GPIO Module

The GPIO module provides an API to configure, read from, and write to the GPIO pins. Functions fall into the two categories, control and data. Control functions configure properties like direction, pin muxing, and qualification. Data functions allow you to read the value on a pin or write a value to it.

Most functions will configure a single pin at a time. The pin to be configured will be specified using its GPIO number. Refer to the device's datasheet to learn what numbers are valid for that part number. Also note that even if a GPIO number is valid for a part number, it may not be valid for all possible features. For instance, GPIO_setAnalogMode() is only usable for a fraction of the GPIO numbers.

For information and functions to configure a pin for low-power mode wake-up, see the SysCtl module.

group **gpio_api**

Defines

GPIO_CTRL_REGS_STEP ((GPIO_O_GPBCTRL - GPIO_O_GPACTRL) / 2U)

GPIO_DATA_REGS_STEP ((GPIO_O_GPBDAT - GPIO_O_GPADAT) / 2U)

GPIO_DATA_READ_REGS_STEP ((GPIO_O_GPBDAT_R - GPIO_O_GPADAT_R) / 2U)

GPIO_GPxCTRL_INDEX (GPIO_O_GPACTRL / 2U)

GPIO_GPxQSEL_INDEX (GPIO_O_GPAQSEL1 / 2U)

GPIO_GPxMUX_INDEX (GPIO_O_GPAMUX1 / 2U)

GPIO_GPxDIR_INDEX (GPIO_O_GPADIR / 2U)

GPIO_GPxAMSEL_INDEX (0x00000014U / 2U)

GPIO_GPxPUD_INDEX (GPIO_O_GPAPUD / 2U)

GPIO_GPxINV_INDEX (GPIO_O_GPAINV / 2U)

GPIO_GPxODR_INDEX (GPIO_O_GPAODR / 2U)

GPIO_GPxGMUX_INDEX (GPIO_O_GPAGMUX1 / 2U)

GPIO_GPxCSEL_INDEX (GPIO_O_GPACSEL1 / 2U)

GPIO_GPxLOCK_INDEX (GPIO_O_GPALOCK / 2U)

GPIO_GPxCR_INDEX (GPIO_O_GPACR / 2U)

GPIO_GPxDAT_INDEX (GPIO_O_GPADAT / 2U)

GPIO_GPxSET_INDEX (GPIO_O_GPASET / 2U)

GPIO_GPxCLEAR_INDEX (GPIO_O_GPACLEAR / 2U)

GPIO_GPxTOGGLE_INDEX (GPIO_O_GPATOGGLE / 2U)

GPIO_GPxDAT_R_INDEX (GPIO_O_GPADAT_R / 2U)

GPIO_MUX_TO_GMUX (GPIO_O_GPAGMUX1 - GPIO_O_GPAMUX1)

GPIO_PIN_TYPE_STD 0x0000U

Push-pull output or floating input.

GPIO_PIN_TYPE_PULLUP 0x0001U

Pull-up enable for input.

GPIO_PIN_TYPE_INVERT 0x0002U

Invert polarity on input.

GPIO_PIN_TYPE_OD 0x0004U

Open-drain on output.

Enums

enum **GPIO_Direction**

Values that can be passed to GPIO_setDirectionMode() as the *pinIO* parameter and returned from GPIO_getDirectionMode().

Values:

enumerator **GPIO_DIR_MODE_IN**

Pin is a GPIO input.

enumerator **GPIO_DIR_MODE_OUT**

Pin is a GPIO output.

enum **GPIO_IntType**

Values that can be passed to GPIO_setInterruptType() as the *intType* parameter and returned from GPIO_getInterruptType().

Values:

enumerator **GPIO_INT_TYPE_FALLING_EDGE** = 0x00

Interrupt on falling edge.

enumerator **GPIO_INT_TYPE_RISING_EDGE** = 0x04

Interrupt on rising edge.

enumerator **GPIO_INT_TYPE_BOTH_EDGES** = 0x0C

Interrupt on both edges.

enum **GPIO_QualificationMode**

Values that can be passed to GPIO_setQualificationMode() as the *qualification* parameter and returned by GPIO_getQualificationMode().

Values:

enumerator **GPIO_QUAL_SYNC**

Synchronization to SYSCLKOUT.

enumerator **GPIO_QUAL_3SAMPLE**

Qualified with 3 samples.

enumerator **GPIO_QUAL_6SAMPLE**

Qualified with 6 samples.

enumerator **GPIO_QUAL_ASYNC**

No synchronization.

enum **GPIO_AnalogMode**

Values that can be passed to GPIO_setAnalogMode() as the *mode* parameter.

Values:

enumerator **GPIO_ANALOG_DISABLED**

Pin is in digital mode.

enumerator **GPIO_ANALOG_ENABLED**

Pin is in analog mode.

enum **GPIO_Port**

Values that can be passed to GPIO_readPortData(), GPIO_setPortPins(), GPIO_clearPortPins(), and GPIO_togglePortPins() as the *port* parameter.

Values:

enumerator **GPIO_PORT_A** = 0

GPIO port A.

enumerator **GPIO_PORT_B** = 1

GPIO port B.

enumerator **GPIO_PORT_H** = 7

GPIO port H.

enum **GPIO_ExternalIntNum**

Values that can be passed to GPIO_setInterruptPin(), GPIO_setInterruptType(), GPIO_getInterruptType(), GPIO_enableInterrupt(), GPIO_disableInterrupt(), as the *extIntNum* parameter.

Values:

enumerator **GPIO_INT_XINT1**

External Interrupt 1.

enumerator **GPIO_INT_XINT2**

External Interrupt 2.

enumerator **GPIO_INT_XINT3**

External Interrupt 3.

enumerator **GPIO_INT_XINT4**

External Interrupt 4.

enumerator **GPIO_INT_XINT5**

External Interrupt 5.

Functions

void **GPIO_setInterruptType**(GPIO_ExternalIntNum *extIntNum*, GPIO_IntType *intType*)

Sets the interrupt type for the specified pin.

This function sets up the various interrupt trigger mechanisms for the specified pin on the selected GPIO port.

Parameters

- `extIntNum`: specifies the external interrupt.
- `intType`: specifies the type of interrupt trigger mechanism.

The following defines can be used to specify the external interrupt for the *extIntNum* parameter:

- GPIO_INT_XINT1
- GPIO_INT_XINT2

- GPIO_INT_XINT3
- GPIO_INT_XINT4
- GPIO_INT_XINT5

One of the following flags can be used to define the *intType* parameter:

- **GPIO_INT_TYPE_FALLING_EDGE** sets detection to edge and trigger to falling
- **GPIO_INT_TYPE_RISING_EDGE** sets detection to edge and trigger to rising
- **GPIO_INT_TYPE_BOTH_EDGES** sets detection to both edges

> **Return**
>
> None.

GPIO_IntType **GPIO_getInterruptType**(GPIO_ExternalIntNum *extIntNum*)

Gets the interrupt type for a pin.

This function gets the interrupt type for a interrupt. The interrupt can be configured as a falling-edge, rising-edge, or both-edges detected interrupt.

> **Parameters**
>
> - `extIntNum`: specifies the external interrupt.

The following defines can be used to specify the external interrupt for the *extIntNum* parameter:

- GPIO_INT_XINT1
- GPIO_INT_XINT2
- GPIO_INT_XINT3
- GPIO_INT_XINT4
- GPIO_INT_XINT5

> **Return**
>
> Returns one of the flags described for GPIO_setInterruptType().

void **GPIO_enableInterrupt**(GPIO_ExternalIntNum *extIntNum*)

Enables the specified external interrupt.

This function enables the indicated external interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt. Disabled sources have no effect on the processor.

Parameters

- `extIntNum`: specifies the external interrupt.

The following defines can be used to specify the external interrupt for the *extIntNum* parameter:

- GPIO_INT_XINT1
- GPIO_INT_XINT2
- GPIO_INT_XINT3
- GPIO_INT_XINT4
- GPIO_INT_XINT5

Return

None.

void **GPIO_disableInterrupt**(GPIO_ExternalIntNum *extIntNum*)

Disables the specified external interrupt.

This function disables the indicated external interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt. Disabled sources have no effect on the processor.

Parameters

- `extIntNum`: specifies the external interrupt.

The following defines can be used to specify the external interrupt for the *extIntNum* parameter:

- GPIO_INT_XINT1
- GPIO_INT_XINT2
- GPIO_INT_XINT3
- GPIO_INT_XINT4
- GPIO_INT_XINT5

Return

None.

uint16_t **GPIO_getInterruptCounter**(GPIO_ExternalIntNum *extIntNum*)

Gets the value of the external interrupt counter.

The following defines can be used to specify the external interrupt for the

extIntNum parameter:

Parameters

- `extIntNum`: specifies the external interrupt.

- GPIO_INT_XINT1
- GPIO_INT_XINT2
- GPIO_INT_XINT3

Note: The counter is clocked at the SYSCLKOUT rate.

Return

Returns external interrupt counter value.

uint32_t **GPIO_readPin**(uint32_t *pin*)

Reads the value present on the specified pin.

The value at the specified pin are read, as specified by

pin. The value is returned for both input and output pins.

Parameters

- `pin`: is the identifying GPIO number of the pin.

The pin is specified by its numerical value. For example, GPIO34 is specified by passing 34 as *pin*.

Return

Returns the value in the data register for the specified pin.

uint32_t **GPIO_readPinDataRegister**(uint32_t *pin*)

Reads the data register value for specified pin.

The value available at the data register for the specified pin is read, as specified by

pin. The value is returned for both input and output pins.

Parameters

- `pin` : is the identifying GPIO number of the pin.

The pin is specified by its numerical value. For example, GPIO34 is specified by passing 34 as *pin*.

See

GPIO_readPin()

Return

Returns the value in the data register for the specified pin.

void **GPIO_writePin**(uint32_t *pin*, uint32_t *outVal*)

Writes a value to the specified pin.

Writes the corresponding bit values to the output pin specified by

pin. Writing to a pin configured as an input pin has no effect.

Parameters

- `pin` : is the identifying GPIO number of the pin.
- `outVal` : is the value to write to the pin.

The pin is specified by its numerical value. For example, GPIO34 is specified by passing 34 as *pin*.

Return

None.

void **GPIO_togglePin**(uint32_t *pin*)

Toggles the specified pin.

Writes the corresponding bit values to the output pin specified by

pin. Writing to a pin configured as an input pin has no effect.

Parameters

- `pin` : is the identifying GPIO number of the pin.

The pin is specified by its numerical value. For example, GPIO34 is specified by passing 34 as *pin*.

Return

None.

uint32_t **GPIO_readPortData**(GPIO_Port *port*)

Reads the data on the specified port.

Return

Returns the value available on pin for the specified port. Each bit of the the return value represents a pin on the port, where bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

Parameters

- `port`: is the GPIO port being accessed in the form of **GPIO_PORT_X** where X is the port letter.

uint32_t **GPIO_readPortDataRegister**(GPIO_Port *port*)

Reads the data written in GPIO Data Register.

Reads the data written in GPIO Data Register for the specified port. In previous devices, read of GPIO data registers resulted in read of corespoinding pins. The function

GPIO_readPortData() returns the value on pin.

Parameters

- `port`: is the GPIO port being accessed in the form of **GPIO_PORT_X** where X is the port letter.

See

GPIO_readPortData()

Return

Returns the value in the data register for the specified port. Each bit of the the return value represents a pin on the port, where bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

void **GPIO_writePortData**(GPIO_Port *port*, uint32_t *outVal*)

Writes a value to the specified port.

This function writes the value

outVal to the port specified by the *port* parameter which takes a value in the form of **GPIO_PORT_X** where X is the port letter. For example, use **GPIO_PORT_A** to affect

port A (GPIOs 0-31).

> **Parameters**

- `port`: is the GPIO port being accessed.
- `outVal`: is the value to write to the port.

The *outVal* is a bit-packed value, where each bit represents a bit on a GPIO port. Bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

> **Return**

None.

void **GPIO_setPortPins**(GPIO_Port*port*, uint32_t *pinMask*)

Sets all of the specified pins on the specified port.

This function sets all of the pins specified by the

pinMask parameter on the port specified by the *port* parameter which takes a value in the form of **GPIO_PORT_X** where X is the port letter. For example, use **GPIO_PORT_A** to affect port A (GPIOs 0-31).

> **Parameters**

- `port`: is the GPIO port being accessed.
- `pinMask`: is a mask of which of the 32 pins on the port are affected.

The *pinMask* is a bit-packed value, where each bit that is set identifies the pin to be set. Bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

> **Return**

None.

void **GPIO_clearPortPins**(GPIO_Port*port*, uint32_t *pinMask*)

Clears all of the specified pins on the specified port.

This function clears all of the pins specified by the

pinMask parameter on the port specified by the *port* parameter which takes a value in the form of **GPIO_PORT_X** where X is the port letter. For example, use **GPIO_PORT_A** to affect port A (GPIOs 0-31).

> **Parameters**

- `port`: is the GPIO port being accessed.

- `pinMask`: is a mask of which of the 32 pins on the port are affected.

The *pinMask* is a bit-packed value, where each bit that is **set** identifies the pin to be cleared. Bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

> **Return**
>
> None.

void **GPIO_togglePortPins**(GPIO_Port*port*, uint32_t *pinMask*)

Toggles all of the specified pins on the specified port.

This function toggles all of the pins specified by the

pinMask parameter on the port specified by the *port* parameter which takes a value in the form of **GPIO_PORT_X** where X is the port letter. For example, use **GPIO_PORT_A** to affect port A (GPIOs 0-31).

> **Parameters**

- `port`: is the GPIO port being accessed.
- `pinMask`: is a mask of which of the 32 pins on the port are affected.

The *pinMask* is a bit-packed value, where each bit that is set identifies the pin to be toggled. Bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

> **Return**
>
> None.

void **GPIO_lockPortConfig**(GPIO_Port*port*, uint32_t *pinMask*)

Locks the configuration of the specified pins on the specified port.

This function locks the configuration registers of the pins specified by the

pinMask parameter on the port specified by the *port* parameter which takes a value in the form of **GPIO_PORT_X** where X is the port letter. For example, use **GPIO_PORT_A** to affect port A (GPIOs 0-31).

> **Parameters**

- `port`: is the GPIO port being accessed.
- `pinMask`: is a mask of which of the 32 pins on the port are affected.

The *pinMask* is a bit-packed value, where each bit that is set identifies the pin to be locked. Bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, 0xFFFFFFFF represents all pins on that port, and so on.

Note that this function is for locking the configuration of a pin such as the pin muxing, direction, open drain mode, and other settings. It does not affect the ability to change the value of the pin.

Return

None.

void **GPIO_unlockPortConfig**(GPIO_Port*port*, uint32_t *pinMask*)

Unlocks the configuration of the specified pins on the specified port.

This function unlocks the configuration registers of the pins specified by the

pinMask parameter on the port specified by the *port* parameter which takes a value in the form of **GPIO_PORT_X** where X is the port letter. For example, use **GPIO_PORT_A** to affect port A (GPIOs 0-31).

Parameters

- `port`: is the GPIO port being accessed.
- `pinMask`: is a mask of which of the 32 pins on the port are affected.

The *pinMask* is a bit-packed value, where each bit that is set identifies the pin to be unlocked. Bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, 0xFFFFFFFF represents all pins on that port, and so on.

Return

None.

void **GPIO_commitPortConfig**(GPIO_Port*port*, uint32_t *pinMask*)

Commits the lock configuration of the specified pins on the specified port.

This function commits the lock configuration registers of the pins specified by the

pinMask parameter on the port specified by the *port* parameter which takes a value in the form of **GPIO_PORT_X** where X is the port letter. For example, use **GPIO_PORT_A** to affect port A (GPIOs 0-31).

Parameters

- `port`: is the GPIO port being accessed.
- `pinMask`: is a mask of which of the 32 pins on the port are affected.

The *pinMask* is a bit-packed value, where each bit that is set identifies the pin to be locked. Bit 0 represents GPIO port pin 0, bit 1 represents GPIO port pin 1, 0xFFFFFFFF represents all pins on that port, and so on.

Note that once this function is called, GPIO_lockPortConfig() and GPIO_unlockPortConfig() will no longer have any effect on the specified pins.

> **Return**
>
> None.

void **GPIO_setDirectionMode**(uint32_t *pin*, GPIO_Direction*pinIO*)

Sets the direction and mode of the specified pin.

This function configures the specified pin on the selected GPIO port as either input or output.

> **Parameters**
>
> - `pin`: is the identifying GPIO number of the pin.
> - `pinIO`: is the pin direction mode.

The parameter *pinIO* is an enumerated data type that can be one of the following values:

- **GPIO_DIR_MODE_IN**
- **GPIO_DIR_MODE_OUT**

where **GPIO_DIR_MODE_IN** specifies that the pin is programmed as an input and **GPIO_DIR_MODE_OUT** specifies that the pin is programmed as an output.

The pin is specified by its numerical value. For example, GPIO34 is specified by passing 34 as *pin*.

> **Return**
>
> None.

GPIO_Direction**GPIO_getDirectionMode**(uint32_t *pin*)

Gets the direction mode of a pin.

This function gets the direction mode for a specified pin. The pin can be configured as either an input or output The type of direction is returned as an enumerated data type.

> **Parameters**

- `pin`: is the identifying GPIO number of the pin.

> **Return**
>
> Returns one of the enumerated data types described for GPIO_setDirectionMode().

void **GPIO_setInterruptPin**(uint32_t *pin*, GPIO_ExternalIntNum *extIntNum*)

Sets the pin for the specified external interrupt.

This function sets which pin triggers the selected external interrupt.

> **Parameters**

- `pin`: is the identifying GPIO number of the pin.
- `extIntNum`: specifies the external interrupt.

The following defines can be used to specify the external interrupt for the *extIntNum* parameter:

- **GPIO_INT_XINT1**
- **GPIO_INT_XINT2**
- **GPIO_INT_XINT3**
- **GPIO_INT_XINT4**
- **GPIO_INT_XINT5**

The pin is specified by its numerical value. For example, GPIO34 is specified by passing 34 as *pin*.

> **See**
>
> XBAR_setInputPin()

> **Return**
>
> None.

void **GPIO_setPadConfig**(uint32_t *pin*, uint32_t *pinType*)

Sets the pad configuration for the specified pin.

This function sets the pin type for the specified pin. The parameter

pinType can be the following values:

> **Parameters**

- `pin`: is the identifying GPIO number of the pin.
- `pinType`: specifies the pin type.

- **GPIO_PIN_TYPE_STD** specifies a push-pull output or a floating input
- **GPIO_PIN_TYPE_PULLUP** specifies the pull-up is enabled for an input
- **GPIO_PIN_TYPE_OD** specifies an open-drain output pin
- **GPIO_PIN_TYPE_INVERT** specifies inverted polarity on an input

GPIO_PIN_TYPE_INVERT may be OR-ed with **GPIO_PIN_TYPE_STD** or **GPIO_PIN_TYPE_PULLUP**.

The pin is specified by its numerical value. For example, GPIO34 is specified by passing 34 as *pin*.

> **Return**
>
> None.

uint32_t **GPIO_getPadConfig**(uint32_t *pin*)

Gets the pad configuration for a pin.

This function returns the pin type for the specified pin. The value returned corresponds to the values used in

GPIO_setPadConfig().

> **Parameters**
>
> - `pin`: is the identifying GPIO number of the pin.
>
> **Return**
>
> Returns a bit field of the values **GPIO_PIN_TYPE_STD**, **GPIO_PIN_TYPE_PULLUP**, **GPIO_PIN_TYPE_OD**, and **GPIO_PIN_TYPE_INVERT**.

void **GPIO_setQualificationMode**(uint32_t *pin*, GPIO_QualificationMode*qualification*)

Sets the qualification mode for the specified pin.

This function sets the qualification mode for the specified pin. The parameter

qualification can be one of the following values:

- **GPIO_QUAL_SYNC**
- **GPIO_QUAL_3SAMPLE**
- **GPIO_QUAL_6SAMPLE**

- GPIO_QUAL_ASYNC

> **Parameters**
>
> - `pin`: is the identifying GPIO number of the pin.
> - `qualification`: specifies the qualification mode of the pin.

To set the qualification sampling period, use GPIO_setQualificationPeriod().

> **Return**
>
> None.

GPIO_QualificationMode **GPIO_getQualificationMode**(uint32_t *pin*)

Gets the qualification type for the specified pin.

> **Return**
>
> Returns the qualification mode in the form of one of the values **GPIO_QUAL_SYNC**, **GPIO_QUAL_3SAMPLE**, **GPIO_QUAL_6SAMPLE**, or **GPIO_QUAL_ASYNC**.

> **Parameters**
>
> - `pin`: is the identifying GPIO number of the pin.

void **GPIO_setQualificationPeriod**(uint32_t *pin*, uint32_t *divider*)

Sets the qualification period for a set of pins

This function sets the qualification period for a set of

8 pins, specified by the *pin* parameter. For instance, passing in 3 as the value of *pin* will set the qualification period for GPIO0 through GPIO7, and a value of 98 will set the qualification period for GPIO96 through GPIO103. This is because the register field that configures the divider is shared.

> **Parameters**
>
> - `pin`: is the identifying GPIO number of the pin.
> - `divider`: specifies the output drive strength.

To think of this in terms of an equation, configuring *pin* as **n** will configure GPIO (n & ~(7)) through GPIO ((n & ~(7)) + 7).

divider is the value by which the frequency of SYSCLKOUT is divided. It can be 1 or an even value between 2 and 510 inclusive.

Return

None.

void **GPIO_setAnalogMode**(uint32_t *pin*, GPIO_AnalogMode*mode*)

Sets the analog mode of the specified pin.

This function configures the specified pin for either analog or digital mode. Not all GPIO pins have the ability to be switched to analog mode, so refer to the technical reference manual for details. This setting should be thought of as another level of muxing.

Parameters

- `pin`: is the identifying GPIO number of the pin.
- `mode`: is the selected analog mode.

The parameter *mode* is an enumerated data type that can be one of the following values:

- **GPIO_ANALOG_DISABLED** - Pin is in digital mode
- **GPIO_ANALOG_ENABLED** - Pin is in analog mode

The pin is specified by its numerical value. For example, GPIO34 is specified by passing 34 as *pin*.

Return

None.

void **GPIO_setPinConfig**(uint32_t *pinConfig*)

Configures the alternate function of a GPIO pin.

This function configures the pin mux that selects the peripheral function associated with a particular GPIO pin. Only one peripheral function at a time can be associated with a GPIO pin, and each peripheral function should only be associated with a single GPIO pin at a time (despite the fact that many of them can be associated with more than one GPIO pin).

Parameters

- `pinConfig`: is the pin configuration value, specified as only one of the **GPIO_::_????** values.

The available mappings are supplied in `pin_map.h`.

Return

None.

The first step to configuring GPIO is to figure out the peripheral muxing. The function to configure the mux registers is GPIO_setPinConfig(). The values to be passed to this function to specify the functionality the pin should have are found in pin_map.h.

Next, use GPIO_setPadConfig() to configure any properties like internal pullups, open-drain, or an inverted input signal. GPIO_setQualificationMode() and GPIO_setQualificationPeriod() can be used to configure any needed input qualification.

Then, for pins configured as GPIOs, use GPIO_setDirectionMode() to select a direction. Take care to write the desired initial value for that pin using GPIO_writePin() before configuring a pin as an output to avoid any glitches.

Several functions are provided for the configuration of external interrupts. These functions use the device's XINT module. The Input X-BAR is also leveraged to configure the pin on which an event will cause an interrupt. These functions are GPIO_setInterruptType(), GPIO_getInterruptType(), GPIO_enableInterrupt(), GPIO_disableInterrupt(), and GPIO_setInterruptPin().

Most functions operate on one pin at a time. However, there are a few functions that can operate on an entire port at once for the sake of efficiency. These are the data functions GPIO_readPortData(), GPIO_writePortData(), GPIO_setPortPins(), GPIO_clearPortPins(), and GPIO_togglePortPins(). Other data functions that affect a single pin at a time are GPIO_readPin(), GPIO_writePin(), and GPIO_togglePin().

The code for this module is contained in driverlib/gpio.c, with driverlib/gpio.h containing the API declarations for use by applications.