GPIO Library

The XMOS GPIO library allows you to access XMOS ports as low-speed GPIO

Although XMOS 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 the individual pins independently. It also allows accessing ports across separate XMOS tiles or separate XMOS chips.

Features

- Abstract interface to GPIO functionality of XMOS ports
- Allow separate access to multibit ports
- Allow access to ports across tiles

Operating modes

- Multibit output for individual access to the pins of a multibit output port
- Multibit input for individual access to the pins of a multibit input port
- Multibit input for individual access to the pins of a multibit input port allowing the application to react to events on those pins

Software version and dependencies

This document pertains to version 1.0.0 of this library. It is known to work on version 14.0.0 of the xTIMEcomposer tools suite, it may work on other versions.

This library depends on the following other libraries:

- lib_xassert (>=2.0.0)

Related application notes

The following application notes use this library:

- AN00166 - How to access individual pins of a multi-bit port

1 Connecting external signals to multibit ports

Multi-bit ports can be connected to independent signals in either an all output configuration (see Figure 2) or an all input configuration (see Figure 1). 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.

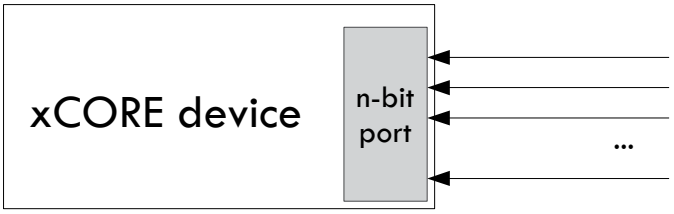

Figure 1: Input configuration

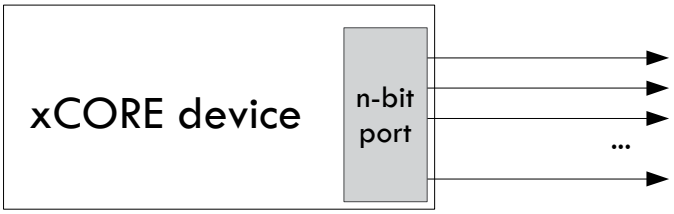

Figure 2: Output configuration

1.1 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 to using the ports.

- The internal buffering, serializing and de-serializing features of the port are not available.
- The software locking and multiplexing between individual bits of the port limits performance in line with the performance of the speed of the logical core that is driving the port. As such, toggling pins at speed above 1Mhz, for example, is not achievable (on a 62.5Mhz logical core). Lower speeds will depend on the other calculation on the core and the use of other pins of the port.

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

2 Usage

2.1 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:

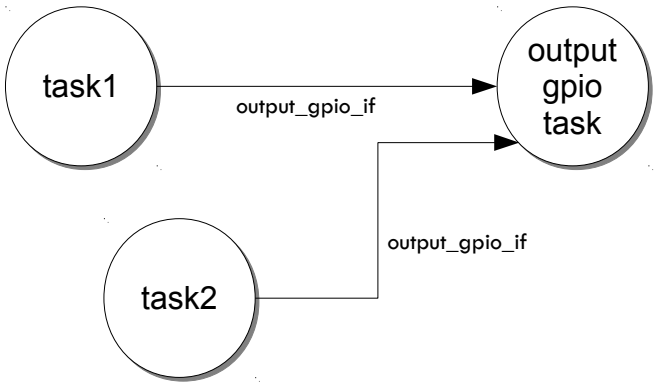

Figure 3: Output GPIO task diagram

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 a different pin 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);
  delay_milliseconds(200);
  gpio1.output(0);
  gpio2.output(1);
  ...
}
```

More information on interfaces and tasks can be be found in the XMOS Programming Guide (see XM-

004440-PC). By default the output GPIIO 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).

2.2 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:

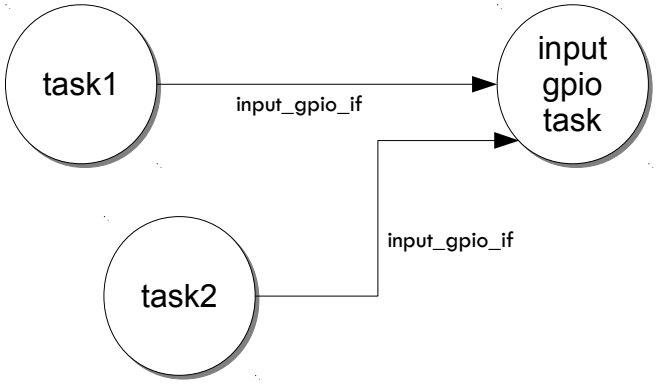

Figure 4: Input GPIO task diagram

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 a different pin 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 (see XM-004440-PC). By default the output GPIIO 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).

2.3 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;
}
```

2.4 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);
    ...
```

3 Output GPIO API

All gpio functions can be accessed via the gpio.h header:

```
#include <gpio.h>
```

You will also have to add lib_gpio to the USED_MODULES field of your application Makefile.

3.1 Output GPIO components

Function	output_gpio	
Description	Task that splits a multibit port into several 1 bit GPIO interfaces. This component allows other tasks to access the individual bits of a multi-bit output port.	
Type	[[distributable]] void output_gpio(server output_gpio_if i[n], static const size_t n, out port p, char(& ?pin_map)[n])	
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 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...}.

3.2 Output GPIO interface

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

Function	output	
Description	Perform an output on a GPIO.	
Type	`void output(unsigned data)`	
Parameters	`data`	The value to be output. The least significant bit represents the 1-bit value to be output.

Function	output_and_timestamp	
Description	Perform an output on a GPIO and get a timestamp of when the output occurs.	
Type	`gpio_time_t output_and_timestamp(unsigned data)`	
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 on a timebase relative to the GPIO.	

4 Input GPIO API

4.1 Input GPIO components

Function	input_gpio
Description	Task that splits a multibit 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.

Type	`[[distributable]]`
	`void`
	`input_gpio(server input_gpio_if i[n],`
	`  static const size_t n,`
	`  in port p,`
	`  char(& ?pin_map)[n])`

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 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...}.

Function	input_gpio_with_events
Description	

Type	`[[combinable]]`
	`void`
	`input_gpio_with_events(server input_gpio_if i[n],`
	`  static const size_t n,`
	`  in port p,`
	`  char(& ?pin_map)[n])`

Function	input_gpio_1bit_with_events
Description	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.

Type	`[[combinable]]`
	`void`
	`input_gpio_1bit_with_events(server input_gpio_if i,`
	`in port p)`

Parameters	`i` The interface to connect to other tasks. `p` The input port.

4.2 Input GPIO interface

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

Functions

Function	input
Description	Perform an input on a GPIO.
Type	`unsigned input(void)`
Returns	The value input from the port in the least significant bit. The rest of the value will be zero extended.

Function	input_and_timestamp
Description	Perform an input on a GPIO and get a timestamp.
Type	`unsigned input_and_timestamp(gpio_time_t &timestamp)`
Parameters	`timestamp` This pass-by-reference parameter will be set to the time the value was input. This timestamp is on a timebase relative to the GPIO.
Returns	The value input from the port in the least significant bit. The rest of the value will be zero extended.

Function	event_when_pins_eq
Description	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.
Type	`[[clears_notification]] void event_when_pins_eq(unsigned val)`
Parameters	`val` The value to match.

Continued on next page

Type	input_gpio_if (continued)

Function	event
Description	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.
Type	[[notification]] slave void event(void)

APPENDIX A - Known Issues

No known issues.

APPENDIX B - GPIO library change log

B.1 1.0.0

- Initial version