Startkit support library

This library provides support for accessing the available functionaility of the startKIT development board.

Features

- Ability to access on-board ADC.
- Ability to access LEDs and buttons.
- Ability to access the board's capacitive sensors.

Resource usage

This following table shows typical resource usage in some different configurations. Exact resource usage will depend on the particular use of the library by the application.

Configuration	Pins	Ports	Clocks	Ram	Logical cores
Simple LED control	9	1 (32-bit)	0	~2.2K	0
LED, buttons and cap-sense	9	1 (32-bit), 2 (4-bit)	1	~14.2K	≤ 1
ADC	5	1 (1-bit)	0	~3.7K	≤ 1
Cap-sense slider	9	1 (4-bit)	1	~12.1K	≤ 1

Software version and dependencies

This document pertains to version 3.0.1 of this library. It is known to work on version 14.1.1 of the xTIMEcomposer tools suite, it may work on other versions.

The library does not have any dependencies (i.e. it does not rely on any other libraries).

Related application notes

The following application notes use this library:

- AN00173 - A startKIT accelerometer demo
- AN00174 - A startKIT glowing LED demo
- AN00175 - A startKIT LED demo
- AN00176 - A startKIT noughts and crosses game (tic-tac-toe)
- AN00177 - A startKIT ADC example

1 Usage

To access any of the library functionality the application `Makefile` needs to add `lib_startkit_support` to its build modules:

```
USED_MODULES = ... lib_startkit_support ...
```

The GPIO functions can be found by using the following header:

```
#include <startkit_gpio.h>
```

The ADC functions can be found by usings the following header:

```
#include <startkit_adc.h>
```

The capacitive sensing functions can be found by usings the following header:

```
#include <startkit_slider.h>
```

1.1 Simple LED control

The `startkit_led_driver` task allows your program to drive LEDS on and off in the 3x3 LED array on the startKIT. This task only allows simple on-off control but has low resource usage.

The driver is instantiated as a parallel task that run in a `par` statement. The application communicates to this tasks using the `startkit_led_if` interface.

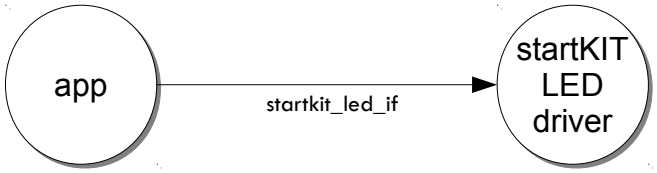

Figure 1: Simple LED control task diagram

For example, the following code instantiates the LED driver component and connects to it:

```
#include <platform.h>
#include <startkit_gpio.h>

port p_gpio = XS1_PORT_32A;

int main() {
  interface startkit_led_if i_led[1];
  par {
    on tile[0]: startkit_led_driver(i_led, 1, p_gpio);
    on tile[0]: app(i_led[0]);
  }
}
```

The task must be passed port 32A. The interface connection is an array (so several tasks can access the LEDs).

The application can then communicate with the task via the interface e.g.:

```
void app(client startkit_led_if led)
{
  ...
  // Set the top middle LED (x=1, y=0) on
  led.set(1, 0, LED_ON);
  // Set the bottom left LED (x=0, y=2) off
  led.set(0, 2, LED_OFF);
  ...
```

1.2 Controlling LEDs, buttons and capacitive sensing together

On the startKIT board, the LEDs, buttons and capacitive sensors are all either on the same xCORE ports or are routed close enough on the board to cause possible cross-talk. For this reason, they all need to be controlled synchronized together so they do not intefere.

The `startkit_gpio_driver` gives your application access to all these hardware interfaces. It is a task that supplies several software interfaces to the application:

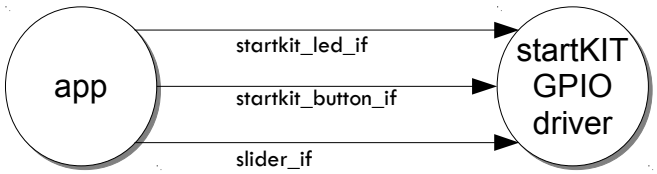

Figure 2: GPIO task diagram

The driver tasks can be instatiated in the top level of the program as in this example:

```
// The port structure required for the GPIO task
startkit_gpio_ports gpio_ports =
  {XS1_PORT_32A, XS1_PORT_4B, XS1_PORT_4A, XS1_CLKBLK_1};

int main() {
  startkit_button_if i_button;
  startkit_led_if i_led;
  slider_if i_slider_x, i_slider_y;
  par {
    on tile[0]: startkit_gpio_driver(i_led, i_button,
                                     i_slider_x, i_slider_y,
                                     gpio_ports);
    on tile[0]: app(i_led, i_button, i_slider_x, i_slider_y);
  }
  return 0;
}
```

The slider interface is described in §1.3.

1.2.1 The PWM LED interface

When using the GPIO driver. The LED interface can set a level for the LED which is driven via a PWM signal. For each LED, the interface accepts a range from 0 to LED_ON. So for example, the following code will set an LED to 50% illumination:

```
void app(client startkit_led_if led, ...) {

    ...
    led.set(1, 1, LED_ON/2);
```

1.2.2 The button interface

The button interface causes an changed event that can be selected on using xC `select` construct whenever a change occurs in the button. The `get_value` function can then be used to get the button state e.g.:

```
void app(.., client startkit_button_if button, ...) {
   ..
   select {
     case button.changed():
        if (button.get_value() == BUTTON_DOWN) {
           // handle button down event
        } else {
           // handle button up event
        }
        break;
     ...
   }
   ...
```

1.3 Using the capacative sensor

The capacitive sensor can be access via the `slider_if` interface. Two interfaces are provided - one in the horizontal (x) direction and one in the vertical (y) direction.

The interface will cause a `changed_state` event when it changes state that can be reacted to in an xC `select` statement e.g.:

```
void app(.., client slider_if i_slider_x, ...) {
   ..
   select {
      case i_slider_x.changed_state():
         sliderstate state = i_slider_x.get_slider_state();
         if (state == LEFTING) {
            // handle the event when the user swipes left
         } else if (state == RIGHTING) {
            ...
```

1.4 Accessing the ADC

The ADC functions can be found by using the following header:

```
#include <startkit_adc.h>
```

The ADC component is instantiated as a parallel task that run in a par statement. The application communicates to this tasks using the `startkit_adc_if` interface. The `adc_task` then needs to be connected to the special `startkit_adc` service which attaches to the analogue hardware.

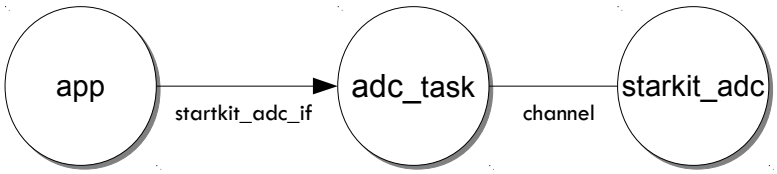

Figure 3: ADC task diagram

For example, the following code instantiates the ADC component and connects to it:

```
#include <platform.h>
#include <startkit_adc.h>

out port adc_sample = ADC_TRIG_PORT;

int main() {
  chan c_adc;
  interface startkit_adc_if i_adc;
  par {
    startkit_adc(c_adc);
    on tile[0]: adc_task(i_adc, c_adc, 0, adc_sample);
    on tile[0]: app(i_adc);
  }
}
```

The application can then communicate with the task via the interface e.g.:

```
void app(client startkit_adc_if adc)
{
  ...
  adc.trigger();  // Fire the ADC!
  select {        // Wait for the ADC to complete.
    case adc.complete():
       short vals[4];
       adc.read(vals); // Read analogue data into vals array
       ...
```

More information on interfaces and tasks can be be found in the XMOS Programming Guide (see XM-004440-PC).

2 Startkit ADC API

Type	startkit_adc_if
Description	ADC Interface.
Functions	

Function	trigger
Description	Initiates a trigger sequence. If trigger already in progress, this call is ignored
Type	`[[guarded]]` `void trigger(void)`

Function	read
Description	Reads the 4 ADC values and places them in array of unsigned shorts passed. Value is 0 to 65520 unsigned. Actual ADC is 12b so bottom 4 bits always zero. Ie. left justified. Optionally returns the ADC state - 1 if ADC trigger/aquisition complete, or 0 if in progress.
Type	`[[clears_notification]]` `int read(unsigned short adc_val[4])`

Function	complete
Description	Call to client to indicate aquisition complete. Behaves a bit like ADC finish interrupt. Optional.
Type	`[[notification]]` `slave void complete(void)`

Function	adc_task
Description	Runs ADC task. Very low MIPS consumption so is good candidate for combining with other low speed tasks Pass i_adc control inteface and automatic trigger period in microseconds. If trigger period is set to zero, ADC will only convert on `trigger()` call.

Type	`[[combinable]]`
	`void`
	`adc_task(server startkit_adc_if i_adc,`
	`chanend c_adc,`
	`int trigger_period,`
	`out port adc_sample)`

3 Startkit GPIO API

Type	led_val
Description	Enum for controlling led levels. Leds can be set in the range LED_OFF .. LED_ON. The exact resolution depends on the driver.
Values	LED_OFF LED_ON

Type	startkit_led_if
Description	Interface for controlling leds on the startkit.
Functions	

Function	set
Description	Set an led output level. Use this function to set a single led in the range LED_OFF .. LED_ON. 'x' takes a value 0,1,2 which refer to columns A,B,C. 'y' takes a value 0,1,2 which refer to rows 1,2,3.
Type	`void set(unsigned x, unsigned y, unsigned val)`

Function	set_multiple
Description	Set multiple led values. Use this function to set the level of several leds at once. The first argument is a bitmask where the least signifcant nine bits map to the led array in the following way: bit = (x + y*3) Where 'x' takes a value 0,1,2 which refer to columns A,B,C and 'y' takes a value 0,1,2 which refer to rows 1,2,3. If the bit is set in the mask then the led is set to the second argument. If the bit is not set then the led is set to LED_OFF.
Type	`void set_multiple(unsigned mask, unsigned val)`

Type	button_val
Description	Enum for representing button state.
Values	BUTTON_UP BUTTON_DOWN

Type	startkit_button_if
Description	Interface for interacting with buttons.
Functions	

Function	changed
Description	This notification occurs when the button changes state (i.e. goes up->down or down->up). You can select on this in your program e.g. void f(client startkit_button_if i_button) { ... select { case i_button.:c:func:*change*: button_val_t val = get_value(); ....
Type	[[notification]] slave void changed()

Function	get_value
Description	Get the current value of the button. This returns either BUTTON_UP or BUTTON_DOWN.
Type	[[clears_notification]] button_val_t get_value()

Function	startkit_led_driver
Description	Simple LED driver. This task will drive leds according to commands received via the interface startkit_led_if. It has no resolution or pwm, so leds will either be turned on or off depending on whether the level is greater or less than LED_ON/2. The drive is distributable, so will not take up any compute on a logical core of its own unless it is connected to clients on a different tile.

Type	`[[distributable]]`
	`void`
	`startkit_led_driver(server startkit_led_if i_led[n],`
	`unsigned n,`
	`port p32)`

Type	startkit_gpio_ports
Description	Ports/clocks for startkit GPIO, the ports should be XS1_PORT_32A, XS1_PORT_4B, XS1_PORT_4A.
Fields	port p32 port capx port capy clock clk

Function	startkit_gpio_driver
Description	startKIT gpio driver. This task drives pwm output on the leds to varying brightness and also reads the button on the board. It requires the startKIT's 32 bit port (XS1_PORT_32A) to be passed as the last argument. Clients can connect via the first two arguments. Several clients can connect to set led levels. The function is combinable so can share a logical core with other combinable tasks. If combined is will use cooperative multitasking to periodically drive the pwm and sample the button value.
Type	[[combinable]] void startkit_gpio_driver(server startkit_led_if ?i_led, server startkit_button_if ?i_button, server slider_if ?i_slider_x, server slider_if ?i_slider_y, startkit_gpio_ports &ps)

4 Startkit Slider API

Type	sliderstate
Description	Type that enumerates the possible activities that may have happened on a slider.
Values	IDLE PRESSING PRESSED LEFTING RIGHTING RELEASED

Type	slider_if
Description	Interface for querying the slider value and state.
Functions	

Function	changed_state
Description	
Type	[[notification]] slave void changed_state()

Function	get_slider_state
Description	
Type	[[clears_notification]] sliderstate get_slider_state()

Function	get_coord
Description	
Type	int get_coord()

Function	slider_task
Description	Function to implement a slider task.

Type	`[[combinable]]`
	`void slider_task(server slider_if i,`
	`port cap,`
	`const clock clk,`
	`static const int n_elements,`
	`static const int N,`
	`int threshold_pressed,`
	`int threshold_unpressed)`

Parameters	`i` interface used to communicate with this task.
	
	`cap` port on which the cap sense is connected
	
	`clk` clock block to be used.
	
	`n_elements`
	number of segments on this slider. Typically 4 or 8. Note that at present this is hardcoded in the capsens.h file too and set to 4.
	
	`N`
	
	`threshold_pressed`
	Value above which something is pressed. Set to 1000
	
	`threshold_unpressed`
	Value below which something is no longer pressed. Set to 200

APPENDIX A - Known Issues

No known issues.

APPENDIX B - Startkit support library change log

B.1 3.0.1

- Update to source code license and copyright

B.2 3.0.0

- Move the declaration of 'adc_sample' port out of the library.
- Correct XS1_PORT_4B, XS1_PORT_4A mapping onto SLIDER X and SLIDER Y
- Alter led interface to use x,y coordinates
- Alter slider state behaviour