AN14744

S32M27 Periodic Interrupt Timer (PIT) Example in S32 Design Studio

Rev. 1.0 — 1 July 2025 Application note

Document information

Information	Content
Keywords	NXP, S32M27, S32 Design Studio, PIT, LLD, RTD
Abstract	Periodic Interrupt Timer (PIT) implementation using Low Level Drivers (LLD) and Real-Time Drivers (RTD) is a key feature for time-critical applications on S32M2 microcontrollers. PIT enhances software precision by offering consistent time bases for task execution. With the LLD/ RTD architecture, developers gain low-latency response and grater control over pheripheral configuration. This application note demostrates the setup and operation of the PIT module using AUTOSAR-compliant RTDs emphasizing its importance in scheduling, task triggering, and real-time behaviour management across power domains. This strengthens the reliability and predictability of time-driven applications on the S32M2 platform.

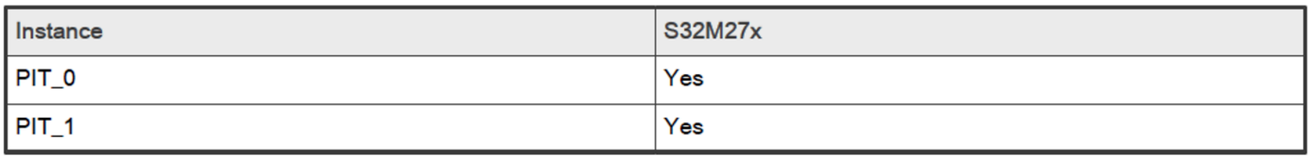
1 Introduction

The S32M2 microcontroller family from NXP is a highly integrated solution tailored for 12V motor control applications. Built on a system-in-package (SiP) design, it combines high-voltage analog components, such as MOSFET gate pre-drivers, LIN/CAN FD interfaces, and voltage regulators, with a robust embedded MCU core based on the Arm® Cortex®-M4 or M7. This architecture supports functional safety up to ISO 26262 ASIL B and enables advanced motor diagnostics, noise reduction algorithms, and seamless firmware-over-the-air (FOTA) updates through the S32 Automotive Platform.

Within this platform, the Periodic Interrupt Timer (PIT) module enhances real-time control through two instances, each with four 32-bit timers. These timers generate periodic triggers for motor control peripherals and support high-resolution interrupts down to 1 µs. The PIT also includes a Real-Time Interrupt (RTI) function that operates independently—even in low-power Stop mode—making it ideal for time-critical wake-up tasks. Its features support DMA triggering, power-efficient operation, and flexible timing configurations suited for embedded motor control.

2 PIT design

The S32M27 microcontroller features two instances of the Periodic Interrupt Timer (PIT), each comprising four 32-bit timer channels. All PIT instances can generate periodic triggers, which are routed to motor control IPs such as eMIOS, LCU, BCTU, and ADC through TRGMUX.



Instance	S32M27x
PIT_0	Yes
PIT_1	Yes

Figure 1. S32M27 PIT Instances

The module includes a Real-Time Interrupt (RTI) capability, which allows configuring a timer resolution of 1 microsecond independently from other timers. This is achieved by programming LDVAL0–LDVAL3 to 48 when using a 48 MHz FIRC system clock or to 40 for a 40 MHz clock.

General-purpose PIT timers operate on the peripheral bus clock, while the RTI timer runs on an independent oscillator that continues even in Stop mode—enabling the RTI to periodically wake the system. PITn uses AIPS_SLOW_CLK, whereas RTI (within PIT0) uses SIRC_CLK. Key features include one RTI timer for CPU wake-up, the ability to generate DMA trigger pulses and maskable interrupts, support for RTI interrupts even with the bus clock off, and power-saving via the separate RTI input clock.

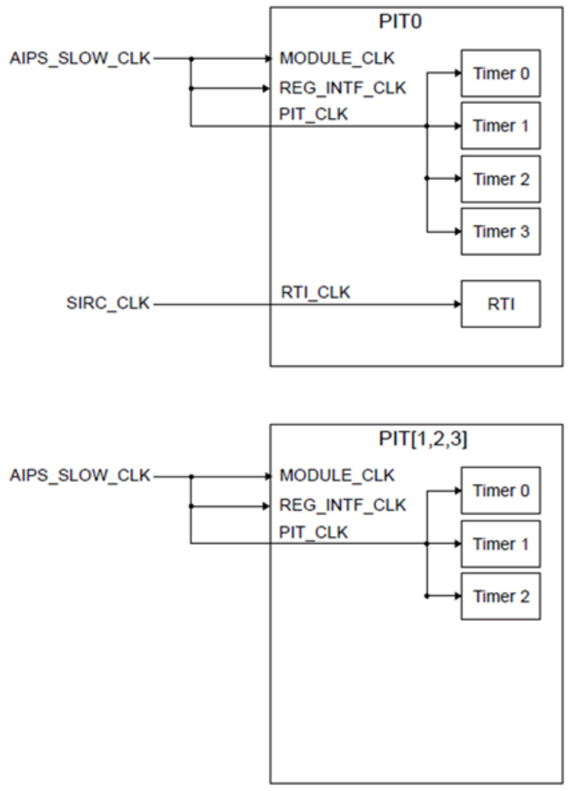


Figure 2. S32M27 PIT Clock

Each timer operates with independent timeout periods, utilizes a down counter, and can be chained into a 64-bit life timer. When enabled, timers use LDVAL registers to load start values, count down to zero, trigger pulses, and reload. The current counter value is accessible via CVAL registers.

Activation of the PIT module requires clearing MCR[MDIS], and timers can be frozen in Debug mode using MCR[FRZ]. Interrupts are enabled with TCTRLn[TIE], and the RTI flag (TIF) is set upon timeout and cleared by writing a 1 to TFLGn[TIF]. Timer periods can be restarted or modified by toggling TCTRLn[TEN], or adjusted on-the-fly by updating LDVAL, with changes applied after the next trigger event.

3 Code description

This project configures the PIT driver with the RTD 4.0.0 P01.

The RTD interface is used for the peripheral configuration.

The application will configure the PIT to trigger an interrupt every 1 second (this can be seen by the toggling of the USER_LED01).

For the implementation of this application, a **S32M27XEVB-C064** will be used.

Function Description

This function set the clock configuration according to pre-defined structure.

```
Clock_Ip_StatusType Clock_Ip_Init(Clock_Ip_ClockConfigType const * Config)
```

This function enable a clock for a given peripheral.

```
void Clock_Ip_EnableModuleClock(Clock_Ip_NameType ClockName)
```

This function configures the pins with the options provided in the given structure.

```
Siul2_Port_Ip_PortStatusType Siul2_Port_Ip_Init(uint32 pinCount, const
 Siul2_Port_Ip_PinSettingsConfig config[])
```

This function initializes the configured interrupts at interrupt controller level.

```
IntCtrl_Ip_StatusType IntCtrl_Ip_Init(const IntCtrl_Ip_CtrlConfigType *pIntCtrlCtrlConfig)
```

This function initializes the given PIT instance.

```
void Pit_Ip_Init(uint8 instance, const Pit_Ip_InstanceConfigType *config)
```

This function initializes the PIT channels.

```
void Pit_Ip_InitChannel(uint8 instance, const Pit_Ip_ChannelConfigType *chnlConfig)
```

This function enables the IRQ corresponding to the PIT timer channel.

```
void Pit_Ip_EnableChannelInterrupt(uint8 instance, uint8 channel)
```

This function starts the PIT timer channel.

```
Pit_Ip_StatusType Pit_Ip_StartChannel(uint8 instance, uint8 channel, uint32 countValue)
```

This function writes the given logical HIGH or LOW value to the specified pin ('0' represents LOW, '1' represents HIGH).

```
void Siul2_Dio_Ip_WritePin(Siul2_Dio_Ip_GpioType * const base, Siul2_Dio_Ip_PinsChannelType pin,
 Siul2_Dio_Ip_PinsLevelType value)
```

Project Creation

1. Download and open S32 Design Studio (S32DS) for S32 Platform version 3.6.2.
2. Download the S32M2xx Development Package version 3.6.0 from S32DS:
   a. Select Help > S32DS Extensions and Updates > S32M2xx development package
   b. Click on Install button and follow the instructions.
3. Select File > New > S32DS Application Project.
4. Write a project name without spaces. For example: *S32M2xx_PIT_Example*.
5. Open folder *Family S32M2xx*, select S32M276 in the *Processors* section and click next.
6. Click on the three dots (…) to select a Software Development Kit (SDK), click ok and click finish.

Pins ConfigTool

1. Expand your project folder in the *Project Explorer* view.
2. Double click on the <Project name>.mex file.
3. Click on the *Pins* view on the toolbar if not already selected.
4. In the pins tab, type PTD15 in the search bar and select the check box on the corresponding pin name to open pin alternatives.
5. Select alternative *SIUL2:gpio,111* > *Output* > *OK* > *Done*.
6. (Optional) In the Routing Details tab, is recommended to give a meaningful identifier to the pin and configure electrical characteristics as needed.

Peripherals ConfigTool

1. Click on the *Peripherals* view near the *Pins* view in the toolbar.

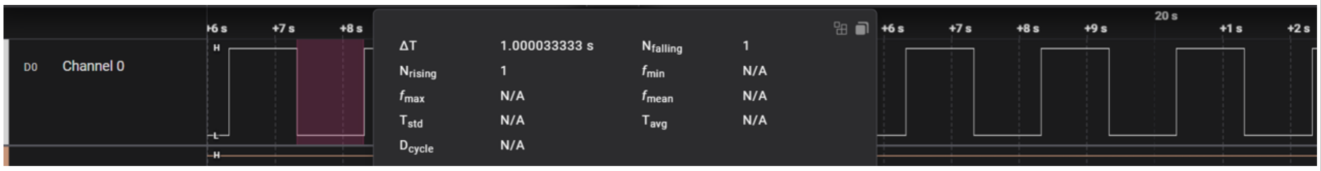
2. In the components tab at the left side, click the Drivers plus "+" button.
3. Select "All" in the components that should be offered in the top center.
4. Type *IntCtrl_Ip* in the search bar and doble click to add the component.
5. Add *Pit* component by repeating steps 2 to 4.
   a. Open Pit component by doble click.
   b. Click on *GptDriverConfiguration* tab in the PIT IPL Configuration tab.
   c. In *GptTimeoutMethod* select OSIF_COUNTER_DUMMY.
   d. Click on *GptHwConfiguration* tab.
   e. Select both check box for *GptHwConfiguration_1.*
   f. Click on *GptChannelConfigSet* tab.
   g. In *PitNotification*, type "Notification_PIT".
6. *Add Siul2_Dio* component by repeating steps 2 to 4.
7. Open *Siul2_Port*.
   a. Click on *PortConfigSet* tab.
   b. In *PortPin Mscr* type "111".
8. Open *IntCtrl_Ip*.
   a. Click on *General Configuration* tab.
   b. Disable *Development errors detection* check box.
   c. Click on *Interrupt Controller* tab.
   d. Add an item by clicking the plus "+" button.
   e. Add a new interrupt by clicking the plus "+" button in *PlatformIsrConfig*.
   f. For this example, select interrupt name *PIT0_IRQn*.
   g. Click on the Interrupt Enabled check box.
   h. In *Priority* type "3".
   i. In *Handler* type "PIT_0_ISR".
9. Update the code by clicking *Update Code* button in the toolbar and click OK.

Main Code

1. To go back to the main file click on *S32DS C/C++* in the toolbar.
2. Replace the auto-generated code with the following:

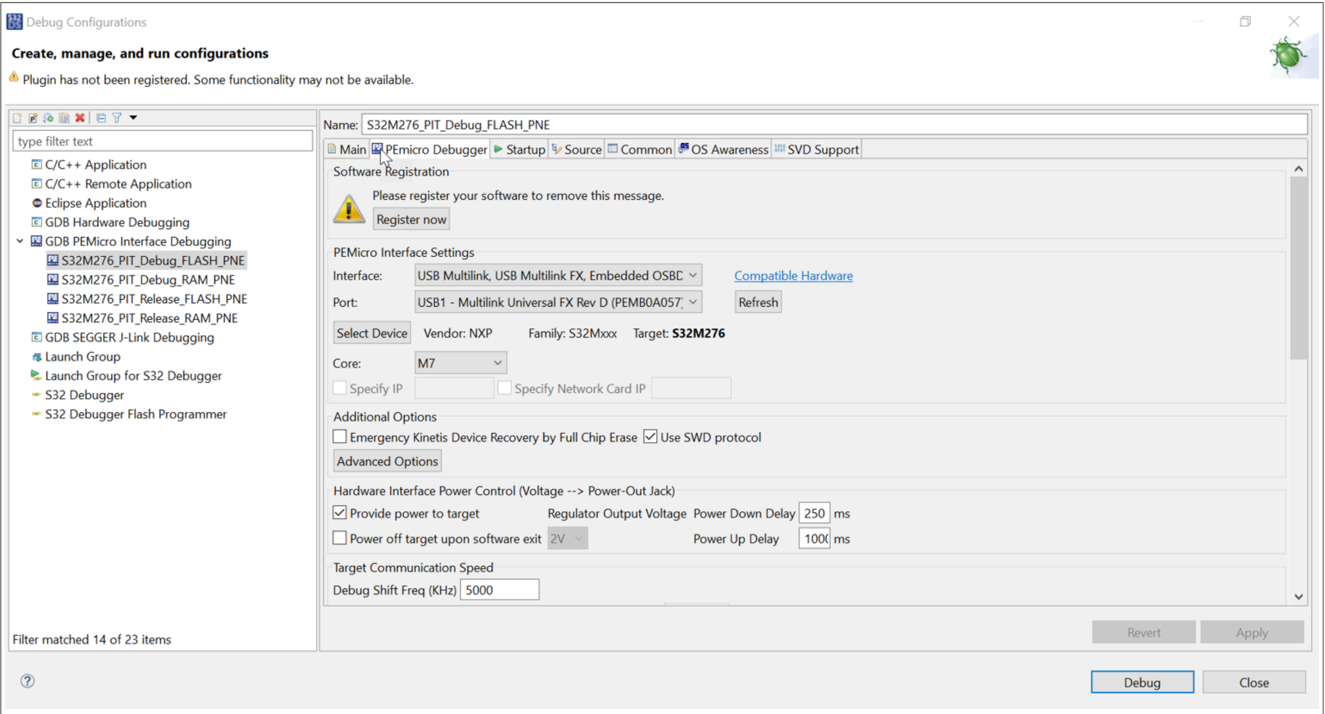
```
/* Including necessary configuration files. */
#include "Clock_Ip.h"
#include "IntCtrl_Ip.h"
#include "Siul2_Port_Ip.h"
#include "Siul2_Dio_Ip.h"
#include "Pit_Ip.h"

/* PIT instance used - 0 */
#define PIT_INST_0                    (0U)
/* PIT Channel used - 0 */
#define CH_0                          (0U)
/* PIT time-out period - equivalent to 1 s*/
#define PIT_PERIOD                    (30000000U)

#define clockConfig &Clock_Ip_aClockConfig[0]

/* Global flag updated in interrupt */
static volatile uint8 toggleLed = 0U;
static volatile uint8 flag = 0U;

void Notification_PIT(void)
{
    flag = 1;
}

int main(void)
{
    /* Init clock */
    Clock_Ip_Init(clockConfig);
    Clock_Ip_EnableModuleClock(PIT0_CLK);

    /* Initialize all pins using the Siul2_Port driver */
    Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS_PortContainer_0_BOARD_InitPeripherals,
    g_pin_mux_InitConfigArr_PortContainer_0_BOARD_InitPeripherals);

    /* Set PIT interupt */
    IntCtrl_Ip_Init(&IntCtrlConfig_0);

    /* Configure PIT */
    Pit_Ip_Init(PIT_INST_0, &PIT_0_InitConfig_PB);
    Pit_Ip_InitChannel(CH_0, PIT_0_ChannelConfig_PB);
    Pit_Ip_EnableChannelInterrupt(PIT_INST_0, CH_0);
    Pit_Ip_StartChannel(PIT_INST_0, CH_0, PIT_PERIOD);

    for(;;)
    {
        if(1 == flag)
        {
            toggleLed = 1 - toggleLed;
            Siul2_Dio_Ip_WritePin(LED_PORT, LED_PIN, toggleLed);
            flag = 0;
        }
    }
}
```

3. Click the hammer icon in the toolbar or "Ctrl+b" to build the project.
4. Connect the board.
5. Click the bug icon in the toolbar to debugg the project as the standard debug configuration.
6. (Optional) Check if Debug Configurations are as follows:
   a. Click on the arrow next to the debug icon.
   b. Select Debug Configurations.
   c. Expand GDB PEMicro Interface Debugging.
   d. Select <project name>_Debug_FLASH_PNE.
   e. Click on the *PEmicro Debugger* tab and review the options in the figure.
   f. Click on Apply and then Debug.
7. Click the play icon in the toolbar to run the application.



ΔT 1.000033333 s
$N_{falling}$ 1
N_{rising} 1
f_{min} N/A
f_{max} N/A
f_{mean} N/A
T_{std} N/A
T_{avg} N/A
D_{cycle} N/A

Figure 3. Application Demostration



Debug Configurations

Create, manage, and run configurations

Plugin has not been registered. Some functionality may not be available.

- C/C++ Application
- C/C++ Remote Application
- Eclipse Application
- GDB Hardware Debugging
- GDB PEMicro Interface Debugging
  - S32M276_PIT_Debug_FLASH_PNE
  - S32M276_PIT_Debug_RAM_PNE
  - S32M276_PIT_Release_FLASH_PNE
  - S32M276_PIT_Release_RAM_PNE
- GDB SEGGER J-Link Debugging
- Launch Group
- Launch Group for S32 Debugger
- S32 Debugger
- S32 Debugger Flash Programmer

Filter matched 14 of 23 items

Name: S32M276_PIT_Debug_FLASH_PNE

Main | PEmicro Debugger | Startup | Source | Common | OS Awareness | SVD Support

Software Registration

Please register your software to remove this message.

Register now

PEMicro Interface Settings

Interface: USB Multilink, USB Multilink FX, Embedded OSBD

Port: USB1 - Multilink Universal FX Rev D (PEMB0A057)

Select Device Vendor: NXP Family: S32Mxxx Target: **S32M276**

Core: M7

Specify IP Specify Network Card IP

Additional Options

☐ Emergency Kinetis Device Recovery by Full Chip Erase ☑ Use SWD protocol

Advanced Options

Hardware Interface Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target Regulator Output Voltage Power Down Delay 250 ms

☐ Power off target upon software exit 2V Power Up Delay 1000 ms

Target Communication Speed

Debug Shift Freq (KHz) 5000

Figure 4. Standard Debug Configuration

4 Conclusion

In conclusion, the **S32M27 microcontroller's PIT module** delivers a highly configurable and efficient timing solution tailored for real-time applications, especially in motor control systems. With dual PIT instances offering multiple 32-bit timers, an independent RTI for precise timekeeping even in low-power modes, and flexible clocking via TRGMUX, the module provides robust support for interrupt generation, DMA triggering, and low-power system wake-up. The ability to chain timers, update values dynamically, and operate autonomously during Stop mode makes the PIT a versatile and reliable component for timing-critical tasks. Overall, it's a powerful tool for enhancing system responsiveness and control precision.

5 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

6 Revision history

Table 1. Revision history

Document ID	Release date	Description
AN14744 v.1.0	01 July 2025	• Initial version

Legal information

Contents

Date of release: 1 July 2025
Document identifier: AN14744