AN0012: General Purpose Input Output

This application note describes usage of the EFM32 and EFR32 general-purpose input/output (GPIO) subsystem.

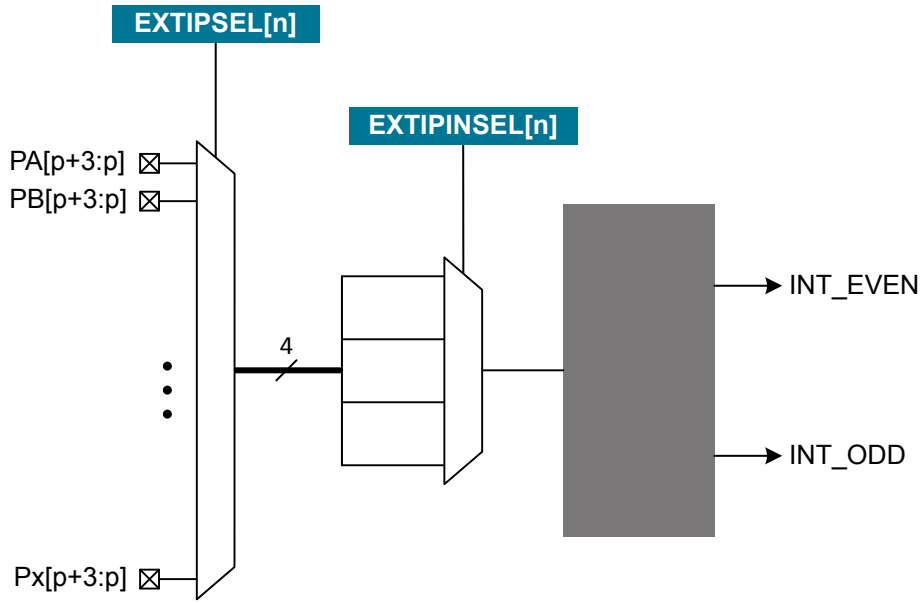
This document discusses configuration, read and writing pin values, peripheral function routing, external interrupt capability, and use of GPIO pins as producers for the Peripheral Reflex System (PRS). Example projects that illustrate these concepts can be run on the Starter Kit boards for many different EFM32 derivatives.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C source code
  - Multiple IDE projects

KEY POINTS

- EFM32 and EFR32 devices have a GPIO subsystem that permits the control of individual pins as inputs and outputs.
- Pins are multiplexed and support peripheral I/O in addition to typical GPIO functionality.
- GPIO pins can act as external interrupt request inputs and trigger the operation of peripherals by acting as producers for the PRS.



Note that p = 0, 4, 8, or 12.

 Copyright © 2025 by Silicon Laboratories

1. Device Compatibility

This application note supports multiple device families, and some functionality is different depending on the device.

For this application note, all sections apply to all Series 0 and Series 1 devices, except for 4.5 Slew Rate, which is only supported on Series 1 and Series 2 devices.

MCU Series 0 consists of:
- EFM32G
- EFM32GG
- EFM32WG
- EFM32LG
- EFM32TG
- EFM32ZG
- EFM32HG

Wireless MCU Series 0 consists of:
- EZR32WG
- EZR32LG
- EZR32HG

MCU Series 1 consists of:
- EFM32JG1/EFM32JG12
- EFM32PG1/EFM32PG12
- EFM32GG11/EFM32GG12
- EFM32TG11

Wireless SoC Series 1 consists of:
- EFR32BG1/EFR32BG12/EFR32BG13
- EFR32FG1/EFR32FG12/EFR32FG13/EFR32FG14
- EFR32MG1/EFR32MG12/EFR32MG13/EFR32MG14

MCU Series 2 consists of:
- EFM32PG22
- EFM32PG23
- EFM32PG26
- EFM32PG28

Wireless SoC Series 2 consists of:
- EFR32BG21/EFR32MG21
- EFR32BG22/EFR32BG22L/EFR32FG22/EFR32MG22
- EFR32FG23/EFR32SG23/EFR32ZG23
- EFR32BG24/EFR32BG24L/EFR32MG24
- EFR32FG25
- EFR32BG26/EFR32MG26
- EFR32BG27/EFR32MG27
- EFR32FG28/EFR32SG28/EFR32ZG28
- EFR32BG29/EFR32MG29

2. About the Examples

2.1 Project Nomenclature

Example projects for this application note are provided for many of the EFM32/EFR32 devices. These projects follow the `<kit>_<example>` naming scheme, where `<kit>` refers to the specific EFM32/EFR32 kit/board on which the example is intended to run, and `<example>` is the particular topic illustrated (e.g. configuration or interrupts). These example projects are located in https://github.com/Silicon-Labs/peripheral_examples.

3. GPIO

3.1 Introduction

The GPIO module controls I/O pins when they are not configured for use with one of the on-chip peripherals. They are organized in ports with up to 16 pins each and are named as P*xn*, where *x* indicates the port (A, B, C...) and *n* indicates the pin number (0, 1,..., 15). Not all ports may be present on a given device, and some ports may have less than 16 pins; refer to the datasheet for the device in question about specific port and pin availability. Each port has its own set of registers for configuration and data reads and writes, and each pin can be configured for a variety of input and output modes.

3.2 Overview

As shown below, the behavior of each pin is controlled by different signals from the GPIO block. In addition to its general-purpose and peripheral (alternate) I/O functionality, each pin is also connected to on-chip analog blocks, interrupt logic, and the Peripheral Reflex System (PRS).

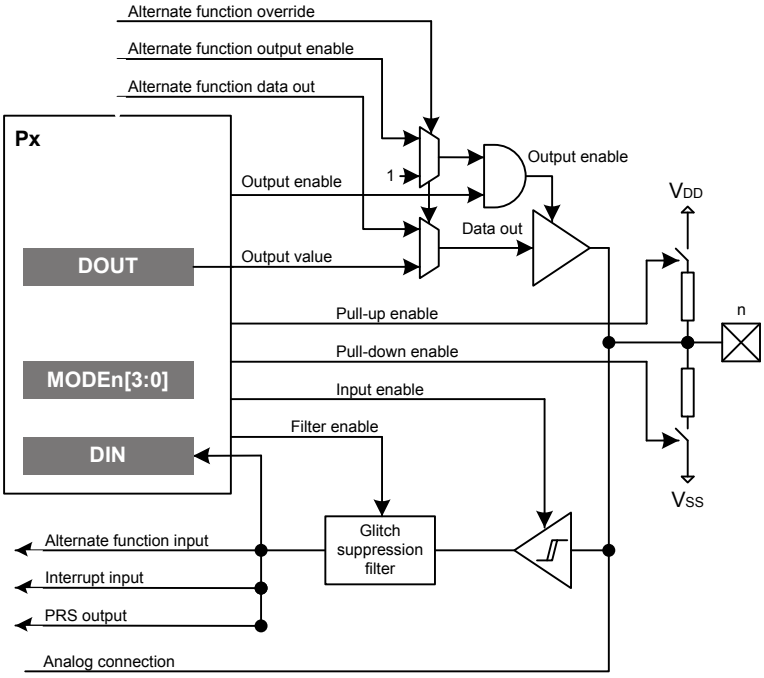


Figure 3.1. Pin Connections Overview

On the output side, a pin is driven by writing to its corresponding bit in the DOUT register. Outputs can be configured as wired-OR, open drain, or push-pull with device-specific drive characteristics.

The state of a pin configured for input is reflected in its corresponding DIN register. A programmable pull-up or pull-down can be enabled for each input along with a selectable filter that can suppress glitches up to 50 ns in duration.

To avoid accidental changes, GPIO configuration can be locked on a per pin basis once set.

4. Configuration

4.1 Overview

Three registers determine the configuration of each pin in PORT x: GPIO_Px_MODEL (pins 0 to 7) or GPIO_Px_MODEH (pins 8 to 15), GPIO_Px_DOUT, and GPIO_Px_CTRL.

The following code configures pin 5 from Port C as a digital input with pull-up and filter. The MODE5 bitfield from GPIO_PC_MODEL register must be set to 0b0011, and the correspondent bit in the GPIO_PC_DOUT register must also be set to 1 to determine the pull direction. A pin can be configured either by using the functions available in the emlib or through a direct register write:

```
GPIO->P[gpioPortC ].DOUT = GPIO->P[gpioPortC ].DOUT | (1 << 5); //Input enabled with pull up and filter

GPIO->P[gpioPortC ].MODEL = (GPIO->P[gpioPortC ].MODEL & ~_GPIO_P_MODEL_MODE5_MASK) |
GPIO_P_MODEL_MODE5_INPUTPULLFILTER;
```

However, code portability is improved by taking advantage of emlib's higher-level functions for configuring and controlling GPIO pins. The following call both implements the same pin configuration as above and is self-documenting:

```
GPIO_PinModeSet(gpioPortC, 5, gpioModeInputPullFilter, 1);
```

with the first parameter being the port, followed by the pin, then pin mode, and finally the DOUT value, which also specifies the pull-up type for inputs (0 = down and 1 = up).

4.2 Reading

The port value is read from the DIN register. Pins must, of course, be configured as inputs before reading DIN.

Any given pin can be read in one of two ways. The first is to read bit (n) of the DIN register of the port by masking all other bits and shifting the result into the LSB position. The second is to use emlib's own pin input function:

```
pin_read = GPIO_PinInGet(gpioPortA, 15);
```

which returns the value of the port and pin specified as parameters.

4.3 Writing

For port writes, pins must be configured in one of the output modes, such as push-pull, after which the most direct way of changing one or more pins is to write the GPIO_Px_DOUT register. Atomic pin toggling (no need for read-modify-write operations) is possible using the GPIO_Px_DOUTTGL register. Some EFM32 devices have GPIO_Px_DOUTSET and GPIO_Px_DOUTCLR registers to perform mask-based port set and clear operations using. These registers work as follows:

- GPIO_Px_DOUT - data written to this register sets the pin values to 0/1 accordingly
- GPIO_Px_DOUTSET - only bits written to 1 are effective and change pin values to 1
- GPIO_Px_DOUTCLR - only bits written to 1 are effective and change pin values to 0
- GPIO_Px_DOUTTGL - only bits written to 1 are effective and toggle pin states

There are also emlib functions for these same operations, and these should be preferred over direct register writes. For set/clear operations, emlib uses, depending on device availability and in order of efficiency, hardware GPIO set/clear registers, ARM Cortex M-series bit-banding, and, finally, read-modify-write operations. Examples of how these functions are used follows:

```
GPIO_PinOutClear(gpioPortA, 3);

GPIO_PinOutSet(gpioPortA, 3);

GPIO_PinOutToggle(gpioPortA, 3);

GPIO_PortOutClear(gpioPortE, 0x0c);
GPIO_PortOutSet(gpioPortE, 0x0c);
GPIO_PortOutToggle(gpioPortE, 0x0c);

GPIO_PortOutSetVal(gpioPortE, 0x0c, 0x0f); port, mask and value
```

The first three functions change the output value of one pin only by clearing (changing the value to 0), setting (changing the value to 1), or toggling the pin, respectively. When possible, these functions are implemented using SET, CLEAR, and TOGGLE registers. The next 3 functions can be used to set output values of multiple pins in a port. The last function uses a mask to set or clear a group of bits in the mask using the value.

4.4 Drive Mode and Drive Strength

4.4.1 Series 0 and Series 1

All the I/O pins have the same default drive strength if no alternate drive strength is configured during GPIO initialization. Default drive strength and programmable drive strength options are device dependent; nominal drive strength settings are described in the reference manual for a given part. If a different drive strength is needed, it is possible to configure an alternate drive strength for each port, and have selected pins use the alternate drive strength. This means that all the pins from the same port have either the default drive strength or the selected alternative drive strength.

Selection of alternate drive strength is controlled by the GPIO_Px_CTRL register and is device-dependent. When GPIO_Px_CTRL includes the DriveMode field, four drive strength selections are available, otherwise the DriveStrength bit selects a strong or weak option. A given device supports only DriveMode or DriveStrength for all GPIO pins, and emlib compiles with only the corresponding function available. For example, devices with DriveMode, would use:

```
GPIO_DriveModeSet(gpioPortB, GPIO_P_CTRL_DRIVEMODE_HIGH);
```

or one of the DriveMode settings like DEFAULT, STANDARD, LOWEST, or LOW. In the case of DriveStrength, the function:

```
GPIO_DriveStrengthSet(gpioPortD, gpioDriveStrengthWeakAlternateStrong);
```

would be used with the other options being gpioDriveStrengthStrongAlternateStrong, gpioDriveStrengthStrongAlternateWeak, and gpioDriveStrengthWeakAlternateWeak.

Regardless of the drive strength options available, be sure the total maximum drive current potentially used across all output pins does not exceed the datasheet-specified maximum for the device in question.

4.4.2 Series 2

Drive strength has now been combined with slew rate. The lower drive strength is achieved by setting the slew rate to the minimum possible 3 bit value (000b). All other slew rate settings result in a higher drive current. Please refer to section 4.5 Slew Rate for more information.

4.5 Slew Rate

Series 1 and Series 2 devices have an additional port setting for slew rate. All the I/O pins have the same default slew rate if no alternate slew rate is configured during GPIO initialization. Default slew rate and programmable slew rate options are device dependent; nominal slew rate settings are described in the reference manual for a given part. If a different slew rate is needed, it is possible to configure an alternate slew rate for each port, and have selected pins use the alternate slew rate. This means that all the pins from the same port have either the default slew rate or the selected alternative slew rate. Note that we do not recommend using the highest slew rate setting.

Selection of alternate slew rate is controlled by the GPIO_Px_CTRL register and is device-dependent. Emlib has a function to support changing the slew rate for a particular port. For example, to change the default slew rate to 5 and the alternate slew rate to 6 on port D, the following function would be used:

```
GPIO_SlewrateSet(gpioPortD, 5, 6);
```

The rise and fall times for a typical device with a pin with a 50 pF load were measured on the bench, and are as follows:

Table 4.1. Slew Rate (Series 1)

	Drive Strength: Strong		Drive Strength: Weak	
Slew Rate Setting	**Rise Time**	**Fall Time**	**Rise Time**	**Fall Time**
7	1.7 ns	1.4 ns	7 ns	4.1 ns
6	2.2 ns	1.8 ns	7.4 ns	4.5 ns
5	3.2 ns	2.6 ns	8.6 ns	7 ns
4	4.1 ns	3 ns	12 ns	8.5 ns
3	6.9 ns	3.9 ns	15 ns	11 ns
2	12 ns	7.1 ns	19 ns	14 ns
1	20 ns	13 ns	25 ns	20 ns
0	24 ns	19 ns	35 ns	26 ns

Note:

1. Measurements were taken with a capacitive load (C_L) of 50 pF.
2. Measurements were taken on one typical device, and may vary from part to part.

Table 4.2. Slew Rate (Series 2)

	Drive Strength: Strong	
Slew Rate Setting	**Rise Time**	**Fall Time**
7	4.4 ns	3.5 ns
6	5.2 ns	4.6 ns
5	8.6 ns	7 ns
4	11.4 ns	9.2 ns
3	25 ns	20 ns
2	41 ns	32 ns
1	55 ns	44 ns
0	66 ns	56 ns

Note:

1. Measurements were taken with a capacitive load (C_L) of 47 pF.
2. Measurements were taken on one typical device, and may vary from part to part.

4.6 Configuration Lock

Unwanted or accidental changes to GPIO configuration can be avoided by using the configuration lock register. Any value other than `0xA534` written to GPIO_LOCK enables the configuration lock. Pins are unlocked by a reset or by writing `0xA534` to the GPIO_LOCK register. Lock status is determined by reading GPIO_GPIOLOCKSTATUS register.0 indicates that the GPIO configuration registers are unlocked, and a 1 indicates that they are locked.

Configuration lock affects the GPIO_Px_MODEL, GPIO_Px_MODEH, GPIO_Px_CTRL, GPIO_Px_PINLOCKN, GPIO_EXTIPSELL, GPIO_EXTIPSELH, GPIO_EXTIPINSELL, GPIO_EXTIPINSELH, GPIO_INSENSE, GPIO_ROUTE, GPIO_ROUTEPEN, and GPIO_ROUTELOC0 registers when they are present on a specific device.

This can be controlled in a granular manner on Series 0 and Series 1 parts using a mask register that is available for each port. The GPIO_Px_PINLOCKN mask register selects which pins of a given port are affected by the lock mechanism.

4.7 Pin Configuration and Reading/Writing Example

Series 0: The `<kit>_gpio_conf` example shows how GPIO pin configuration, reading, and writing are performed. By default, LED0 is configured to be driven with the default drive strength. When the PB0 button is pressed, the EFM32 will change the drive strength of LED0 to its lowest value. Default and alternate drive strength are derivative-specific; check the datasheet for the device in question for drive strength options.

Series 1: The `<kit>_switch_led_polled` example shows how GPIO pin configuration, reading, and writing are performed. The LED0 output pin is set and cleared when PB0 is pressed or released. This is due to how the drive strength changes This is the case because drive strength changes on these devices are not enough to produce a visible change in the LED output.

Series 2: The `<kit>_switch_led_polled` example shows how GPIO pin configuration, reading, and writing are performed. The LED0 output pin is set and cleared when PB0 is pressed or released. This is due to how the drive strength changes This is the case because drive strength changes on these devices are not enough to produce a visible change in the LED output. On Series 2 devices, the drive strength is controlled by setting slew rate.

4.8 Slew Rate Example (Series 1 and Series 2 Only)

Series 1: The `<kit>_gpio_slew_rate` project demonstrates the slew rate settings for the GPIO. A 1 MHz square wave is generated on a pin. The slew rate setting is changed by pressing PB0. The drive strength setting is changed by pressing PB1. The effects of these settings on the pin's slew rate can be observed by placing a 50 pF capacitor on the output pin and observing the change in rise and fall times of the 1 MHz square wave.

Series 2: The `<kit>_gpio_slew_rate` project demonstrates the slew rate settings for the GPIO. A 1 MHz square wave is generated on a pin. The slew rate setting is changed by pressing PB0. The drive strength setting is changed by pressing PB1. The effects of these settings on the pin's slew rate can be observed by placing a 50 pF capacitor on the output pin and observing the change in rise and fall times of the 1 MHz square wave. On series 2 devices, both the slew rate and drive strength settings are controlled by the slew rate setting. Therefore, PB0 changes both settings. The lowest 3 bit slew rate value also results in the lowest drive strength. All other values result in high drive strength.

5. Peripheral Usage

5.1 Routing

With some exceptions, such as the segment LCD controller or USB transceiver, peripheral I/O functions are generally multiplexed onto several different pins to allow for ease of PCB trace routing. Tables in the pinout section of the datasheet for any given device outline I/O multiplexing and GPIO availability.

Modules with I/Os that can be mapped to more than one location have register-based routing and pin enabling functions. On some devices, the term "location" refers to a group of typically adjacent pins associated with a specific GPIO port. Each peripheral on these devices has a dedicated ROUTE register with a bit field that determines the location for all associated I/Os and individual enable bits for these I/Os.

Series 1 devices have a more flexible routing scheme wherein each peripheral has a ROUTEPEN register for enabling and disabling I/Os on a per pin basis and as many ROUTELOCn registers as are needed to allow discrete mapping of each peripheral I/O to one of up to 32 possible locations.

On Series 2 devices, peripheral routing is completely contained in the GPIO peripheral. The concept of "location" no longer exists and port name and pin numbers are used instead to route a peripheral functionality to a pin. To route the TX resource of USART1 to PB3, set PORT to 0x1 and PIN to 0x3 in GPIO_USART1_TXROUTE. Then set the GPIO_USART1_ROUTEN.TXEN bit.

If a pin is configured for both one of its alternate functions and as a general purpose output, the alternate function's output data is given priority and will override the port output data register. Despite this, the pin's configuration registers will remain unchanged, which means that a pin must be set as an output so that any multiplexed peripheral can also use it as an output. Likewise, care must be taken when selecting the pin's output mode. For instance, if the pin is configured as push-pull, the peripheral will be able to drive both high and low values; however, if it is configured as open-drain mode without a pull-up (either internal or external), the peripheral will only be able to drive a low value.

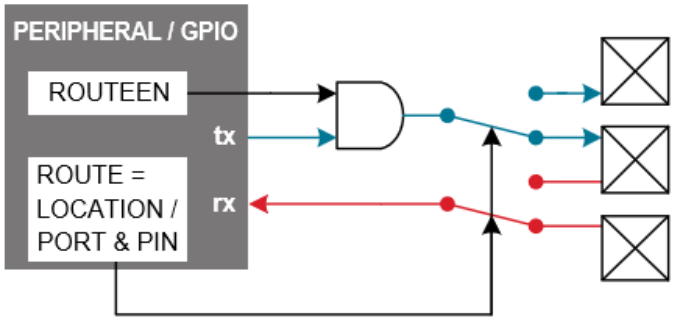


Figure 5.1. Peripheral Routing

It is possible to have two or more peripherals connected to the same pin. This is not recommended and any peripheral conflicts should be resolved so that any pin only has one peripheral connected at a time.

5.2 Routing Example

Series 0: The `<kit>_gpio_periph` project configures one of the pins to drive the output of the low frequency RC oscillator (LFRCO).

Series 1: The `<kit>_gpio_periph` project configures one of the pins to drive the output of the low frequency RC oscillator (LFRCO).

Series 2: The `<kit>_gpio_periph` project configures one of the pins to drive the output of the low frequency RC oscillator (LFRCO). It also sets the slew rate and drive strength of the pin.

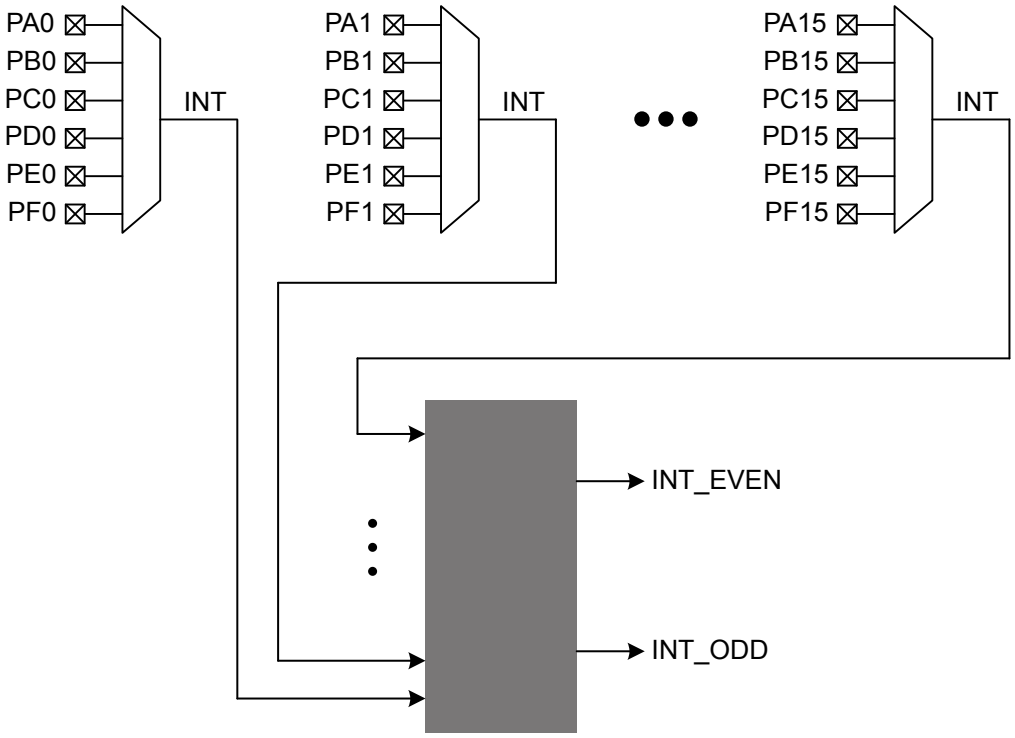
6. Interrupts

6.1 GPIO as External Interrupt Requests

All GPIO pins have interrupt capability; however, they are grouped in such a way that there can be a maximum of 16 external interrupt requests coming from GPIO pins. Furthermore, the external interrupt requests to which the pins are mapped are grouped by pin number with the even-numbered requests being assigned to the interrupt controller's GPIO_EVEN source and odd-numbered requests being assigned to the GPIO_ODD source.

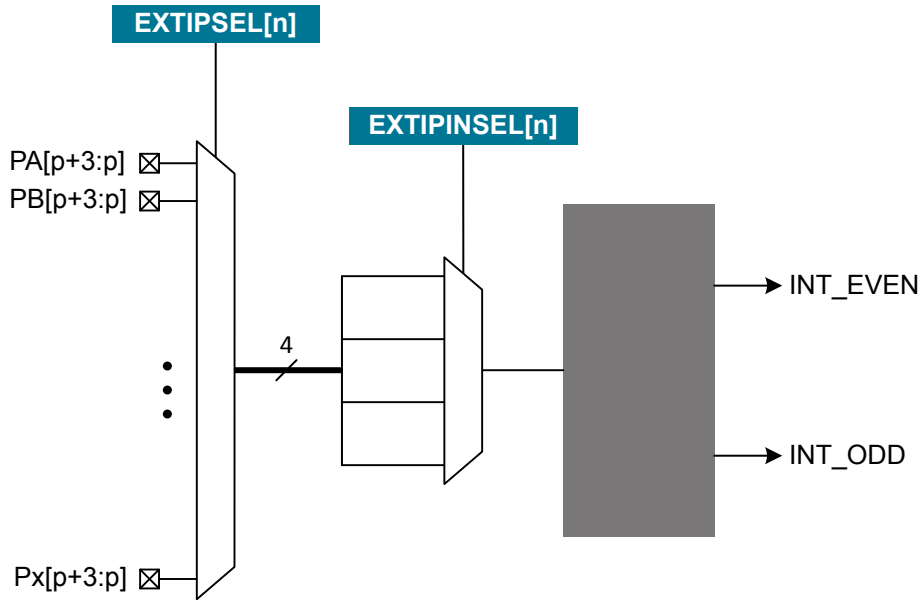
Two different mechanisms are used to select the pins mapped to external interrupt requests across the EFM32/EFR32 family. The first, illustrated below, groups all pins with the same number spanning all ports into a single external interrupt request. Thus PA0, PB0, etc. are used as an external interrupt request 0, PA1, PB1, etc. as an external interrupt request 1, and so on up through external interrupt request 15. As noted previously, some GPIO ports may have less than 16 pins. This has particular significance on devices in small packages because it may not be possible to select a pin for use with some external interrupt requests.

Figure 6.1. Interrupts for Series 0, Series 1, and Series 2 Devices



Only one pin with the same number across all ports can be enabled as an interrupt at any given time. Thus, for example, both port A bit 0 (PA0) and port C bit 0 (PC0) cannot be enabled as interrupts at the same time. Using PC2 instead of PC0 would solve this problem. The suggestion to use PC2 in this case is intentional. As noted above, all even-numbered external interrupt requests are assigned to the interrupt controller's GPIO_EVEN source. Consequently, switching from PC0 to PC1 might not be desirable as PC1 would be mapped to external interrupt request 1, which, in turn, is assigned to the interrupt controller's GPIO_ODD source. The GPIO_EXTIPSELL and GPIO_EXTIPSELH registers select which GPIO port provides the pin for the corresponding external interrupt request.

More flexibility is provided by the second pin-to-external interrupt request mapping mechanism available on EFM32 Series 1 and EFR32 Series 2 devices. Pins across all ports are mapped such that the two MSBs of the pin number match the two MSBs of the external interrupt requests number. Thus, pins [3:0] across all ports are mapped to external interrupt requests [3:0], pins [7:4] to requests [7:4], pins [11:8] to requests [11:8], and pins [15:12] to requests [15:12]. This mapping is not of a correspondence of one pin number to one external interrupt request but rather of the group of four pins collectively to the matching group of four external interrupt requests.



Note that p = 0, 4, 8, or 12.

Figure 6.2. Interrupts for EFM32 Series 1 / EFR32 Series 2 Devices

As shown above, the GPIO_EXTIPSELL and GPIO_EXTIPSELH registers select the port from which the pin is provided for a given external interrupt request. However, because any one pin from a given group of four can be assigned to one of the four corresponding external interrupt requests, a second level of selection logic is needed. The GPIO_EXTIPINSELL and GPIO_EXTIPINSELH registers determine which pin from the group of four associated with the selected GPIO port is connected to the external interrupt request.

This added flexibility can lead to confusion and programming errors if care is not taken with respect to interrupt pin selection. For example, if GPIO_EXTIPSELL is programmed to select port D for external interrupt request 0, port E cannot be selected to do the same. However, if GPIO_EXTIPSELL is programmed to select port D for external interrupt requests 0 and 1, GPIO_EXTIPINSELL can be programmed such that the same pin (e.g., PD0) is used for external interrupt requests 0 and 1. There may be instances in which this behavior is desirable, but it also means that there is no correspondence between even-numbered GPIO pins and the GPIO_EVEN interrupt source and odd-numbered pins and the GPIO_ODD source. In this case, PD0 can be selected as external interrupt request 1 and routed to the GPIO_ODD source of the interrupt controller.
Regardless of the pin selection mechanism, GPIO interrupt programming is the same. First, the transition(s) to be recognized as interrupts is(are) programmed into the corresponding register(s). An external interrupt to be recognized on the rising edge is enabled in the GPIO_EXTRISE register; falling edge interrupts are enabled in GPIO_EXTIFALL. Because these registers are independent, it is possible to interrupt on both rising and falling edges by setting the corresponding bits in both registers.

Second, the desired external interrupt requests must be enabled by setting the corresponding bits in the GPIO_IEN register. Finally, interrupt sensing must be enabled for the GPIO module by setting the INT bit in the GPIO_SENSE register. As with any interrupts, be sure the NVIC is initialized and that the vectors for the GPIO_EVEN and GPIO_ODD sources point to valid interrupt service routines. When an external interrupt occurs, the corresponding bit will be set in the GPIO_IF register, and it will trigger an interrupt request using either the GPIO_EVEN or GPIO_ODD sources.

6.2 Interrupt Example

Series 0, 1, and 2: The `<kit>_switch_led_interrupt` project places the device into a low energy mode, EM3. When the user presses PB0, a GPIO interrupt will fire and LED0 will be toggled. When the user presses PB1, a GPIO interrupt will fire and LED1 will be toggled.The device will then return to EM3 until the next button press.

7. Peripheral Reflex System (PRS) Output

7.1 GPIO as Peripheral Reflex System Producer

The Peripheral Reflex System (PRS) allows peripherals to communicate without CPU intervention, particularly in low energy modes when the CPU is shutdown. Peripherals that are reflex producers send signals (pulses or levels) to other peripherals, reflex consumers, causing them to take action. The PRS chapter in the reference manual for a given device contains a table listing all producers and consumers and their output and input types, respectively.

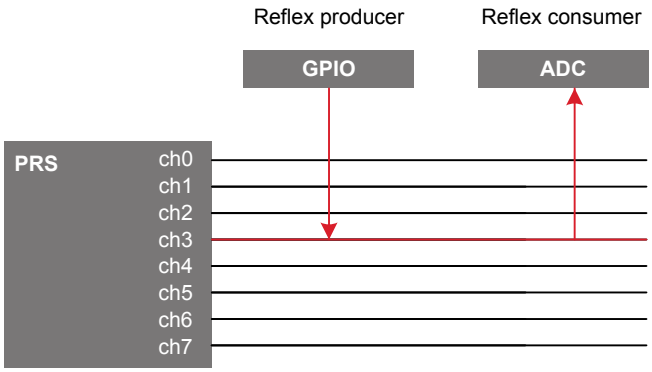


Figure 7.1. PRS output

Every GPIO pin can be a reflex producer. They are grouped in the same way as described previously for external interrupts. Selecting a pin for use as an external interrupt source also selects it for use as a PRS producer with the only difference being that external interrupt requests must also be enabled by setting the corresponding bits in the GPIO_IEN register. The warnings about whether two pins can both be used as the same external interrupt request or whether a single pin can be used for two different external interrupt requests also appy to pins selected as PRS reflex producers.

PRS input sensing must be enabled by setting the PRS bit in the GPIO_INSENSE register on devices which map GPIO pin numbers to external interrupt requests, and therefore, GPIOs used as reflex producers. These are devices which the GPIO_EXTIPSELL and GPIO_EXTIPSELH registers alone to select the port that provides a given external interrupt request and PRS reflex producer. Devices which use the combined GPIO_EXTIPSELL/GPIO_EXTIPSELH and GPIO_EXTIPINSELL/GPIO_EXTIPINSELH mechanism do not have a dedicated PRS input sensing control bit. Pins intended for use as GPIO reflex producers on these devices are active once configured as inputs and selected by the aforementioned registers.

7.2 PRS Example

Series 0 and 1: The `<kit>_gpio_prs` example runs the EFM32/EFR32 in EM1. A falling edge (pushing button PB0) triggers a single ADC conversion. When the conversion ends, the ADC requests an interrupt to wake up the processor, and a delay cycle keeps it running in EM0 for 1 second before re-entering EM1. This allows the change in current draw to be observed using the Advanced Energy Monitor or the EnergyAware Profiler.

Series 2: The `<kit>_iadc_single_gpio_prs_ldma` example runs the EFM32/EFR32 in EM2. A raising edge (pushing button PB0) triggers a single IADC conversion. When the conversion ends, an LDMA data transfer occurs that stores the raw data to memory. The IADC/LDMA operates in EM2; LDMA requests an interrupt to wake EFM32/EFR32 to EM0 to toggle a GPIO before returning to EM2. Because the IADC conversion is done in EM2, only Port A/B pins can be used. However, LDMA transfer complete toggle is performed in the LDMA ISR, operates in EM0, and may utilize pins on any port. This allows the change in current draw to be observed using the Advanced Energy Monitor or the EnergyAware Profiler.

8. Revision History

8.1 Revision 2.05

June, 2025

- Added EFM32PG26, EFR32BG22L, EFR32BG24L, EFR32BG26, EFR32MG26, EFR32BG29, and EFR32MG29 to 1. Device Compatibility.
- Updated the Series 2 example name in 4.7 Pin Configuration and Reading/Writing Example.
- Split the examples based on series in 7.2 PRS Example.

8.2 Revision 2.04

August, 2023

- Added EFM32GG12, EFM32TG11, MCU Series 2, EFR32xG23, EFR32xG24, EFR32FG25, EFR32xG27, and EFR32xG28 family members to 1. Device Compatibility.
- Removed EFM32JG13, EFM32PG13, and EFR32BG14 from 1. Device Compatibility.

8.3 Revision 2.03

March, 2020

- 1. Device Compatibility - Added EFR32xG22 OPNs.
- 2.1 Project Nomenclature - Updated location of example project files.
- 4.7 Pin Configuration and Reading/Writing Example - Split examples based on series.
- 4.8 Slew Rate Example (Series 1 and Series 2 Only) - Fixed typo in which example project name was <kit>_slew_rate instead of <kit>_gpio_slew_rate. Split examples based on series.
- 5.1 Routing - Fixed typo in which Port B referenced 0x02 instead of 0x01.
- 5.2 Routing Example - Split examples based on series.
- 6.2 Interrupt Example - Split examples based on series.
- 7.2 PRS Example - Split examples based on series.
- 8. Revision History - Updated the format.

8.4 Revision 2.02

2019-02-25

- Edited for Series 2 EFR32.

8.5 Revision 2.01

2017-06-29

- Added section on slew rate.
- Added device compatibility page.

8.6 Revision 2.00

2015-11-06

- Added support for EFM32 Series 1
- Updated text and examples

8.7 Revision 1.07

2014-05-07

- Updated example code to CMSIS 3.20.5
- Changed to Silicon Labs license on code examples
- Added example projects for Simplicity IDE
- Removed example makefiles for Sourcery CodeBench Lite

8.8 Revision 1.06

2013-10-14

- New cover layout

8.9 Revision 1.05

2013-05-08

- Added software projects for ARM-GCC and Atollic TrueStudio.

8.10 Revision 1.04

2012-11-12

- Adapted software projects to new kit-driver and bsp structure.

8.11 Revision 1.03

2012-04-20

- Adapted software projects to new peripheral library naming and CMSIS_V3.

8.12 Revision 1.02

2012-03-14

- Fixed makefile-error for CodeSourcery projects.

8.13 Revision 1.01

2010-11-16

- Changed example folder structure, removed build and src folders.
- Added chip-init function.
- Updated register defines in code to match newest efm32lib release.

8.14 Revision 1.00

September 20th, 2010.

- Initial revision.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

Disclaimer
Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

Trademark Information
Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect , n-Link, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress® , Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com