AN0012: General Purpose Input Output

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

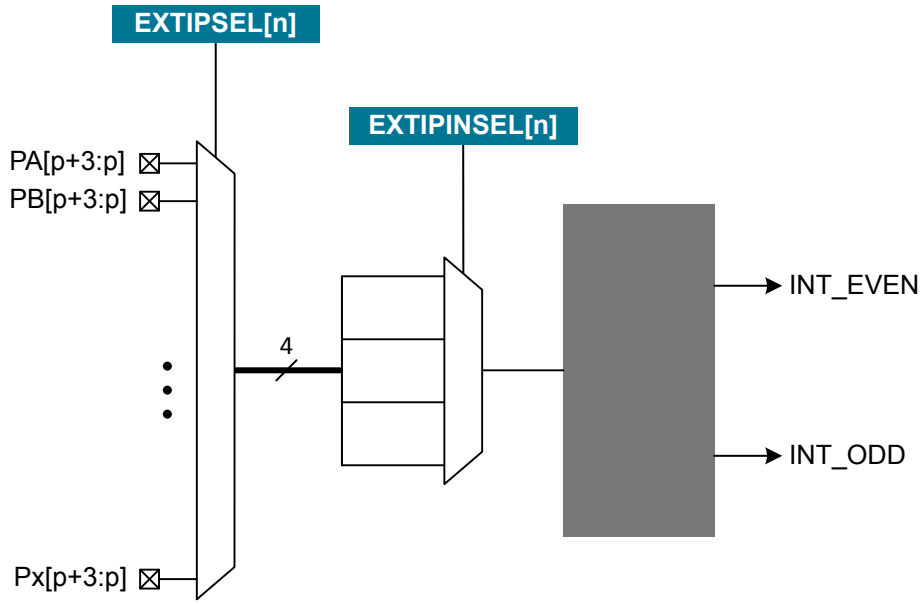
This document discusses configuration, read and writing pin values, peripheral function routing, external interrupt capability, and use of GPIO pins as producers for the Peripheral Reflex System (PRS). Example projects that illustrate these concepts can be run on the Starter Kit boards for many different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

This application note includes:

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

KEY POINTS

- EFM32 devices have a GPIO subsystem that permits the control of individual pins as inputs and outputs.
- EFM32 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.

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 devices.

MCU Series 0 consists of:

- EFM32 Gecko (EFM32G)
- EFM32 Giant Gecko (EFM32GG)
- EFM32 Wonder Gecko (EFM32WG)
- EFM32 Leopard Gecko (EFM32LG)
- EFM32 Tiny Gecko (EFM32TG)
- EFM32 Zero Gecko (EFM32ZG)
- EFM32 Happy Gecko (EFM32HG)

Wireless MCU Series 0 consists of:

- EZR32 Wonder Gecko (EZR32WG)
- EZR32 Leopard Gecko (EZR32LG)
- EZR32 Happy Gecko (EZR32HG)

MCU Series 1 consists of:

- EFM32 Jade Gecko (EFM32JG1/EFM32JG12/EFM32JG13)
- EFM32 Pearl Gecko (EFM32PG1/EFM32PG12/EFM32PG13)
- EFM32 Giant Gecko (EFM32GG11)

Wireless SoC Series 1 consists of:

- EFR32 Blue Gecko (EFR32BG1/EFR32BG12/EFR32BG13)
- EFR32 Flex Gecko (EFR32FG1/EFR32FG12/EFR32FG13)
- EFR32 Mighty Gecko (EFR32MG1/EFR32MG12/EFR32MG13)

2. About the Examples

2.1 Project Nomenclature

Example projects are provided for many of the EFM32 devices and follow the `<kit>_gpio_<example>` naming scheme where `gpio` denotes that these projects are part of the GPIO application note, `<example>` is the particular topic illustrated (e.g. configuration or interrupts), and `<kit>` refers to the specific EFM32 kit on which the example is intended to run.

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).

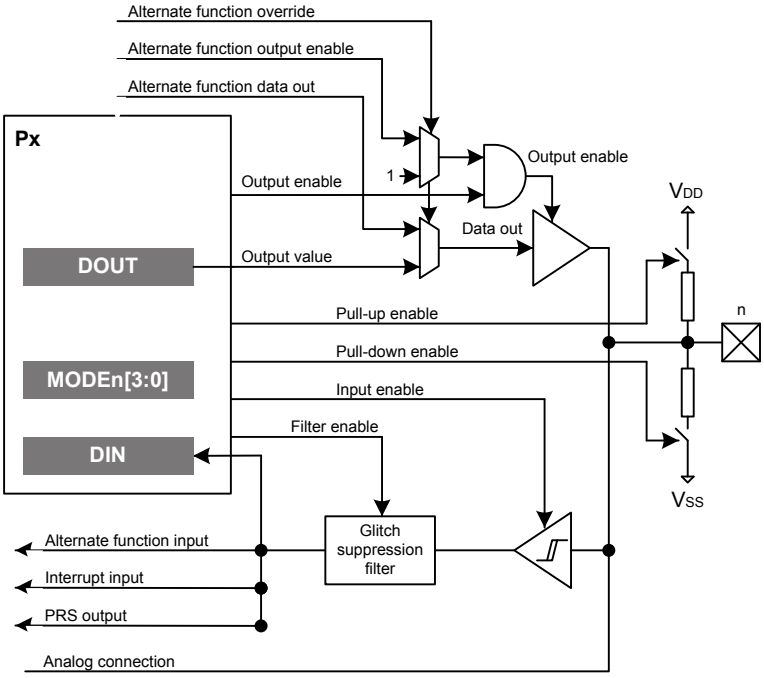


Figure 3.1. Pin Connections Overview

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 pins: GPIO_Px_MODEL (for port pins 0 to 7) or GPIO_Px_MODEH (for port pins 8 to 15), GPIO_Px_DOUT, and GPIO_Px_CTRL.

The following code configures pin 5 from Port C as an 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[2].MODEL = (GPIO->P[2].MODEL
GPIO->P[2].MODEL = (GPIO->P[2].MODEL & ~_GPIO_P_MODEL_MODE5_MASK) |
   GPIO_P_MODEL_MODE5_INPUTPULLFILTER;
GPIO->P[2].DOUT = GPIO->P[2].DOUT | (1 << 5);
```

In the example above, the port registers are accessed directly using GPIO->P[*x*], where *x* corresponds to the I/O port enumeration A = 0, B = 1, etc.

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 perform a bitwise AND of the entire DIN value, masking out all other pins. 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_PortOutSetVal(gpioPortE, 0x0c, 0x0f);
GPIO_PortOutToggle(gpioPortE, 0x0c);
```

The first three functions change the state of one pin only by clearing (changing the value to 0), setting (changing the value to 1), or toggling the pin, respectively. The last four functions change the value of more than one pin, depending on the mask used.

4.4 Drive Mode and Drive Strength

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.

4.5 Slew Rate

Series 1 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.

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

	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.

4.6 Configuration Lock

Unwanted or accidental changes to GPIO configuration can be avoided by using the configuration lock 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. Once all pins across all GPIO ports to be locked are selected by the mask registers, 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_LOCK with a 0 indicating that the GPIO configuration registers are unlocked, and a 1 indicating 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 specifc deivce.

4.7 Pin Configuration and Reading/Writing Example

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.

For Series 1 devices, this example merely sets and clears the output level of the LED0 pin when PB0 is pressed or released. This is the case because drive strength changes on these devices are not enough to produce a visible change in the LED output.

4.8 Slew Rate Example

The `<kit>_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 BTN0. The drive strength setting is changed by pressing BTN1. 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.

The output pin is:

- PD1 on EFM32GG11 STKs (SLSTK3701)
- PA0 on all other series 1 STKs

To test:

1. Place a 50 pF capacitor between the output pin and GND.
2. Upload and run the example.
3. The slew rate setting and drive strength setting are displayed on the kit's LCD.
4. While observing the rise and fall times of the waveform on PA0:
    a. Press BTN0 to change the slew rate setting.
    b. Press BTN1 to change the drive strength setting.

Note: The Slew Rate example only supports Series 1 devices.

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. 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.

Newer 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.

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.

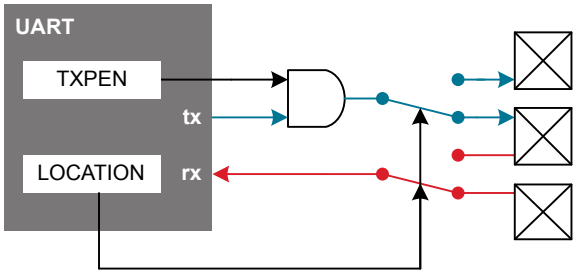


Figure 5.1. Peripheral Routing

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

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

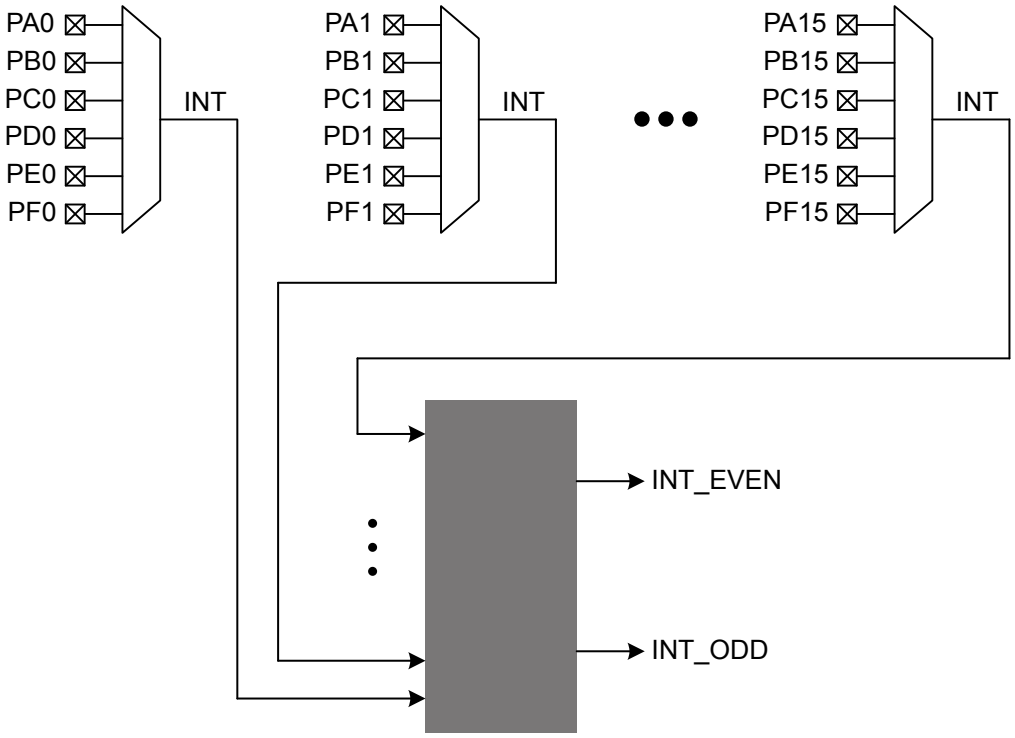
6. Interrupts

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.

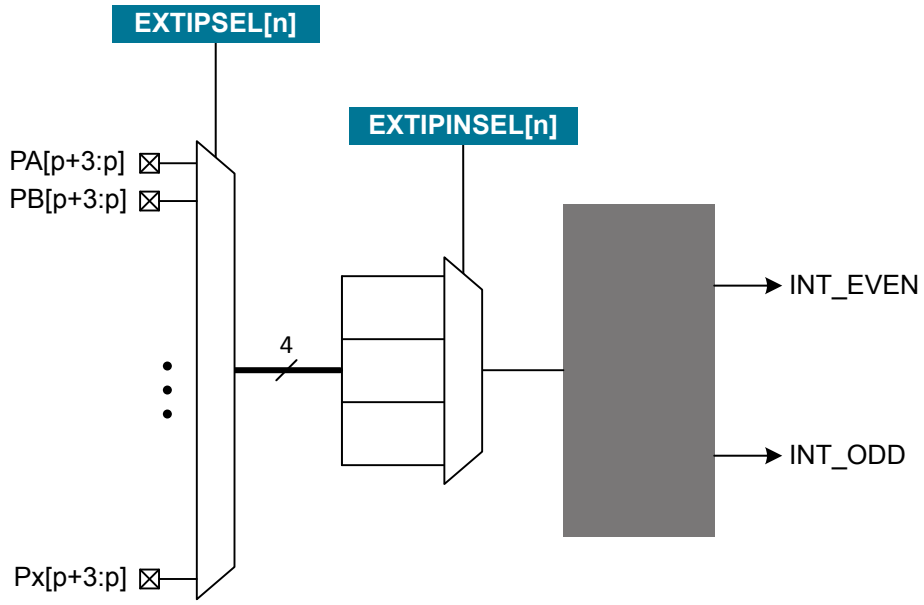
Two different mechanisms are used to select the pins mapped to external interrupt requests across the EFM32 family. The first, illustrated below, groups all pins with the same number spanning all ports into a single external interrupt request. Thus PA0, PB0, etc. are used as an external interrupt request 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 and Series 1 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. 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 correspondonce 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 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

The `<kit>_gpio_int` project places the device into a low energy mode, EM3. When the user presses PB0 or PB1, a GPIO interrupt will fire and LED0 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.

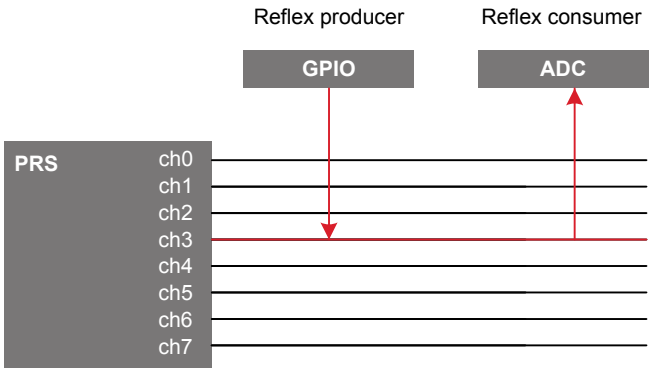


Figure 7.1. PRS output

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

The `<kit>_gpio_prs` example runs the EFM32 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.

8. Revision History

8.1 Revision 2.01

2017-06-29

Added section on slew rate

Added device compatibility page

8.2 Revision 2.00

2015-11-06

Added support for EFM32 Series 1

Updated text and examples

8.3 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

8.4 Revision 1.06

2013-10-14

New cover layout

8.5 Revision 1.05

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

8.6 Revision 1.04

2012-11-12

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

8.7 Revision 1.03

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

8.8 Revision 1.02

2012-03-14

Fixed makefile-error for CodeSourcery projects.

8.9 Revision 1.01

2010-11-16

Changed example folder structure, removed build and src folders.

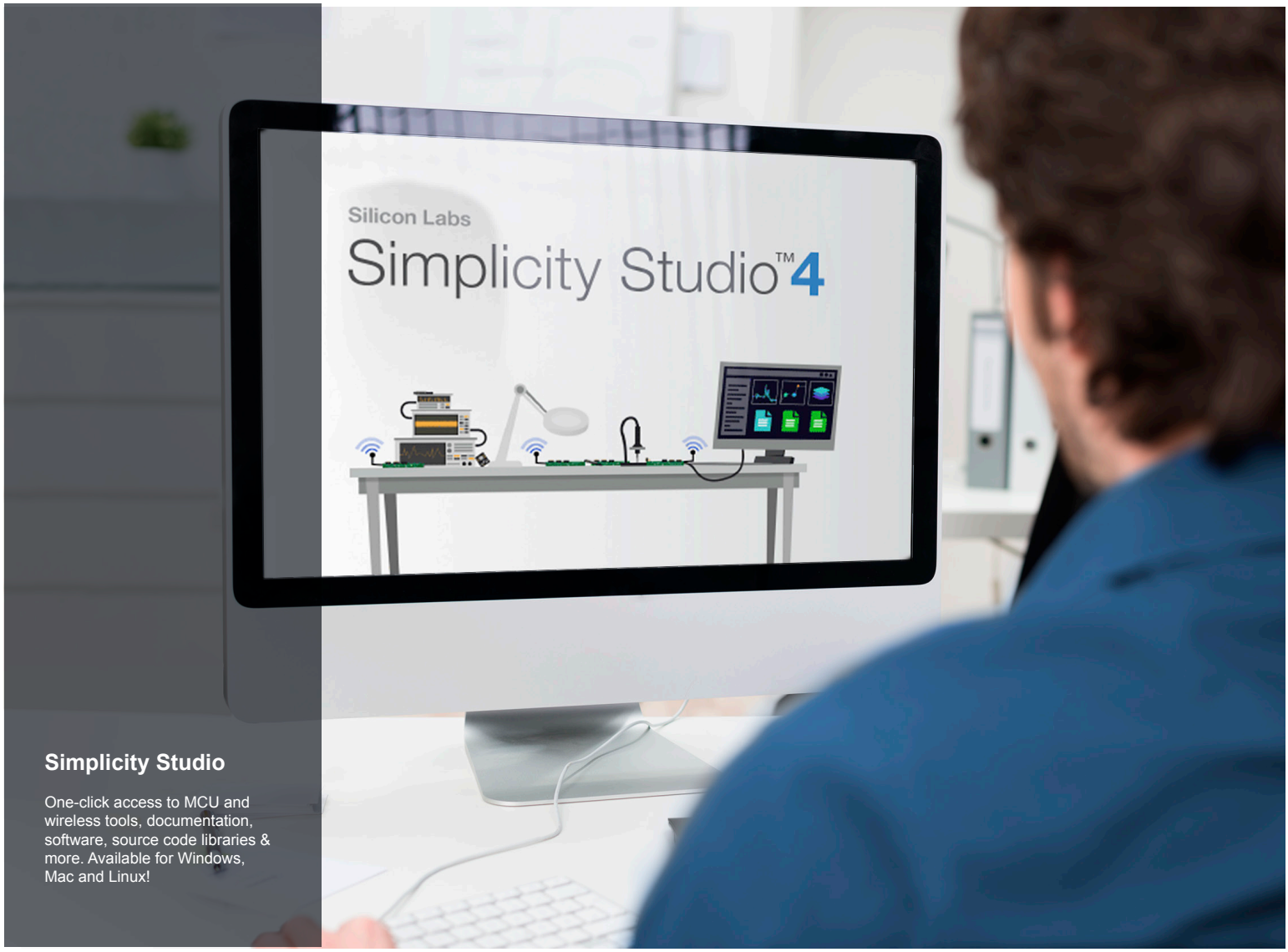
Added chip-init function.

Updated register defines in code to match newest efm32lib release.

8.10 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 and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com