Quantum Computer Outreach Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

Todo List

Global read_external_buttons ()

read buttons

Global set_external_led (int data)

How long should this be?

Global setup_spi (void)

TODO – should clear the ANSEL registers here to It's probably not strictly necessary (already clear)

Global TLC591x_mode_switch (int mode)

mode switcher for LED Driver

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_GLOBAL Struct Reference

Data Fields

- int **strobe_leds**
- int **strobe_state**

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

Description: Include this once at the top of main.

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 **PERIOD** 500000

Functions

- int setup_io (void)
  Set up LEDs and buttons on port D.
- void __attribute__ ((__interrupt__, no_auto_psv))
  Interrupt service routine for timer 4.
- void start_strobe ()
  Set LEDs flashing.
- void stop_strobe ()
  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 set_external_led (int data)

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

  Switch between normal and special mode.
- int read_external_buttons ()

  Read external buttons.

Variables

- LED_GLOBAL led_global = {0}

  Global LED strobing state parameter.

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 Function Documentation

5.2.2.1 __attribute__()

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

Interrupt service routine for timer 4.

Note

I have no idea what this line means...

5.2.2.2 flash_all()

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

Flash all the LEDs a number of times.

Parameters

number	

5.2.2.3 flash_led()

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

Flash LED a number of times.

Parameters

color	
number	

5.2.2.4 read_btn()

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

Read the state of a push button.

Parameters

btn	

Note

How well do you know C

5.2.2.5 read_external_buttons()

```
int read_external_buttons ( )
```

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

5.2.2.6 set_external_led()

```
int set_external_led (
            int data )
```

Turn on an LED via the external 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	the byte to send to LED driver

Set LE(ED1) pin

Todo How long should this be?

5.2.2.7 set_led()

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

Turn a particular LED on or off.

Parameters

color	
state	

5.2.2.8 set_strobe()

```
void set_strobe (
```

```
            int color,
            int state )
```

Set an LED strobing.

Parameters

color	LED pin to strobe
state	help

5.2.2.9 setup_io()

```
int setup_io (
            void  )
```

Set up LEDs and buttons on port D.

Parameters

OE	global var for OE

Set the OE pin high

Set OE(ED2) pin

5.2.2.10 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

5.2.2.11 toggle_strobe()

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

Toggle LED strobe.

Parameters

color	

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

Macros

- #define **red** 0
- #define **amber** 1
- #define **green** 2
- #define **sw1** 6
- #define **sw2** 7
- #define **sw3** 13
- #define **off** 0
- #define **on** 1
- #define **LE** 3
- #define **OE** 4
- #define **SH** 5
- #define **CLK_INH** 8

Functions

- int setup_io (void)
  Set up LEDs and buttons on port D.
- 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.
- void start_strobe ()
  Set LEDs flashing.
- void set_strobe (int color, int state)
  Set an LED strobing.
- void toggle_strobe (int color)
  Toggle LED strobe.
- int set_external_led (int data)
  Turn on an LED via the external display driver.

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 LED a number of times.

Parameters

color	
number	

5.3.2.3 read_btn()

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

Read the state of a push button.

Parameters

btn	

Note

How well do you know C

5.3.2.4 set_external_led()

```
int set_external_led (
            int data )
```

Turn on an LED via the external 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	the byte to send to LED driver

Set LE(ED1) pin

Todo How long should this be?

5.3.2.5 set_led()

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

Turn a particular LED on or off.

Parameters

color	
state	

5.3.2.6 set_strobe()

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

Set an LED strobing.

Parameters

color	LED pin to strobe
state	help

5.3.2.7 setup_io()

```
int setup_io (
            void  )
```

Set up LEDs and buttons on port D.

Parameters

OE	global var for OE

Set the OE pin high

Set OE(ED2) pin

5.3.2.8 toggle_strobe()

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

Toggle LED strobe.

Parameters

color	

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  )
```

Loop to cycle through LEDs 0 - 15

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** (Matrix2 X, Matrix2 Z, Matrix2 H)
- void **make_ops_cmplx** (CMatrix2 X, CMatrix2 Y, CMatrix2 Z, CMatrix2 H)
- void **init_state** (Vector V, State s)
- void **init_state_cmplx** (CVector V, State s)
- void **mat_mul** (Matrix2 M, Vector V)
- void **mat_mul_cmplx** (CMatrix2 M, CVector V)
- void **fix_phase** (Vector V)
- void **fix_phase_cmplx** (CVector V)
- void **clean_state** (Vector V)
- void **clean_state_cmplx** (CVector V)
- void **show_state** (Vector V)
- void **show_state_cmplx** (CVector V)

5.5.1 Detailed Description

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

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**
- typedef Q15 **Complex**[2]
- typedef Q15 **Matrix4**[4][4]
- typedef Q15 **CMatrix4**[4][4][2]
- typedef Q15 **Matrix2**[2][2]
- typedef Q15 **CMatrix2**[2][2][2]
- typedef Q15 **Vector**[2]
- typedef Q15 **CVector**[2][2]

Enumerations

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

Functions

- void **make_ops** (Matrix2 X, Matrix2 Z, Matrix2 H)
- void **make_ops_cmplx** (CMatrix2 X, CMatrix2 Y, CMatrix2 Z, CMatrix2 H)
- void **init_state** (Vector V, State s)
- void **init_state_cmplx** (CVector V, State s)
- void **mat_mul** (Matrix2 M, Vector V)
- void **mat_mul_cmplx** (CMatrix2 M, CVector V)
- void **fix_phase** (Vector V)
- void **fix_phase_cmplx** (CVector V)
- void **clean_state** (Vector V)
- void **clean_state_cmplx** (CVector V)
- void **show_state** (Vector V)
- void **show_state_cmplx** (CVector V)

5.6.1 Detailed Description

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

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)
- int **send_byte_spi_1** (int data)
- int **read_byte_spi_3** ()

5.7.1 Detailed Description

Description: Functions for communicating with serial devices.

5.7.2 Function Documentation

5.7.2.1 setup_spi()

```
int setup_spi (
            void  )
```

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

Todo TODO – should clear the ANSEL registers here to It's probably not strictly necessary (already clear)

```
@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)
- int **send_byte_spi_1** (int data)
- int **read_byte_spi_3** ()

5.8.1 Detailed Description

Description: SPI communication functions.

5.8.2 Function Documentation

5.8.2.1 setup_spi()

```
int setup_spi (
            void  )
```

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

Todo TODO – should clear the ANSEL registers here to It's probably not strictly necessary (already clear)

```
@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** ()
- void **mat_mul_test_cmplx** ()
- void **one_qubit** ()
- void **one_qubit_cmplx** ()
- void **dim_leds** ()
- void **multi_led_strobe** ()

5.9.1 Detailed Description

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

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** ()
- void **one_qubit** ()
- void **one_qubit_cmplx** ()
- void **dim_leds** ()
- void **multi_led_strobe** ()

5.10.1 Detailed Description

Description: Header file containing all the tests we performed.

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.

5.12 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.

Index