AN12835
RT600 secure GPIO and usage

Rev. 1 — 25 June 2020 Application Note

Contents

1 Introduction

This application note explains how a secure GPIO must be used and configured in secure mode. It also demonstrates how a non-secure mode can access a peripheral's pin state regardless of the pin function whether the peripheral function is secure or non-secure which results in secure information leakage.

2 Overview

RT600 has secure GPIO module whose usage is closely related to normal GPIO, TrustZone, and secure AHB Controller. This section briefly introduces these functions. For more information on these functions, refer to RT600 User Manual.

2.1 TrustZone and secure AHB Controller

2.1.1 TrustZone

TrustZone for Armv8-M are available on all RT600 devices to protect secure resources from access by malicious code. Such secure resource may include secure memory blocks (code/data) and secure peripherals. It is achieved by segmentation of address space into either secure (S) or non-secure (NS). TrustZone can filter address access from CPU0 based on specific security attribute (S, NS) assigned to that address space.

As an example shown in Figure 1, CM33 CPU in secure state (CPU-S) can execute instructions from secure memory (S-memory), but is not allowed to directly execute instructions from non-secure memory (NS-memory). However, CPU-S can access data in both S-memory and NS-memory.

CPU-NS can execute instructions only from NS-memory and is not allowed to execute instructions from S-memory. CPU-NS can access data only in NS-memory but is not allowed to access data from S-memory.

In summary:

- NS application code "trust" that secure code do not corrupt or modify NS code or data inadvertently or on purpose to create malfunction or hazard.
- S application code does not "trust" NS application code and disallows access to a CPU-NS.

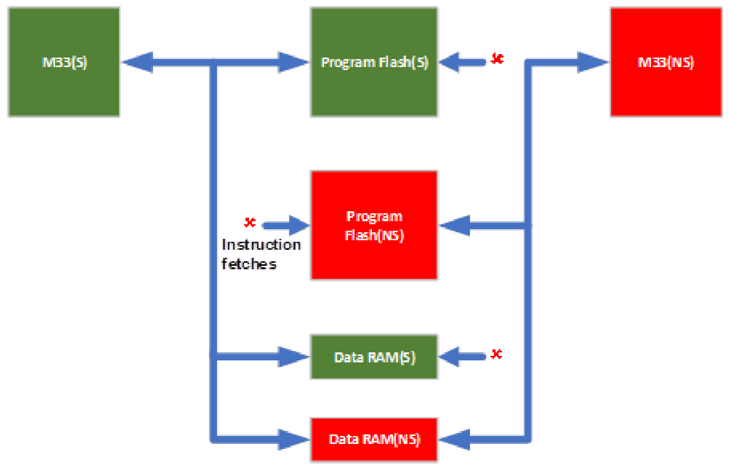


Figure 1. Secure state and non-secure state view

2.1.2 Secure AHB Controller

The RT600 implements second layer of protection with secure AHB Controller to provide secure trusted execution at system-level.

With secure AHB Controller, we can configure security access rules for each peripheral.

By default, CM33 CPU in secure state (CPU-S) can access the peripherals in both S-state and NS-state. CM33 CPU in non-secure state (CPU-NS) can only access the peripherals in NS-state.

2.2 Normal GPIO

Normal GPIO is the most common digital peripheral in a microcontroller. Normal GPIO of RT6xx MCU is very flexible and powerful like SPI, UART, and so on. A normal GPIO is also a digital peripheral in the MCU. Table 1shows a simple block diagram of the normal GPIO. As you can see that normal GPIO can read a pin state regardless of pin function configured, for example, if this pin is configured as UART, then this pin state can be read via normal GPIO read.

3 Secure GPIO and secure GPIO Mask

Due to the architecture of the normal GPIO, all digital IO pins states are readable through normal GPIO module from the GPIO read path, independent of which function is chosen for this pin. As a result, there is a possibility of leaking information from secure resource(S).

For example, when a UART is configured as a secure peripheral, which means that this UART is only allowed to be accessed by the secure mode that is, code, not by the non-secure mode.

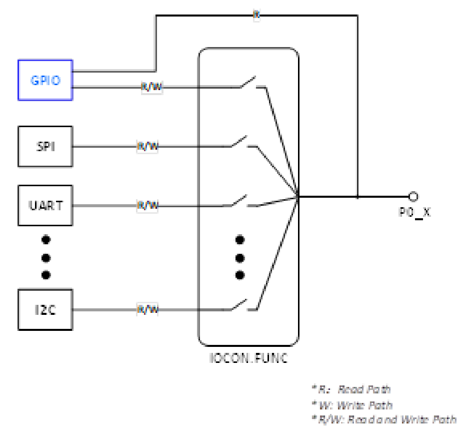
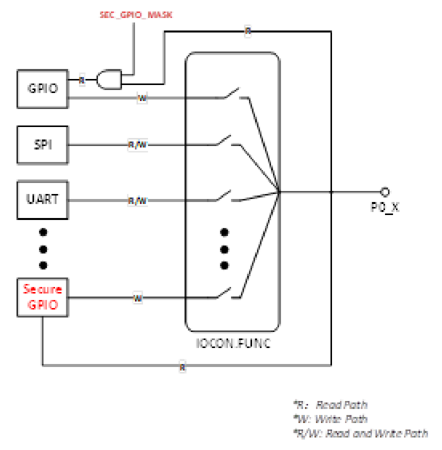
However, in this case, the UART pin states can still be monitored by non-secure mode through normal GPIO read path as shown below. Hence, the non-secure mode can get all the secure UARTs information.

To solve this issue and safeguard incoming data on secure peripherals, a secure GPIO Mask register is implemented on RT600. This register contains bit assignments for each peripheral such that if set, the GPIO pins states can only be read if in secure mode. In addition, it can be used to receive certain input pattern from external device for secure signaling.

The secure GPIO is standard GPIO peripheral supporting 32 pins on port 0 only. Having two GPIO peripherals allows user to configure one GPIO peripheral into secure world and another one into normal world. Thus for all pins on port 0, user can select whether the pin is controlled from secure or non-secure domain (using IOCON).

The following is a simple block diagram of the normal GPIO & secure GPIO and secure GPIO Mask.

Table 1. Normal GPIO & Secure GPIO and Secure GPIO Mask

 R GPIO R/W SPI R/W UART R/W I2C R/W IOCON.FUNC P0_X *R: Read Path *W: Write Path *R/W: Read and Write Path	 SEC_GPIO_MASK R GPIO R W SPI R/W UART R/W Secure GPIO W IOCON.FUNC P0_X R *R: Read Path *W: Write Path *R/W: Read and Write Path

3.1 Secure GPIO Mask

Each GPIO has a secure GPIO MASK. As shown in the Table 1, we can think of the secure GPIO Mask as one input of the AND gate. Its default value is 1. Thanks to the secure GPIO Mask, we can control the on/off state of the normal GPIO read path.

3.2 Secure GPIO

As shown in the Table 1, Secure GPIO has the same functions as normal GPIO. However, just as its name implied 'secure', the access rules to this secure GPIO for different secure levels shall be configured through the secure AHB controller which can only be accessed in secure state. The Secure GPIO port has its own instance of the two GPIO interrupts.

4 Secure GPIO Usage

This section introduces with usage of secure GPIO with code snippets.

4.1 Use secure GPIO mask to protect secure digital peripherals which need IO.

SEC_GPIO_MASK register is used for controlling secure GPIO Mask. By default, this register value is all 1s, which means NS code can still read secure peripheral states by reading its pin states as shown in left side of Figure 2 below.

To prevent this risk of secure information leakage, the normal GPIO shall be masked by setting the corresponding bits in SEC_GPIO_MASK to 0 as shown in the right side of below Figure 2.

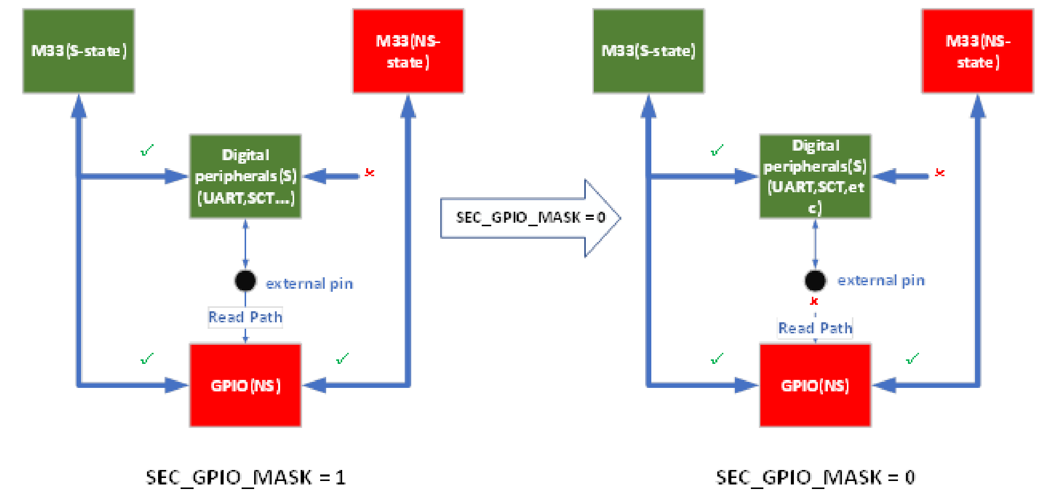


Figure 2. Usage of SEC_GPIO_MASK

The following code snippet shows how to mask P0_27 pin by using secure GPIO MASK:

```
AHB_SECURE_CTRL->SEC_GPIO_MASK0 &=
~AHB_SECURE_CTRL_SEC_GPIO_MASK0_PIO0_PIN27_SEC_MASK(0x1U);
```

4.2 Set one IO to secure GPIO

Follow below steps to configure an I/O pin as secure pin:

- Configure the corresponding bit of SEC_GPIO_MASK to 0
- Configure the secure GPIO module to secure through secure AHB Controller. This prevents non-secure mode from accessing the secure GPIO
- Configure the IOCON block to secure through secure AHB Controller. This prevents non-secure mode from accessing the IOCON.
- Configure the corresponding pin function to secure GPIO (FUNC=8) through secure IOCON block
- Enable secure GPIOs clock

Afterwards, you can use it like a normal GPIO pin.

The following code snippets taking P0_27 pin as an example.

- Configure the SEC_GPIO_MASK of P0_27 to 0:

```
AHB_SECURE_CTRL->SEC_GPIO_MASK0 &=
~AHB_SECURE_CTRL_SEC_GPIO_MASK0_PIO0_PIN27_SEC_MASK(0x1U);
```

- Set secure GPIO as secure:

```
AHB_SECURE_CTRL->AHB_PERIPH3_SLAVE_RULE =
AHB_SECURE_CTRL_AHB_PERIPH3_SLAVE_RULE_SECURE_GPIO_RULE3(0x3U);
```

- Make the IOCON block secure:

```
AHB_SECURE_CTRL->APB_BRIDGE[0].APB_GRP0_MEM_RULE0 =
AHB_SECURE_CTRL_APB_BRIDGE_APB_GRP0_MEM_RULE0_IOPCTL_RULE4(0x3U);
```

- Configure P0_27 pin function to secure GPIO (FUNC=8):

```
const uint32_t port0_pin27_config = (
                                        /* Pin is configured as SEC_P0_27 */
                                        IOPCTL_PIO_FUNC8 |
                                        /* Disable pull-up / pull-down function */
                                        IOPCTL_PIO_PUPD_DI |
                                        /* Enable pull-down function */
                                        IOPCTL_PIO_PULLDOWN_EN |
                                        /* Enables input buffer function */
                                        IOPCTL_PIO_INBUF_EN |
                                        /* Normal mode */
                                        IOPCTL_PIO_SLEW_RATE_NORMAL |
                                        /* Normal drive */
                                        IOPCTL_PIO_FULLDRIVE_DI |
                                        /* Analog mux is disabled */
                                        IOPCTL_PIO_ANAMUX_DI |
                                        /* Pseudo Output Drain is disabled */
                                        IOPCTL_PIO_PSEDRAIN_DI |
                                        /* Input function is not inverted */
                                        IOPCTL_PIO_INV_DI);
/* PORT0 PIN27 (coords: B3) is configured as SEC_P0_27 */
IOPCTL_PinMuxSet(IOPCTL, 0U, 27U, port0_pin27_config);
```

- Enable secure GPIOs clock:

```
CLOCK_EnableClock(kCLOCK_ShsGpio0);
```

- Making PINT for secure GPIO as secure (Optional step):

```
AHB_SECURE_CTRL->APB_BRIDGE[1].APB_GRP1_MEM_RULE0 =
AHB_SECURE_CTRL_APB_BRIDGE_APB_GRP1_MEM_RULE0_GPIO_INTR_CTRL_RULE5_MASK( 0x3U);
```

5 Sample Example Application

5.1 Environment

5.1.1 Hardware environment

- Board
  - MIMXRT685EVK
- Debugger
  - Integrated CMSIS-DAP debugger on the board
- Miscellaneous
  - 1 Micro USB cable
  - PC
- Board Setup
  - Connect the micro USB cable between PC and J5 link on the board for loading and running a demo.

5.1.2 Software environment

- Tool chain

  — IAR embedded workbench 8.50.1 or MCUXpresso IDE 11.1.1 or Keil 5.29

- Software package
  — SDK_2.7.0_EVK-MIMXRT685

5.2 Steps and result

1. Follow the Getting Started with MCUXpresso SDK for MIMXRT600 (can be found inside SDK->docs) in order to go through the steps for running secure_gpio demo (SDK\boards\evkmimxrt685\trustzone_examples\secure_gpio) using MCUXpresso, IAR, or Keil.

   NOTE

   The instruction for a TrustZone based application are a little different as compared to other application. Follow steps for TrustZone based application in getting started guide.

2. Connect the development platform to your PC via USB cable.
3. Result

   Two LEDs are used in this example, Blue LED indicates that the pin state is read by normal GPIO, whereas green LED indicates that the pin state is read by secure GPIO. After reset, code is running in secure mode, and it initializes the system including above two LEDs, and then it jumps to non-secure mode. In non-secure mode, P0_10 pin (SW2 button on EVK) is read in idle loop and in secure mode P0_10 pin is read every 5 ms using a System Timer tick. The secure mode also sets the secure GPIO mask based on SW1 button. The Secure GPIO mask is cleared if SW1 is pressed, while secure GPIO mask is set if SW1 is released. By default secure GPIO Mask for P0_10 is 1 allowing both the secure mode and non-secure mode to read the pin state. When SW2 is pressed and hold down, it turns on the blue LED and green LED as now both normal GPIO and secure GPIO read all 0 from this pin.

   When SW1 is pressed, it jumps to secure mode, clears secure GPIO Mask, and then jump back to non-secure mode. When secure GPIO Mask is set to 0, only secure mode can read the P0_10 pin state while non-secure mode will always read 0 as pin state. Thus in this case blue LED remains on irrespective of SW2 button state. While green LED turns on only when SW2 is pressed and hold down and turns off as soon as it is released. The secure GPIO mask can be set again by releasing the SW1 button.

6 Conclusion

The example shows that non-secure mode can access a peripheral's pin state regardless of the pin function whether the peripheral function is secure or non-secure which results in secure information leakage. To prevent, a secure GPIO must be used and it shall be configured and used in secure mode. Whereas, the normal GPIO shall be used in non-secure mode.

7 References

- RT600 user manual
- RT600 data sheet
- MCUXpresso SDK Release Notes for EVK-MIMXRT685 (Available inside SDK)
- Getting Started with MCUXpresso SDK for EVK-MIMXRT685 (Available inside SDK)
- MCUXpresso SDK API Reference Manual

8 Revision history

Rev. Number	Date	Substantive Changes

Table continues on the next page...

Table continued from the previous page...

0	05 May 2020	Initial release
1	25 June 2020	Updated section 5.2

Date of release: 25 June 2020
Document identifier: AN12835