# AURIX™ TC3xx and OPTIREG™ PMIC TLF3558x - low power states

## AURIX™ 32-bit microcontroller family

## About this document

### Scope and purpose

This application note describes how to set the system, composed of the AURIX™ TC3xx and OPTIREG™ PMIC TLF3558x, in low power modes, such as sleep mode or standby mode.

### Intended audience

This document is intended to support engineers using state transition of OPTIREG™ PMIC TLF3558x on AURIX™ TC3xx boards.

# Table of contents

**About this document** ..... **1**
**Table of contents** ..... **2**
**1 Introduction** ..... **3**
1.1 Tools ..... 3
**2 PMIC overview** ..... **4**
2.1 PMIC state ..... 5
2.2 Normal state ..... 5
2.3 STANDBY state ..... 6
2.4 SLEEP state ..... 7
2.5 STANDBY wake-up ..... 8
2.6 SLEEP wake-up ..... 8
2.7 Failed transition ..... 9
**3 PMIC integration on AURIX™ TC3xx boards** ..... **10**
3.1 Enabling system state transition ..... 11
3.2 How to transition the AURIX™ TC3xx board to the lowest power mode possible ..... 13
**4 Standby controller (SCR)** ..... **15**
**5 Software implementation** ..... **17**
5.1 Set-up and prerequisites ..... 18
5.2 Project overview ..... 19
5.3 Initialization function ..... 20
5.4 State function ..... 23
5.5 Main loop ..... 24
5.5.1 Low power mode ..... 25
5.6 Standby Controller ..... 27
**6 State transition related measurement** ..... **29**
6.1 Time measurement ..... 29
6.2 Transition from NORMAL to STANDBY ..... 30
6.3 Transition from STANDBY to RUN ..... 30
6.4 Current measurement ..... 32
**References** ..... **33**
**Revision history** ..... **34**
**Disclaimer** ..... **35**

# 1 Introduction

This document is based on the power management system architecture, where the AURIX™ microcontroller is supplied by the Infineon companion Power Management IC (PMIC), OPTIREG™ TLF3558x, and the reference Infineon boards considered in the next pages are the AURIX™ TC3xx Application Kits and the AURIX™ TC3xx TriBoards.

The OPTIREG™ PMIC TLF3558x is an integrated circuit designed for the power management of AURIX™ microcontrollers. On the AURIX™ TC3xx boards considered in this document, the embedded PMIC can control the main supply voltages applied to the microcontroller, as well as additional supply rails for external peripherals.

The considered PMIC has several outputs and different operating states. Depending on the state, the configuration of the output rails is adapted, and the overall system power consumption is subsequently impacted. More detailed information can be found in the OPTIREG™ PMIC TLF3558x datasheets (TLF35584).

Changing the state of the PMIC to a low-power mode (SLEEP or STANDBY) helps reduce the current consumption of the entire system, which is useful for saving energy in many application fields.

The SLEEP-state is a low-power state that the microcontroller may enter to reduce the current consumption when the application is not in use (for example, microcontroller is in STOP mode). In the NORMAL state, the microcontroller may configure the status of PMIC regulators and the safety functions via SPI commands.

The STANDBY state is a low-power state that the microcontroller may enter to reduce the current consumption to a minimum level when the application is not in use for a long time. During the standby state, the functionalities and all the CPUs of the microcontroller are disabled, except the stand-by controller (SCR), which could be enabled to execute simple standby operations.

On all the AURIX™ TC3xx boards considered in this document, it is possible to control the power modes with the PMIC. This document explains how to change the state of the power subsystem of an AURIX™ TC3xx application to SLEEP and STANDBY modes. The last case presents, additionally, the wake-up performed by the SCR.

## 1.1 Tools

The equipment considered in this application note consists of the AURIX™ TC397 board (Application Kit or TriBoard) and the TLF35584QVHS (PMIC). All the code examples have been developed on the AURIX™ Development Studio (ADS) and can be found in the following GIT repository:

- TC397_APPKIT_5V_ADS_PMIC_SLEEP_MODE
- TC397_APPKIT_5V_ADS_PMIC_STANDBY_MODE
- TC397_TRIBOARD_ADS_PMIC_SLEEP_MODE
- TC397_TRIBOARD_ADS_PMIC_STANDBY_MODE

# 2 PMIC overview

The OPTIREG™ PMIC TLF3558x is a multi-voltage safety microprocessor supply with many features embedded, such as voltage regulators, QSPI, safety signals, and watchdog that can provide four supply outputs for the AURIX™ microcontrollers (QST, QUC, QVR, and QCO). To communicate with the microcontroller, the PMIC uses the QSPI protocol.

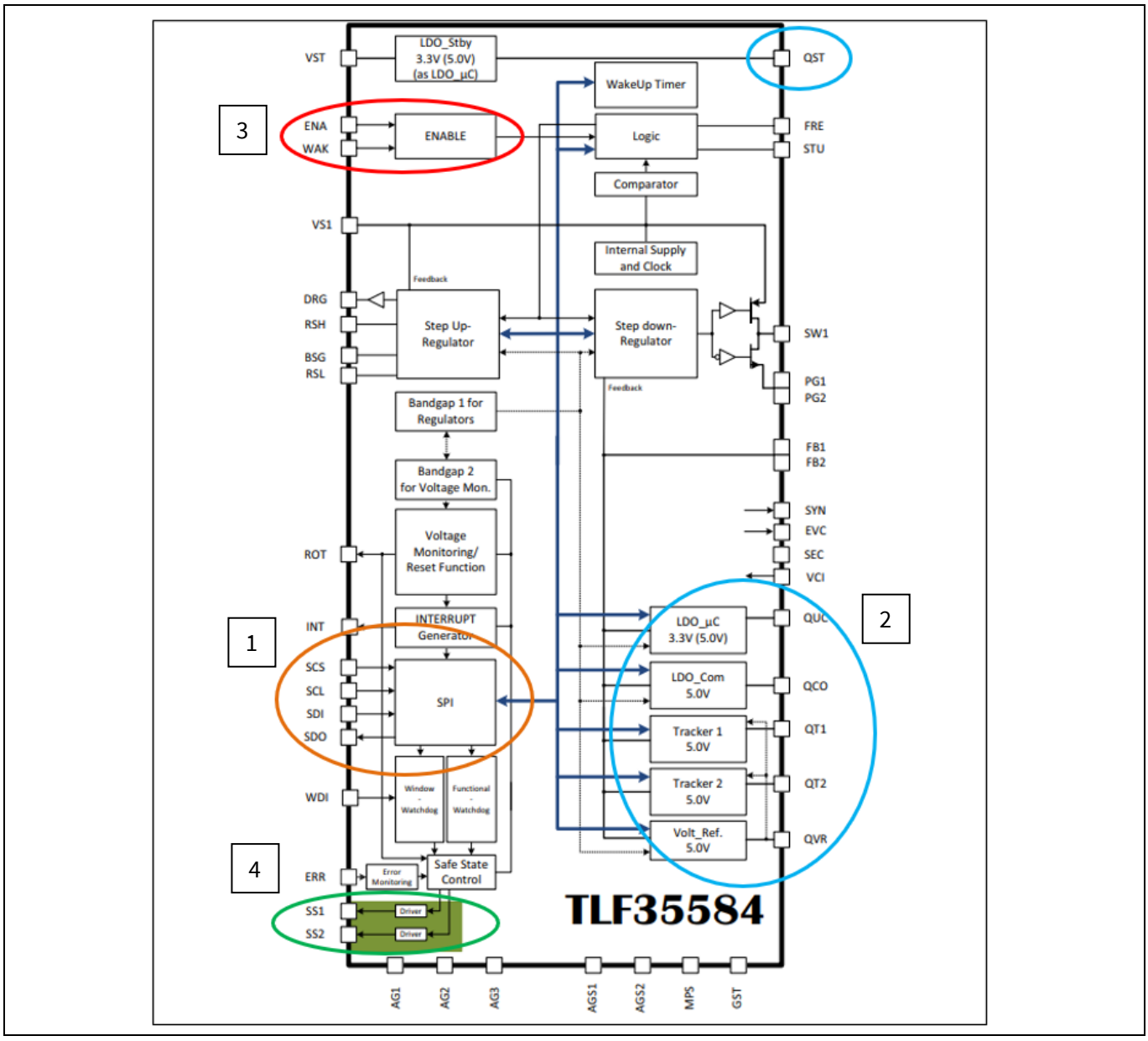


**Figure 1 TLF35584 bloc diagram**

- The microcontroller communicates with the PMIC via SPI (1)
- The PMIC has 4 voltage outputs, and 2 tracker outputs (2)
- The PMIC has 2 wake up inputs (ENA and WAK) (3)
- The PMIC has 2 safe state signals (4)

## 2.1 PMIC state

The OPTIREG™ PMIC TLF3558x can operate between seven operative states; the ones used in this document are the following:

- **INIT state**: It is the initialization state, which the PMIC enters when it is powered on
- **NORMAL state**: It is the normal operative state in which the PMIC stays while it is running with all the main functions active
- **STANDBY state**: It is a low-power state, where most of the main functions are turned off
- **SLEEP state**: It is a low-power state, where the IC remains in a safe state, but the main functions can be turned on.

The transition between the different states is described in the PMIC's state machine.

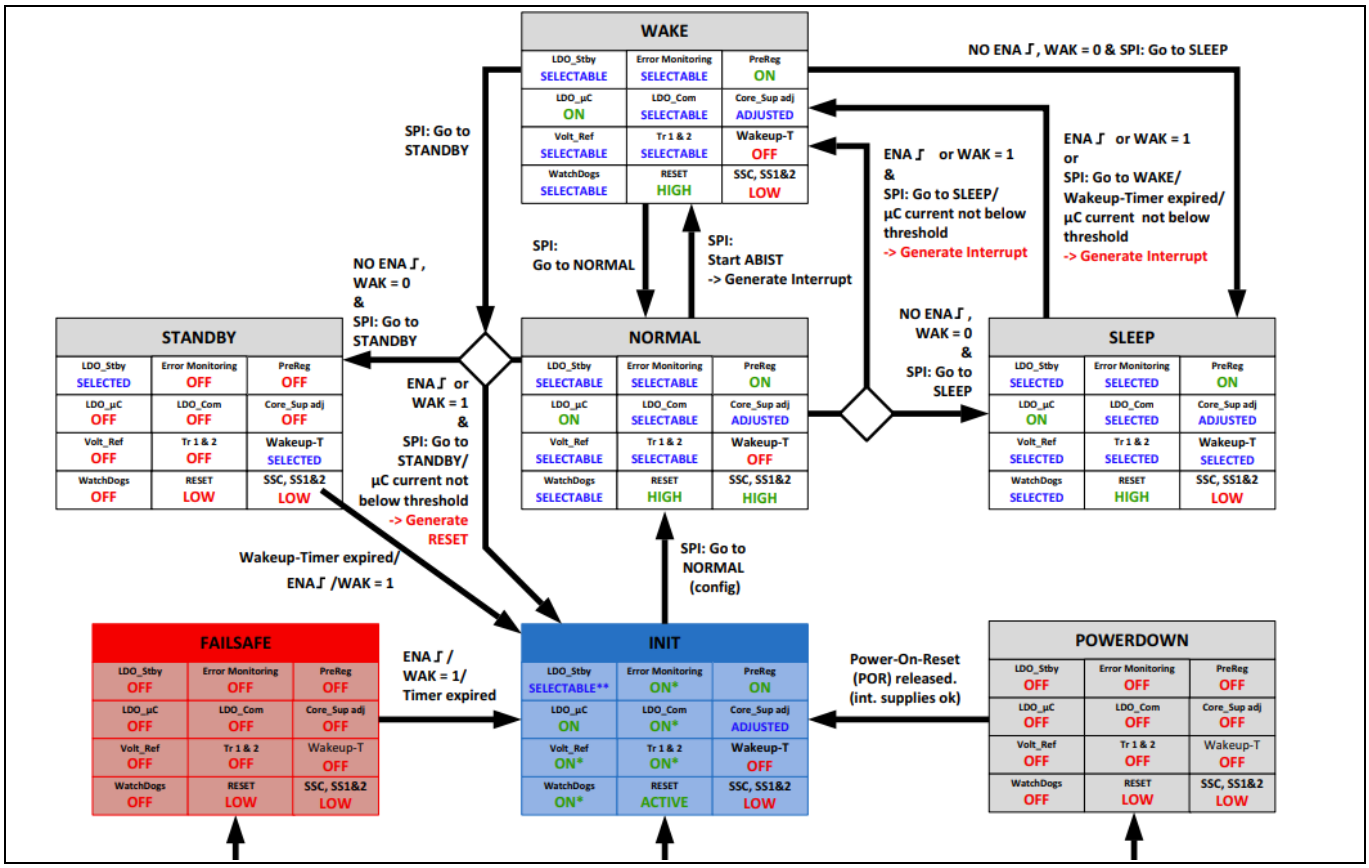


**Figure 2 TLF35584 state machine**

## 2.2 Normal state

In the NORMAL state, the PMIC is supplying the microcontroller and the applications. Safety and monitoring functions, such as reset and safe state control, are active. The microcontroller may configure several post regulators of the device and the wake-up timer using SPI commands.

To trigger the NORMAL state after the initialization of the board, the following prerequisites must be fulfilled:

- A delay of 60 µs must be considered to ensure proper release of internal validation signals

- Watchdog(s) need to be serviced once according to default configuration or according to reconfiguration within the INIT timer
- ERR monitor needs to be serviced with a valid signal (minimum 3 periods) or disabled within the INIT timer. The error valid signal is described in Table 18 of TLF3558x datasheets and exemplified by the square wave, “Error monitoring” as shown in Figure 3
- If the functional watchdog is activated, a valid FWD triggering needs to be provided according to the definition explained in TLF3558x datasheets. Only window watchdog, WWD, not FWD, is shown in Figure 3

The PMIC will go to the NORMAL state after receiving from AURIX™ SPI command “Go to NORMAL”.

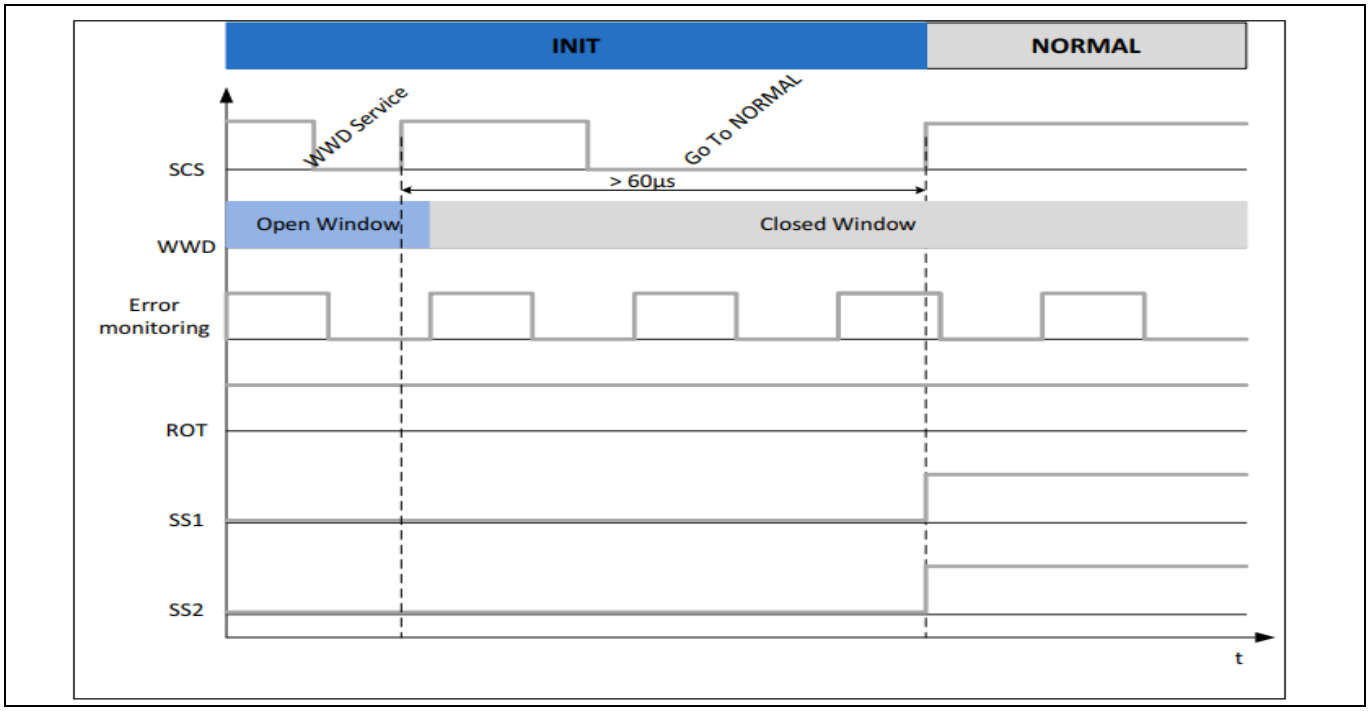


**Figure 3 Transition from INIT to NORMAL state**

## 2.3 STANDBY state

The STANDBY state is a low-power state in which the microcontroller may enter to reduce the current consumption to a minimum; the system application is in a safe state during standby.

The transition to the STANDBY is initiated by AURIX™ through the SPI command “GO TO STANDBY”. Before and during the transition to STANDBY, the following prerequisites should be fulfilled:

- Before going to standby, the PMIC needs to be in the NORMAL state
- The WAK and ENA signals should remain low and should not be triggered during the process
- The LDO_µC current (QUC regulator) should drop below a specific threshold to allow the transition (as specified in the TLF3558x “ILDO_µC, att” datasheet parameter)

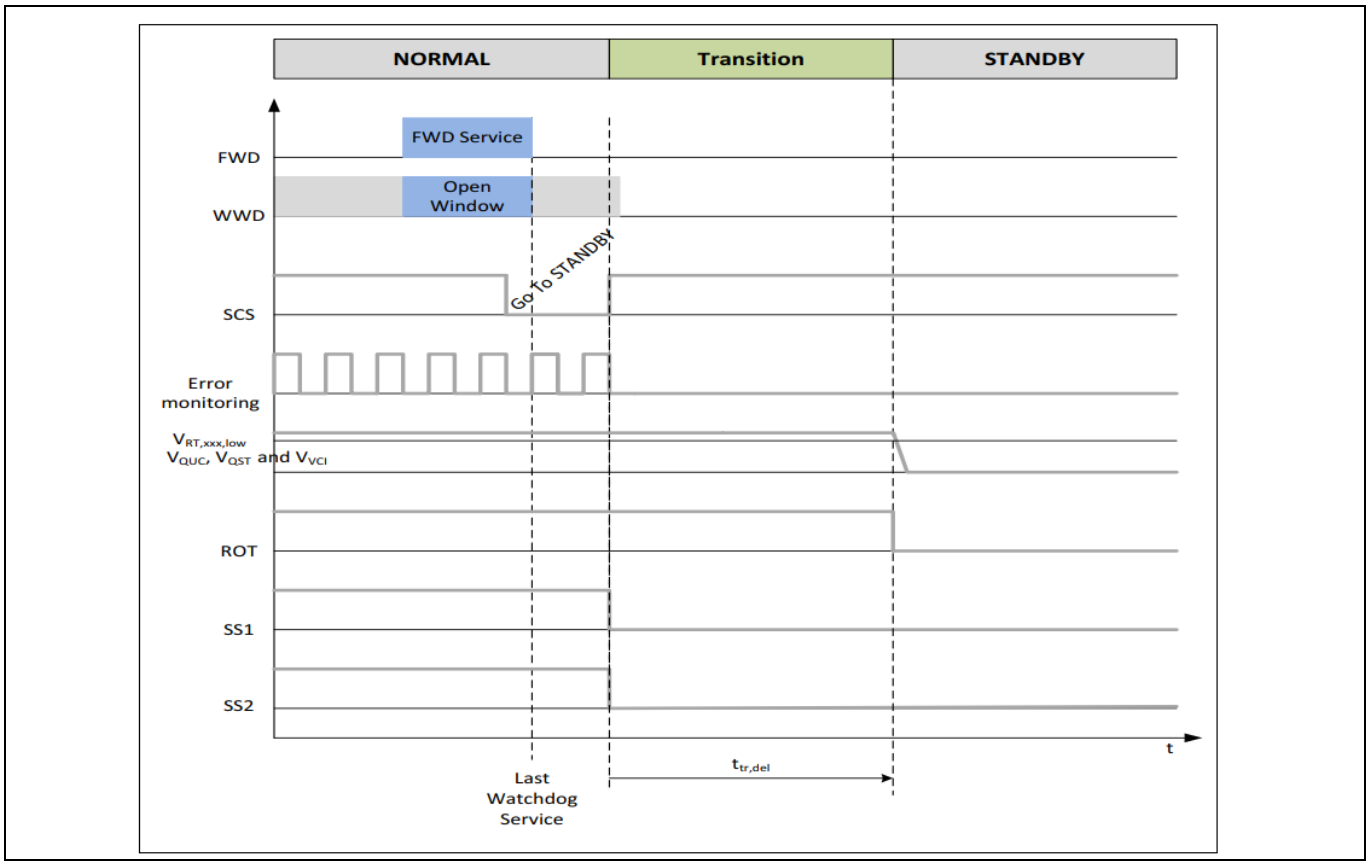


**Figure 4 Transition from NORMAL to STANDBY state**

## 2.4 SLEEP state

The SLEEP state is a low-power state in which the microcontroller may enter to reduce the current consumption to a minimum without going into standby mode. The application is in a safe state during sleep.

The transition to the SLEEP state is initiated by the AURIX through the SPI command “GO TO SLEEP”. Before and during the transition to SLEEP, the following prerequisites should be fulfilled:

- Before going to SLEEP, the PMIC needs to be in the NORMAL state
- The WAK and ENA signals should remain low and should not be triggered during the process
- The LDO_µC current (QUC regulator) should drop below a specific threshold to allow the transition (as specified in the TLF3558x “ILDO_µC, att” datasheet parameter)

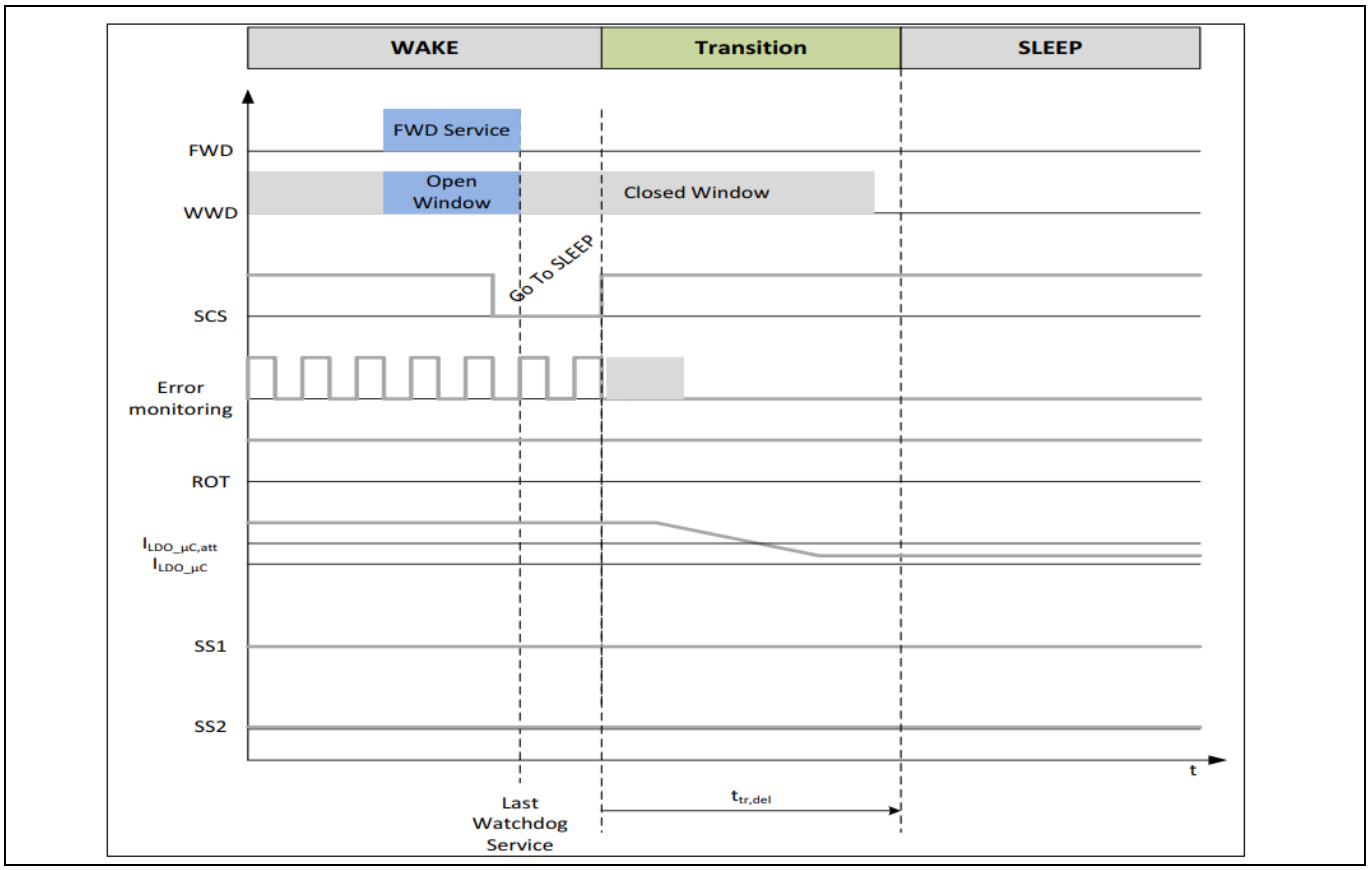


**Figure 5 Transition from NORMAL to SLEEP state**

## 2.5 STANDBY wake-up

The PMIC can operate a wake-up from STANDBY state to INIT state. The wake-up can be triggered by the following events:

- The wake timer
- A trigger on the ENA signal
- The WAK signal is set to '1'

After waking up the PMIC, when the microcontroller's related regulator (VQUC) is over the voltage reset threshold, the output reset (ROT) is activated.

## 2.6 SLEEP wake-up

The PMIC can operate a wake-up from the SLEEP state to the WAKE state. The wake-up can be triggered by the following events:

- The wake timer (embedded on PMIC)
- A trigger on the ENA signal (from the TriBoard's button 102)
- The WAK signal is set to '1' (from the onboard circuitry; refer to Section 3.1)
- The LDO_µC current is above the threshold value (as specified in the TLF3558x "ILDO_µC, att" datasheet parameter)
- SPI command "Go to Wake" (from AURIX™)

## 2.7 Failed transition

The PMIC's state transition fails when some of the prerequisites are not fulfilled. If the transition to the STANDBY state fails, then the PMIC goes to the INIT state instead, and the output reset (ROT) is activated. If the transition to SLEEP state fails, the PMIC goes to the WAKE state. The main reasons for failures are the following:

- Before the transition, the PMIC is not in the NORMAL state
- The ENA signal was set to '1' or was triggered during the transition
- The WAK signal was set to '1' during the transition
- The LDO_µC current was not below the threshold during the transition

*Note: Section 3.1 describes how to avoid such failures on the AURIX™ TC397 Application Kit and TriBoard.*

# 3 PMIC integration on AURIX™ TC3xx boards

On AURIX™ TC397 boards, the PMIC is always linked to the CPU with the same SPI connection (QPSI2). Even if they are similar in their architecture, the TriBoard and the application kit remain different. Therefore, they cannot be configured in the same way.

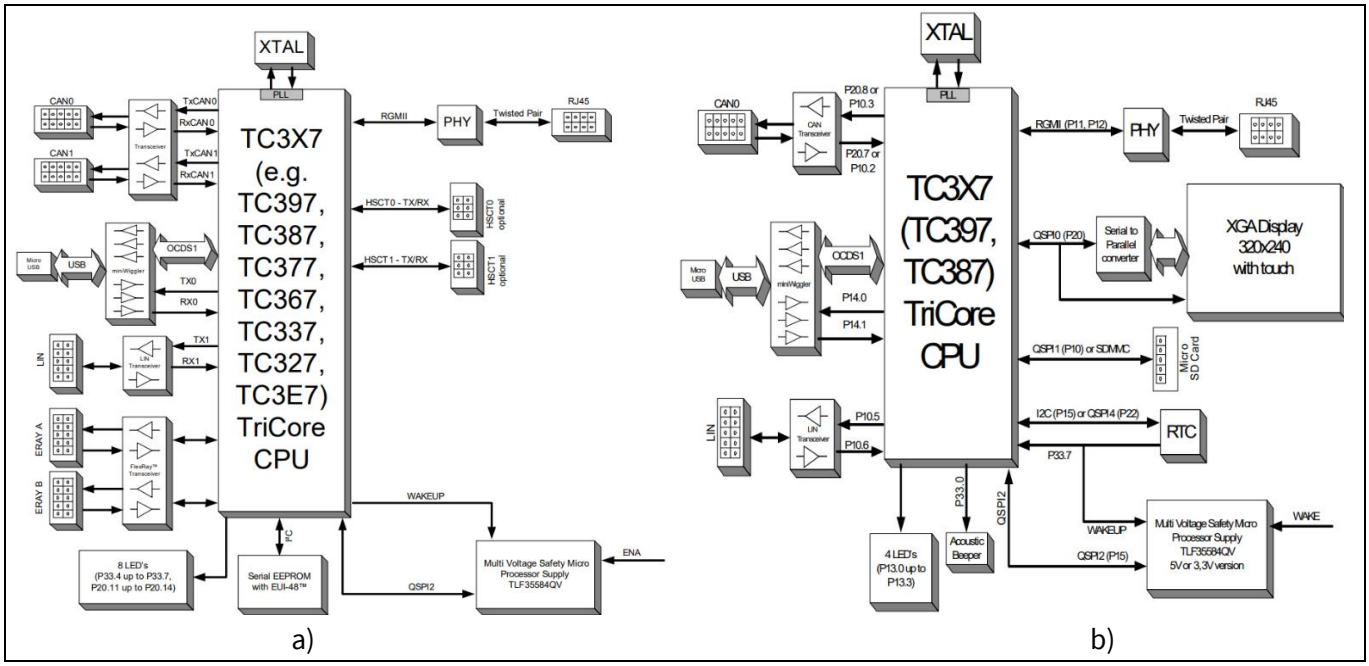


**Figure 6 (a) TriBoard block schematic and (b) Application kit block schematic**

On the TriBoard, the ENA signal is linked to the ENA button of the board, and the WAK signal is linked to a General Purpose Input Output (GPIO) pin.

On the application kit, the ENA signal is linked to the WAKE button of the board, and the WAK signal is linked to the Real-Time Clock (RTC) of the device.

On both boards, the QUC voltage is supplying the VEXT rail (External supply), which can be used as a trigger for changing the state of the AURIX™ TC3x7 (i.e., entering or exiting the STANDBY state of the microcontroller).

## 3.1 Enabling system state transition

This part will explain how to ensure the state transition against possible failures when changing state to a low-power mode (SLEEP or STANDBY) on both boards (TriBoard and Application kit). The possible failures are due to the ENA signal, LDO_uc current, or the WAK signal.

*Note:* *The solutions provided in this document are based only on these two boards. For other boards, this guide is not guaranteed; however, it can provide clues for solving the issue.*

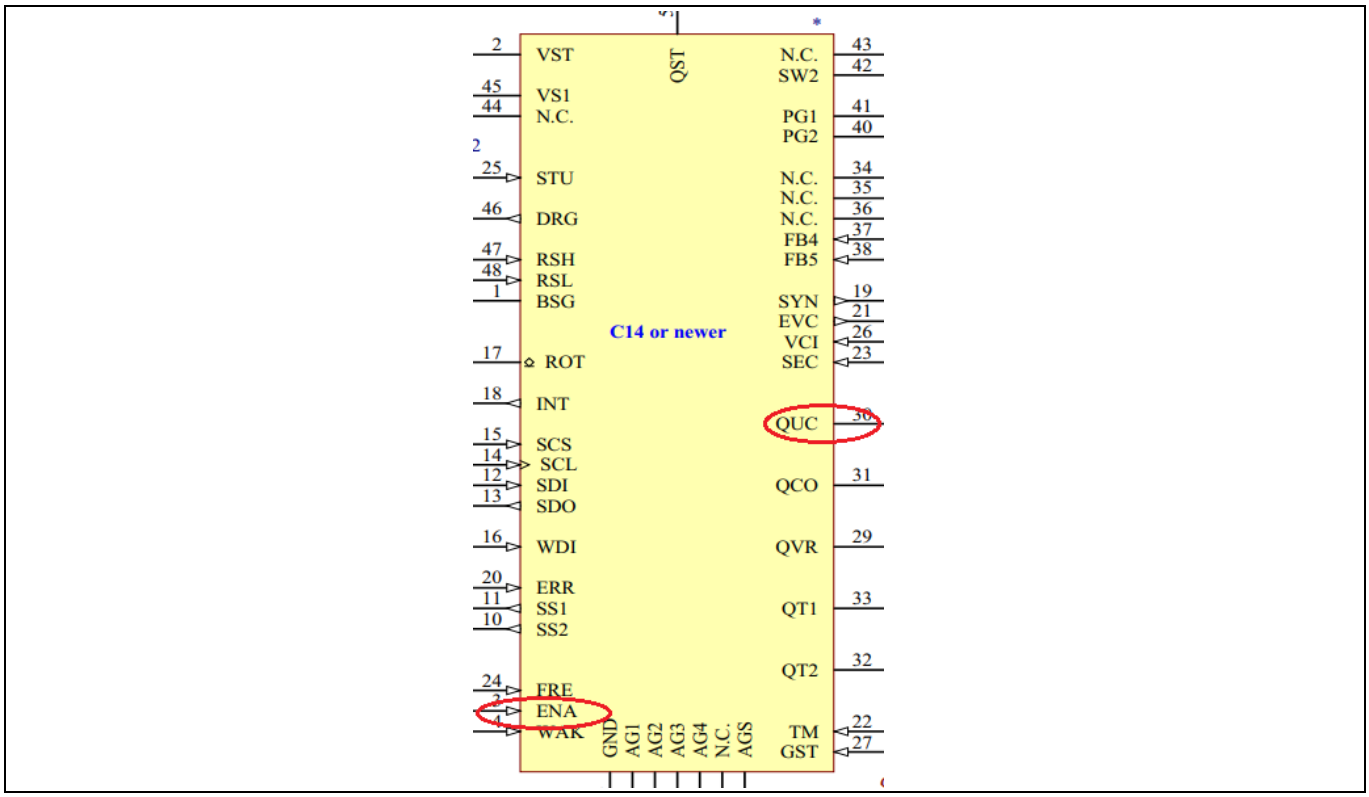


**Figure 7 TLF3558x pinout**

**ENA signal:**
Generally, this signal is connected to a button using direct logic (while the button is not pressed, the signal is low). In case of transition failure, this pin should be monitored with a scope to ensure that its logic level is low as expected.

**LDO_uc (QUC / VEXT) current:**
This failure is due to the power consumption of the microcontroller: the AURIX™ VEXT current consumption is too high, so the QUC value is never below the threshold (as specified in the TLF3558x datasheet parameter “ILDO_µC, att”).

There are several ways to reduce the current consumption of the board, such as putting the microcontroller in sleep mode, turning off the unused modules and peripherals of the microcontroller, or reducing their clock frequency.

For the AURIX™ TC397 TriBoard and Application Kit, the solution adopted in the example is to turn OFF the peripherals and set all the CPUs except CPU0 to idle.

*Note: The best practice is to monitor the current sunk from QUC to ensure it is below its threshold value.*

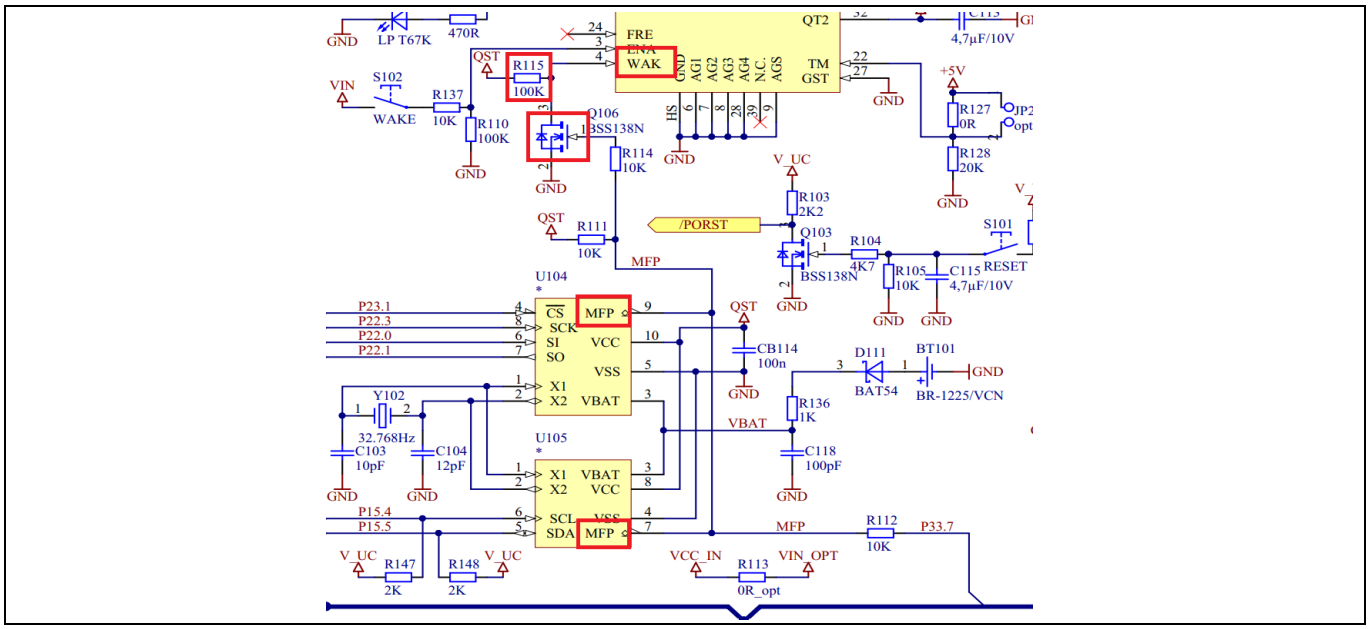


**Figure 8 TriBoard schematic – WAK path**

**WAK signal:**
This signal can be linked to different peripherals or ICs; therefore, it is suggested to ensure that this signal remains low by measuring it with a probe.

For the specific case of Application Kit AURIX™ TC397, the WAK signal is linked to the port multifunctional pin (MFP) of the RTC (component U105).

Based on the schematic, the WAK pin is driven by the RTC through the NMOS (Q106), which works as a switch, and the pull up resistor R115. To set the WAK signal to ‘0’, the transistor must be set to ON; therefore, the MFP signal must be set to ‘1’.

The RTC communicates with the microcontroller via I2C. A request to this IC must be sent from AURIX™ to enable the MFP output.

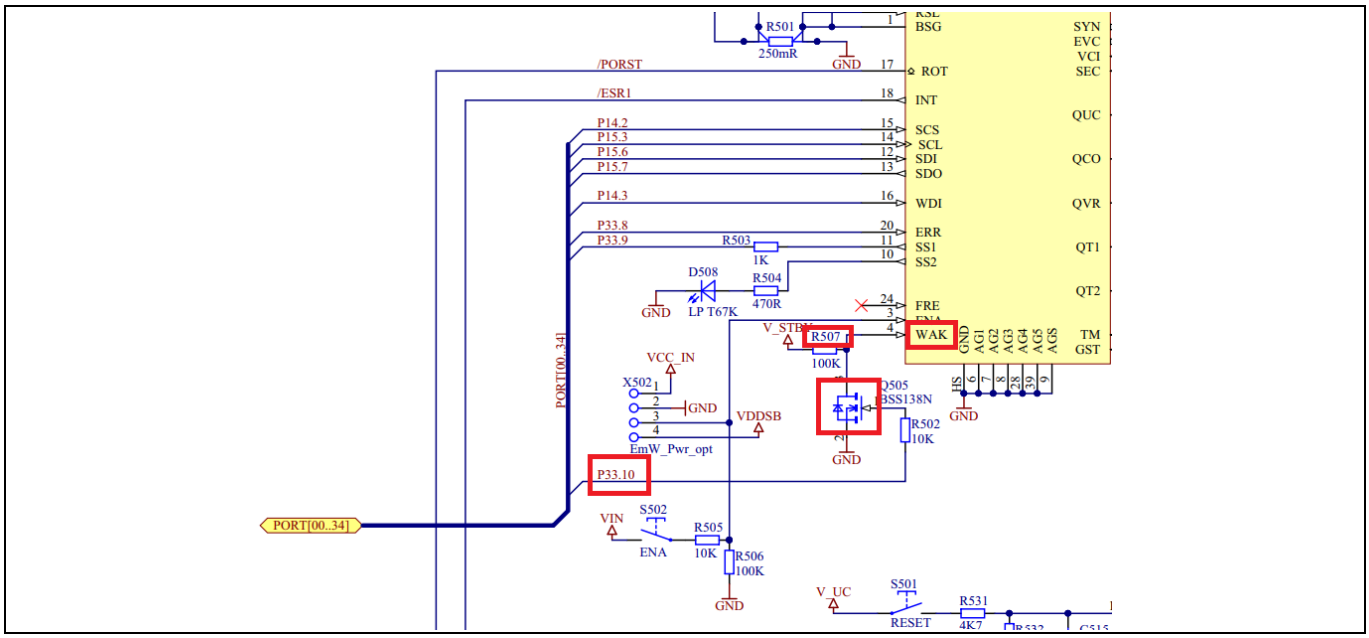


**Figure 9 Application Kit schematic - WAK path**

For the TriBoard AURIX™ TC397, the WAK signal is linked to the pin P33.10.

Following the schematic, the WAK pin is driven by the NMOS (Q505) , which works as a switch, and the related pull-up resistor R507. To set the WAK signal to '0', the transistor must be set to ON; therefore, the pin P33.10 must be set to '1'.

*Note: The standby controller (SCR) ensures that the pin is in a high state during the STANDBY state.*

## 3.2 How to transition the AURIX™ TC3xx board to the lowest power mode possible

As described in the AURIX™ TC3xx User Manual, PMS related chapter, the microcontroller may go to STANDBY mode according to one of the following events:

- Standby entry on a secondary under-voltage event during VEXT supply ramp-down, if configured in PMSWCR0.VEXTSTBYEN bits
- Standby entry on SW request (PMCSRx.REQSLP = 11B ) if configured via SCU_PMSWCR1.STBYEV register bit field. The Standby request is issued only after CPUx ENDINIT bit is set back again. The safety ENDINIT mechanism will not be used by a CPUx to issue a Standby request
- Standby entry on ESR1 (NMI) edge event if configured via SCU_PMSWCR1.STBYEV register bit field

Considering the first option above, it is possible to trigger the standby transition of the board by leading also the PMIC to standby. On a subsequent VEXT supply ramp-down event (managed by the PMIC) the microcontroller will go to standby as well.

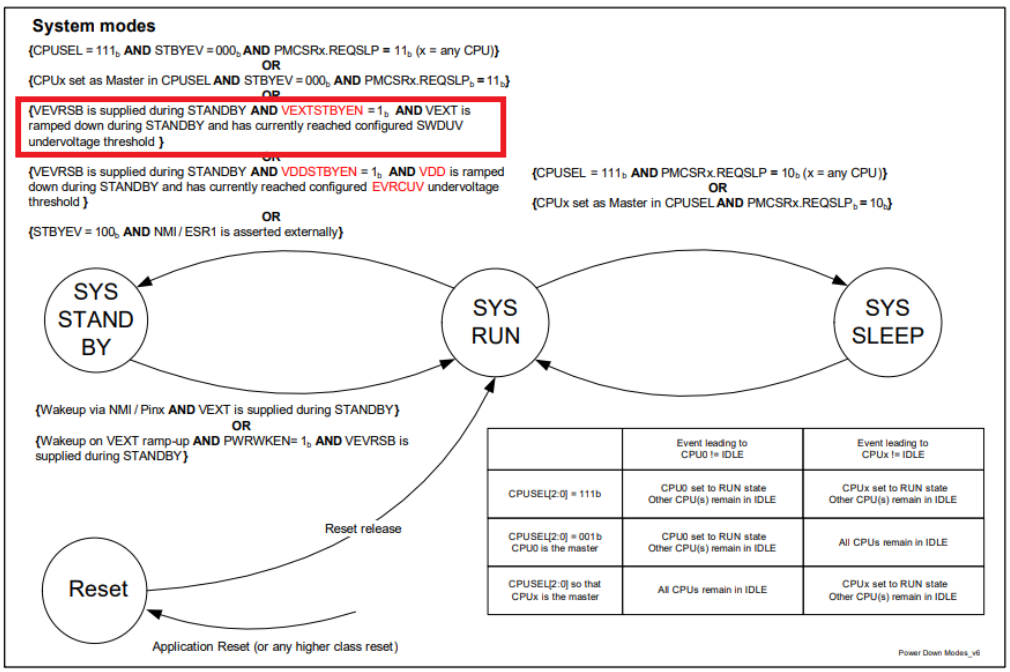


**Figure 10 AURIX™ TC3xx low power mode transitions (AURIX™ TC3xx user manual)**

*Note: The opposite mechanism works for the wake-up transition, when a VEXT supply ramp-up event occurs (managed by the PMIC), the microcontroller wakes up again.*

# 4 Standby controller (SCR)

The standby controller (SCR) is an 8-bit microcontroller inside the AURIX™ device that can still operate while the system is in standby state. The SCR is composed of a set of interrupts, a clocking system, an output port, and has its own space memory.

The main code of the SCR must be copied to the XRAM data. The SCR can operate using a 20 MHz clock or a 70 kHz clock.

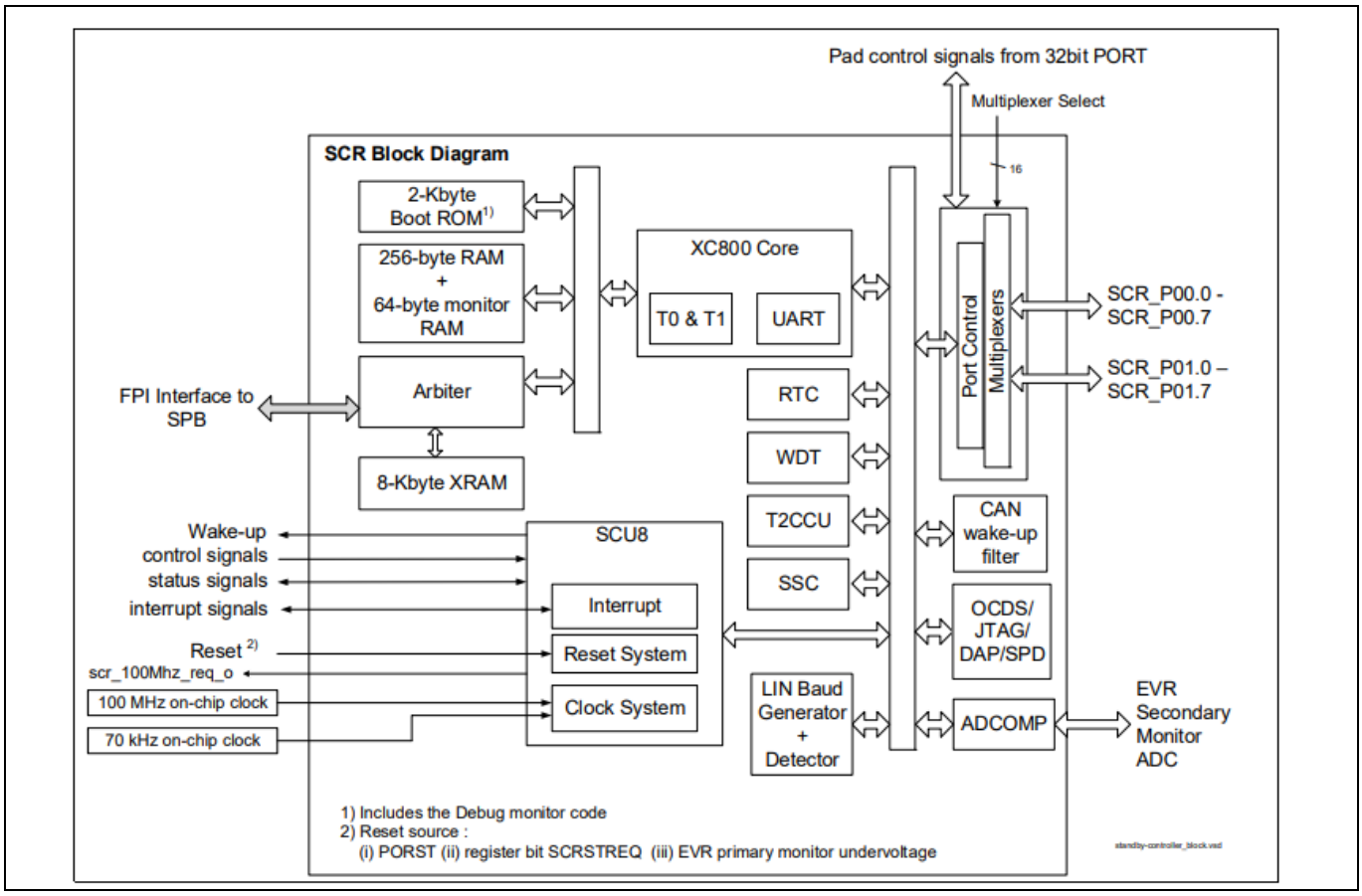


**Figure 11 SCR block diagram**

The SCR can control the pins P33.0 to pin P34.1 of the board, which are divided between pin ports SCR_P00 and SCR_P01 as reported in following tables:

| SCR Ports | Main Ports | SCR Ports | Main Ports |
|---|---|---|---|
| SCR_P00.0 | P33.0 | SCR_P01.0 | P34.1 |
| SCR_P00.1 | P33.1 | SCR_P01.1 | P33.9 |
| SCR_P00.2 | P33.2 | SCR_P01.2 | P33.10 |
| SCR_P00.3 | P33.3 | SCR_P01.3 | P33.11 |
| SCR_P00.4 | P33.4 | SCR_P01.4 | P33.12 |
| SCR_P00.5 | P33.5 | SCR_P01.5 | P33.13 |
| SCR_P00.6 | P33.6 | SCR_P01.6 | P33.14 |
| SCR_P00.7 | P33.7 | SCR_P01.7 | P33.15 |

**Figure 12 SCR ports and pins**

## Standby controller (SCR)

The SCR, when active in standby, can be used to wake up the board by setting up an interrupt that will trigger the WAK signal during the standby state.

The pin P33.10 is linked to the wake-up signal (WAK) on the AURIX™ TC397 TriBoard, while the pin P33.11 is linked to the button S202, and both are controlled by the standby controller. To wake up the board, the external interrupt (EXIN T5B) controlled by the button S202 needs to be enabled, and its function will have to toggle the state of the pin P33.10.

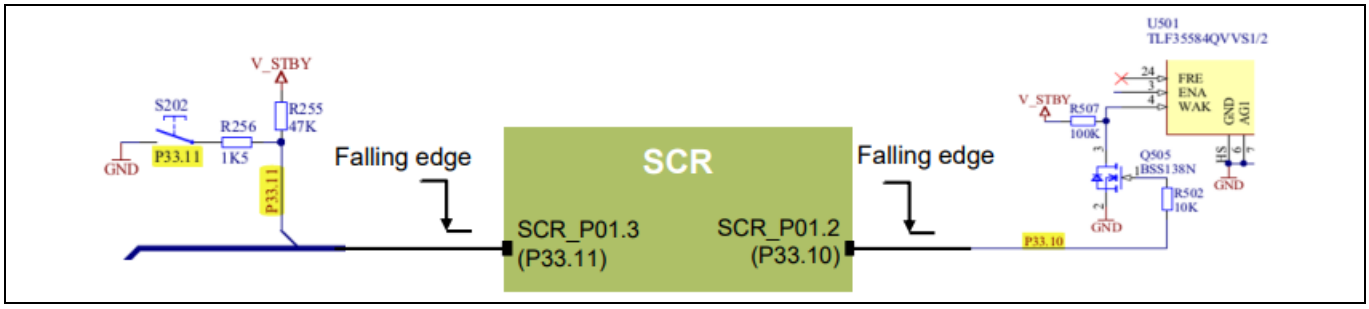


**Figure 13 TriBoard block schematic**

In order to reduce the power consumption, the clock frequency must be set to 70 kHz.

# 5 Software implementation

The following part of the document describes a software solution for transitioning the board into a low-power state (STANDBY) and then waking it up with a button. The described code is based on the AURIX™ TC397 TriBoard and application kit and was implemented using the AURIX™ Development Studio (ADS).

**Scope:**

For both boards, the program sends a request to the PMIC via QSPI to go to STANDBY. Once the PMIC is in standby state, the PMIC regulator (QUC) output is turned off, which results in the AURIX™ TC397 board entering standby mode (triggered by VEXT ramp-down). In the case of the TriBoard, the button S202 (P33.11) is used as an external trigger for the wake-up process, since it is linked to the WAK signal through the SCR.

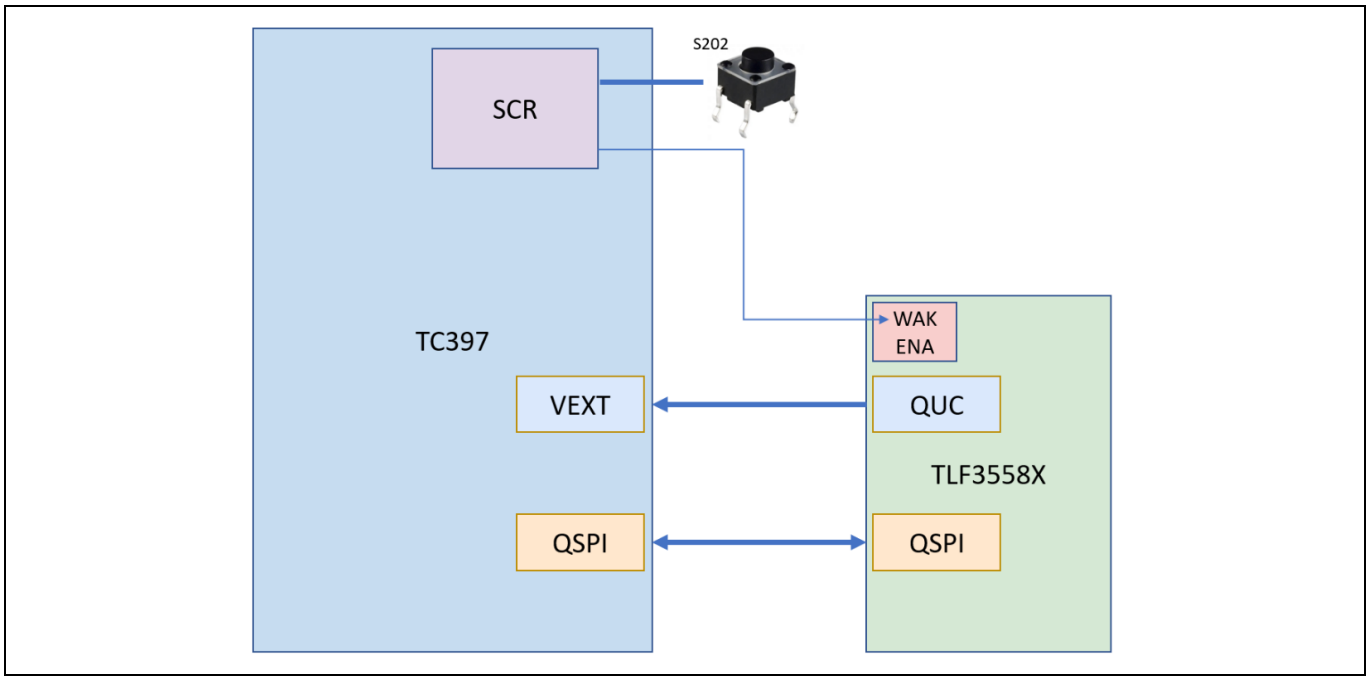


**Figure 14 State transition system diagram**

## 5.1 Set-up and prerequisites

When creating a project in ADS, in order to configure the SCR properly, the option “Add support for StandBy controller” must be enabled after selecting the board for the project and clicking “Next”.

*Note: The ADS version should be 1.9.12 or higher.*

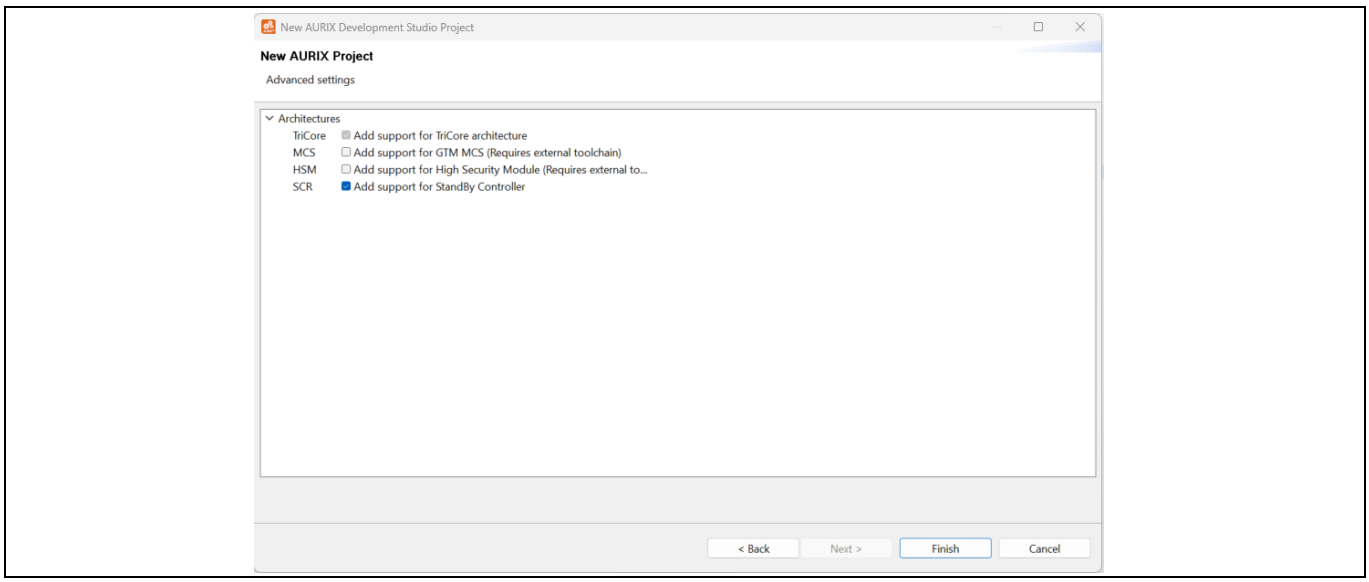


**Figure 15 New ADS Project setup**

After creating the project, the function that enables and copies the SCR main code to the XRAM memory is in the *SCR.c* file. The *SCR* folder contains the SCR library and the main program (*main.c*).

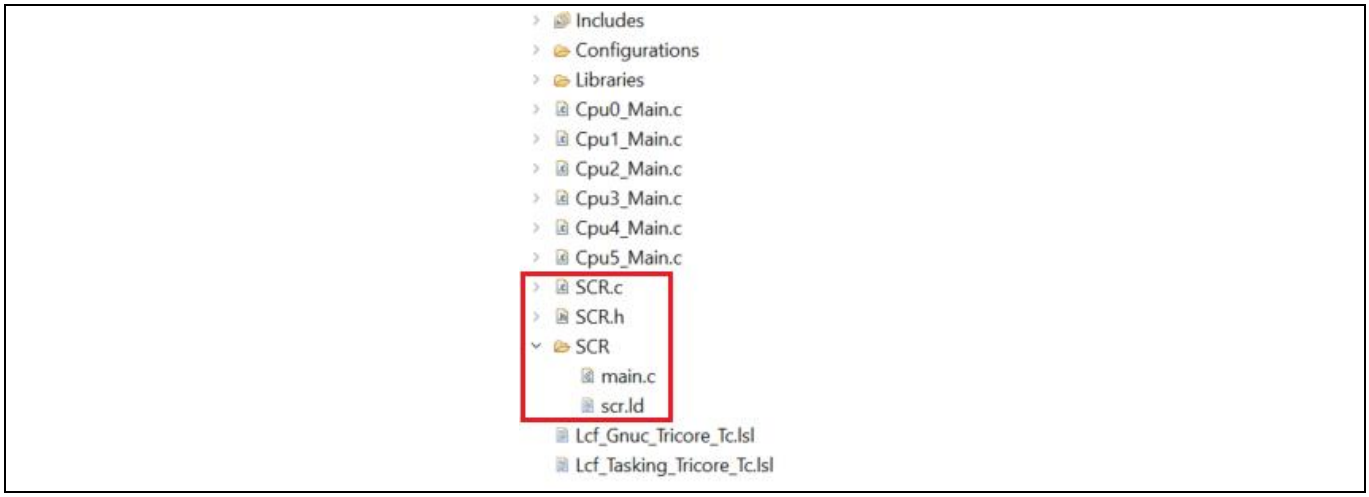


**Figure 16 ADS project - SCR related files**

## 5.2 Project overview


**Figure 17 ADS project file tree**

For the rest of the document, the following project is used as a code example:

1. **serialio**: UART initialization function, redefinition of the *printf()* function for displaying the results in the terminal
2. **Cpu0_main**: Main code and the calling of the initialization function
3. **I2C_Read_Ext_Device**: I2C function
4. **powerConfig**: Low/high power function and standby initialization function
5. **SCR**: SCR function
6. **SPI_init**: SPI Function (initialization and transmission function)
7. **StandBy_conf**: Function related to the transition state
8. **TLF35584**: Function related to the PMIC
9. **SCR**: Main code for the standby controller (main.c)

## 5.3 Initialization function

The Cpu0_Main.c file must be configured as follows before changing the PMIC state:

- The QSPI communication between the CPU and the PMIC (*initQSPI()*) is initialized
- The watchdog and error monitor of the PMIC are disabled, and the PMIC registers are configured for standby transition (*initTLF35584()*)
- The module I2C for the communication between the RTC and the OPTIREG™ PMIC TLF3558x (only for the Application Kit) (*init_I2C_module()*is initialized
- The standby controller (*standbyControllerInit()*) must be initialized
- The standby register of the board is configured to go to standby *(standbyInit()*)
- For configuring the QSPI module, depending on the selected board, the SPI pins need to be associated with the right port pins on the board in the *SPI_init.c* file.

```
/* Select the port pins for communication */
const IfxQspi_SpiMaster_Pins qspi2MasterPins = {
   // &IfxQspi2_SCLK_P15_8_OUT, IfxPort_OutputMode_pushPull,          /* SCLK Pin                        (CLK)      */
    &IfxQspi2_SCLK_P15_3_OUT, IfxPort_OutputMode_pushPull,
    &IfxQspi2_MTSR_P15_6_OUT, IfxPort_OutputMode_pushPull,          /* MasterTransmitSlaveReceive pin (MOSI)     */
    &IfxQspi2_MRSTB_P15_7_IN, IfxPort_InputMode_pullDown,           /* MasterReceiveSlaveTransmit pin (MISO)     */
    IfxPort_PadDriver_cmosAutomotiveSpeed3                          /* Pad driver mode                           */
};
spiMasterConfig.pins = &qspi2MasterPins;                            /* Assign the Master's port pins             */
```

**Figure 18 SPI Initialization function**

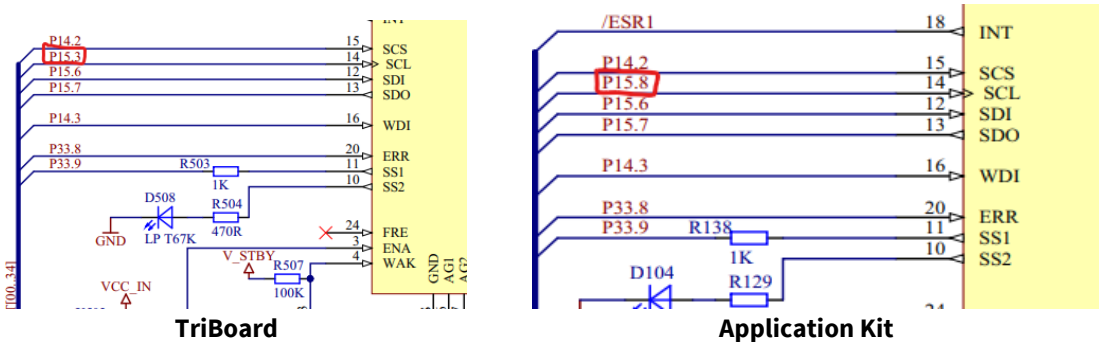


**Figure 19 SPI schematics**

- For the TLF3558x, the error pin monitor and the watchdog are disabled to avoid conflict with the main program. In the *STANDBYConfig* function, the wake timer can be enabled and configured via the registers Device configuration (DEVCFG0) and Wake timer configuration (WKTIMCFG0). The current threshold for QUC voltage is selected via register DEVCFG2 (max. 100 mA).

```
void STANDBYConfig(tlf35584 *tlfDevice){

    tlf35584SpiFrameType spiFrame1, spiFrame2, spiFrame3, spiFrame4, spiFrame5, spiFrame6;

    spiFrame1.U = transferDataTLF35584(SpiCommand_read, DEVCFG0RegAddr, (uint8)DUMMY_DATA);
    spiFrame2.U = transferDataTLF35584(SpiCommand_read, DEVCFG1RegAddr, (uint8)DUMMY_DATA);
    spiFrame3.U = transferDataTLF35584(SpiCommand_read, DEVCFG2RegAddr, (uint8)DUMMY_DATA);

    spiFrame4.U = transferDataTLF35584(SpiCommand_read, Wktimcfg0, (uint8)DUMMY_DATA);
    spiFrame5.U = transferDataTLF35584(SpiCommand_read, Wktimcfg1, (uint8)DUMMY_DATA);
    spiFrame6.U = transferDataTLF35584(SpiCommand_read, Wktimcfg2, (uint8)DUMMY_DATA);

    tlfDevice->DEVCFG0.U = spiFrame1.B.data;
    tlfDevice->DEVCFG1.U = spiFrame2.B.data;
    tlfDevice->DEVCFG2.U = spiFrame3.B.data;
    tlfDevice->WKTIMCFG0.U = spiFrame4.B.data;
    tlfDevice->WKTIMCFG1.U = spiFrame5.B.data;
    tlfDevice->WKTIMCFG2.U = spiFrame6.B.data;

    tlfDevice->DEVCFG0.B.WKTIMCYC = Enable;
    tlfDevice->DEVCFG0.B.WKTIMEN = Enable;
    tlfDevice->DEVCFG0.B.TREDL = 0xF;
    tlfDevice->DEVCFG2.B.CMONEN = Enable;
    tlfDevice->DEVCFG2.B.CTHR = 0x3;
    tlfDevice->WKTIMCFG0.B.TIMVALL = 0xF4;
    tlfDevice->WKTIMCFG1.B.TIMVALM = 0x01;

    transferDataTLF35584(SpiCommand_write, DEVCFG0RegAddr, tlfDevice->DEVCFG0.U);
    transferDataTLF35584(SpiCommand_write, DEVCFG2RegAddr, tlfDevice->DEVCFG2.U);
    transferDataTLF35584(SpiCommand_write, Wktimcfg0, tlfDevice->WKTIMCFG0.U);
    transferDataTLF35584(SpiCommand_write, Wktimcfg1, tlfDevice->WKTIMCFG1.U);
}
```

WAKE TIMER

WAKE timer reload value

**Figure 20 Wakeup timer**

- For the Application Kit, the signal connecting to the WAK port must be low
- As previously mentioned, to put WAK to a low state, the pin MFP of the RTC IC needs to be set to ‘1’
- The function *read_ext_device_address()* in the *I2C_Read_Ext_Device.c* file uses the I2C connection of the microcontroller to set the pin MFP of the RTC to ‘1’

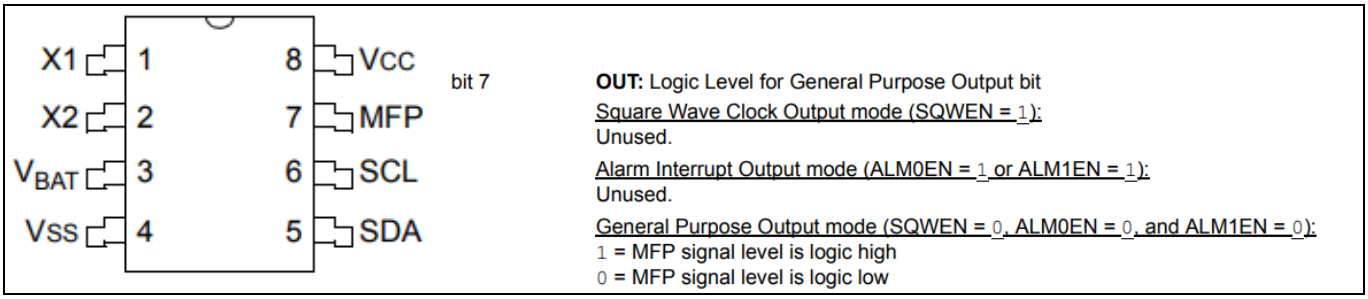


**Figure 21 RTC component pinout**

The bit 7 of register 0x07 in the RTC must be set to ‘1’ using the function *read_ext_device_address().*

```
void read_ext_device_address(void)
{
    /* Initialize register address and the message */
    uint8 i2cTxBuffer = ADDRESS_OF_REGISTER;
    uint8 i2cMessage[2] = {ADDRESS_OF_REGISTER, DATA};

    /* Write data to device */
    while(IfxI2c_I2c_write(&g_i2cDevEeprom, &i2cMessage[0], WRITING_LENGTH) == IfxI2c_I2c_Status_nak);

    /* Read the register */
    while(IfxI2c_I2c_write(&g_i2cDevEeprom, &i2cTxBuffer, LENGTH_OF_ADDRESS) == IfxI2c_I2c_Status_nak);
    while(IfxI2c_I2c_read(&g_i2cDevEeprom, &g_macAddr[0], LENGTH_OF_ADDRESS) == IfxI2c_I2c_Status_nak);
}
```

**Figure 22 RTC setup function**

- For the standby controller, the SCR needs to be enabled with the function *IfxScr_enableSCR().* The SCR program is copied into the XRAM with the function *IfxScr_copyProgram()*, then the XRAM is programmed with the function *IfxScr_init(n),* where n is the boot mode of register PMSWCR4. The pin P33 is enabled for SCR via register P33_PCSR.

```
/* activation of the stand by controller */
void standbyControllerInit(void) {

    IfxScuWdt_clearSafetyEndinit(IfxScuWdt_getCpuWatchdogPassword());
    IfxScr_disableSCR();
    //enable SCR
    IfxScr_enableSCR();
    IfxScuWdt_setSafetyEndinit(IfxScuWdt_getCpuWatchdogPassword());

    IfxScuWdt_clearSafetyEndinit(IfxScuWdt_getCpuWatchdogPassword());
    //boot_mode=0 - XRAM not programmed
    IfxScr_init(0);
    IfxScuWdt_setSafetyEndinit(IfxScuWdt_getCpuWatchdogPassword());

    // Write SCR program to XRAM
    IfxScr_copyProgram();

    IfxScuWdt_clearSafetyEndinit(IfxScuWdt_getCpuWatchdogPassword());
    //boot_mode=1 - XRAM programmed
    IfxScr_init(1);
    IfxScuWdt_setSafetyEndinit(IfxScuWdt_getCpuWatchdogPassword());

    if(SCU_RSTSTAT.B.STBYR)
    {
       IfxScuWdt_clearCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
       SCU_RSTCON2.B.CLRC = 1;
       IfxScuWdt_setCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
    }

    while(SCU_RSTSTAT.B.STBYR){}

    /* Set P33 under the control of SCR */
    uint8 mode = 3;
    pinEnable(mode);
}
```

**Figure 23 SCR setup function**

For the standby configuration, on the AURIX™ TC397 microcontroller side, the register PMSWCR0 must be modified. To allow the board to go to standby on VEXT ramp-down (PMIC goes to standby), the bits Standby Wake-up Enable on VEXT Supply ramp-up (PWRKEN) and Standby Entry on VEXT Supply ramp-down (VEXTSBYEN) are enabled, and all the other Wake-up enable (WKEN) bit fields are disabled. The bit Standby Controller Wake-up enable from Standby (SCRWKEN) allows the SCR to be woken up in standby state. The Blanking Filter delay for Wake-up (BLNKFIL) and Standby RAM supply in Standby Mode (STBYRAMSEL) should also be configured according to the application needs.

**PMSWCR0**
**Standby and Wake-up Control Register 0 ($00B4_H$) LVD Reset Value: $0010\ 02D0_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| WUTWKEN | PORSTWKEN | SCRWKEN | PWRWKEN | PINBWKEN | PINAWKEN | ESR1WKEN | ESR0WKEN | BLNKFIL | | | | 0 | STBYRAMSEL | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | | | | r | rw | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PINBEDCON | | PINBDFEN | PINAEDCON | | PINADFEN | ESR1EDCON | | ESR1DFEN | ESR0EDCON | | ESR0DFEN | VDDSTBYEN | VEXTSTBYEN | 0 | |
| rw | | rw | rw | | rw | rw | | rw | rw | | rw | rw | rw | r | |

**Figure 24 PMSWCR0 fields**

## 5.4 State function

- ***goToNormal()*:** This function sets the microcontroller in a normal power consumption mode if it was not before (*highPower()*). The transition is made by the *setSystemStatusFlagsTLF35584(STATE)* function. If the transition fails, the program is stuck in a loop, and a PORST is required.

```
void goToNormal(void)
{
    /* Set the microcontroller in normal configuration in case it was working in Low Power mode */
    if (powerMode == 1){
        highPower();
        }
    /* clear the status and wakeup flag */
    if (getSystemStatusFlagsTLF35584(&g_tlfDevice) != 0)
        {
            clearSystemStatusFlagsTLF35584(&g_tlfDevice);
            clearWakeupStatusFlagsTLF35584(&g_tlfDevice);
        }

  //  IfxStm_waitTicks(BSP_DEFAULT_TIMER, IfxStm_getTicksFromMicroseconds(BSP_DEFAULT_TIMER, TLF_INIT_DELAY_TIME_US));

    /* go to normal and verify if the transition succeed, block in loop instead */
    setStateTransitionTLF35584(&g_tlfDevice, DeviceStateTransition_normal);
    if (getSystemStatusFlagsTLF35584(&g_tlfDevice) != 0)
        {
            while(1);
        }
    if(getCurrentStateTLF35584(&g_tlfDevice) != DeviceStateTransition_normal)
        {
            while(1);
        }
}
```

**Figure 25 Microcontroller transition to normal state function**

- ***goToStandby()*:** This function sets the board in a low-power consumption mode (*lowPower()*). The transition is made by the setSystemStatusFlagsTLF35584(*STATE*) function.

```
void goToStandby(void)
{
    /* clear the status and wakeup flag */
    if (getSystemStatusFlagsTLF35584(&g_tlfDevice) != 0)
        {
            clearSystemStatusFlagsTLF35584(&g_tlfDevice);
            clearWakeupStatusFlagsTLF35584(&g_tlfDevice);
        }

    /* Low Power activation */
    lowpower();
    powerMode = 1;

    /* go to stand by */
    IfxStm_waitTicks(BSP_DEFAULT_TIMER, IfxStm_getTicksFromMicroseconds(BSP_DEFAULT_TIMER, TLF_NORMAL_TRANSITION_TIME_US));

    setStateTransitionTLF35584(&g_tlfDevice, DeviceStateTransition_standby);

}
```

**Figure 26 Microcontroller transition to standby state function**

In the setSystemStatusFlagsTLF35584(*STATE*) function, the *STATE* variable is the number associated with the PMIC state, as follows:

```
typedef enum
{
    DeviceStateTransition_none     = 0,        /* NONE     */
    DeviceStateTransition_init     = 1,        /* INIT     */
    DeviceStateTransition_normal   = 2,        /* NORMAL   */
    DeviceStateTransition_sleep    = 3,        /* SLEEP    */
    DeviceStateTransition_standby  = 4,        /* STANDBY  */
    DeviceStateTransition_wake     = 5,        /* WAKE     */
    DeviceStateTransition_reserved = 6,        /* RESERVED */
} statereqType;
```

**Figure 27 PMIC states definition**

## 5.5 Main loop

In this code example, the main program (*CoreConf()*) is accessed by triggering the ASCLIN0 reception interrupt (*asclin0_Rx_ISR*). If the UART connection via USB is used, pressing a key on the keyboard will activate the interrupt.

```
void CoreConf (void)
{
    // User enter the loop when they press a key on the keyboard
    if (mode == 1)
    {
        /* Clear the wakeup flag */
        if(PMS_PMSWSTAT2.B.PWRWKP == 1){

            IfxScuWdt_clearSafetyEndinit(IfxScuWdt_getSafetyWatchdogPassword());
            IfxScuWdt_clearCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
            PMS_PMSWSTATCLR.B.PWRWKPCLR = 1;
            IfxScuWdt_setSafetyEndinit(IfxScuWdt_getSafetyWatchdogPassword());
            IfxScuWdt_setCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
        }
        /* go to standby or sleep pattern */
        printf("Go to normal\n\r");
        goToNormal();
        printf("%i\n\r\n\r",getCurrentStateTLF35584(&g_tlfDevice));
        printf("Go to standby\n\r");
        goToStandby();
        printf("%i\n\r\n\r",getCurrentStateTLF35584(&g_tlfDevice));
     //   GigaSTATUSView(&g_tlfDevice);
        mode = 0;
    }
}
```

**Figure 28 Main loop**

The main program clears the status flag, then the PMIC changes its state. To go to standby state, the function *goToNormal()* is called first and then *goToStandby()* is called (cf PMIC state machine).

### 5.5.1 Low power mode

As presented above, on the Application Kit AURIX™ TC397 and the TriBoard AURIX™ TC397, the embedded hardware requires too much current from QUC to fall under the threshold “ILDO_µC”, which prevents the PMIC from changing its state. To get the PMIC in low-power mode, some settings need to be performed to reduce the global power consumption of the board (*lowPower*()). To reduce the consumption, before going into standby or sleep mode, all the CPUs, except CPU0, are put in idle mode.

```
//CPU settings
IfxScuWdt_clearSafetyEndinitInline(IfxScuWdt_getSafetyWatchdogPasswordInline());
IfxScuWdt_clearCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
IfxCpu_setCoreMode(&MODULE_CPU1, IfxCpu_CoreMode_idle);
IfxCpu_setCoreMode(&MODULE_CPU2, IfxCpu_CoreMode_idle);
IfxCpu_setCoreMode(&MODULE_CPU3, IfxCpu_CoreMode_idle);
IfxCpu_setCoreMode(&MODULE_CPU4, IfxCpu_CoreMode_idle);
IfxCpu_setCoreMode(&MODULE_CPU5, IfxCpu_CoreMode_idle);
IfxScuWdt_setSafetyEndinitInline(IfxScuWdt_getSafetyWatchdogPasswordInline());
IfxScuWdt_setCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
```

**Figure 29 CPU’s settings to reduce the power consumption**

The clock frequency of the microcontroller and the peripherals can also decrease the power consumption if they are set lower or turned off. The unused modules and peripherals are turned off, or their clock frequency is reduced.

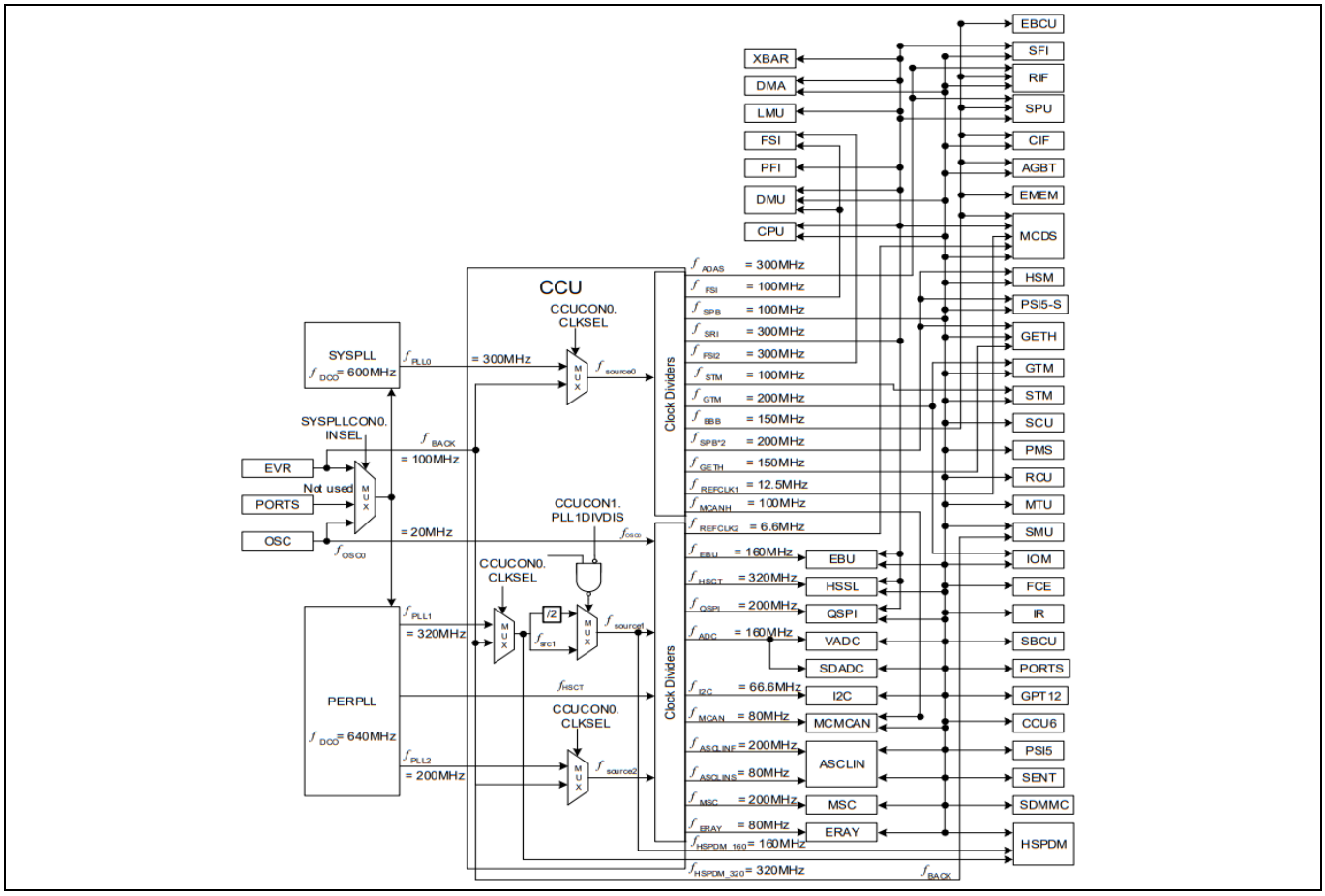


**Figure 30 Schematic of clocking system of the AURIX™ TC3xx**

The System PLL (SYSPLL) generates the frequency for the main modules, while the Peripheral PLL (PERPLL) provides the frequency for the peripherals. In order to reduce their frequency, the divider must be set to its maximum value.

```
//PLL settings
IfxScuWdt_clearSafetyEndinitInline(IfxScuWdt_getSafetyWatchdogPasswordInline());
IfxScuWdt_clearCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
SCU_SYSPLLCON0.U = 0x40013A00;
while(SCU_SYSPLLSTAT.B.K2RDY == 0){}
SCU_SYSPLLCON1.B.K2DIV = 0x7;
SCU_PERPLLCON1.B.K2DIV = 0x7;
IfxScuWdt_setSafetyEndinitInline(IfxScuWdt_getSafetyWatchdogPasswordInline());
IfxScuWdt_setCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
```

**Figure 31 PLL settings for reduce frequency**

The CCUCON registers can control and turn off the main and peripheral handles of the PLLs. In this document, only the QSPI (communication with PMIC) and the ASCLIN (display for the terminal) modules are needed, so they will be enabled. The other modules are stopped.

```
//Peripherals settings
IfxScuWdt_clearSafetyEndinitInline(IfxScuWdt_getSafetyWatchdogPasswordInline());
IfxScuWdt_clearCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
SCU_CCUCON0.U = 0x170F0F00;
SCU_CCUCON1.U = 0x1F000000;
SCU_CCUCON2.U = 0x05000101;
SCU_CCUCON5.U = 0x00000000;
SCU_CCUCON6.U = 0x0000003F;
IfxScuWdt_setSafetyEndinitInline(IfxScuWdt_getSafetyWatchdogPasswordInline());
IfxScuWdt_setCpuEndinit(IfxScuWdt_getCpuWatchdogPassword());
```

**Figure 32 Peripherals settings**

**CCU Clock Control Register 1**

**CCUCON1**
**CCU Clock Control Register 1** **(0034$_H$)** **System Reset Value: 1010 0302$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LCK | 0 | CLKSELQSPI | | QSPIDIV | | | | 0 | | CLKSELMSC | | MSCDIV | | | |
| rh | rw | rw | | rw | | | | rw | | rw | | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | I2CDIV | | | | PLL1DIVDIS | 0 | CLKSELMCAN | | MCANDIV | | | |
| rw | | | | rw | | | | rw | rw | rw | | rw | | | |

**Figure 33 QSPI bitfield in CCUCON1 should not be modified**

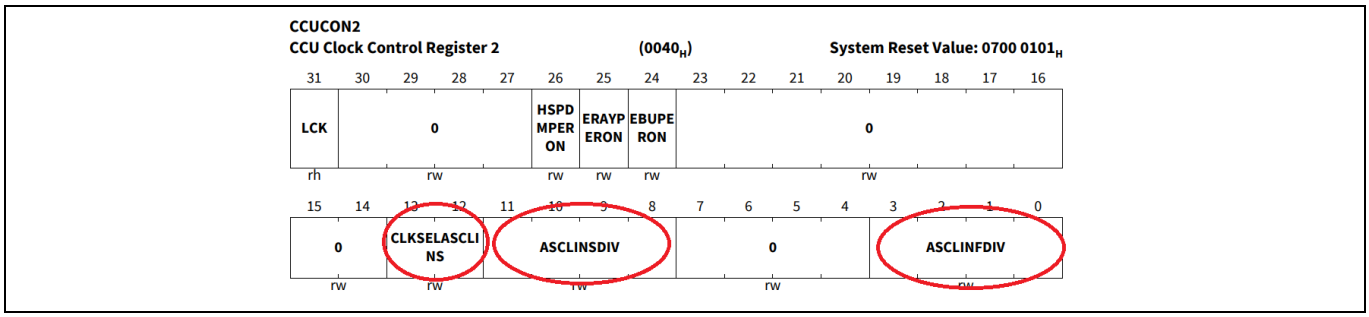


**Figure 34 ASCLIN bitfield in CCUCON2 should not be modified**

## 5.6 Standby Controller

After enabling the SCR and booting the main code in SCR XRAM, the main SCR program can be executed. Each register is organized into pages, before accessing a register, the page must be updated (SCU_PAGE).

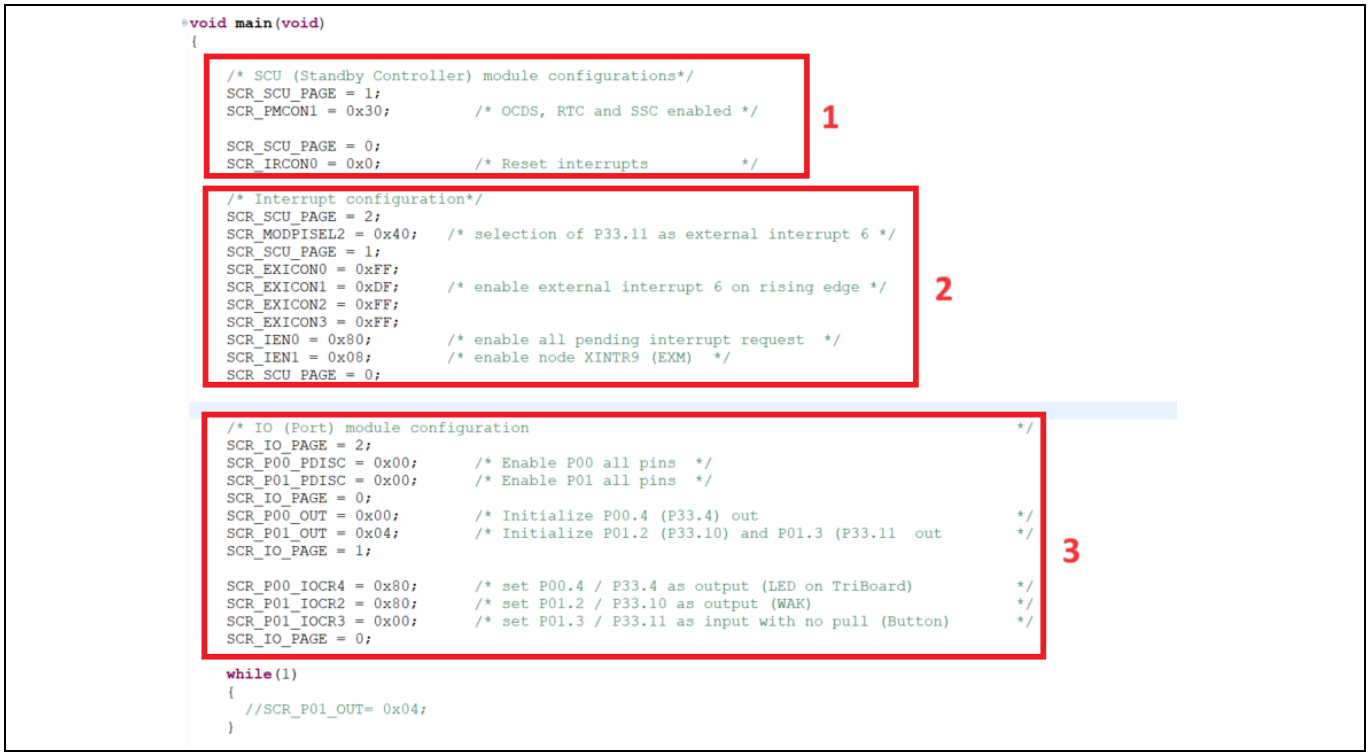


**Figure 35 Main loop**

The initialization process involves the following steps:

1. Enable the main peripherals except the watchdog and LIN
2. Configure of the external interrupt
3. Configure of the external port

For the interrupt link to the button, the external interrupt 5 (EXINT5) is used. Configuration steps are the following:

1. Enable EXINT5 (MODPISEL2), which is linked to P33.10
2. Select the trigger event, for example, a rising edge (EXICON1)
3. Enable the node XINTR9, which is linked to interrupt 5 (IEN1 and IEN0)

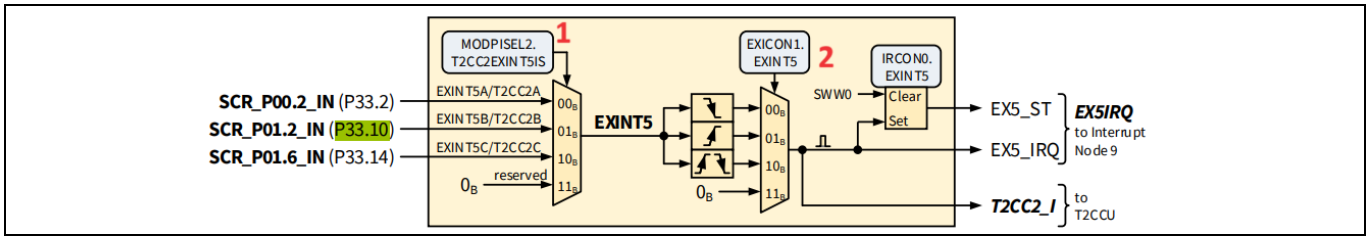


**Figure 36 External interrupt 5 connections**

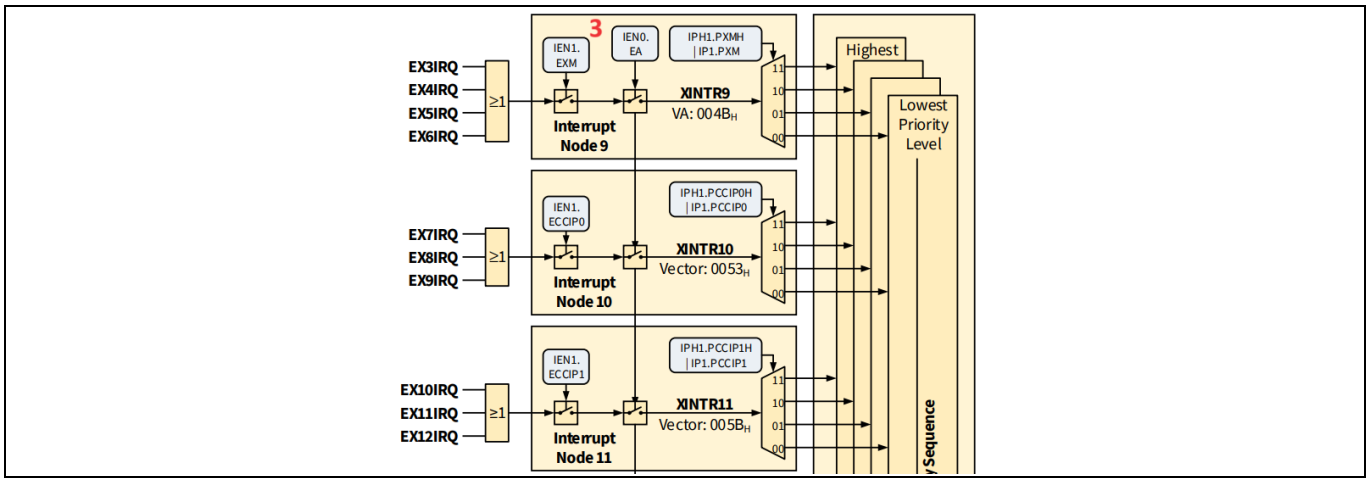


**Figure 37 Interrupt node 9**

In the case of interrupt 6 function, the state of the pin P33.10 can be toggled with the command: "*SCR_P01_OMTR = 0x04*".

```
//interrupt function which toggle the sate of the PIN 33.10
void ex6_interrupt(void) __interrupt(XINTR9)
{
    unsigned char ioPage, scuPage;
    int count = 0;

    scuPage = SCR_SCU_PAGE;
    SCR_SCU_PAGE = 0;
    ioPage = SCR_IO_PAGE;
    SCR_IO_PAGE = 0;
    SCR_IRCON0 = (1 << 6);

    SCR_P01_OMTR = 0x04;

    SCR_IO_PAGE = ioPage;
    SCR_SCU_PAGE = scuPage;

}
```

**Figure 38 SCR interrupt function**

# 6 State transition related measurement

In order to verify if the state transition is done correctly on the TriBoard, the WAK signal, the V_uC signal, and the SS2 signal will be measured.

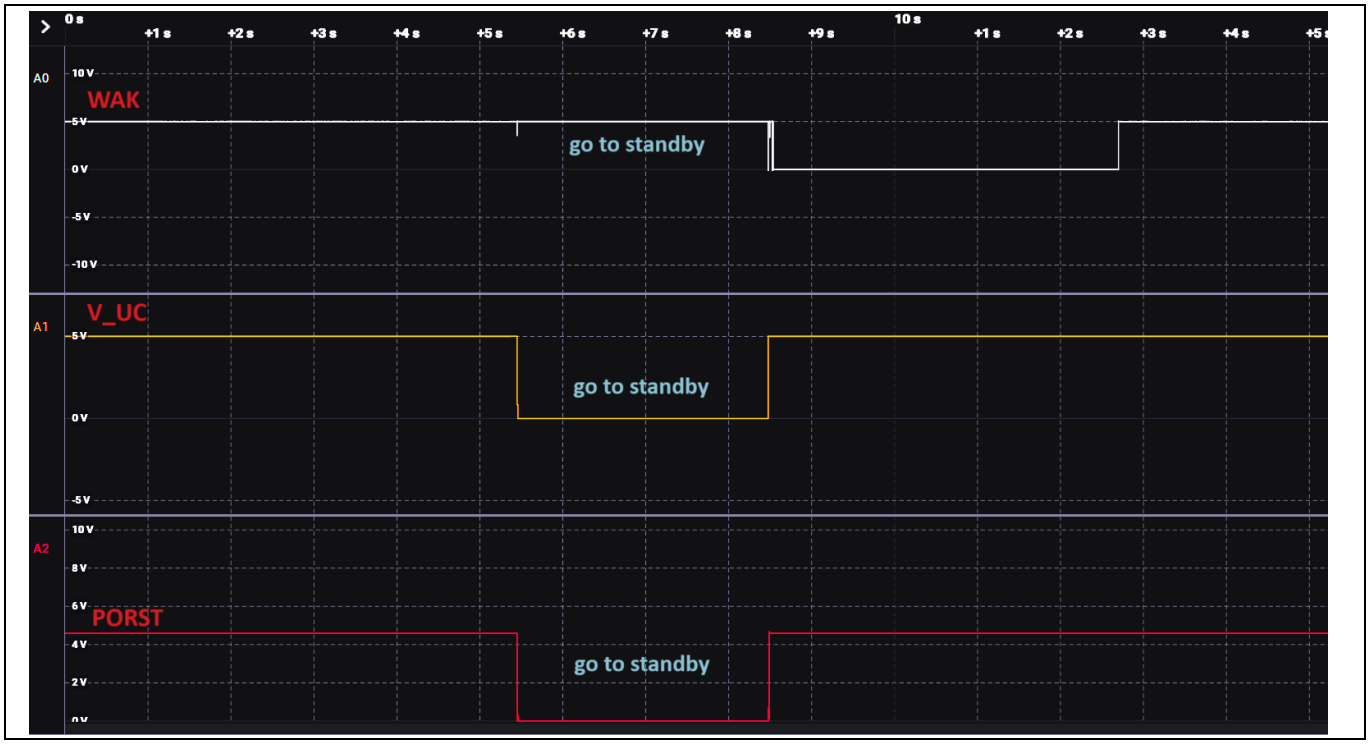


**Figure 39 Transition to standby**

## 6.1 Time measurement

During the transition from NORMAL to STANDBY, the SS2 signal and V_uc (QUC)should be turned OFF (0 V). To measure the transition time from NORMAL to STANDBY, the state of the GPIO pin is visualized (P33.13 alias “PIN_state” on screenshots). When the SS2 signal is turned OFF, the transition starts, and when the pin goes to low state, it means that the board entered standby.

During the state switch from STANDBY to NORMAL, the WAKE signal triggers the transition when it is a in low state, then the V_uc signal is turned ON again. To measure the transition time from STANDBY to INIT, the state of a GPIO pin is visualized (P33.13 alias “PIN_state” on screenshots). When the WAKE signal is triggered, the transition begins, and when the pin is set to a high state, that means the board has been initialized and is in an operative state, ready to execute the main program.

## 6.2 Transition from NORMAL to STANDBY

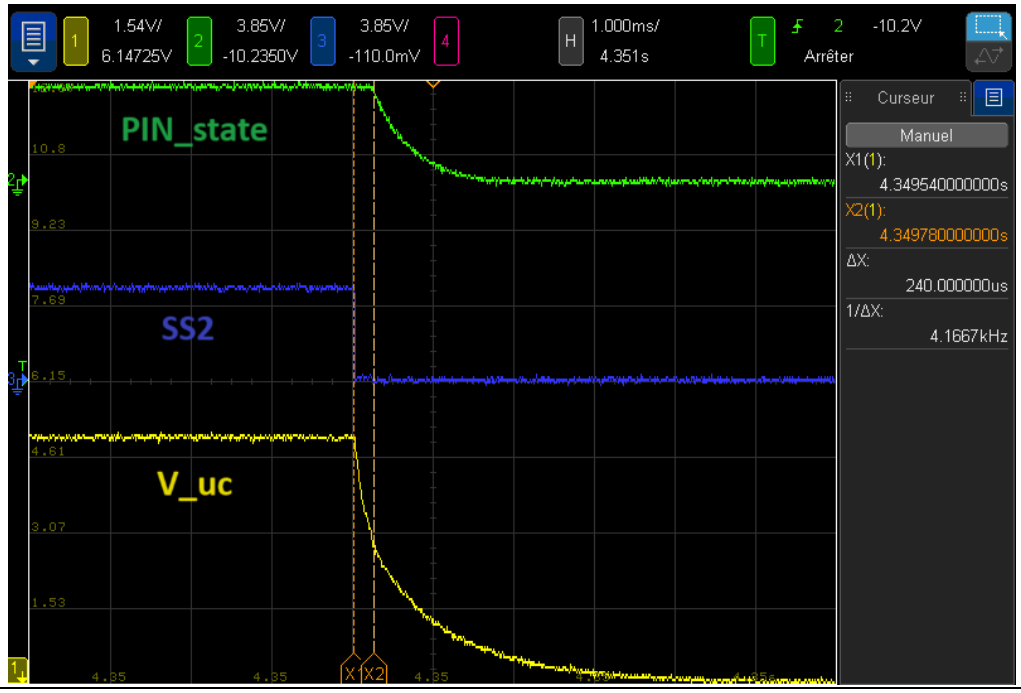


**Figure 40** **AURIX™ RUN to STANDBY - Transition from normal state to standby state triggered by SS2 PMIC pin**

## 6.3 Transition from STANDBY to RUN

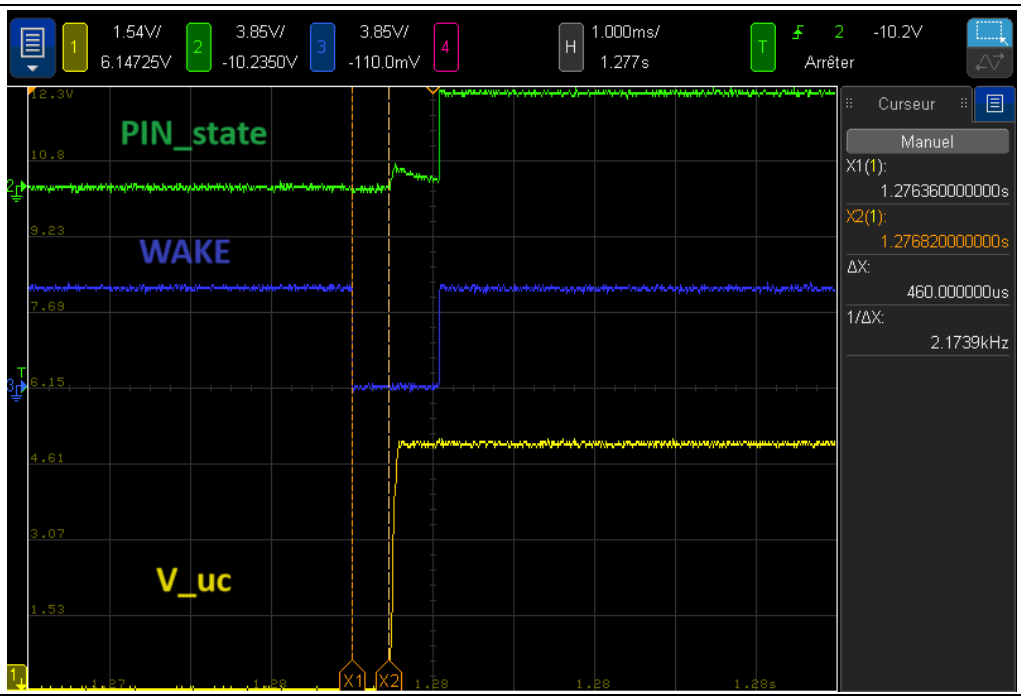


**Figure 41** **STANDBY to PMIC INIT - PMIC wakeup time measurement after WAKE signal is triggered**

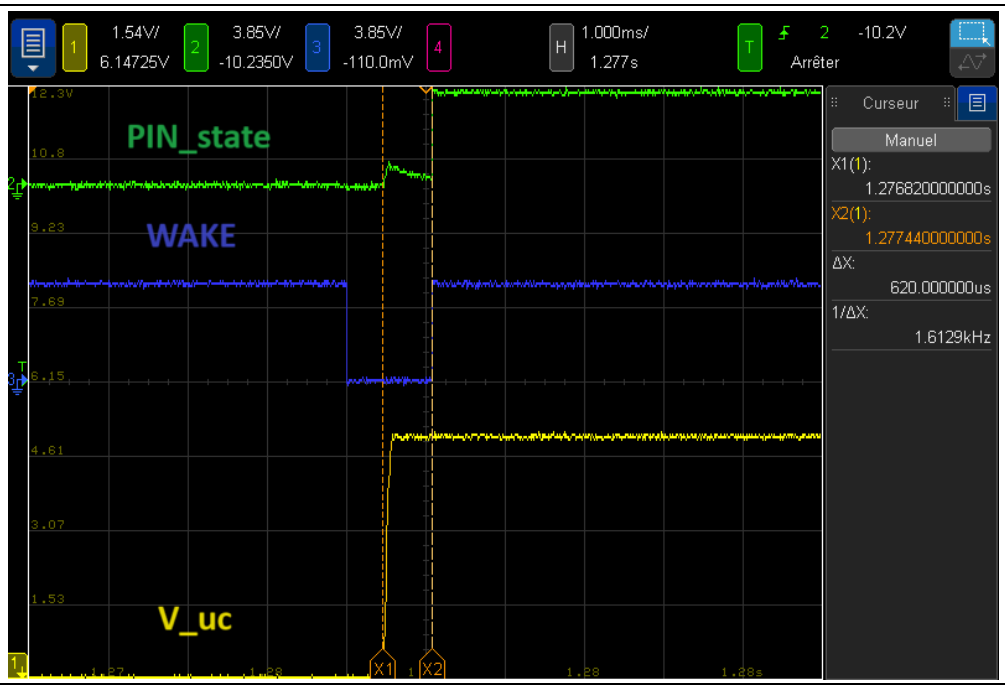


**Figure 42** **PMIC INIT to AURIX™ INIT - time after V-uC (VEXT) rising edge triggers AURIX™ wakeup**

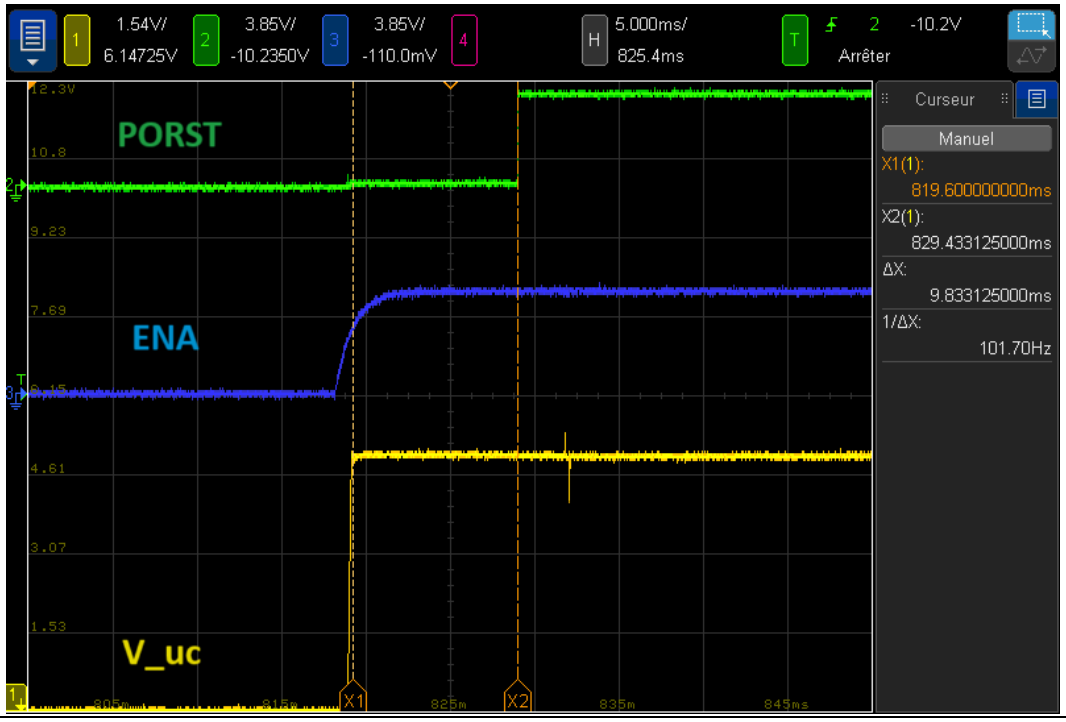


**Figure 43** **PMIC INIT to AURIX™ RUN - Microcontroller startup time after V-uC (VEXT) rising edge triggers the wakeup**

**Table 1 Transition times**

| Transition | Transition time (us) |
|---|---|
| AURIX™ RUN to STANDBY | 240 |
| STANDBY to PMIC INIT | 460 |
| PMIC INIT to AURIX™ INIT | 620 |
| PMIC INIT to AURIX™ RUN | 9833 |

*Note: The pin P33.13 is a fast pin. A slow pin will need more time for the transition.*

## 6.4 Current measurement

On the TriBoard, the overall consumption of the board, provided by the main 12 V CD power line, is 184 mA when the board is in normal mode (with the power consumption reduced as described in Section 3.1 – LDO_uC (QUC / VEXT) current), and 0.5 mA in standby mode with the standby controller active and only two pins controlled by the SCR.

## References

[1] Infineon Technologies AG: *AURIX™ TC3xx user manual vol 2 part 1*; Available online

[2] Infineon Technologies AG: *AURIX™ TC3xx user manual vol 2 part 2*; Available online

[3] Infineon Technologies AG: *TLF35584QVHSx datasheet*; Available online

[4] Microchip: *MCP79411 datasheet*; Available online

[5] Infineon Technologies AG: *TriBoard Manual TC3X7*; Available online

[6] Infineon Technologies AG: *Application kit Manual TC3X7*; Available online

[7] Infineon Technologies AG: *Community - Standby in tc397 via VEXT ramp down*; Available online

## Revision history

| Document revision | Date | Description of changes |
|---|---|---|
| V 1.0 | 2025-02-07 | Initial release |

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

**Edition 2025-02-07**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**



**Do you have a question about this document?**

**Email:** erratum@infineon.com

**Document reference**

**AN0022**

**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.

**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.