# Quantum Computer Outreach Project

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# Todo List

**Global __attribute__ ((__interrupt__, no_auto_psv))**

turn on all the LEDs back on Reset all the counters

**Global BTN_CHIP_NUM**

read buttons

**Global led_cycle_test ()**

This won't work now: write_display_driver(counter);

**Global read_external_buttons ()**

How long should this be?

button remappings...

**Global setup_timer ()**

distinguish between the two different timers here...

**Global TLC591x_mode_switch (int mode)**

mode switcher for LED Driver

**Global update_display_buffer (int led_index, int R, int G, int B)**

hmmm...

**Global update_display_buffer (int led_index, int R, int G, int B)**

hmmm...

**Global write_display_driver (int ∗data)**

Does the high byte or low byte go first?

How long should this be?

# Chapter 2

# Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

# Data Structure Documentation

## 4.1 LED Struct Reference

Each LED has the following type.

`#include <io.h>`

### Data Fields

- int **R**
- int G

  *The line number for red.*
- int B

  *the line number for green*
- unsigned _Fract N_R

  *The line number for blue.*
- unsigned _Fract N_G

  *The R brightness.*
- unsigned _Fract N_B

  *The G brightness.*
- unsigned _Fract n_R

  *The B brightness.*
- unsigned _Fract n_G

  *Counter for R – do not modify.*
- unsigned _Fract n_B

  *Counter for G – do not modify.*

### 4.1.1 Detailed Description

Each LED has the following type.

The type holds the information about the position of the RGB lines in the display driver array and also the brightness of the RGB lines. The counters are used by a timer interrupt service routine pulse the RGB LEDs at a specified rate.

The type of the counter is *Fract to facilitate easy comparison with the N$*$* variables which used the fractional type.

The documentation for this struct was generated from the following file:

- dspic33e/qcomp-sim-c.X/io.h

## 4.2 LED_GLOBAL Struct Reference

pin mappings Pins for LE and OE on port D OE = RD4 = uC:81 = J1:28 = J10:14 LE = RD3 = uC:78 = J1:40 = J11:18

```
#include <io.h>
```

### Data Fields

- int strobe_leds

  *Bit set the LEDs which are strobing.*
- int strobe_state

  *Bit zero is the current state (on/off)*

### 4.2.1 Detailed Description

pin mappings Pins for LE and OE on port D OE = RD4 = uC:81 = J1:28 = J10:14 LE = RD3 = uC:78 = J1:40 = J11:18

Pins for SH and CLK_INH on port D SH = RD5 = uC:82 = J1:25 = J10:13 CLK_INH = RD8 = uC:68 = J1:58 = J11:25Global LED strobing state parameter

The documentation for this struct was generated from the following file:

- dspic33e/qcomp-sim-c.X/io.h

# Chapter 5

# File Documentation

## 5.1 dspic33e/qcomp-sim-c.X/config.h File Reference

General config settings #pragma for microcontroller.

This graph shows which files directly or indirectly include this file:

## 5.2 dspic33e/qcomp-sim-c.X/io.c File Reference

Contains all the functions for reading buttons and writing to LEDs.

```
#include "io.h"
#include "time.h"
#include "spi.h"
```

Include dependency graph for io.c:

### Macros

- #define **DISPLAY_CHIP_NUM** 2
- #define **PERIOD** 500000
- #define BTN_CHIP_NUM 2

  *Read external buttons.*

### Functions

- int led_color_int (int device, int R, int G, int B)

  *Takes led number & RGB -> returns integer for sending via SPI to set the LED.*
- int setup_io (void)

  *Set up LEDs and buttons on port D.*
- void __attribute__ ((__interrupt__, no_auto_psv))

  *The max value for isr_counter.*
- void setup_external_leds ()

  *Set external variable RGB LEDs.*

- void stop_external_leds ()

  *Stop LEDs flashing.*
- void set_strobe (int color, int state)

  *Set an LED strobing.*
- void toggle_strobe (int color)

  *Toggle LED strobe.*
- int set_led (int color, int state)

  *Turn a particular LED on or off.*
- int read_btn (int btn)

  *Read the state of a push button.*
- void leds_off (void)

  *Turn all the LEDs off.*
- void flash_led (int color, int number)

  *Flash LED a number of times.*
- void flash_all (int number)

  *Flash all the LEDs a number of times.*
- int update_display_buffer (int index, int R, int G, int B)
- int write_display_driver (int ∗data)

  *Turn on an LED via the external display driver.*
- int TLC591x_mode_switch (int mode)

  *Switch between normal and special mode.*
- int set_external_led (int index, unsigned _Fract R, unsigned _Fract G, unsigned _Fract B)
- int read_external_buttons ()

  *Update the buttons array (see declaration above)*
- int led_cycle_test ()

  *Loop to cycle through LEDs 0 - 15.*

## Variables

- int buttons [16]

  *Contains the button states.*
- LED_GLOBAL led_global = {0}
- LED led [LED_NUM]

  *The LED array. Not to be used globally.*
- int display_buf [DISPLAY_CHIP_NUM] = {0}

  *Display buffer to be written to display driver.*
- unsigned _Fract isr_counter = 0

  *Counter for the interrupt service routine _T5Interrupt.*
- unsigned _Fract isr_res = 0.01

  *Counter value.*
- unsigned _Fract isr_limit = 0.99

  *Counter resolution.*

### 5.2.1 Detailed Description

Contains all the functions for reading buttons and writing to LEDs.

**Author**

J Scott

**Date**

8/11/18

### 5.2.2 Macro Definition Documentation

#### 5.2.2.1 BTN_CHIP_NUM

```
#define BTN_CHIP_NUM 2
```

Read external buttons.

The external buttons are interfaced to the microcontroller via a shift register. Data is shifted in a byte at a time using the SPI 3 module. The sequence to read the buttons is as follows:

1) Momentarily bring SH low to latch button data into the shift registers 2) Bring CLK_INH low to enable the clock input on the shift register 3) Start the SPI 3 clock and read data in via the SDI 3 line

The control lines SH and CLK_INH are on port D

**Todo** read buttons

### 5.2.3 Function Documentation

#### 5.2.3.1 __attribute__()

```
void __attribute__ (
             (__interrupt__, no_auto_psv)  )
```

The max value for isr_counter.

Interrupt service routine for timer 4

Interrupt service routines are automatically called by the microcontroller when an event occurs. In this case, _↩ T5Interrupt is called when the 32 bit timer formed from T4 and T5 reaches its preset period. The silly name and sill attributes are so that the compiler can correctly map the function in the microcontroller memory. More details of interrupts and interrupt vectors can be found in the compiler manual and the dsPIC33E datasheet.

The job of this routine is to control the modulated brightnesses of the RBG LEDs. This routine is set to be called periodically with a very long period on the time scale of microcontroller operations, but very fast in comparison to what the eye can see. For example, once every 100us.

Each time the routine is called, it increments counters corresponding to RGB line of every LED. Once these counters reach thresholds, that have been set globally in another function, the interrupt routine turns off the corresponding LED line. Once the Increment the LED RGB counter

Increment the LED RGB counter

Increment the LED RGB counter

Reset the counter

**Todo** turn on all the LEDs back on Reset all the counters

#### 5.2.3.2 flash_all()

```
void flash_all (
             int number )
```

Flash all the LEDs a number of times.

**Parameters**

| | |
|---|---|
| *number* | |

#### 5.2.3.3 flash_led()

```
void flash_led (
             int color,
             int number )
```

Flash LED a number of times.

Flash one LED a number of times.

#### 5.2.3.4 led_color_int()

```
int led_color_int (
             int device,
             int R,
             int G,
             int B )
```

Takes led number & RGB -> returns integer for sending via SPI to set the LED.

**Parameters**

| | |
|---|---|
| *device* | input LED number to change |
| *R* | red value between 0 & 1 |
| *G* | green value between 0 & 1 |
| *B* | blue value between 0 & 1 |

**Returns**

> Returns int to be sent to LED Driver

convention RGB -> 000

Each LED takes 3 lines, assumes there are no gaps between LED channels "device" goes between 0 to 2^n -1

#### 5.2.3.5 led_cycle_test()

```
int led_cycle_test ( )
```

Loop to cycle through LEDs 0 - 15.

**Todo** This won't work now: write_display_driver(counter);

#### 5.2.3.6 read_btn()

```
int read_btn (
             int btn )
```

Read the state of a push button.

**Parameters**

| | |
|---|---|
| *btn* | |

**Note**

How well do you know C

#### 5.2.3.7 read_external_buttons()

```
int read_external_buttons ( )
```

Update the buttons array (see declaration above)

SH pin

**Todo** How long should this be?

**Todo** button remappings...

#### 5.2.3.8 set_external_led()

```
int set_external_led (
             int index,
             unsigned _Fract R,
             unsigned _Fract G,
             unsigned _Fract B )
```

**Parameters**

| | |
|---|---|
| *led_index* | |
| *R* | red value between 0 & 1 |
| *G* | green value between 0 & 1 |
| *B* | blue value between 0 & 1 |

**Returns**

0 if successful, -1 otherwise

Use the function to set the RGB level of an LED. The LED is chosen using the

**Parameters**

| | |
|---|---|
| *led_index.* | The |
| *R* | |

#### 5.2.3.9 set_led()

```
int set_led (
             int color,
             int state )
```

Turn a particular LED on or off.

**Parameters**

| | |
|---|---|
| *color* | |
| *state* | |

#### 5.2.3.10 set_strobe()

```
void set_strobe (
             int color,
             int state )
```

Set an LED strobing.

**Parameters**

| | |
|---|---|
| *color* | |
| *state* | |

#### 5.2.3.11 setup_external_leds()

```
void setup_external_leds ( )
```

Set external variable RGB LEDs.

Setup up external LED lines

Turn all LEDs off

#### 5.2.3.12 setup_io()

```
int setup_io (
             void  )
```

Set up LEDs and buttons on port D.

< Set port c digital for spi3

Set the OE pin high

Set OE(ED2) pin

Set the SH pin high

Set SH pin

set CLK_INH high while buttons are pressed

#### 5.2.3.13 TLC591x_mode_switch()

```
int TLC591x_mode_switch (
             int mode )
```

Switch between normal and special mode.

The mode switch for the TLC591x chip is a bit tricky because it involves synchronising the control lines LE(ED1) and OE(ED2) on Port D with the SPI 1 clock. To initiate a mode switch, OE(ED2) must be brought low for one clock cycle, and then the value of LE(ED1) two clock cycles later determines the new mode. See the diagrams on page 19 of the datasheet

So long as the timing is not strict, we can probably implement the mode switch by starting a non-blocking transfer of 1 byte to the device (which starts the SPI 1 clock), followed by clearing OE(ED2) momentarily and then setting the value of LE(ED1) as required. So long as those two things happen before the SPI 1 clock finishes the procedure will probably work. (The reason is the lack of max timing parameters on page 9 for the setup and hold time for ED1 and ED2, which can therefore presumably be longer than one clock cycle.)

**Parameters**

| | |
|---|---|
| *mode* | |

**Todo** mode switcher for LED Driver

#### 5.2.3.14 toggle_strobe()

```
void toggle_strobe (
             int color )
```

Toggle LED strobe.

**Parameters**

|  |  |
|---|---|
| *color* |  |

#### 5.2.3.15 update_display_buffer()

```
int update_display_buffer (
             int index,
             int R,
             int G,
             int B )
```

**Parameters**

|  |  |
|---|---|
| *index* | LED number to modify |
| *R* | Intended value of the R led |
| *G* | Intended value of the G led |
| *B* | Intended value of the B led |

**Returns**

0 if successful

Could this get any worse!

This function is supposed to make the display writing process more efficient. It updates a global display buffer which is written periodically to the led display drivers. Instead of the display driver function re-reading the desired state of all the LED lines every time it is called, this function can be used to update only the lines that have changed.

There are quite a few potential bugs in here, mainly array out of bounds if the DISPLAY_CHIP_NUM is not set correctly or the LED RGB lines are wrong. (Or if there are just bugs.)

**Todo** hmmm...

#### 5.2.3.16 write_display_driver()

```
int write_display_driver (
             int * data )
```

Turn on an LED via the external display driver.

Send a byte to the display driver.

On power on, the chip (TLC591x) is in normal mode which means that the clocked bytes sent to the chip set which LEDs are on and which are off (as opposed to setting the current of the LEDs)

To write to the device, use the SPI module to write a byte to the SDI 1 pin on the chip. Then momentarily set the LE(ED1) pin to latch the data onto the output register. Finally, bring the OE(ED2) pin low to enable the current sinking to turn on the LEDs. See the timing diagram on page 17 of the datasheet for details.

LE(ED1) and OE(ED2) will be on Port D

**Parameters**

| | |
|---|---|
| *data[ ]* | an array of bytes to send to LED driver |

**Todo** Does the high byte or low byte go first?

Set LE(ED1) pin

**Todo** How long should this be?

## 5.2.4 Variable Documentation

### 5.2.4.1 buttons

```
int buttons[16]
```

Contains the button states.

Each entry in the array is either 1 if the button is pressed or 0 if not. The array is accessed globally using 'extern buttons;' in a ∗.c file. Read buttons array us updated by calling read_external_buttons

### 5.2.4.2 isr_counter

```
unsigned _Fract isr_counter = 0
```

Counter for the interrupt service routine _T5Interrupt.

These variables are for keeping track of the interrupt based LED pulsing. The type is _Fract because it is easier to directly compare two _Fracts than attempt multiplication of integers and _Fracts (which isn't supported) The limit is not 1 because _Fract types do not go up to 1.

### 5.2.4.3 led_global

```
LED_GLOBAL led_global = {0}
```

**Parameters**

| | |
|---|---|
| *led_global* | Global LED strobing state parameter |

## 5.3 dspic33e/qcomp-sim-c.X/io.h File Reference

Description: Header file for input output functions.

```
#include "p33EP512MU810.h"
#include "xc.h"
```

Include dependency graph for io.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct LED_GLOBAL

  *pin mappings Pins for LE and OE on port D OE = RD4 = uC:81 = J1:28 = J10:14 LE = RD3 = uC:78 = J1:40 = J11:18*
- struct LED

  *Each LED has the following type.*

### Macros

- #define red 0

  *Locations of LEDs and buttons on Port D.*
- #define **amber** 1
- #define **green** 2
- #define **sw1** 6
- #define **sw2** 7
- #define **sw3** 13
- #define **off** 0
- #define **on** 1
- #define LE 3

  *Control for TLC591x chip on Port D.*
- #define **OE** 4
- #define SH 5

  *COntrol lines for SNx4HC165 chip.*
- #define **CLK_INH** 8
- #define LED_NUM 4

  *The number of external LEDs.*

### Functions

- int setup_io (void)

  *Set up LEDs and buttons on port D.*
- void setup_external_leds ()

  *Set external variable RGB LEDs.*
- int set_led (int color, int state)

  *Turn a particular LED on or off.*
- int read_btn (int btn)

  *Read the state of a push button.*
- void leds_off (void)

  *Turn all the LEDs off.*
- void flash_led (int color, int number)

  *Flash one LED a number of times.*

- void flash_all (int number)

  *Flash all the LEDs a number of times.*
- void set_strobe (int color, int state)

  *Set an LED strobing.*
- void toggle_strobe (int color)

  *Toggle LED strobe.*
- int update_display_buffer (int led_index, int R, int G, int B)
- int write_display_driver (int ∗data)

  *Send a byte to the display driver.*
- int set_external_led (int led_index, unsigned _Fract R, unsigned _Fract G, unsigned _Fract B)
- int led_color_int (int device, int R, int G, int B)

  *Takes led number & RGB -> returns integer for sending via SPI to set the LED.*
- int led_cycle_test ()

  *Loop to cycle through LEDs 0 - 15.*
- int read_external_buttons ()

  *Update the buttons array (see declaration above)*

## 5.3.1 Detailed Description

Description: Header file for input output functions.

Include it at the top of any C source file which uses buttons and LEDs. It also defines various constants representing the positions of the buttons and LEDs on port D.

## 5.3.2 Function Documentation

### 5.3.2.1 flash_all()

```
void flash_all (
             int number )
```

Flash all the LEDs a number of times.

**Parameters**

| | |
|---|---|
| *number* | |

### 5.3.2.2 flash_led()

```
void flash_led (
             int color,
             int number )
```

Flash one LED a number of times.

**Parameters**

|  |  |
|---|---|
| *color* |  |
| *number* |  |

Flash one LED a number of times.

#### 5.3.2.3 led_color_int()

```
int led_color_int (
             int device,
             int R,
             int G,
             int B )
```

Takes led number & RGB -> returns integer for sending via SPI to set the LED.

**Parameters**

|  |  |
|---|---|
| *device* | input LED number to change |
| *R* | red value between 0 & 1 |
| *G* | green value between 0 & 1 |
| *B* | blue value between 0 & 1 |

**Returns**

Returns int to be sent to LED Driver

convention RGB -> 000

Each LED takes 3 lines, assumes there are no gaps between LED channels "device" goes between 0 to 2^n -1

#### 5.3.2.4 led_cycle_test()

```
int led_cycle_test ( )
```

Loop to cycle through LEDs 0 - 15.

**Todo** This won't work now: write_display_driver(counter);

#### 5.3.2.5 read_btn()

```
int read_btn (
             int btn )
```

Read the state of a push button.

**Parameters**

| | |
|---|---|
| *btn* | |

**Note**

How well do you know C

#### 5.3.2.6 read_external_buttons()

```
int read_external_buttons ( )
```

Update the buttons array (see declaration above)

SH pin

**Todo** How long should this be?

**Todo** button remappings...

#### 5.3.2.7 set_external_led()

```
int set_external_led (
             int index,
             unsigned _Fract R,
             unsigned _Fract G,
             unsigned _Fract B )
```

**Parameters**

| | |
|---|---|
| *led_index* | |
| *R* | red value between 0 & 1 |
| *G* | green value between 0 & 1 |
| *B* | blue value between 0 & 1 |

**Returns**

0 if successful, -1 otherwise

Use the function to set the RGB level of an LED. The LED is chosen using the

**Parameters**

| | |
|---|---|
| *led_index.* | The |
| *R* | |

#### 5.3.2.8 set_led()

```
int set_led (
             int color,
             int state )
```

Turn a particular LED on or off.

**Parameters**

| | |
|---|---|
| *color* | |
| *state* | |

#### 5.3.2.9 set_strobe()

```
void set_strobe (
             int color,
             int state )
```

Set an LED strobing.

**Parameters**

| | |
|---|---|
| *color* | |
| *state* | |

#### 5.3.2.10 setup_external_leds()

```
void setup_external_leds ( )
```

Set external variable RGB LEDs.

Setup up external LED lines

Turn all LEDs off

#### 5.3.2.11 setup_io()

```
int setup_io (
             void )
```

Set up LEDs and buttons on port D.

< Set port c digital for spi3

Set the OE pin high

Set OE(ED2) pin

Set the SH pin high

Set SH pin

set CLK_INH high while buttons are pressed

#### 5.3.2.12 toggle_strobe()

```
void toggle_strobe (
             int color )
```

Toggle LED strobe.

**Parameters**

| | |
|---|---|
| *color* | |

#### 5.3.2.13 update_display_buffer()

```
int update_display_buffer (
             int index,
             int R,
             int G,
             int B )
```

**Parameters**

| | |
|---|---|
| *led_index* | LED number to modify |
| *R* | Intended value of the R led |
| *G* | Intended value of the G led |
| *B* | Intended value of the B led |

**Returns**

0 if successful

**Parameters**

| | |
|---|---|
| *index* | LED number to modify |
| *R* | Intended value of the R led |
| *G* | Intended value of the G led |
| *B* | Intended value of the B led |

**Returns**

0 if successful

Could this get any worse!

This function is supposed to make the display writing process more efficient. It updates a global display buffer which is written periodically to the led display drivers. Instead of the display driver function re-reading the desired state of all the LED lines every time it is called, this function can be used to update only the lines that have changed.

There are quite a few potential bugs in here, mainly array out of bounds if the DISPLAY_CHIP_NUM is not set correctly or the LED RGB lines are wrong. (Or if there are just bugs.)

**Todo** hmmm...

#### 5.3.2.14 write_display_driver()

```
int write_display_driver (
            int * data )
```

Send a byte to the display driver.

**Parameters**

| | |
|---|---|
| *data* | Don't use this function to write to LEDs – use the set_external_led function |

Send a byte to the display driver.

On power on, the chip (TLC591x) is in normal mode which means that the clocked bytes sent to the chip set which LEDs are on and which are off (as opposed to setting the current of the LEDs)

To write to the device, use the SPI module to write a byte to the SDI 1 pin on the chip. Then momentarily set the LE(ED1) pin to latch the data onto the output register. Finally, bring the OE(ED2) pin low to enable the current sinking to turn on the LEDs. See the timing diagram on page 17 of the datasheet for details.

LE(ED1) and OE(ED2) will be on Port D

**Parameters**

| | |
|---|---|
| *data[ ]* | an array of bytes to send to LED driver |

**Todo** Does the high byte or low byte go first?

Set LE(ED1) pin

**Todo** How long should this be?

## 5.4 dspic33e/qcomp-sim-c.X/main.c File Reference

The main function.

```
#include "p33EP512MU810.h"
#include "xc.h"
#include "config.h"
#include "time.h"
#include "io.h"
#include "quantum.h"
#include "tests.h"
#include "spi.h"
```

Include dependency graph for main.c:

### Functions

- int main (void)

### 5.4.1 Detailed Description

The main function.

**Author**

J R Scott

**Date**

8/11/18

Description: Contains an example of fixed precision 2x2 matrix multiplication for applying operations to a single qubit. The only operations included are H, X and Z so that everything is real (this can be extended later).

All the functions have now been moved into separate files. io.h and io.c contain functions for reading and controlling the buttons and LEDs, and quantum.h/quantum.c contain the matrix arithmetic for simulating one qubit.

Compile command: make (on linux). But if you want to program the micro- controller too or if you're using windows you're better of downloading and installing MPLAB-X https://www.microchip.com/mplab/mplab-x-ide.

**Note**

You also need the microchip xc16 compilers which are available from https://www.microchip.com/mplab/compilers

### 5.4.2 Function Documentation

#### 5.4.2.1 main()

```
int main (
            void )
```

Reading button state

The button states are written into an array of type BUTTON_ARRAY whose

Global variable for button state

Update the buttons variable

Do something if button 0 has been pressed...

Example use of RGB LEDs – won't do anything yet

Just pass the values of R, G and B to the function along with the led index (which can just be an integer, like qubit number). The exact mapping of indices to LED lines in the display driver will be in the io.h file.

## 5.5 dspic33e/qcomp-sim-c.X/quantum.c File Reference

Description: Contains matrix and vector arithmetic for simulating one qubit.

```
#include "io.h"
#include "quantum.h"
```

Include dependency graph for quantum.c:

### Functions

- void **cadd** (Complex a, Complex b, Complex result)
- void **cmul** (Complex a, Complex b, Complex result)
- void make_ops_cmplx (CMatrix2 X, CMatrix2 Y, CMatrix2 Z, CMatrix2 H)

  *Create complex X, Y, Z and H.*
- void init_state_cmplx (CVector V, State s)

  *Initialise a complex state vector.*
- void mat_mul_cmplx (CMatrix2 M, CVector V)

  *2x2 complex matrix multiplication*
- void fix_phase (Vector V)

  *Add a global phase to make first amplitude positive.*
- void fix_phase_cmplx (CVector V)

  *Add a global phase to make first complex amplitude positive.*
- void clean_state (Vector V)

  *Clean the state: return the closest state out of 0>, |1>, |+> and |->*
- void clean_state_cmplx (CVector V)

  *Clean the state: return the closest state out of |0>, |1>, |+> , |->, |D> and |A>*
- void show_state_cmplx (CVector V)

  *Show the qubit state on the LEDs.*

## 5.5.1 Detailed Description

Description: Contains matrix and vector arithmetic for simulating one qubit.

## 5.5.2 Function Documentation

### 5.5.2.1 clean_state()

```
void clean_state (
             Vector V )
```

Clean the state: return the closest state out of 0>, |1>, |+> and |->

**Parameters**

| | |
|---|---|
| *V* | real vector |

### 5.5.2.2 clean_state_cmplx()

```
void clean_state_cmplx (
             CVector V )
```

Clean the state: return the closest state out of |0>, |1>, |+> , |->, |D> and |A>

**Parameters**

| | |
|---|---|
| *V* | complex vector |

### 5.5.2.3 fix_phase()

```
void fix_phase (
             Vector V )
```

Add a global phase to make first amplitude positive.

**Parameters**

| | |
|---|---|
| *V* | vector |

#### 5.5.2.4 fix_phase_cmplx()

```
void fix_phase_cmplx (
             CVector V )
```

Add a global phase to make first complex amplitude positive.

**Note**

This only works for certain states (zero, one, plus, minus, etc.)

**Parameters**

| | |
|---|---|
| *V* | complex vector |

#### 5.5.2.5 init_state_cmplx()

```
void init_state_cmplx (
             CVector V,
             State s )
```

Initialise a complex state vector.

**Parameters**

| | |
|---|---|
| *V* | complex vector |
| *s* | complex state |

#### 5.5.2.6 make_ops_cmplx()

```
void make_ops_cmplx (
             CMatrix2 X,
             CMatrix2 Y,
             CMatrix2 Z,
             CMatrix2 H )
```

Create complex X, Y, Z and H.

**Parameters**

| | |
|---|---|
| *X* | Pauli X c-Matrix |
| *Z* | Pauli Z c-matrix |
| *H* | Hadamard c-matrix |
| *Y* | Pauli Y c-matrix |

#### 5.5.2.7 mat_mul_cmplx()

```
void mat_mul_cmplx (
             CMatrix2 M,
             CVector V )
```

2x2 complex matrix multiplication

**Parameters**

| | |
|---|---|
| *M* | complex matrix |
| *V* | complex vector |

#### 5.5.2.8 show_state_cmplx()

```
void show_state_cmplx (
             CVector V )
```

Show the qubit state on the LEDs.

**Parameters**

| | |
|---|---|
| *V* | complex vector |

## 5.6 dspic33e/qcomp-sim-c.X/quantum.h File Reference

Description: Header file containing all the matrix arithmetic for simulating a single qubit.

```
#include "p33EP512MU810.h"
#include "xc.h"
```

Include dependency graph for quantum.h: This graph shows which files directly or indirectly include this file:

### Typedefs

- typedef signed _Fract Q15

  *Basic fractional time.*
- typedef Q15 Complex[2]

  *Complex type.*
- typedef Q15 Matrix4[4][4]

  *Matrix4 type.*
- typedef Q15 **CMatrix4**[4][4][2]
- typedef Q15 Matrix2[2][2]

*Matrix2 type.*

- typedef Q15 **CMatrix2**[2][2][2]
- typedef Q15 Vector[2]

  *Vector type.*
- typedef Q15 **CVector**[2][2]

## Enumerations

- enum State {
  **ZERO**, **ONE**, **PLUS**, **MINUS**,
  **iPLUS**, **iMINUS** }

  *Basis states.*

## Functions

- void make_ops (Matrix2 X, Matrix2 Z, Matrix2 H)

  *Create real? X, Z, H.*
- void make_ops_cmplx (CMatrix2 X, CMatrix2 Y, CMatrix2 Z, CMatrix2 H)

  *Create complex X, Y, Z and H.*
- void init_state (Vector V, State s)

  *Initialise a real state vector.*
- void init_state_cmplx (CVector V, State s)

  *Initialise a complex state vector.*
- void mat_mul (Matrix2 M, Vector V)

  *2x2 matrix multiplication*
- void mat_mul_cmplx (CMatrix2 M, CVector V)

  *2x2 complex matrix multiplication*
- void fix_phase (Vector V)

  *Add a global phase to make first amplitude positive.*
- void fix_phase_cmplx (CVector V)

  *Add a global phase to make first complex amplitude positive.*
- void clean_state (Vector V)

  *Clean the state: return the closest state out of 0>, |1>, |+> and |->*
- void clean_state_cmplx (CVector V)

  *Clean the state: return the closest state out of |0>, |1>, |+> , |->, |D> and |A>*
- void show_state (Vector V)

  *Show the qubit state on the LEDs.*
- void show_state_cmplx (CVector V)

  *Show the qubit state on the LEDs.*

### 5.6.1 Detailed Description

Description: Header file containing all the matrix arithmetic for simulating a single qubit.

### 5.6.2 Function Documentation

#### 5.6.2.1 clean_state()

```
void clean_state (
            Vector V )
```

Clean the state: return the closest state out of 0>, |1>, |+> and |->

**Parameters**

| | |
|---|---|
| *V* | real vector |

#### 5.6.2.2 clean_state_cmplx()

```
void clean_state_cmplx (
            CVector V )
```

Clean the state: return the closest state out of |0>, |1>, |+> , |->, |D> and |A>

**Parameters**

| | |
|---|---|
| *V* | complex vector |

#### 5.6.2.3 fix_phase()

```
void fix_phase (
            Vector V )
```

Add a global phase to make first amplitude positive.

**Parameters**

| | |
|---|---|
| *V* | vector |

#### 5.6.2.4 fix_phase_cmplx()

```
void fix_phase_cmplx (
            CVector V )
```

Add a global phase to make first complex amplitude positive.

**Note**

This only works for certain states (zero, one, plus, minus, etc.)

**Parameters**

| | |
|---|---|
| *V* | complex vector |

### 5.6.2.5 init_state()

```
void init_state (
             Vector V,
             State s )
```

Initialise a real state vector.

**Parameters**

| | |
|---|---|
| *V* | vector |
| *s* | state |

### 5.6.2.6 init_state_cmplx()

```
void init_state_cmplx (
             CVector V,
             State s )
```

Initialise a complex state vector.

**Parameters**

| | |
|---|---|
| *V* | complex vector |
| *s* | complex state |

### 5.6.2.7 make_ops()

```
void make_ops (
             Matrix2 X,
             Matrix2 Z,
             Matrix2 H )
```

Create real? X, Z, H.

**Parameters**

| | |
|---|---|
| *X* | Pauli x matrix |
| *Z* | Pauli z matrix |
| *H* | Hadamard matrix |

#### 5.6.2.8 make_ops_cmplx()

```
void make_ops_cmplx (
             CMatrix2 X,
             CMatrix2 Y,
             CMatrix2 Z,
             CMatrix2 H )
```

Create complex X, Y, Z and H.

**Parameters**

| | |
|---|---|
| *X* | Pauli X c-Matrix |
| *Z* | Pauli Z c-matrix |
| *H* | Hadamard c-matrix |
| *Y* | Pauli Y c-matrix |

#### 5.6.2.9 mat_mul()

```
void mat_mul (
             Matrix2 M,
             Vector V )
```

2x2 matrix multiplication

**Parameters**

| | |
|---|---|
| *M* | real matrix |
| *V* | real vector |

#### 5.6.2.10 mat_mul_cmplx()

```
void mat_mul_cmplx (
             CMatrix2 M,
             CVector V )
```

2x2 complex matrix multiplication

**Parameters**

| | |
|---|---|
| *M* | complex matrix |
| *V* | complex vector |

#### 5.6.2.11 show_state()

```
void show_state (
            Vector V )
```

Show the qubit state on the LEDs.

**Parameters**

| | |
|---|---|
| *V* | real vector |

#### 5.6.2.12 show_state_cmplx()

```
void show_state_cmplx (
            CVector V )
```

Show the qubit state on the LEDs.

**Parameters**

| | |
|---|---|
| *V* | complex vector |

## 5.7 dspic33e/qcomp-sim-c.X/spi.c File Reference

Description: Functions for communicating with serial devices.

```
#include "spi.h"
```

Include dependency graph for spi.c:

### Functions

- int setup_spi (void)

  *Set up serial peripheral interface.*
- int send_byte_spi_1 (int data)

  *Send a byte to the SPI1 peripheral.*
- int read_byte_spi_3 ()

  *Recieve a byte from the SPI3 peripheral.*

### 5.7.1 Detailed Description

Description: Functions for communicating with serial devices.

### 5.7.2 Function Documentation

#### 5.7.2.1 send_byte_spi_1()

```
int send_byte_spi_1 (
            int data )
```

Send a byte to the SPI1 peripheral.

**Parameters**

| | |
|---|---|
| *data* | byte to be sent to SPI1 |

#### 5.7.2.2 setup_spi()

```
int setup_spi (
            void  )
```

Set up serial peripheral interface.

Pin mappings — Pin mappings and codes — J10:41 = J1:91 = uC:70 = RPI74 (PPS code: 0100 1010) J10:44 = J1:93 = uC:9 = RPI52 (PPS code: 0011 0100) J10:47 = J1:101 = uC:34 = RPI42 (PPS code: 0010 1010) J10:43 = J1:95 = uC:72 = RP64 (PPS reg: RPOR0_L; code: 0100 0000) J10:46 = J1:97 = uC:69 = RPI73 (PPS code: 0100 1001) J10:7 = J1:13 = uC:3 = RP85 (PPS reg: RPOR6_L; code: 0101 0101) J10:5 = J1:7 = uC:5 = RP87 (PPS reg: RPOR6_H) J10:55 = J1:117 = uC:10 = RP118 (PPS reg: RPOR13_H)

— Pin mappings for SPI 1 module — SPI 1 Clock Out (SCK1) PPS code: 000110 (0x06) SPI 1 Data Out (SDO1) PPS code: 000101 (0x05) SPI 1 Slave Select PPS code: 000111

— Pin mappings for SPI 3 module — SPI 3 Clock Out (SCK3) PPS code: 100000 (0x20) SPI 3 Data Out (SDO3) PPS code: 011111 (0x1F) SPI 3 Slave Select PPS code: 100001

Configure the SPI 1 pins

$<$ Put SCK1 on J10:43

$<$ Put SDO1 on J10:55

The clock pin also needs to be configured as an input

$<$ Set SCK1 on J10:43 as input

Configure the SPI 3 output pins

$<$ Put SCK3 on J10:7

$<$ Put SDO3 on J10:5

$<$ Put SDI3 on J10:44

$<$ Set SCK3 on J10:7 as input

```
    @note
```

SPI 1 clock configuration

SCK1 = F_CY / (Primary Prescaler $*$ Secondary Prescaler)

Assuming that F_CY = 50MHz, and the prescalers are 4 and 1, the SPI clock frequency will be 12.5MHz.

## 5.8 dspic33e/qcomp-sim-c.X/spi.h File Reference

Description: SPI communication functions.

```
#include "p33EP512MU810.h"
#include "xc.h"
```

Include dependency graph for spi.h: This graph shows which files directly or indirectly include this file:

### Functions

- int setup_spi (void)

  *Set up serial peripheral interface.*
- int send_byte_spi_1 (int data)

  *Send a byte to the SPI1 peripheral.*
- int read_byte_spi_3 ()

  *Recieve a byte from the SPI3 peripheral.*

### 5.8.1 Detailed Description

Description: SPI communication functions.

### 5.8.2 Function Documentation

#### 5.8.2.1 send_byte_spi_1()

```
int send_byte_spi_1 (
              int data )
```

Send a byte to the SPI1 peripheral.

**Parameters**

| | |
|---|---|
| *data* | byte to be sent to SPI1 |

#### 5.8.2.2 setup_spi()

```
int setup_spi (
              void   )
```

Set up serial peripheral interface.

Pin mappings — Pin mappings and codes — J10:41 = J1:91 = uC:70 = RPI74 (PPS code: 0100 1010) J10:44 = J1:93 = uC:9 = RPI52 (PPS code: 0011 0100) J10:47 = J1:101 = uC:34 = RPI42 (PPS code: 0010 1010) J10:43 = J1:95 = uC:72 = RP64 (PPS reg: RPOR0_L; code: 0100 0000) J10:46 = J1:97 = uC:69 = RPI73 (PPS code: 0100 1001) J10:7 = J1:13 = uC:3 = RP85 (PPS reg: RPOR6_L; code: 0101 0101) J10:5 = J1:7 = uC:5 = RP87 (PPS reg: RPOR6_H) J10:55 = J1:117 = uC:10 = RP118 (PPS reg: RPOR13_H)

— Pin mappings for SPI 1 module — SPI 1 Clock Out (SCK1) PPS code: 000110 (0x06) SPI 1 Data Out (SDO1) PPS code: 000101 (0x05) SPI 1 Slave Select PPS code: 000111

— Pin mappings for SPI 3 module — SPI 3 Clock Out (SCK3) PPS code: 100000 (0x20) SPI 3 Data Out (SDO3) PPS code: 011111 (0x1F) SPI 3 Slave Select PPS code: 100001

Configure the SPI 1 pins

< Put SCK1 on J10:43

< Put SDO1 on J10:55

The clock pin also needs to be configured as an input

< Set SCK1 on J10:43 as input

Configure the SPI 3 output pins

< Put SCK3 on J10:7

< Put SDO3 on J10:5

< Put SDI3 on J10:44

< Set SCK3 on J10:7 as input

```
    @note
```

SPI 1 clock configuration

SCK1 = F_CY / (Primary Prescaler ∗ Secondary Prescaler)

Assuming that F_CY = 50MHz, and the prescalers are 4 and 1, the SPI clock frequency will be 12.5MHz.

## 5.9 dspic33e/qcomp-sim-c.X/tests.c File Reference

Description: Contains all the tests we have performed on the micro- controller.

```
#include "tests.h"
#include "io.h"
#include "quantum.h"
#include "time.h"
```

Include dependency graph for tests.c:

## Functions

- void mat_mul_test_cmplx ()

    *Testing the speed of 2^15 2x2 real matrix multiplications void mat_mul_test() {.*
- void one_qubit_cmplx ()

    *Simulating one qubit.*
- void **dim_leds** ()

### 5.9.1 Detailed Description

Description: Contains all the tests we have performed on the micro- controller.

### 5.9.2 Function Documentation

#### 5.9.2.1 mat_mul_test_cmplx()

```
void mat_mul_test_cmplx ( )
```

Testing the speed of 2^15 2x2 real matrix multiplications void mat_mul_test() {.

Define state vector |0> = (1,0) |1> = (0,1) Vector V; init_state(V, ZERO);

Matrix2 X = {{0}}, Z = {{0}}, H = {{0}}; make_ops(X, Z, H);

Start the timer start_timer();

Do a matrix multiplication test unsigned int n = 0; while (n < 32768) { mat_mul(X, V); n++; }

Read the timer unsigned long int time = read_timer();

Show that the test is finished set_led(red, on);

wait (add a breakpoint here) while(1 == 1);

}

#### 5.9.2.2 one_qubit_cmplx()

```
void one_qubit_cmplx ( )
```

Simulating one qubit.

Buttons apply H, X and Z and LEDs display the state of the qubit. void one_qubit() {

Define quantum operations Matrix2 X = {{0}}, Z = {{0}}, H = {{0}}; make_ops(X, Z, H);

Define state vector |0> = (1,0) |1> = (0,1) Vector V; init_state(V, ZERO);

Show qubit state show_state(V);

while (1 == 1) {

Wait for user to choose an operation int btn1 = off, btn2 = off, btn3 = off; while ((btn1 == off) && (btn2 == off) && (btn3 == off)) { btn1 = read_btn(sw1); btn2 = read_btn(sw2); btn3 = read_btn(sw3); }

Apply operation if (btn1 == on) mat_mul(H, V); // Multiply H by V, put result in V if (btn2 == on) mat_mul(X, V); // Multiply X by V, put result in V if (btn3 == on) mat_mul(Z, V); // Multiply Z by V, put result in V

Add a global phase to make first amplitude positive fix_phase(V);

Clean state clean_state(V);

Show qubit state show_state(V);

Wait for all the buttons to be released while ((btn1 == on) || (btn2 == on) || (btn3 == on)) { btn1 = read_btn(sw1); btn2 = read_btn(sw2); btn3 = read_btn(sw3); }

Short delay to stop button bouncing unsigned long int cnt = 0; // 32 bit int while (cnt < 100000) cnt++;

} }

## 5.10 dspic33e/qcomp-sim-c.X/tests.h File Reference

Description: Header file containing all the tests we performed.

```
#include "p33EP512MU810.h"
#include "xc.h"
```

Include dependency graph for tests.h: This graph shows which files directly or indirectly include this file:

### Functions

- void **mat_mul_test** ()
- void mat_mul_test_cmplx ()

  *Testing the speed of 2$^\wedge$ 15 2x2 real matrix multiplications void mat_mul_test() {.*
- void **one_qubit** ()
- void one_qubit_cmplx ()

  *Simulating one qubit.*
- void **dim_leds** ()
- void **multi_led_strobe** ()

### 5.10.1 Detailed Description

Description: Header file containing all the tests we performed.

### 5.10.2 Function Documentation

#### 5.10.2.1 mat_mul_test_cmplx()

```
void mat_mul_test_cmplx ( )
```

Testing the speed of 2^15 2x2 real matrix multiplications void mat_mul_test() {.

Define state vector |0> = (1,0) |1> = (0,1) Vector V; init_state(V, ZERO);

Matrix2 X = {{0}}, Z = {{0}}, H = {{0}}; make_ops(X, Z, H);

Start the timer start_timer();

Do a matrix multiplication test unsigned int n = 0; while (n < 32768) { mat_mul(X, V); n++; }

Read the timer unsigned long int time = read_timer();

Show that the test is finished set_led(red, on);

wait (add a breakpoint here) while(1 == 1);

}

#### 5.10.2.2 one_qubit_cmplx()

```
void one_qubit_cmplx ( )
```

Simulating one qubit.

Buttons apply H, X and Z and LEDs display the state of the qubit. void one_qubit() {

Define quantum operations Matrix2 X = {{0}}, Z = {{0}}, H = {{0}}; make_ops(X, Z, H);

Define state vector |0> = (1,0) |1> = (0,1) Vector V; init_state(V, ZERO);

Show qubit state show_state(V);

while (1 == 1) {

Wait for user to choose an operation int btn1 = off, btn2 = off, btn3 = off; while ((btn1 == off) && (btn2 == off) && (btn3 == off)) { btn1 = read_btn(sw1); btn2 = read_btn(sw2); btn3 = read_btn(sw3); }

Apply operation if (btn1 == on) mat_mul(H, V); // Multiply H by V, put result in V if (btn2 == on) mat_mul(X, V); // Multiply X by V, put result in V if (btn3 == on) mat_mul(Z, V); // Multiply Z by V, put result in V

Add a global phase to make first amplitude positive fix_phase(V);

Clean state clean_state(V);

Show qubit state show_state(V);

Wait for all the buttons to be released while ((btn1 == on) || (btn2 == on) || (btn3 == on)) { btn1 = read_btn(sw1); btn2 = read_btn(sw2); btn3 = read_btn(sw3); }

Short delay to stop button bouncing unsigned long int cnt = 0; // 32 bit int while (cnt < 100000) cnt++;

} }

## 5.11 dspic33e/qcomp-sim-c.X/time.c File Reference

Description: Functions to control the on chip timers.

```
#include "time.h"
```

Include dependency graph for time.c:

### Functions

- void **setup_clock** ()
- void setup_timer ()
- void **reset_timer** ()
- void **start_timer** ()
- void **stop_timer** ()
- unsigned long int **read_timer** ()

### 5.11.1 Detailed Description

Description: Functions to control the on chip timers.

### 5.11.2 Function Documentation

#### 5.11.2.1 setup_timer()

```
void setup_timer ( )
```

**Todo** distinguish between the two different timers here...

## 5.12 dspic33e/qcomp-sim-c.X/time.h File Reference

Description: Header file containing all the timing functions.

```
#include "p33EP512MU810.h"
#include "xc.h"
```

Include dependency graph for time.h: This graph shows which files directly or indirectly include this file:

### Functions

- void **setup_clock** ()
- void setup_timer ()
- void **reset_timer** ()
- void **start_timer** ()
- void **stop_timer** ()
- unsigned long int **read_timer** ()

### 5.12.1 Detailed Description

Description: Header file containing all the timing functions.

### 5.12.2 Function Documentation

#### 5.12.2.1 setup_timer()

```
void setup_timer ( )
```

**Todo** distinguish between the two different timers here...

# Index