuit diagram for PMIC with load switch 1

- PMIC EN polarity is HIGH for enable. PMIC PG polarity is HIGH for PG.

  RESET_PMIC is generated when PG of PMIC goes LOW, which causes HV reset in the MCU.
- PMIC feedback signal is connected to load switch on PMIC side.
- Load switch is connected between PMIC output and V_{CCD} of the MCU.
- Gate signal of load switch and PG pin of PMIC are connected to EXT_PS_CTL0 pin of the MCU. PG pulls up to the power rail (V_{DDD}), however:

  When Vin voltage for PMIC starts to ramp up, PG terminal voltage may rise before PMIC pulls down PG because of incomplete Vin voltage and Hi-z status of PMIC PG terminal. As a result, load switch may turn ON

at an unintended timing. One of the ways to prevent the unintended turning ON is to connect the pull-up resistor of EXT_PS_CTL0 (PMIC Power Good) to EXT_PS_CTL1 (PMIC EN) instead of V_{DDD}.

- EN pin of PMIC is connected to EXT_PS_CTL1 pin of the MCU. A pull-down resistor on PCB or in PMIC is required to disable PMIC during XRES and Hibernate.
- If EN pin does not have the internal pull-down resistor, an external pull-down resistor must be placed to keep PMIC disabled during POR.
- V_{CCD} capacitor (C1) depends on the MCU requirement. (See the device datasheet **[1]** for the capacitance)
- Output voltage setting resistors (R1, R2) and output capacitor (C2) depend on the selected PMIC.

The subsequent software flow assumes that the register setting of PMIC enable polarity is “1” and PG status polarity is “1”.

Enable Polarity in the `ConfigureRegulator` API specifies the polarity of the enable signal to PMIC, and Reset Polarity in the `ConfigureRegulator` API specifies the polarity of power good state from PMIC. Note that Reset polarity is set to PG de-asserted polarity. These settings depend on PMIC specifications.

In this use case, internal regulator configuration uses OCD, and PMIC is disabled in DeepSleep power mode.

4.4.1.2 Handover timing chart

Figure 32 shows an example of the handover sequence with load switch 1. This example includes regulator configuration, handover from internal regulators to PMIC, transition to DeepSleep power mode, wake up from Deep Sleep power mode, and handover from PMIC to internal regulators. This section assumes the register setting of PMIC enable polarity is “1” and PG status polarity is “1”.

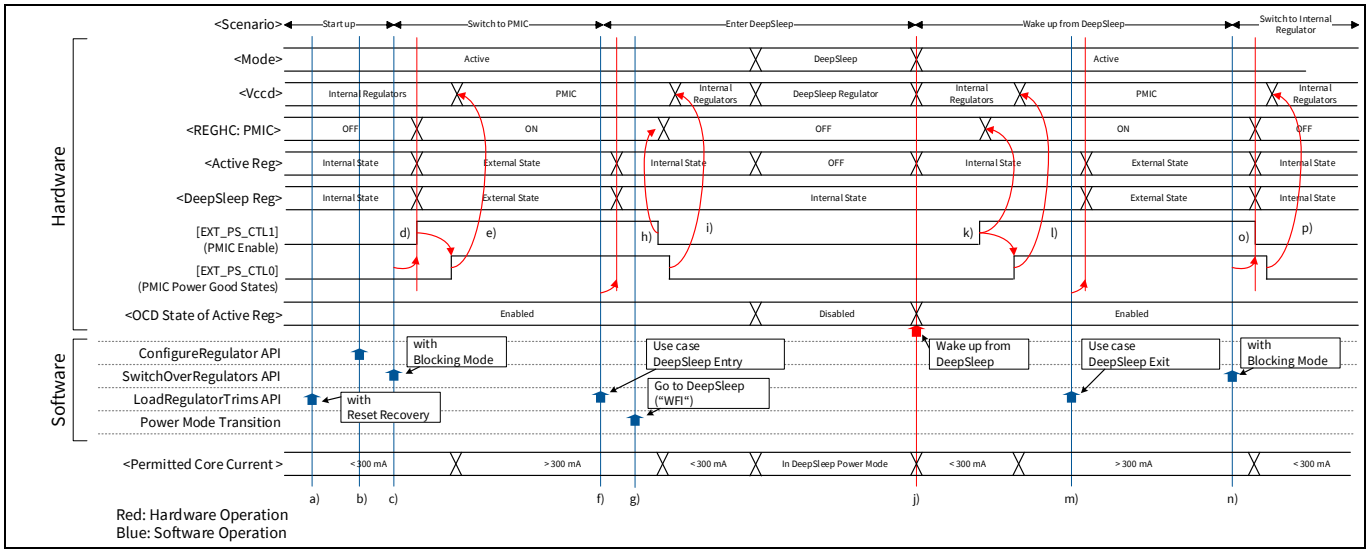

Figure 32 Handover sequence example of PMIC with load switch 1

a) Call the `LoadRegulatorTrims` API with reset recovery use case, if PWR_REGHC_CTL.REGHC_CONFIGURED is set to “1”. See the **reset recovery** for details.

b) Call the `ConfigureRegulator` API for REGHC with PMIC configuration after start up.

c) Call the `SwitchOverRegulators` API for handover to PMIC from internal regulators

Active Regulator is enabled to use OCD, and changed to external state. The DeepSleep Regulator changes to external state. EXT_PS_CTL1 outputs PMIC enable signal by the `SwitchOverRegulators` API, and PMIC gets enabled.

PMIC (switching regulator)

d) EXT_PS_CTL0 is in power good state after PMIC is ready.

e) Load switch turns ON when EXT_PS_CTL0 is in power good state. PMIC supplies power to the MCU.

f) Call the `LoadRegulatorTrims` API with DeepSleep Entry use case for changing the state of the internal regulators. The Active and DeepSleep Regulators change to internal state

g) Reduce the current consumption to be within the limit of the Active Regulator, which is 300 mA. Then, you can execute the transition to DeepSleep power mode using the “WFI” instruction.

h) Transition to DeepSleep triggers the handover to internal regulators. EXT_PS_CTL1 outputs PMIC disable signal, and EXT_PS_CTL0 is not in power good states.

i) Load switch is OFF when EXT_PS_CTL0 is not in power good state. Internal regulators supply power to the MCU.

j) A power mode transition from DeepSleep to Active happens after the wake-up event, and then the Active Regulator starts with internal state.

k) EXT_PS_CTL1 outputs PMIC enable signal, and PMIC is enabled. EXT_PS_CTL0 outputs PG state signal after PMIC is ready.

l) Load switch is ON when EXT_PS_CTL0 is in power good state. PMIC supplies power to the MCU.

m) Call the `LoadRegulatorTrims` API with DeepSleep Exit use case for changing the external state of the regulator

n) Call the `SwitchOverRegulators` API for the handover from PMIC to internal regulators.

Internal regulators change to internal state.

o) EXT_PS_CTL1 outputs PMIC disable signal, and EXT_PS_CTL0 is not power good states.

p) Load switch is OFF when EXT_PS_CTL0 is not in power good state. Internal regulators supply power to the MCU.

4.4.1.3 Software flow

Handover from internal regulator to PMIC

Figure 33 shows the handover from internal regulator to PMIC.

PMIC (switching regulator)

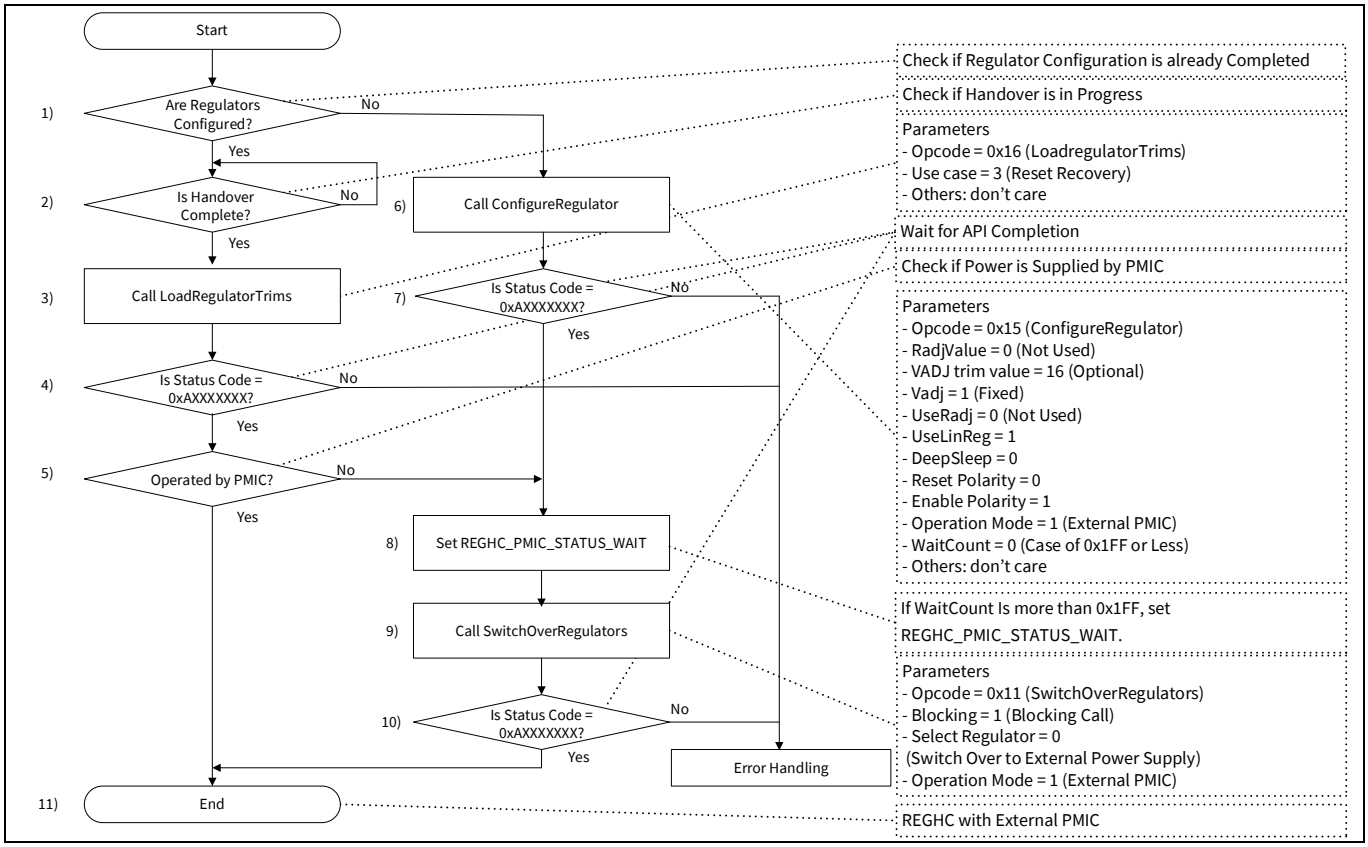

Figure 33 Handover flow from internal regulator to PMIC with load switch 1

1. Check if regulator configuration is already completed. Once REGHC is setup, it can be enabled without configuring again.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to "0", the configuration is not complete. Go to (6).

2. Wait for the handover completion.

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = "0" and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN

3. Call the `LoadRegulatorTrims` API with reset recovery:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 3 (Reset Recovery)
4. Wait for the completion of the `LoadRegulatorTrims` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.
5. Check if MCU operates with PMIC. When PWR_REGHC_STATUS.REGHC_ENABLED is set to "1", the PMIC is supplying power. Go to (11). When PWR_REGHC_STATUS.REGHC_ENABLED is set to "0", the internal regulator is supplying power. Go to (8).
6. Call the `ConfigureRegulator` API with the following parameters:
   - Opcode = 0x15 (Call the `ConfigureRegulator` API)
   - RadjValue = 0 (Not used)
   - VADJ trim value = 16
   - Vadj = 1 (Fixed)
   - UseRadj = 0 (Not used)

PMIC (switching regulator)

- UseLinReg = 1 (Internal Active Regulator is kept as enabled after PMIC is enabled)
- DeepSleep = 0 (PMIC is disabled in DeepSleep power mode)
- Reset Polarity = 0 (PMIC PG signal indicates power bad status with “0”)
- Enable Polarity = 1 (PMIC is enabled by “1”)
- Operating Mode = 1 (External PMIC)
- WaitCount = 0 (0x1FF or less)

7. Wait for the `ConfigureRegulator` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
8. When you need to set the WaitCount to a value greater than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register. If not, this process is unnecessary because it can be set by the `ConfigurRegulator` API. See the `ConfigureRegulator` API in **System Call API** for more details.

To enable the REGHC with PMIC, perform these steps:

9. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 0 (Switch over to external power supply)
   - Operating Mode = 1 (External PMIC) This parameter must match the Operating Mode in the `ConfigureRegulator` API
10. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
11. The device is now operating with PMIC. Active and DeepSleep Regulators operate in external state.

DeepSleep entry and exit

Figure 34 shows the transition to DeepSleep power mode in PMIC with load switch 1.

PMIC (switching regulator)

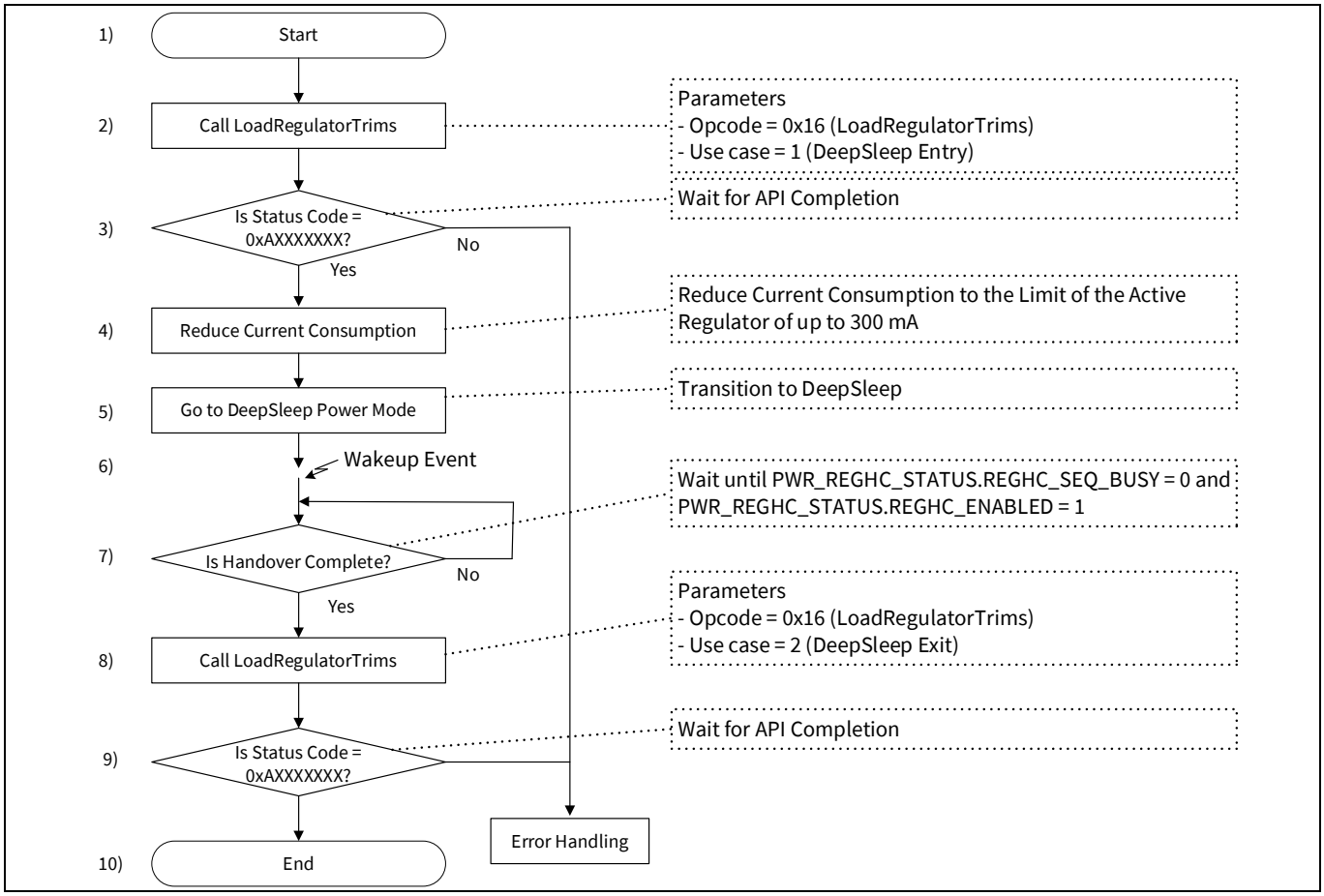

Figure 34 DeepSleep entry and exit flow in PMIC with load switch 1

1. The MCU is powered by PMIC.
2. Call the `LoadRegulatorTrims` API with the following parameters before transitioning to DeepSleep power mode:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 1 (DeepSleep Entry)
3. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. Reduce the current consumption to be within the limit of Active Regulator, which is 300 mA.
5. Start the transition to DeepSleep power mode using the “WFI” instruction. Perform handover to internal regulator, and PMIC is disabled. Then, Active Regulator is disabled, and DeepSleep Regulator supplies power during DeepSleep power mode.

The MCU is powered by DeepSleep Regulator.

6. Due to wakeup event, the MCU transitions to Active Power mode. Then, the hardware enables PMIC.
7. Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1
8. Call the `LoadRegulatorTrims` API with the following parameters. Active and DeepSleep Regulators are set to external state:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 2 (DeepSleep Exit)

PMIC (switching regulator)

9. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
10. The MCU is powered by PMIC.

Handover to internal regulator from PMIC

Figure 35 shows handover from PMIC to internal regulator.

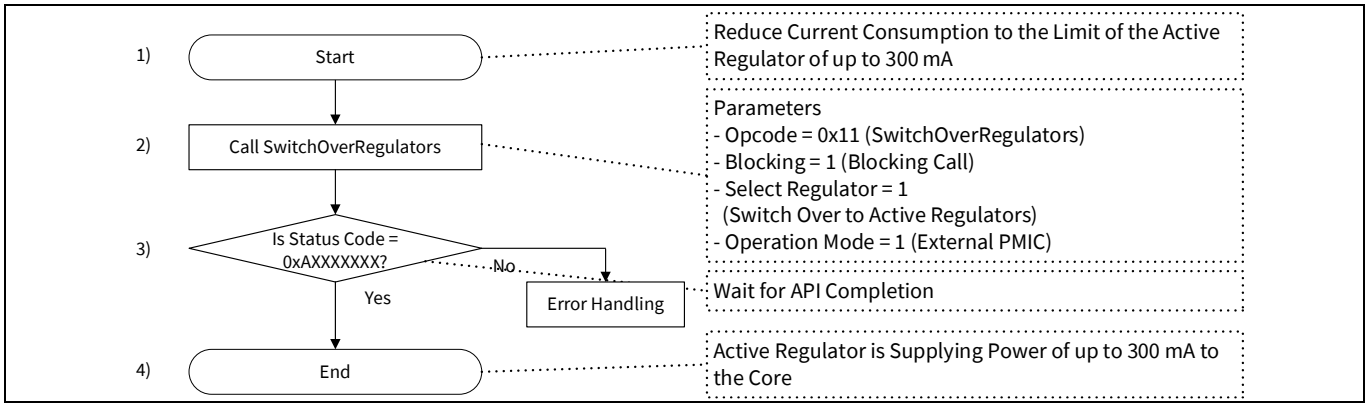

Figure 35 Handover flow from PMIC with load switch 1 to internal regulator

To transition from REGHC with PMIC to internal regulator:

1. Reduce the current consumption to be within the limit of Active Regulator, which is 300 mA.
2. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 1 (Blocking call)
   - Select regulator = 1 (Switch over to Active Regulator)
   - Operating Mode = 1 (External PMIC)
3. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. The device is now operating with internal regulator.

4.4.2 Case 2

In this case, OCD is not used in Active mode, PMIC is enabled in DeepSleep power mode. (Using SwitchOverRegulators with Blocking Call) The load switch gate is controlled by REGHC pin.

Load switch is required for isolating the PMIC from V_{CCD} depending on the choice of PMIC topology. See **Required PMIC specification** for more details. REGHC with PMIC configuration does not initialize by LV resets (Software, FAULT, WDT, MCWDT, CSV) but by HV resets. (XRES, POR, BOD, OVD, OCD, HIBERNATE wakeup) See the “Reset System” in the architecture TRM **[2]** for reset sources.

4.4.2.1 Hardware configuration

Figure 36 shows the circuit diagram for PMIC with load switch 2.

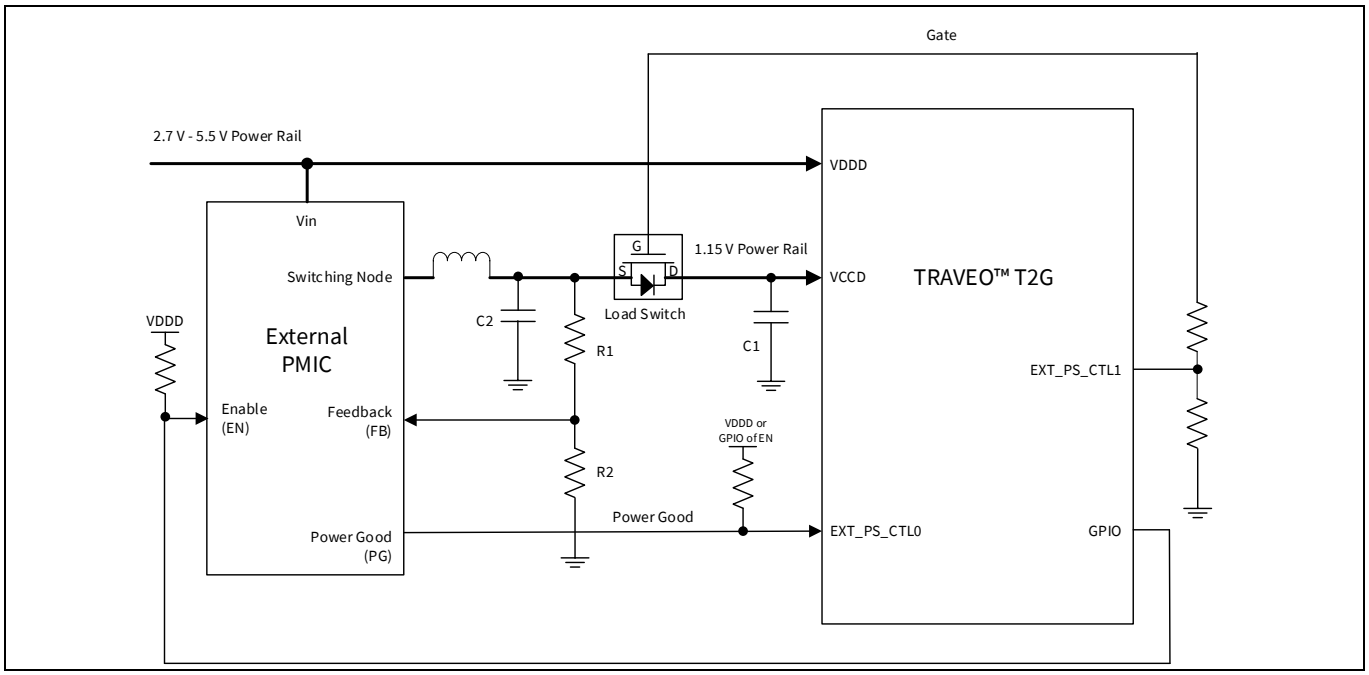

Figure 36 Circuit diagram for PMIC with load switch 2

- PMIC EN polarity is HIGH for enable. PMIC PG polarity is HIGH for PG.

  RESET_PMIC is generated when PG of PMIC goes LOW. This is HV reset. HV reset is applied and the microcomputer including REGHC with PMIC is initialized.
- PMIC feedback signal is connected to load switch on PMIC side.
- Load switch is connected between PMIC output and V_{CCD} of MCU
- Gate signal of load switch is connected to EXT_PS_CTL1 pin of MCU.
- EN pin of PMIC is connected to GPIO of MCU. To maintain correct operation during DeepSleep mode, use GPIO in the power domain that keeps the power even during DeepSleep mode.
- A pull-up resistor is connected to GPIO (EN) signal to enable the PMIC during LV reset.
- A resistor to restrict current of load switch gate charge to less than 1 mA is put between EXT_PS_CTL1 and gate of the load switch.
- V_{CCD} capacitor (C1) depends on the MCU requirement (See the device datasheet **[1]** for the capacitance).
- Output voltage setting resistors (R1, R2) and output capacitor (C2) depends on the selected PMIC.

Note: In this use case, the current consumption increases during DeepSleep. If this is acceptable, you can use this use case.

The following software flow assumes that the register setting of PMIC enable polarity is “1” and PG status polarity is “1”.

Enable Polarity in the `ConfigureRegulator` API specifies the polarity of the enable signal to PMIC, and Reset Polarity in the `ConfigureRegulator` API specifies the polarity of PG state from PMIC. Note that Reset polarity is set to power good de-asserted polarity. These settings depend on PMIC specifications.

PMIC (switching regulator)

In this use case, internal regulator configuration does not use OCD, and PMIC is enabled in DeepSleep power mode.

4.4.2.2 Handover timing chart for PMIC

Figure 37 shows an example of the handover sequence with load switch 2. This example includes regulators configuration, handover from internal regulators to the PMIC, transition to DeepSleep power mode, and wake up from DeepSleep power mode. In this configuration, the API cannot be used to switch back from the PMIC to the internal regulator. To switch back to the internal regulator, you need to reset the entire chip (HV reset). If your system requires to run the handover from PMIC to internal regulators by the `SwitchOverRegulators` API, see **Case 3** for details.

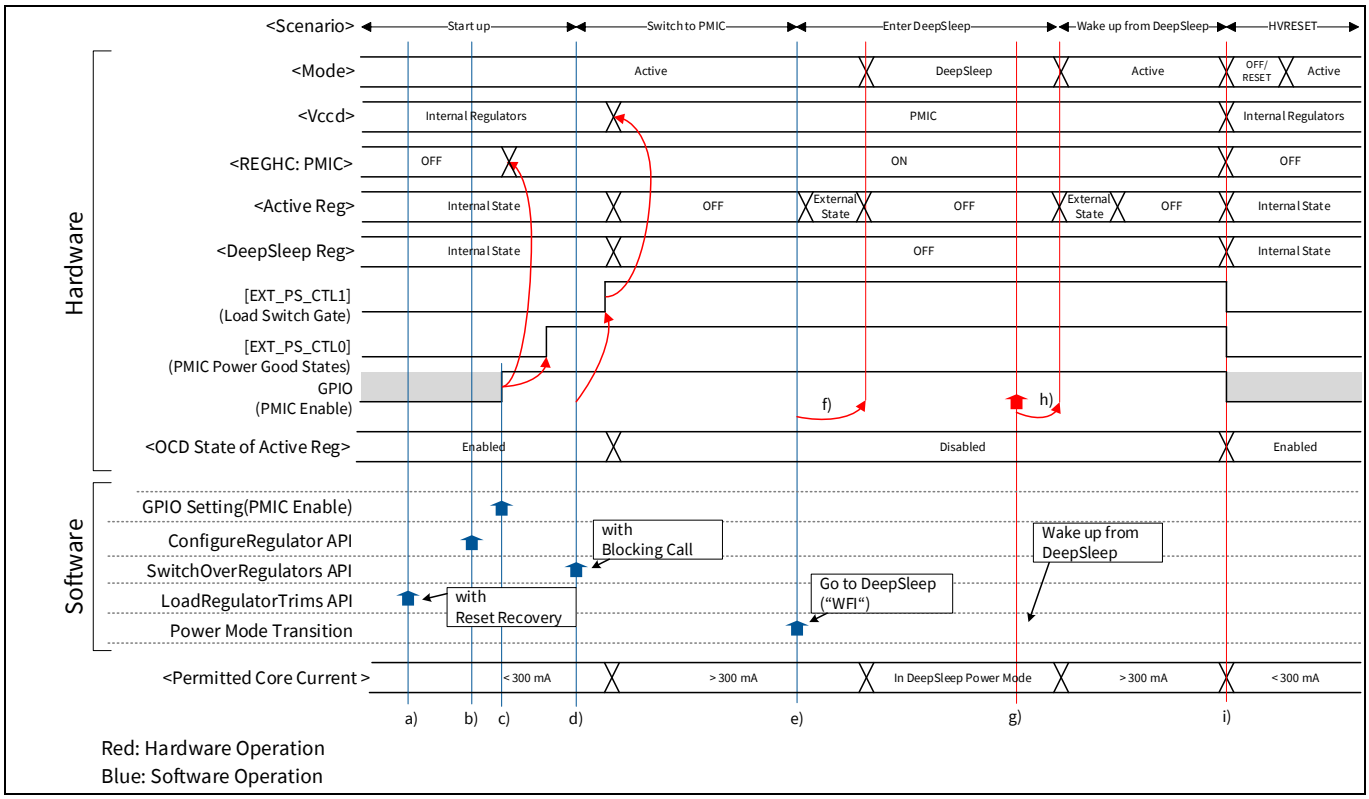

Figure 37 Handover sequence example of PMIC with load switch 2 (case 2)

The initial state of GPIO (PMIC Enable) depends on the pin termination.

a) Call the `LoadRegulatorTrims` API with reset recovery use case, if PWR_REGHC_CTL.REGHC_CONFIGURED is set to "1". See the **reset recovery** for details.

b) Call the `ConfigureRegulator` API for REGHC with PMIC configuration after starting up.

c) User software sets GPIO to "1" to enable PMIC. Then, PMIC is enabled, and EXT_PS_CTL0 is in power good state after PMIC ready. Note that PMIC is enabled here, but no handover has been performed.

d) Call the `SwitchOverRegulators` API with blocking call for the handover from internal regulators to PMIC.

   EXT_PS_CTL1 turns ON the load switch using the `SwitchOverRegulators` API, then the MCU is supplies power from PMIC. Active Regulator and DeepSleep Regulator are disabled.

   If your system requires to run the handover from PMIC to internal regulators using the `SwitchOverRegulators` API, you need to call the `SwitchOverRegulators` API with non-blocking call when running the handover to PMIC.

PMIC (switching regulator)

e) You can execute the transition to DeepSleep power mode using the “WFI” instruction. In this case, the system can transition to DeepSleep power mode directly without reducing current consumption.

When transitioning to DeepSleep power mode, PMIC is still enabled, and Active and DeepSleep Regulators are disabled.

f) MCU transitions to DeepSleep power mode.

g) A power mode transition from DeepSleep to Active power mode happens after the wake-up event.

h) Active power mode transition is completed, PMIC will continue to supply power. Active and DeepSleep Regulators are disabled.

i) When HV reset occurs, power system is initialized. Therefore, MCU starts up with Active Regulator after reset is released.

4.4.2.3 Software flow

Transitioning from internal regulator to PMIC

Figure 38 shows the transition from internal regulator to PMIC.

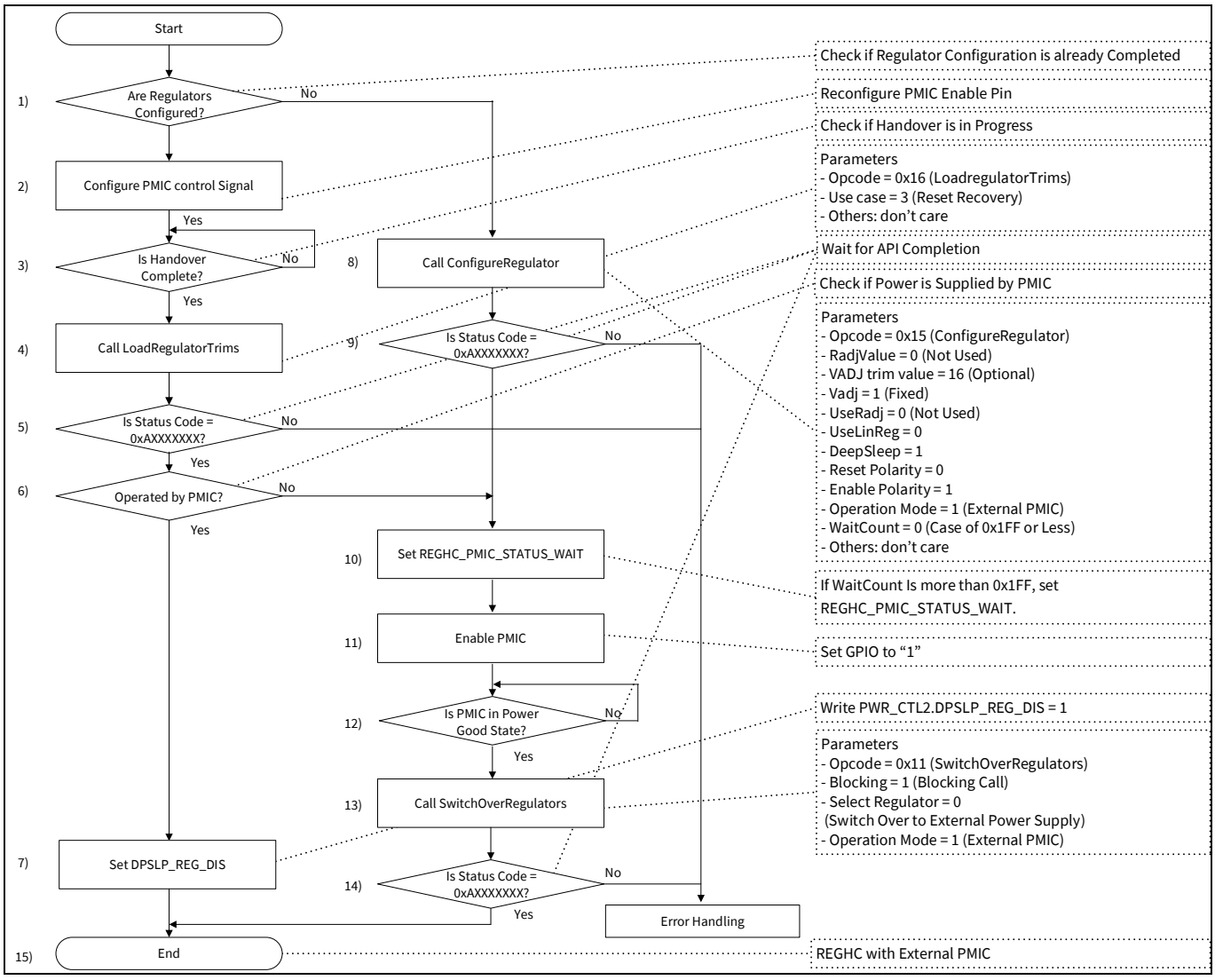

Figure 38 Handover flow from internal regulator to PMIC with load switch 2 (case 2)

PMIC (switching regulator)

1. Check if regulator configuration is already completed. Once REGHC is setup, it can be enabled without configuring again.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to “0”, the configuration is not complete. Go to (8).

2. Configure GPIO for PMIC enable control. In this case, PMIC enable state before LV reset is reconfigured. The following is an example of GPIO reconfiguration:

   a) Write ~ (PWR_REGHC_CTL2.REGHC_EN ^ PWR_REGHC_CTL.REGHC_PMIC_CTL_POLARITY) to GPIO data register.

   b) Configure to GPIO output.

3. Wait for handover completion

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = “0” and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN

4. Call the `LoadRegulatorTrims` API with reset recovery:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 3 (Reset Recovery)
5. Wait for the completion of the `LoadRegulatorTrims` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.
6. Check if MCU operates with PMIC. When PWR_REGHC_STATUS.REGHC_ENABLED is set to “1”, the PMIC is supplying power. Go to (7). When PWR_REGHC_STATUS.REGHC_ENABLED is set to “0”, the internal regulator is supplying power. Go to (12).
7. Set PWR_CTL2.DPSLP_REG_DIS to “1”.

   In case 2, PMIC is enabled in DeepSleep power mode. Therefore, if the handover to the PMIC has already been completed, the user software needs to set PWR_CTL2.DPSLP_REG_DIS to "1". If not, PWR_CTL2.DPSLP_REG_DIS is set by the `SwitchOverRegulators` API with blocking mode.

8. Call the `ConfigureRegulator` API with the following parameters:
   - Opcode = 0x15 (Call the `ConfigureRegulator` API)
   - RadjValue = 0 (Not used)
   - VADJ trim value = 16
   - Vadj = 1 (Fixed)
   - UseRadj = 0 (Not used)
   - UseLinReg = 0 (Internal Active Regulator is disabled after PMIC is enabled)
   - DeepSleep = 1 (PMIC is enabled in DeepSleep power mode)
   - Reset Polarity = 0 (PMIC PG signal indicates power bad status with “0”)
   - Enable Polarity = 1 (PMIC is enabled by “1”)
   - Operating Mode = 1 (External PMIC)
   - WaitCount = 0 (0x1FF or less)

Note: *In this configuration, a resistor to restrict current is placed between EXT_PS_CTL1 (PMIC enable signal output) and load switch. The load switch may not turn ON immediately due to this resistor and capacitance of the load switch. Therefore, the handover completion need to wait for the load switch to completely turn ON. You can use a hardware timer to wait for the load switch to turn ON. This wait time can be configured using WaitCount in the `ConfigureRegulator` API. See **Load switch** for calculating the required wait time.*

PMIC (switching regulator)

9. Wait for the `ConfigureRegulator` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
10. When you need to set the WaitCount to a value greater than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register. If not, this process is unnecessary because it can be set by the `ConfigurRegulator` API. See the `ConfigureRegulator` API in **System Call API** for more details.
11. Set GPIO to "1" for enabling PMIC. Then, PMIC is enabled.
12. Wait until PWR_REGHC_STATUS.REGHC_PMIC_STATUS_OK = 1. It means waiting for the PMIC to become a power good state. If PMIC indicates PG state before the PMIC output stabilizes sufficiently, software can wait for the PMIC output to stabilize.

To enable the REGHC with PMIC, perform these steps:

13. Call the `SwitchOverRegulators` API with the following parameters:
    - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
    - Blocking = 1 (Blocking call)
    - Select regulator = 0 (Switch over to external power supply)
    - Operating Mode = 1 (External PMIC) This parameter must match the Operating Mode in the `ConfigureRegulator` API

Note: If your system requires to run the handover from PMIC to internal regulators using the `SwitchOverRegulators` API, you need to call the `SwitchOverRegulators` API with non-blocking call when running handover to PMIC.

14. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
15. The device is now operating with PMIC. Active and DeepSleep Regulators are disabled.

DeepSleep entry and exit

Figure 39 shows the transition to DeepSleep power mode in PMIC with load switch 2. In this case, you can transition to DeepSleep power mode without any additional action.

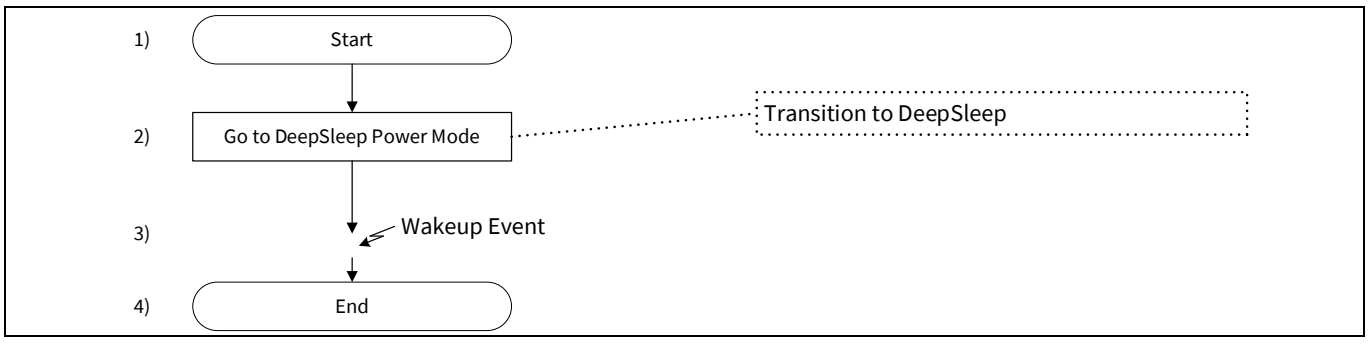

Figure 39 DeepSleep entry and exit flow in PMIC with load switch 2 (case 2)

1. The MCU is powered by PMIC.
2. Start the transition to DeepSleep power mode using the "WFI" instruction. Active Regulator is disabled, and PMIC continues to supply power.
3. Due to wakeup event, the MCU transitions to Active power mode. Active Regulator changes to external state.

4. The MCU is powered by PMIC.

4.4.3 Case 3

In this case, OCD is not used in Active mode, PMIC is enabled in DeepSleep power mode (Using SwitchOverRegulators with Non-blocking Call) The load switch gate is controlled by REGHC pin

The circuit diagram of this case is the same as **Figure 36**.

Figure 40 shows an example of the handover sequence with load switch 2. This example includes regulators configuration, handover from internal regulators to the PMIC, and handover from PMIC to internal regulators.

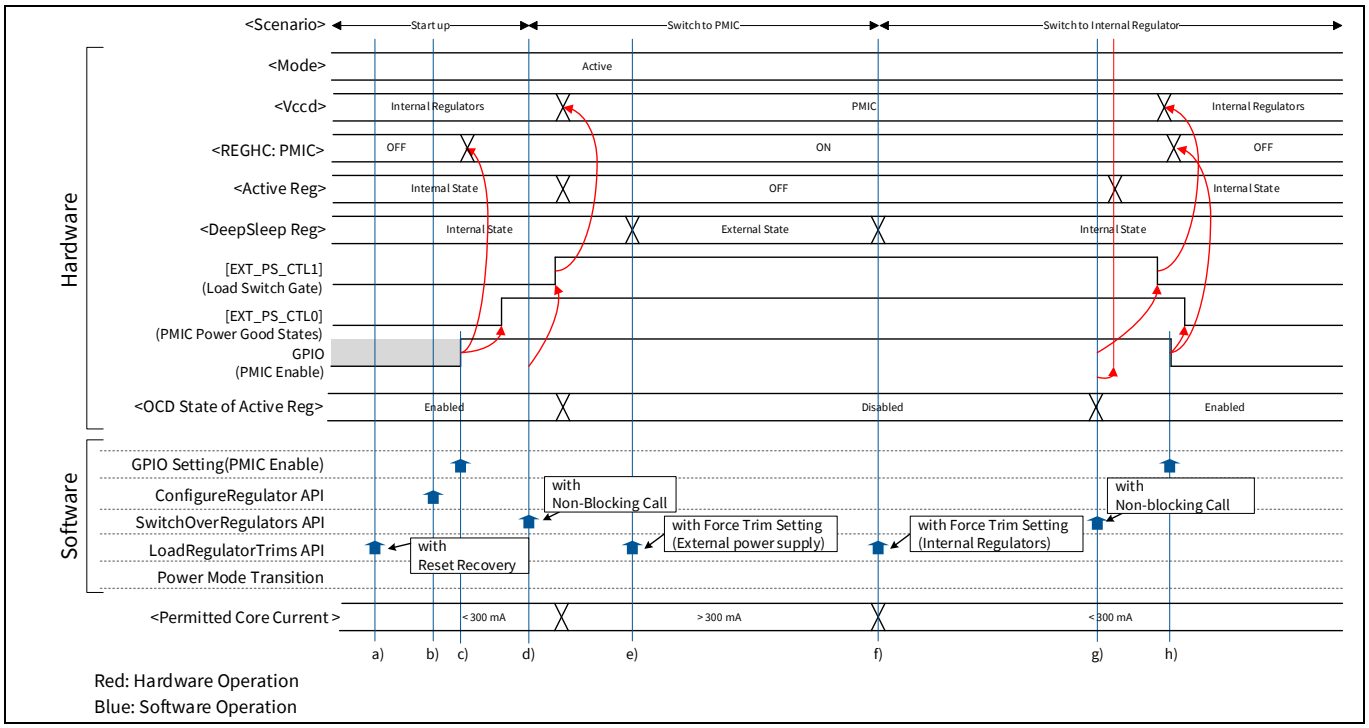

Figure 40 Handover sequence example of PMIC with load switch 2 (case 3)

The initial state of GPIO (PMIC Enable) depends on the pin termination.

a) Call the `LoadRegulatorTrims` API with reset recovery use case, if PWR_REGHC_CTL.REGHC_CONFIGURED is set to “1”. See the **reset recovery** for details.

b) Call the `ConfigureRegulator` API for REGHC with PMIC configuration after starting up.

c) User software sets GPIO to “1” to enable PMIC. Then, PMIC is enabled, and EXT_PS_CTL0 is in power good state after PMIC is ready. Note that PMIC is enabled here, but no handover has been performed.

d) Call the `SwitchOverRegulators` API with non-blocking call for the handover from Internal regulators to PMIC.

Note: When calling the `SwitchOverRegulators` API with non-blocking call, the API does not set the PWR_CTL2.DPSLP_REG_DIS to “1”. Therefore, you can switch back to internal regulators from PMIC using the `SwitchOverRegulators` API. However, if the application software sets the PWR_CTL2.DPSLP_REG_DIS to “1”, you cannot switch back to internal regulators from PMIC.

PMIC is enabled. Active Regulator is disabled, because the OCD feature is not requested.

e) Call the `LoadRegulatorTrims` API with Force trim setting use case and External Power Supply operating mode for changing the Internal Regulator state. The DeepSleep Regulator changes to external state.

f) Call the `LoadRegulatorTrims` API with Force trim setting use case and internal regulators operating mode for changing internal regulator state. The DeepSleep Regulator changes to internal state.

g) Call the `SwitchOverRegulators` API with non-blocking call for the handover from PMIC to internal regulators. Then, EXT_PS_CTL1 turns OFF the load switch. User software must wait for handover completion.

h) User software sets GPIO to “0” to disable PMIC. Then, PMIC is disabled and EXT_PS_CTL0 is in power good de-asserted state. PMIC is disabled.

4.4.3.1 Software flow

Transitioning from internal regulator to PMIC

Figure 41 shows the transition from internal regulator to PMIC.

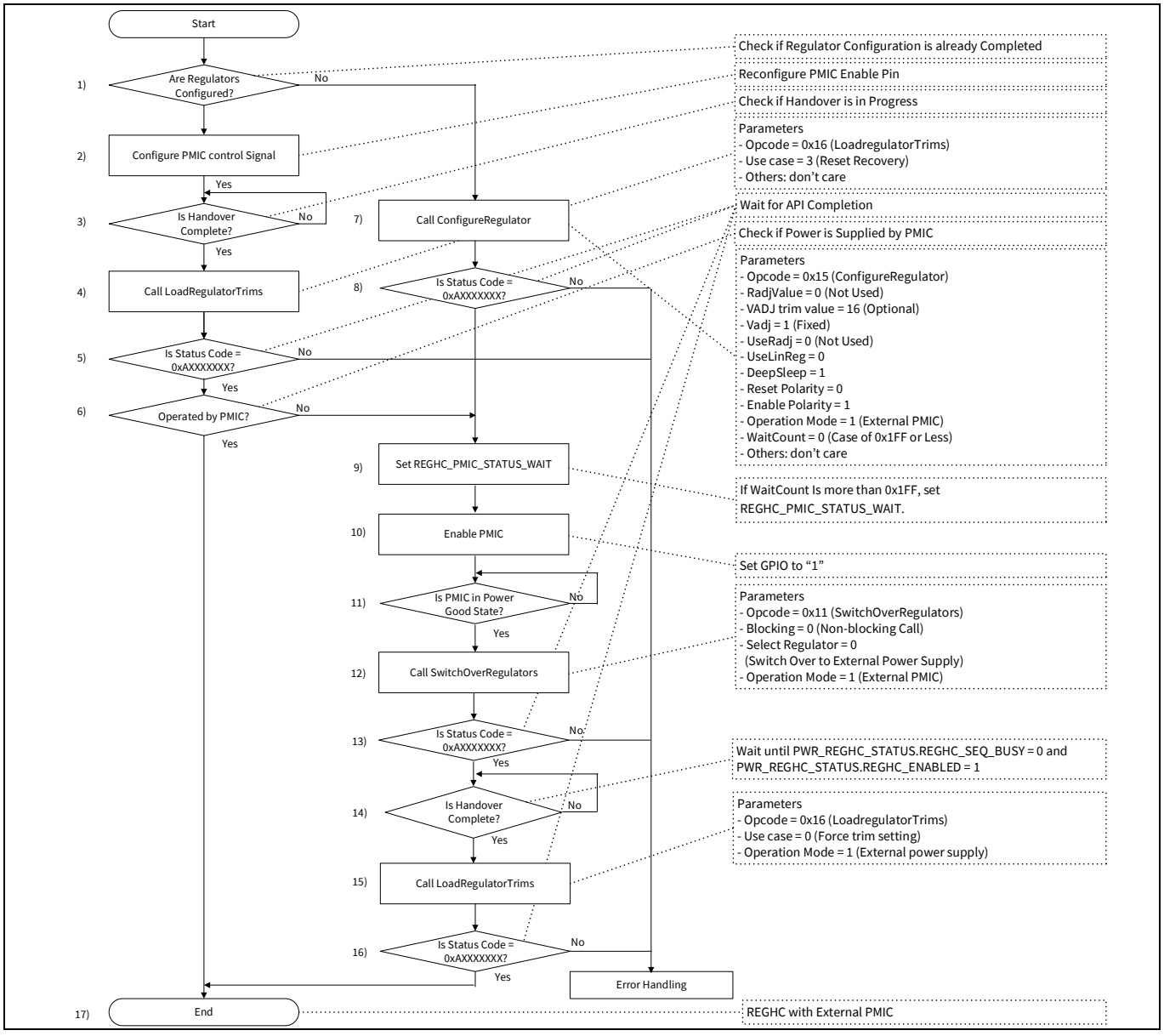

Figure 41 Handover flow from internal regulator to PMIC with load switch 2 (case 3)

PMIC (switching regulator)

1. Check if regulator configuration is already completed. Once REGHC is setup, it can be enabled without reconfiguring.

   When PWR_REGHC_CTL.REGHC_CONFIGURED is set to “0”, the configuration is not complete. Go to (7).

2. Configure GPIO for PMIC enable control. In this case, PMIC enable state before LV reset is reconfigured. The following is an example of GPIO reconfiguration:

   a) Write ~ (PWR_REGHC_CTL2.REGHC_EN ^ PWR_REGHC_CTL.REGHC_PMIC_CTL_POLARITY) to GPIO data register.

   b) Configure to GPIO output.

3. Wait for handover completion:

   Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = “0” and PWR_REGHC_STATUS.REGHC_ENABLED = PWR_REGHC_CTL2.REGHC_EN

4. Call the `LoadRegulatorTrims` API with reset recovery
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 3 (Reset Recovery)
5. Wait for the completion of the `LoadRegulatorTrims` API. If the status code is other than 0xAXXXXXXX, perform appropriate error handling according to your system.
6. Check if MCU operates with PMIC. When PWR_REGHC_STATUS.REGHC_ENABLED is set to “1”, the PMIC is supplying power. Go to (17). When PWR_REGHC_STATUS.REGHC_ENABLED is set to “0”, the internal regulator is supplying power. Go to (9).
7. Call the `ConfigureRegulator` API with the following parameters:
   - Opcode = 0x15 (Call the `ConfigureRegulator` API)
   - RadjValue = 0 (Not used)
   - VADJ trim value = 16
   - Vadj = 1 (Fixed)
   - UseRadj = 0 (Not used)
   - UseLinReg = 0 (Internal Active Regulator is disabled after PMIC is enabled)
   - DeepSleep =1 (PMIC is enabled in DeepSleep power mode)
   - Reset Polarity = 0 (PMIC PG signal indicates power bad status with “0”)
   - Enable Polarity = 1 (PMIC is enabled by “1”)
   - Operating Mode = 1 (External PMIC)
   - WaitCount = 0 (0x1FF or less)

Note: *In this configuration, a resistor to restrict current is placed between EXT_PS_CTL1 (PMIC enable signal output) and load switch. The load switch may not turn ON immediately due to this resistor and capacitance of the load switch. Therefore, the handover completion need to wait for the load switch to completely turn ON. You can use a hardware timer to wait for the load switch to turn ON. This wait time can be configured using WaitCount in the `ConfigureRegulator` API. See Load Switch for calculating the required wait time. See **Load switch** for calculating the required wait time.*

8. Wait for the `ConfigureRegulator` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
9. When you need to set the WaitCount to a value greater than 0x1FF, set the PWR_REGHC_CTL.REGHC_PMIC_STATUS_WAIT register. If not, this process is unnecessary because it can be

PMIC (switching regulator)

set by the `ConfigurRegulator` API. See the `ConfigureRegulator` API in **System Call API** for more details.

10. Set GPIO to “1” for enabling PMIC. Then, PMIC is enabled.
11. Wait until PWR_REGHC_STATUS.REGHC_PMIC_STATUS_OK = 1. It means waiting for the PMIC to become a power good state. If PMIC indicates PG state before the PMIC output stabilizes sufficiently, software can wait for the PMIC output to stabilize.

To enable the REGHC with PMIC, perform these steps:

12. Call the `SwitchOverRegulators` API with the following parameters:
    - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
    - Blocking = 0 (Non-blocking call)
    - Select regulator = 0 (Switch over to external power supply)
    - Operating Mode = 1 (External PMIC) This parameter must match the Operating Mode in the `ConfigureRegulator` API

Note: *The `SwitchOverRegulators` API with non-blocking call does not set PWR_CTL2.DPSLP_REG_DIS to "1". See **SwitchOverRegulators API non-blocking call handling** for details.*

13. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
14. Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 1.
15. Call the `LoadRegulatorTrims` API with force trim setting:
    - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
    - Use case = 0 (Force trim setting)
    - Operating Mode = 1 (External power supply)

This process changes internal regulator output state to external state.

16. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
17. The device is now operating with PMIC. Active Regulator is in OFF state, and DeepSleep Regulator operates in external state.

Handover from PMIC to internal regulator

Figure 42 shows the handover flow from PMIC to internal regulator.

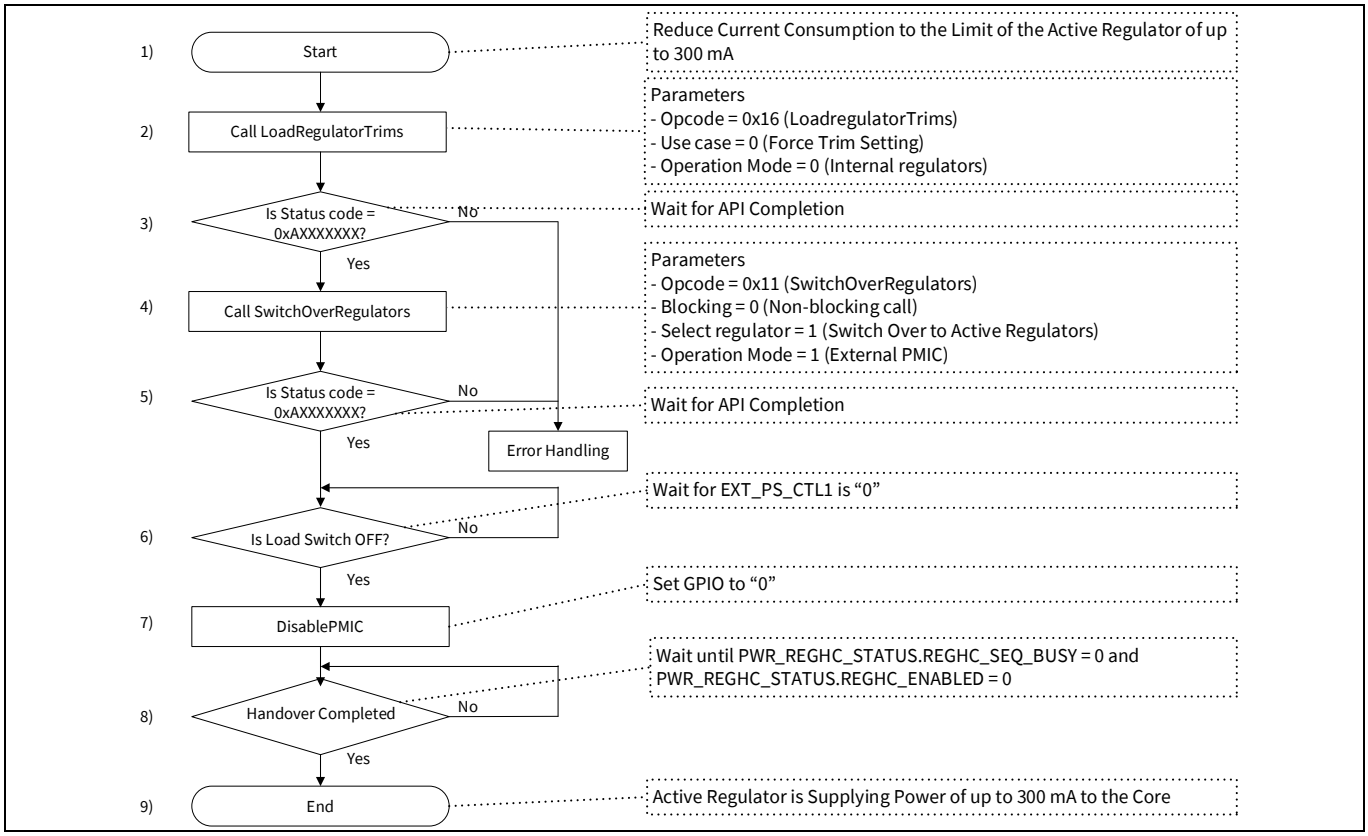

Figure 42 Handover flow from PMIC to internal regulator with load switch 2 (case 3)

To transition from PMIC to internal regulator:

1. Reduce the current consumption to be within the limit of Active Regulator, which is 300 mA.
2. Call the `LoadRegulatorTrims` API with force trim setting:
   - Opcode = 0x16 (Call the `LoadRegulatorTrims` API)
   - Use case = 0 (Force trim setting)
   - Operating Mode = 1 (internal regulators)

This process changes internal regulator output state to internal state.

3. Wait for the `LoadRegulatorTrims` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
4. Call the `SwitchOverRegulators` API with the following parameters:
   - Opcode = 0x11 (Call the `SwitchOverRegulators` API)
   - Blocking = 0 (Non-blocking call)
   - Select regulator = 1 (Switch over to Active Regulator)
   - Operating Mode = 1 (External PMIC)
5. Wait for the `SwitchOverRegulators` API to complete. If the status code is other than 0xAXXXXXXX, you need to perform appropriate error handling according to your system.
6. Wait for software timer to end, which means wait for the load switch to be completely turned OFF.

Note: In this configuration, a resistor to restrict current is placed between EXT_PS_CTL1 (PMIC enable signal output) and load switch. The load switch may not turn OFF immediately after PMIC disable output due to this resistor and capacitance of using load switch. Therefore, the handover

*completion need to wait for the load switch to completely turn OFF. Implement the timer using software, and wait for the load switch to turn OFF. See **Load switch** for calculating the required waiting time.*

7. Set GPIO to “0” for disabling PMIC.
8. Wait until PWR_REGHC_STATUS.REGHC_SEQ_BUSY = 0 and PWR_REGHC_STATUS.REGHC_ENABLED = 0.
9. The device is now operating with internal regulator.

4.5 Component selection

This section describes the selection of components.

4.5.1 Peripheral components of PMIC

In PMIC Direct Connection, the output capacitance of PMIC is shared with the V_{CCD} capacitance. For more details on selecting the output capacitor, see **Component selection**.

Also, the output voltage setting resistor should consider the bias current by itself. Note that this bias current also flows during DeepSleep power mode.

For other PMIC components, see the PMIC datasheet.

4.5.2 Load switch

Table 19 lists the load switch requirement.

Table 19 Load switch requirement specification

Parameter	Symbol	Value	Unit
Source-Drain voltage	V_{DSO}	≥ V_{DDD}	V
Source-Gate voltage	V_{GSO}	≥ V_{DDD}	V
Drain current	I_D	≥ 0.6	A

R_{ON} of the load switch may cause a V_{CCD} voltage drop. V_{CCD} drop voltage by R_{ON} is calculated using **Equation 3**.

Equation 3

$$V_{CCD\_drop} = R_{ON\_max} \times I_{VCCD\_max}$$

Where:

- V_{CCD_drop}: V_{CCD} drop voltage (V)
- R_{ON_max}: Load switch maximum on resistance (Ω)
- I_{VCCD_max}: Maximum V_{CCD} load current (A)

Select a load switch that fits the V_{CCD} voltage requirement of MCU including this voltage drop. Low R_{ON} is recommended in consideration of voltage drop when current flows.

Source-Gate voltage of the load switch during the ON state is Source-Gate minimum voltage of the load switch in **Equation 4**. The load switch must turn ON with this voltage.

Equation 4

$$V_{GS\_min} = V_{G\_min} - V_{CCD\_max}$$

PMIC (switching regulator)

Where:

- V_{GS_min}: Source-Gate minimum voltage of the load switch (V)
- V_{G_min}: Ground-Gate minimum voltage of the load switch (V)
  - For PMIC with Load Switch Case 1, V_{G_min} means power good pull-up minimum voltage.
  - For PMIC with Load Switch Case 2, V_{G_min} means minimum voltage of EXT_PS_CTL1=H.
- V_{CCD_max}: Maximum V_{CCD} voltage (V)

In case of PMIC with Load Switch 2, EXT_PS_CTL1 might supply overcurrent momentarily to drive the load switch gate. In this case, current restriction resistor is needed to satisfy maximum current of EXT_PS_CTL1. (For more details, see SID23 and SID30 in the **Body Controller High Family** datasheet **[1]**).

Required resistance of the load switch gate is calculated using **Equation 5**.

Equation 5

$$R_{Load\ switch\ gate} \geq V_{DDD\_max}/I_{EXT\_PS\_CTL1\_peak}$$

Where:

- $R_{Load\_switch\_gate}$: EXT_PS_CTL1 current restriction resistor (Ω)
- $I_{EXT\_PS\_CTL1\_peak}$: EXT_PS_CTL1 peak current to drive the load switch gate (A)
- V_{DDD_max}: Maximum V_{DDD} voltage (V)

For example, if V_{DDD} maximum voltage is 5.5 V, a resistor more than 5.5 kΩ is required to reduce the current to 1 mA or less.

Furthermore, by this current restriction resistor and Ciss of the load switch, voltage waveform of the load switch gate becomes a discharge curve. As a result, ON/OFF status transient of load switch is delayed. This time constant is calculated using **Equation 6**.

Equation 6

$$T_{EXT\_PS\_CTL1} = R_{Load\ switch\ gate} \times C_{iss\_load\ switch}$$

Where:

- $T_{EXT\_PS\_CTL1}$: Time constant of load switch gate and current restriction resistor (s)
- $C_{iss\_load\_switch}$: input capacitance of load switch (F)
- $R_{Load\_switch\_gate}$: EXT_PS_CTL1 current restriction resistor (Ω)

The load switch gate delay time that rises to 95% of EXT_PS_CTL1 high voltage, is three times the value of τ. The load switch gate delay time that falls to 5% of EXT_PS_CTL1 high voltage, is also the same.

4.6 Layout design guidelines

Here are a few PCB layout design guidelines:

- The output of PMIC and load switch should be located as close as possible to the V_{CCD} terminal of the MCU.

For other precautions, follow the PMIC layout guidelines from the PMIC supplier.

5 Appendix A. Hardware sequence

The following shows an example sequence of CYT3B/4B series.

Handover sequence between pass transistor and active regulator

Figure 43 shows the handover sequence of hardware between Pass Transistor and Active Regulator.

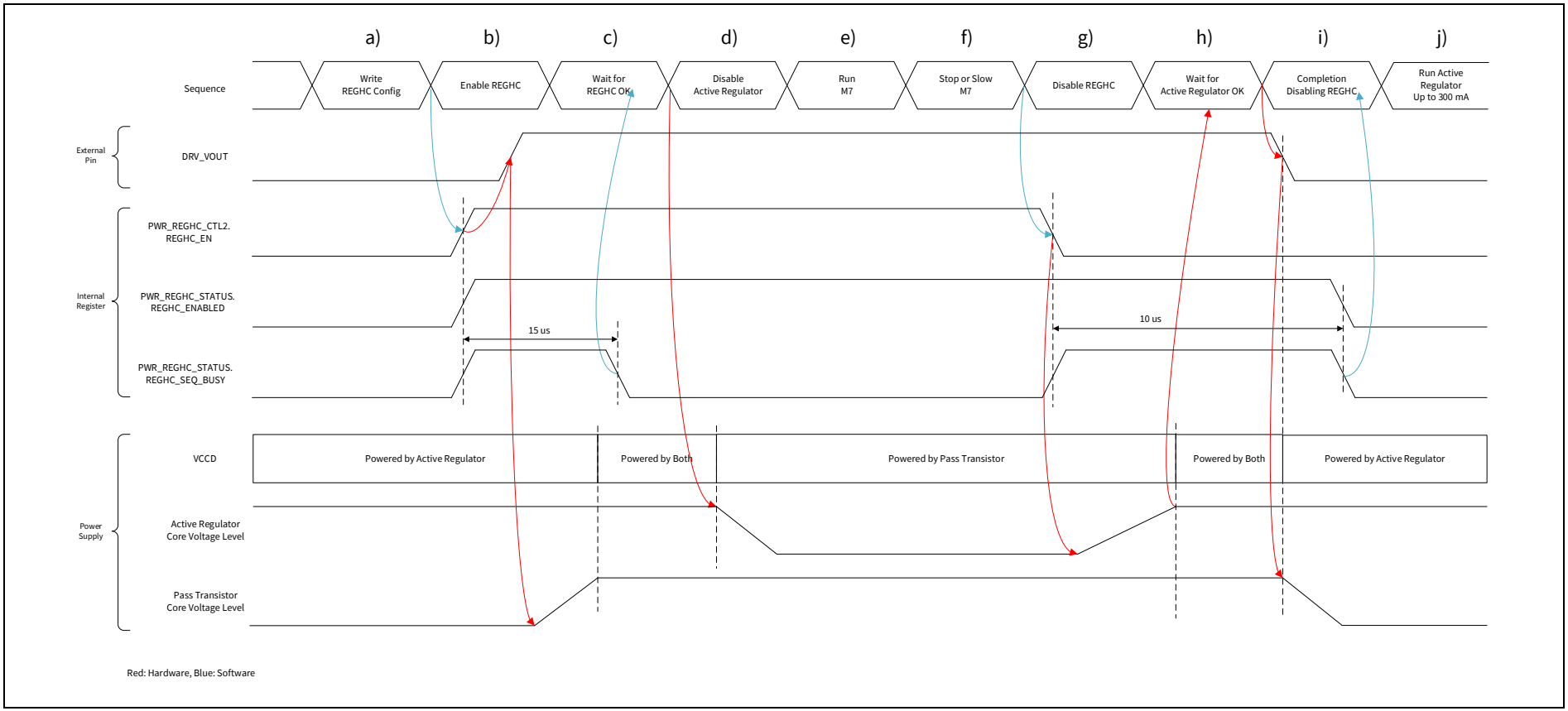

Figure 43 Handover sequence between pass transistor and active regulator

MCU starts with the Active Regulator during startup:

a) SW: Configures REGHC with Pass Transistor in external pass transistor mode.
b) SW: Enables REGHC with Pass Transistor. Then, REGHC starts to operate the Pass Transistor by driving DRV_VOUT pin.
c) SW: Waits for the REGHC OK flag, until the external core voltage level is reached.
d) HW: Disables Active Regulator.
e) MCU runs with REGHC (Pass Transistor).

Note: *The application software should increase the operating clock frequency step-by-step to avoid V_{CCD} voltage undershoot.*

f) Prior to the transition to the Active Regulator, the current consumption of the application must be within the limit of the Active Regulator.
g) SW: Disables REGHC with Pass Transistor
h) HW: Waits for Active Regulator to be fully active.
i) HW: Completes the REGHC disabling procedure.
j) MCU runs with Active Regulator.

Handover sequence between PMIC and active regulator

Figure 44 shows the handover sequence of hardware between PMIC and Active Regulator

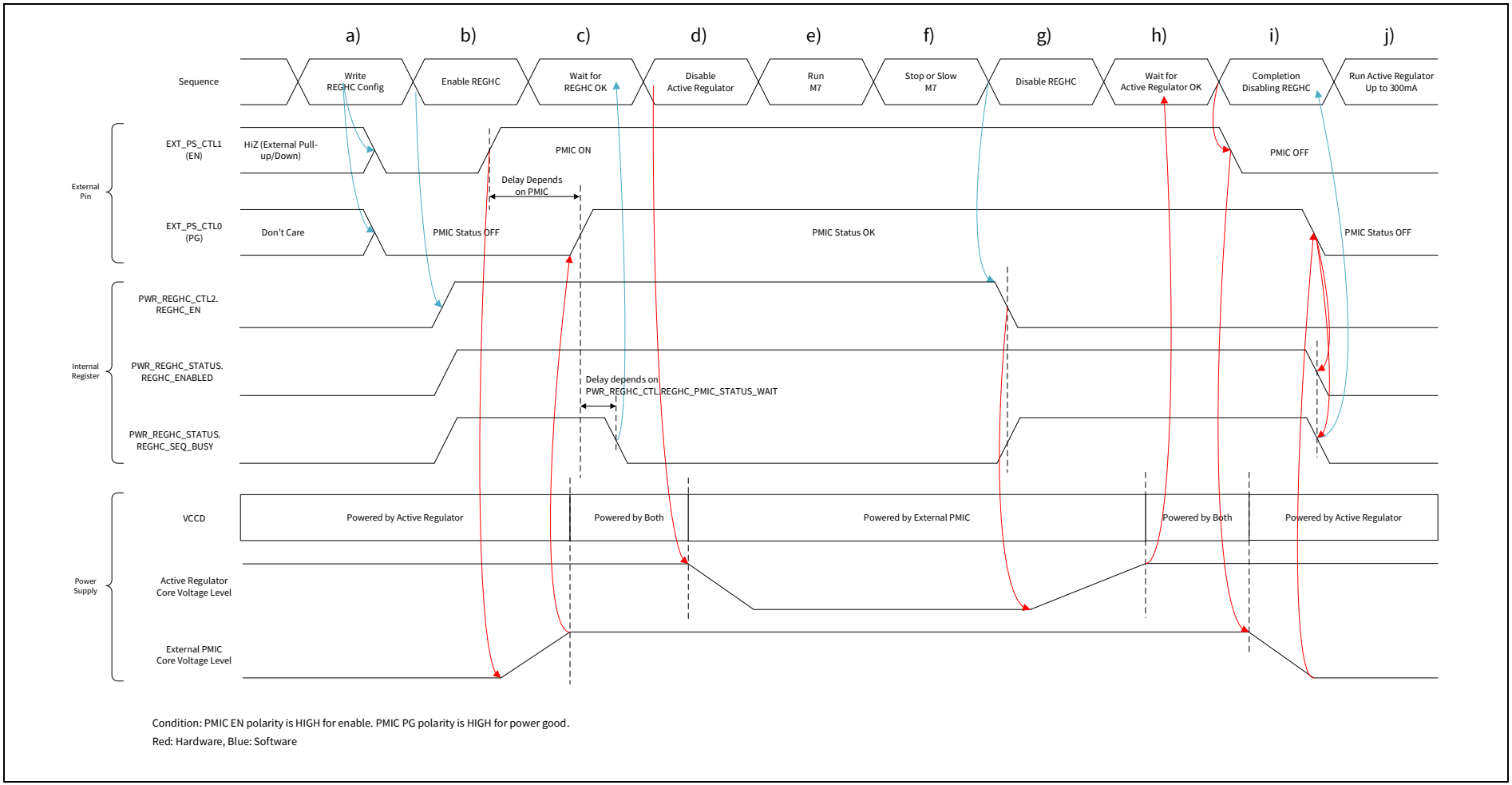

Figure 44 Handover sequence between PMIC and active regulator

MCU starts with the Active Regulator during startup:

a) SW: Configures REGHC with PMIC.

b) SW: Enables REGHC with PMIC. Then, external PMIC starts to operate.

c) SW: Waits for REGHC OK flag. REGHC OK flag is set when EXT_PS_CTL0 is in power good state after PMIC is ready.

d) If EXT_PS_CTL0 is in power good state during soft-start before 100% of the external target voltage is reached, the internal Active Regulator may be turned OFF and V_{CCD} voltage undershoot might occur after an immediate increase in the current consumption of the application (for example, PLL). To mitigate this problem, you can use the REGHC_PMIC_STATUS_WAIT to cover the settling time of the PMIC. This is used by the hardware sequencer to set additional settling time before disabling the internal Active Regulator. HW disables Active Regulator. V_{CCD} is powered by external PMIC.

e) MCU can run with external PMIC.

Note: The application software must increase the operating clock frequency should be increased step-by-step to avoid V_{CCD} voltage undershoot.

f) MCU needs to run below the limit of Active Regulator prior to transitioning to Active Regulator.

g) SW: Starts the procedure of handover from PMIC to internal regulator. Disables REGHC with PMIC.

h) HW: Waits for Active Regulator to be fully active.

i) HW: Completes the disabling of external PMIC.

j) MCU runs with Active Regulator.

Abbreviations

Abbreviation	Description
API	Application Programming Interface
BOD	Brown-out-Detection
BOM	Bill of Material
EP	Exposed Pad
GPIO	General Purpose I/O
HVD	High-Voltage-Detection
HV reset	High-Voltage Reset
LDO regulator	Low-Dropout regulator
LV reset	Low-Voltage Reset
MCU	Microcontroller Unit
OCD	Over-Current-Detection
OVD	Over-Voltage-Detection
PCB	Printed Circuit Board
PG	Power Good output signal from PMIC
PFM	Pulse Frequency Modulation
PMIC	Power Management Integrated Circuit
PWM	Pulse Width Modulation
POR	Power-On-Reset
REGHC	High Current Regulator Controller
SROM	Supervisory ROM
WFI	Wait For Interrupt CPU instruction

References

The following are the TRAVEO™ T2G family series datasheets and technical reference manuals. Contact **Technical Support** to obtain these documents.

[1] Device datasheets:

- **CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family**
- CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-24601)
- **CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family**
- CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-27763)

[2] Architecture technical reference manuals:

- **TRAVEO™ T2G automotive body controller high family architecture technical reference manual (TRM)**
- TRAVEO™ T2G automotive cluster 2D family architecture technical reference manual (TRM) (Doc No. 002-25800)

[3] Registers technical reference manuals:

- **TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF**
- **TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB**
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN (Doc No. 002-25923)
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT3DL (Doc No. 002-29854)

[4] Application notes:

- **AN220118 - Getting started with the TRAVEO™ T2G family**
- **AN201006 - Thermal considerations and parameters**

Revision history

Document version	Date of release	Description of changes
**	2020-08-04	Initial release
*A	2020-09-23	Deleted DRV_VOUT function in PMIC configuration Changed WaitCount parameter setting and flow in `ConfigureRegulator` API Clarification of DeepSleep power mode entry instruction.
*B	2021-01-07	Added part number. (CYT3 series) Added description for Cluster 2D family
*C	2021-08-10	Added note in section **System Call API** Changed pass transistor configuration

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-08-10

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Go to www.cypress.com/support

Document reference

002-26698 Rev. *C

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.