lib_gpio: GPIO abstraction for multibit ports

Publication Date: 2025/6/18
Document Number: XM-010422-UG v2.2.0

XMOS

IN THIS DOCUMENT

1 Inroduction

The XMOS GPIO library allows accessing xcore ports as low-speed GPIO.

Although xcore ports can be directly accessed via the xC programming language this library allows more flexible usage. In particular, it allows splitting a multi-pin output/input port to be able to use the individual pins independently. It also allows accessing ports across separate XMOS tiles or separate XMOS chips.

lib_gpio is intended to be used with the XCommon CMake , the *XMOS* application build and dependency management system.

To use this library, include **lib_gpio** in the application's **APP_DEPENDENT_MODULES** list in *CMakeLists.txt*, for example:

```
set(APP_DEPENDENT_MODULES "lib_gpio")
```

Applications should then include the **gpio.h** header file.

2 Connecting external signals to multi-bit ports

Multi-bit ports can be connected to independent signals in either an all output configuration (see *Output configuration*) or an all input configuration (see *Input configuration*). This implies two important restrictions:

- **Bi-directional signals cannot use this library**
- **The signals on the same port must go in the same direction**

To use bi-directional signals, a dedicated 1-bit hardware port needs to be used.

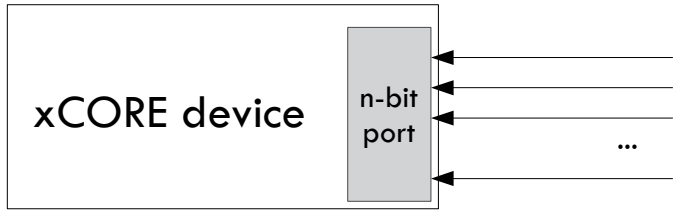

Fig. 1: Input configuration

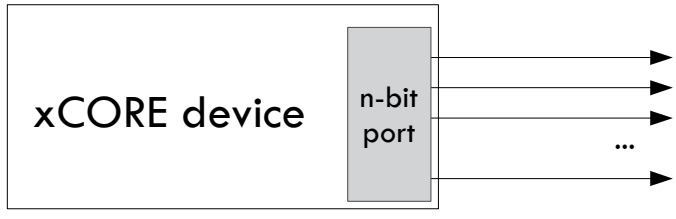

Fig. 2: Output configuration

2.1 Performance restrictions

This library allows independent access to the pins of mulit-bit ports by multiplexing the port output or input in software. This means that there are some performance implications, namely:

- The internal buffering, serializing and de-serializing features of the xCORE port are not available.
- The software locking and multiplexing between individual bits of the port limits performance. As such, toggling pins at speed above 1Mhz, for example, is not achievable (on a 62.5Mhz logical core). The limit may be lower depending on the other code is running on the core and how the other pins of the port are being driven.

As such, sharing multi-bit ports is most suitable for slow I/O such as LEDs, buttons and reset lines.

3 Usage

There are three ways to use the GPIO library:

GPIO type	Description
Output GPIO	Allows control of individual bits of a multi-bit output port.
Input GPIO	Allows reading of individual bits of a multi-bit input port.
Input GPIO with events	Allows reading of individual bits of a multi-bit input port and reacting to events on those pins.

3.1 Output GPIO usage

Output GPIO components are instantiated as parallel tasks that run in a `par` statement. These components connect to the hardware ports of the xCORE device. The application can connect via an interface connection using an array of the `output_gpio_if` interface type like in *Output GPIO task diagram*.

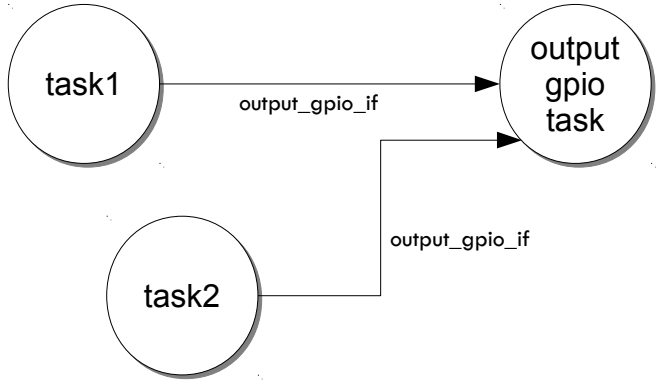

Fig. 3: Output GPIO task diagram

For example, the following code instantiates an output GPIO component for the first 3 pins of a port and connects to it

```
port p = XS1_PORT_4C;

int main(void) {
  output_gpio_if i_gpio[3];
  par {
    output_gpio(i_gpio, 3, p, null);
    task1(i_gpio[0], i_gpio[1]);
    task2(i_gpio[2]);
  }
  return 0;
}
```

Note that the connection is an array of interfaces, so several tasks can connect to the same component instance, each controlling different pins of the port.

The application can use the client end of the interface connection to perform GPIO operations e.g.

```
void task1(client output_gpio_if gpio1, client output_gpio_if gpio2)
{
  // ...
  gpio1.output(1);
  gpio2.output(0);
```

(continues on next page)

(continued from previous page)

```
  delay_milliseconds(200);
  gpio1.output(0);
  gpio2.output(1);
  // ...
}
```

More information on interfaces and tasks can be be found in the XMOS Programming Guide. By default the output GPIO component does not use any logical cores of its own. It is a *distributed* task which means it will perform its function on the logical core of the application task connected to it (provided the application task is on the same tile).

3.2 Input GPIO usage

There are two types of input GPIO component: those that support events and those that do not support events. In both cases, input GPIO components are instantiated as parallel tasks that run in a `par` statement. These components connect to the hardware ports of the xCORE device. The application can connect via an interface connection using an array of the `input_gpio_if` interface type like in *Input GPIO task diagram*.

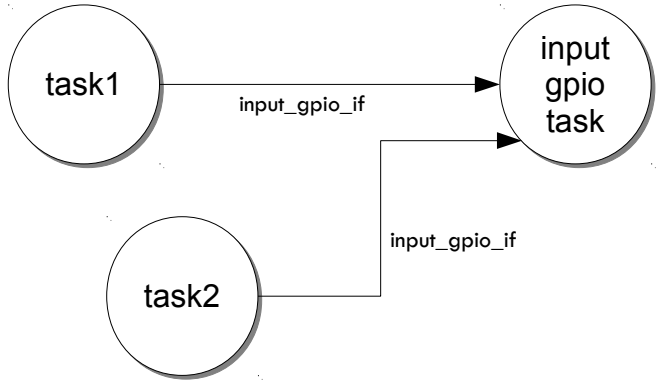

Fig. 4: Input GPIO task diagram

For example, the following code instantiates an input GPIO component for the first 3 pins of a port and connects to it

```
port p = XS1_PORT_4C;

int main(void) {
  input_gpio_if i_gpio[3];
  par {
    input_gpio(i_gpio, 3, p, null);
    task1(i_gpio[0], i_gpio[1]);
    task2(i_gpio[2]);
  }
  return 0;
}
```

Note that the connection is an array of interfaces, so several tasks can connect to the same component instance, each controlling different pins of the port.

The application can use the client end of the interface connection to perform GPIO operations e.g.

```
void task1(client input_gpio_if gpio1, client input_gpio_if gpio2)
{
  // ...
  val1 = gpio1.input();
  val2 = gpio2.input();
  // ...
  val1 = gpio1.input();
  val2 = gpio2.input();
  // ...
}
```

More information on interfaces and tasks can be be found in the XMOS Programming Guide. By default the output GPIO component does not use any logical cores of its own. It is a *distributed* task which means it will perform its function on the logical core of the application task connected to it (provided the application task is on the same tile).

3.3 Input GPIO using events

The *input_gpio_with_events()* and *input_gpio_1bit_with_events()* functions support the event based functions of the input GPIO interface

```
port p = XS1_PORT_4C;

int main(void) {
  input_gpio_if i_gpio[3];
  par {
    input_gpio_with_events(i_gpio, 3, p, null);
    task1(i_gpio[0], i_gpio[1]);
    task2(i_gpio[2]);
  }
  return 0;
}
```

In this case the application can request an event on a pin change and then select on the event happening e.g.

```
gpio.event_when_pins_eq(1);
select {
  case gpio.event():
    // This event was caused by the pin value being 1
    // ...
    break;
}
```

3.4 Pin maps

The GPIO tasks all take a `pin_map` argument. If this is `null` then the elements of the inteface array will correspond with the a bit of the port based on the array element index. So the first element of the array will control bit 0, the second with control bit 1 and so on.

Alternatively an array can be provided mapping array elements to pins. For example, the following will map the array indices to pins 3, 2 and 7 of the port

```
char pin_map[3] = {3, 2, 7};

int main() {
  // ...
  par {
    output_gpio(i_gpio, 3, p, pin_map);
    // ...
```

4 GPIO APIs

4.1 Output GPIO API

Output GPIO components

void **output_gpio**(SERVER_ARRAY_OF_SIZE(output_gpio_if, i, *n*), static_const_size_t n, out_port_t p, NULLABLE_ARRAY_OF_SIZE(char, pin_map, *n*))

Task that splits a multi-bit port into several 1-bit GPIO interfaces.

This component allows other tasks to access the individual bits of a multi-bit output port.

Parameters

- **i** – The array of interfaces to connect to other tasks.
- **n** – The number of interfaces connected.
- **p** – The output port to be split.
- **pin_map** – This array maps the connected interfaces to the pin(s) of the port. For example, if 3 clients are connected to split a 8-bit port and the array {2,5,3} is supplied. Then bit 2 will go to interface 0, bit 5 to inteface 1 and bit 3 to inteface 2. If null is supplied for this argument then the pin map is assumed to be {0,1,2...}.

Output GPIO interface

group **output_gpio_if**

This interface provides access to a GPIO that can perform output operations only. All GPIOs are single bit.

Functions

void **output**(unsigned data)

Perform an output on a GPIO.

Parameters

- **data** – The value to be output. The least significant bit represents the 1-bit value to be output.

gpio_time_t **output_and_timestamp**(unsigned data)

Perform an output on a GPIO and get a timestamp of when the output occurs.

Parameters

- **data** – The value to be output. The least significant bit represents the 1-bit value to be output.

Returns

The time the value was input. This timestamp is the 16-bit port timer value. The port timer is driven at the rate of the port clock.

4.2 Input GPIO API

Input GPIO components

void **input_gpio**(SERVER_ARRAY_OF_SIZE(input_gpio_if, i, *n*), static_const_size_t n, in_port_t p, NULLABLE_ARRAY_OF_SIZE(char, pin_map, *n*))

Task that splits a multi-bit input port into several 1-bit GPIO interfaces (no events).

This component allows other tasks to access the individual bits of a multi-bit input port. It does not support events but is distributable so requires no specific logical core to run on. If the *event_when_pins_eq()* function is called then the component will trap.

Parameters

- **i** – The array of interfaces to connect to other tasks.
- **n** – The number of interfaces connected.
- **p** – The input port to be split.
- **pin_map** – This array maps the connected interfaces to the pin(s) of the port. For example, if 3 clients are connected to split a 8-bit port and the array {2,5,3} is supplied. Then bit 2 will go to interface 0, bit 5 to inteface 1 and bit 3 to inteface 2. If null is supplied for this argument then the pin map is assumed to be {0,1,2...}.

void **input_gpio_with_events**(SERVER_ARRAY_OF_SIZE(input_gpio_if, i, *n*), static_const_size_t n, in_port_t p, NULLABLE_ARRAY_OF_SIZE(char, pin_map, *n*))

Task that splits a multi-bit input port into several 1-bit GPIO interfaces (with events).

This component allows other tasks to access the individual bits of a multi-bit input port. It does support events so requires a logical core to run on (but can be combined with other tasks on the same core).

Parameters

- **i** – The array of interfaces to connect to other tasks.
- **n** – The number of interfaces connected.
- **p** – The input port to be split.
- **pin_map** – This array maps the connected interfaces to the pin(s) of the port. For example, if 3 clients are connected to split a 8-bit port and the array {2,5,3} is supplied. Then bit 2 will go to interface 0, bit 5 to inteface 1 and bit 3 to inteface 2. If null is supplied for this argument then the pin map is assumed to be {0,1,2...}.

void **input_gpio_1bit_with_events**(SERVER_INTERFACE(input_gpio_if, i), in_port_t p)

Convert a 1-bit port to a single 1-bit GPIO interface.

This component allows other tasks to access a 1-bit port as a GPIO interface. It is more efficient that using input_gpio_with_events() for the restricted case where a 1-bit port is used.

Parameters

- **i** – The interface to connect to other tasks.
- **p** – The input port.

Input GPIO interface

group **input_gpio_if**

This interface provides access to a GPIO that can perform input operations only. All GPIOs are single bit.

Functions

unsigned **input**(void)

Perform an input on a GPIO

Returns

The value input from the port in the least significant bit. The rest of the value will be zero extended.

unsigned **input_and_timestamp**(REFERENCE_PARAM(gpio_time_t, timestamp))

Perform an input on a GPIO and get a timestamp

Parameters

- **timestamp** – This pass-by-reference parameter will be set to the time the value was input. This timestamp is the 16-bit port timer value. The port timer is driven at the rate of the port clock.

Returns

The value input from the port in the least significant bit. The rest of the value will be zero extended.

void **event_when_pins_eq**(unsigned val)

Request an event when the pin is a certain value.

This function will cause a notification to occur when the pins match the specified value.

Parameters

- **val** – The least significant bit represents the 1-bit value to match.

void **event**(void)

A pin event has occurred.

This notification will occur when a pin event has occurred. Events can be requested using the *event_when_pins_eq()* call.