EFM32®

... the world's most energy friendly microcontrollers

General Purpose Input Output

AN0012 - Application Note

Introduction

This application note describes how to use the EFM32 GPIO Module to configure port pins. Modes include digital I/O, analog input, interrupt and PRS input and alternative functions (peripheral usage).

This application note includes:

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

ARM ZERO TINY GECKO LEOPARD GIANT WONDER

SILICON LABS

1 GPIO

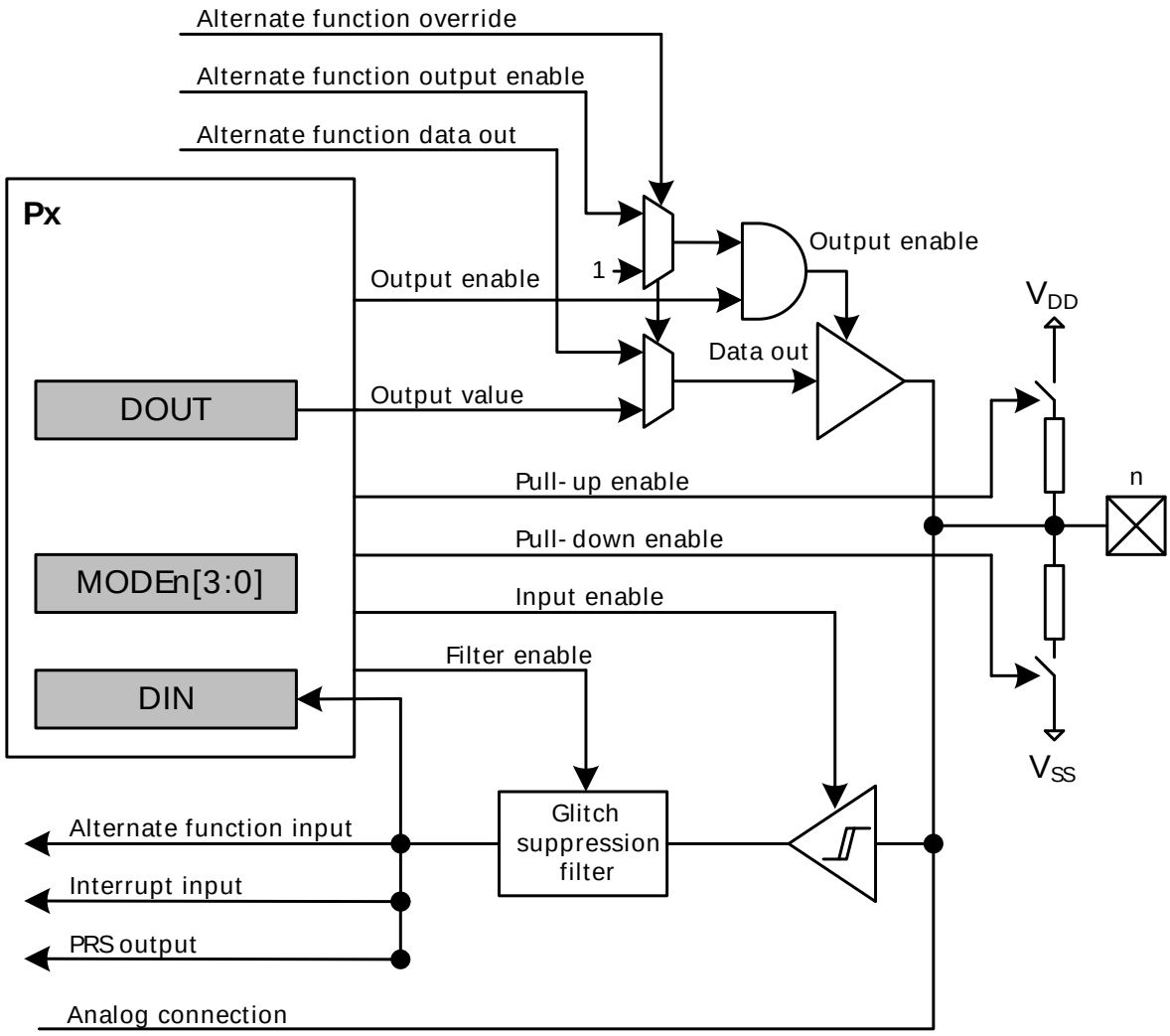
1.1 Introduction

The GPIO module allows the configuration of the I/O pins, which are used by the microcontroller to interact with any external system components. They are organized in ports with up to 16 pins each and are named as Pxn, where x indicates the port (A, B, C...) and n indicates the pin number (0, 1,..., 15). Some ports may have less than 16 pins, so the reader must refer to the device datasheet on this matter. Each port has its own set of configuration registers. These can be used to configure the pins with one of the available of functions. And also allow a flexible peripheral routing.

1.2 Overview

The next image illustrates the internal connections to each of the I/O pins.

Figure 1.1. Pin connections overview



On the output side, it is possible to drive the pins by writing the corresponding bit in the DOUT register. It is also possible to configure the pin for peripheral usage (e.g. UART or LCD connection). All pins can be configured with internal pull ups and pull downs.

In input mode each pin has a selectable filter which can suppress glitches up to 50ns. The pin value is read from the DIN register, and the pin can also be used as an external interrupt source, analog input or PRS producer, connecting it to one of the 8 PRS channels. A configuration lock functionality is also available to avoid accidental changes to the pins' configuration.

2 Configuration

2.1 Overview

The I/O Pins on the EFM32 can have several configurations such as open source or push pull. The reader must refer to the device reference manual for detailed information on each of the modes.

To configure a pin there are 4 registers that must be used: GPIO_Px_MODEL (for pins 0 to 7), GPIO_Px_MODEH (for 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_MODEL_MODE5_MASK) |
GPIO_P_MODEL_MODE5_INPUTPULLFILTER;

GPIO->P[2].DOUT = GPIO->P[2].DOUT | (1 << 5);
```

The proper way to access GPIO Port registers is GPIO->P[x], where x is the Port number (A=0, B=1, ...).

As an alternative to direct register write, the following function can be used:

```
GPIO_PinModeSet(GPIO_Port_TypeDef port, unsigned int pin, GPIO_Mode_TypeDef
mode, unsigned out)
```

The first function parameter would be the port, followed by the pin, functional mode and also the DOUT value, if applicable (for "don't care", either 0 or 1 is valid).

2.2 Reading

The port value is read from the DIN register. Please make sure to set the correct pin mode before reading DIN.

If a specific pin reading is to be done, there are two ways of doing it. The first option is to have a bitwise AND operation with the entire DIN value, masking all the unwanted pins. The other way is using the following function:

```
unsigned int GPIO_PinInGet(GPIO_Port_TypeDef port, unsigned int pin)
```

This function will return the value of the port pin entered as parameter.

2.3 Writing

For a writing operation, the pin must be configured to an output mode such as push pull. Details are on the reference manual. After performing this configuration, the most direct way of changing one or more pins is to write the GPIO_Px_DOUT register. As alternative, registers GPIO_Px_DOUSET, GPIO_Px_DOUTCLR and GPIO_Px_DOUTTGL can also be used and will act on the pins as follows:

- GPIO_Px_DOUT - writing 0/1 to this register will set the pin values to 0/1 correspondingly
- GPIO_Px_DOUTSET - only writes to 1 are effective and will change the pin values to 1
- GPIO_Px_DOUTCLR - only writes to 1 are effective and will change the pin values to 0
- GPIO_Px_DOUTTGL - only writes to 1 are effective and will toggle the pin values

There are also functions available to perform all of these register writes, as follows:

```
void GPIO_PinOutClear(GPIO_Port_TypeDef port, unsigned int pin)

void GPIO_PinOutSet(GPIO_Port_TypeDef port, unsigned int pin)

void GPIO_PinOutToggle(GPIO_Port_TypeDef port, unsigned int pin)

void GPIO_PortOutClear(GPIO_Port_TypeDef port, uint32_t pins)

void GPIO_PortOutSet(GPIO_Port_TypeDef port, uint32_t pins)

void GPIO_PortOutSetVal(GPIO_Port_TypeDef port, uint32_t val, uint32_t mask)

void GPIO_PortOutToggle(GPIO_Port_TypeDef port, uint32_t pins)
```

The first three functions will change the value for one pin only. The first will clear the pin (change the value to 0), the second will set the pin (change the value to 1) and the third will toggle the pin value. The four last functions will change the value of more than one pin, depending on the mask used.

2.4 Drive Strength

All the I/O pins have the same default drive strength, which means that if no alternate drive strength is configured by the user, the current drive will be set to 6mA. If a different current is needed, it is possible to configure an alternate drive strength for each port, and have the selected pins configured to use that alternate drive (see functional modes for details). This means that all the pins from the same port will either have the default drive strength of 6mA or the alternative drive strength configured for the correspondent port.

To configure an alternate drive strength the user can either write directly on GPIO_Px_CTRL register, or use the appropriate function from the emlib.

Refer to the datasheet for maximum combined current drive.

2.5 Configuration Lock

In order to avoid any unwanted or accidental changes to the configurations, it is possible to lock the configuration registers. This includes GPIO_Px_MODEL, GPIO_Px_MODEH, GPIO_Px_CTRL, GPIO_Px_PINLOCKN, GPIO_EXTIPSELL, GPIO_EXTIPSELH, GPIO_INSENSE and GPIO_ROUTE. To enable the lock write any value other than `0xA534` to the GPIO_LOCK register. After writing the unlocking code `0xA534`, the registers will be unlocked. If a reset occurs the registers will also be unlocked.

A reading of the GPIO_LOCK register will indicate the locking status. If it reads 0, the registers are unlocked, and if reads 1 they are locked.

2.6 Pin configuring and read/writing example

On the `gpio_conf_gecko` and `gpio_conf_gg` project there is an example for the Starter Kit. When the PB0 button is pressed, the EFM32 will drive LEDs 0 and 1 with different strengths. LED 0 will have the default of 6 mA and LED 1 will have an alternate drive strength of 0.5 mA.
In the `gpio_conf_tg` pushing the button will toggle the drive strength of the USER LED.

As for the `gpio_conf_dvk`, is to be used with the Development Kit. The user should put either a high or low value on PC12 (pin P4.15 on the protoboard) which is continuously read and outputted to PD8 (pin 5.11 on the protoboard).

3 Peripheral Usage

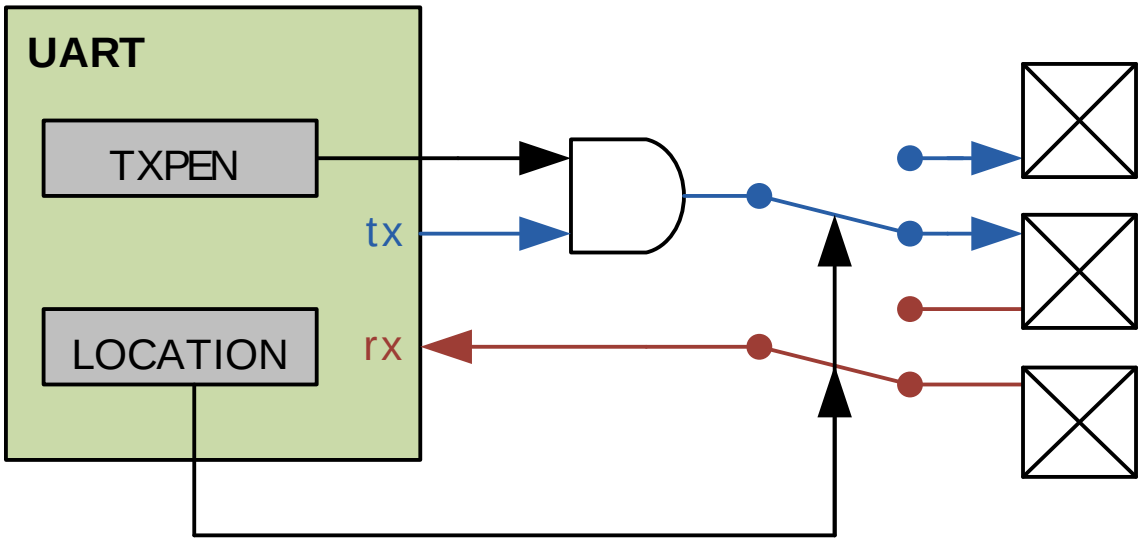
3.1 Routing

Some peripherals such as the USART, the EBI or the LCD driver utilizes one or more I/O pins. Some, such as the LCD, have only one location, but others like the USART, have more than one. For more information on the different locations, the reader must refer to the device datasheet.

The modules with more than one location have a ROUTE register. This register contains a bitfield which selects the pins where the module will be connected. It also has bits which enable or disable the peripherals pin connections, the xxPEN bits. Peripheral pin connections are called alternate functions on the GPIO module.

If both alternate function and output are enabled for the same pin, the alternate function's output data will overwrite the output driver. Nevertheless, the state of the pin's configuration registers will remain. This means that a pin must be set as an output so that the peripherals are able to override it and use it as an output pin. Also, the user must be careful on which output mode the pin is configured to. For instance, if the pin is configured as push pull, the peripheral will be able to drive both high and low values, but if it is configured to open-drain mode without a pull up (either internal or external), the peripheral will only be able to drive a low value.

Figure 3.1. Peripheral Routing



It is also possible to have two or more peripherals connected to the same pin. However, this is not recommended, and the user should solve all peripheral conflicts so that each pin only has one peripheral connected at a time.

3.2 Routing Example

The `gpio_periph` project can be used in both DVK and STK. This example will configure pins PC12 (P4.15 on the protoboard) and PD8 (P5.11 on the protoboard) to output the High and Low Frequency RC Oscillators respectively. In the `gpio_periph_tg` and `gpio_periph_gg` project the Low Frequency RC Oscillators will be output on PD8.

4 Interrupts

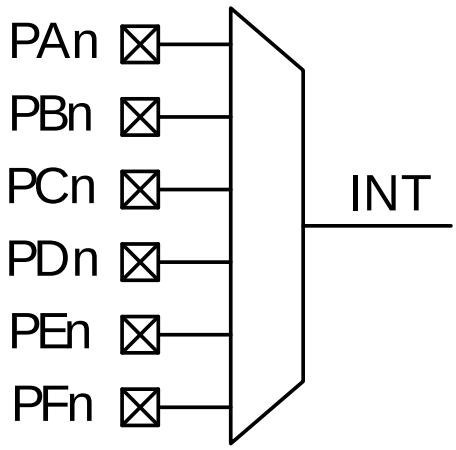
4.1 GPIO as external interrupt source

Any of the GPIO pins can generate an interrupt, which in turn can wake up the core from all energy modes except EM4.

The pins with the same number from each of the ports are grouped together to generate an interrupt, meaning that the user can't, for instances, use pin 0 from both PORT A and B as external interrupt sources. There can be up to 16 external interrupt sources, which are connected to 2 interrupt lines. The even numbered pins will generate an interrupt on the even interrupt line, and the odd numbered pins on the odd line. The interrupts can be generated by a rising edge transition, falling edge or both.

It can be advisable to enable the input glitch suppression filter on the interrupt pins.

Figure 4.1. Interrupts



To setup an interrupt, registers GPIO_EXTIPSELL and GPIO_EXTIPSELH are used to select which pins will be used as interrupt sources. The interrupt triggering edge must be selected using registers GPIO_EXTIRISE and/or GPIO_EXTIFALL. Next, the bits for the correspondent pins on GPIO_IEN must be set to activate the interrupts and the interrupt sensing must be activated on GPIO_INSENSE register. Also, the corresponding interrupt vector must be activated on the NVIC. Refer to the reference manual for details. When an interrupt occurs, a bit will be set on GPIO_IF, and it will trigger an interrupt request on either IRQ_GPIO_EVEN or IRQ_GPIO_ODD lines.

4.2 Interrupt Example

To exemplify the external interrupt feature, the code is very similar to the configuration example. For the STK (`gpio_int_stk`) , when the user presses PB0 LEDs 0 and 1 will be driven with different currents. But the EFM32 will be in EM3 when the button is not pressed.

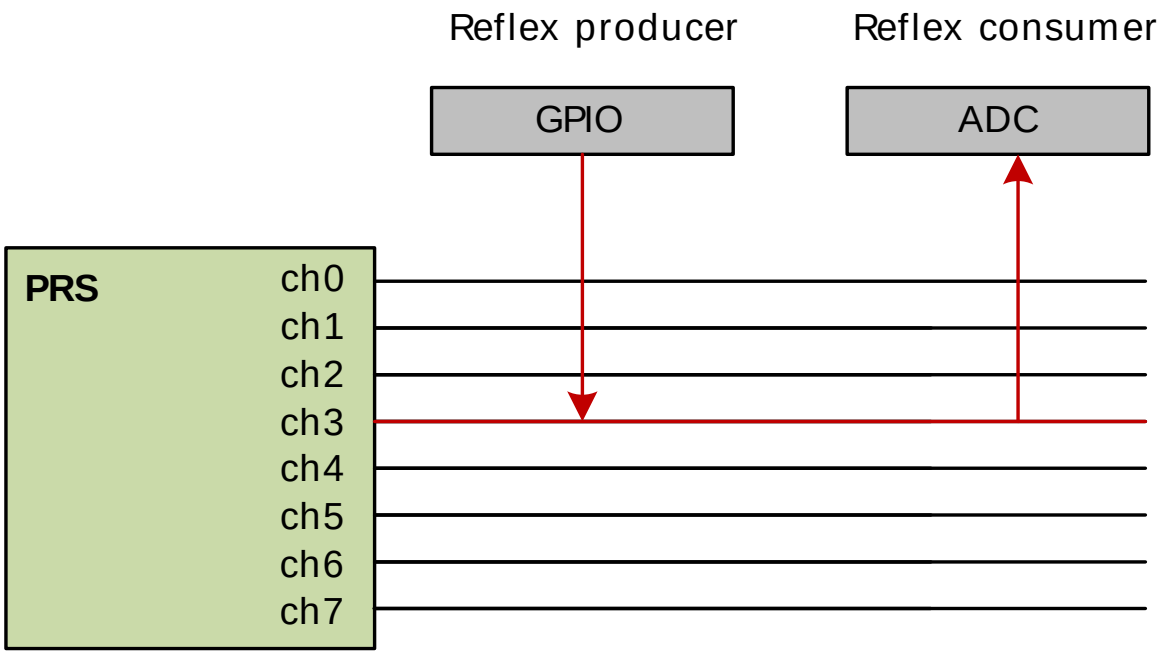
On the DVK (`gpio_int_dvk`) the user can use the advanced energy monitoring to observe the current variation when the EFM32 goes to run mode and then back to EM3. To achieve that, PORTE1 (pin 6.4 on the protoboard) must be grounded, which will cause an interrupt, but no output pins will be driven on this example.

5 Peripheral Reflex System (PRS) output

5.1 GPIO as PRS producer

The PRS allows communication between peripherals without CPU intervention. This enables peripherals to interact in energy modes where the CPU is shutdown. There are Reflex producers, which are the peripherals who send Reflex signals (can be both pulses or levels), and Reflex consumers, the peripherals who take actions when they receive those signals. Please refer to the device reference manual for a table listing all producers and consumers and they're respective output and action type.

Figure 5.1. PRS output



Every GPIO pin can be a Reflex producer. They are grouped in the same way as described previously for the external interrupt feature. This means that the same numbered pins from different ports can not be configured as PRS producers simultaneously. To configure a pin as a PRS producer, it must be selected as an interrupt source on the GPIO_EXTIPSELL or GPIO_EXTIPSELH and then enable the PRS Sense on the GPIO_INSENSE register. Also, on the PRS module, the pin must be configured as an input source for one or more of the available channels.

5.2 PRS Example

On the `gpio_prs_dvk` and `gpio_prs_dvk` projects, the EMF32 will be in EM1. A falling edge variation on a pin triggers an ADC single conversion. When the conversion ends, the ADC will cause an interrupt to wake up the processor, and a delay cycle will keep it running for 1 second. This is for the user to be able to see the current variation using the AEM or EnergyAware Profiler. On the STK, the user should press PB0 button to trigger the ADC, and on the DVK should ground PE0 (pin 6.4 on the protoboard).

6 Revision History

6.1 Revision 1.06

2013-09-03

New cover layout

6.2 Revision 1.05

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

6.3 Revision 1.04

2012-11-12

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

6.4 Revision 1.03

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

6.5 Revision 1.02

2012-03-14

Fixed makefile-error for CodeSourcery projects.

6.6 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.

6.7 Revision 1.00

September 20th, 2010.

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories 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 Laboratories 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 Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. 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 Laboratories products are generally not intended for military applications. Silicon Laboratories 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.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, the Silicon Labs logo, Energy Micro, EFM, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

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

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

silabs.com


ZERO
ARM Cortex-M0+
TINY
ARM Cortex-M3
GECKO
ARM Cortex-M3
LEOPARD
ARM Cortex-M3
GIANT
ARM Cortex-M3
WONDER
ARM Cortex-M4