

## My Project

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Todo List</b>	<b>1</b>
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>File Documentation</b>	<b>5</b>
3.1	ADC.c File Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Function Documentation . . . . .	6
3.1.2.1	changeADC(int channel) . . . . .	6
3.1.2.2	clearADC(int channel) . . . . .	6
3.1.2.3	getADC(int channel) . . . . .	6
3.1.2.4	initADC(int channel) . . . . .	7
3.1.2.5	ISR(ADC_vect) . . . . .	7
3.1.3	Variable Documentation . . . . .	7
3.1.3.1	adch . . . . .	7
3.1.3.2	adcl . . . . .	7
3.1.3.3	readNewChannel . . . . .	7
3.2	arm.c File Reference . . . . .	8
3.2.1	Detailed Description . . . . .	9
3.2.2	Function Documentation . . . . .	9
3.2.2.1	controlArmWithButtons() . . . . .	9
3.2.2.2	controlJoint1WithButtons() . . . . .	9
3.2.2.3	controlPositionWithButtons() . . . . .	9

3.2.2.4	convertVoltsToDACVal(float volts)	9
3.2.2.5	drawTriangleWithButtons()	10
3.2.2.6	getCurrent(int joint)	10
3.2.2.7	getGs(int axis)	10
3.2.2.8	getJointAngle(int joint)	11
3.2.2.9	getTimeSeconds()	12
3.2.2.10	inPosition(int theta1, int theta2)	12
3.2.2.11	ISR(TIMER0_COMPA_vect)	12
3.2.2.12	printJointAngle(int joint)	12
3.2.2.13	printLogLineJoint2(int setPoint)	13
3.2.2.14	setJointAngles(int lowerJoint, int upperJoint)	13
3.2.3	Variable Documentation	13
3.2.3.1	degreesPerJoint1Val	13
3.2.3.2	degreesPerJoint2Val	13
3.2.3.3	lowerAngle	13
3.2.3.4	servicePID	13
3.2.3.5	timerCount	14
3.2.3.6	upperAngle	14
3.2.3.7	x_coord	14
3.2.3.8	y_coord	14
3.3	button.c File Reference	14
3.3.1	Detailed Description	15
3.3.2	Function Documentation	15
3.3.2.1	lastButtonPressed()	15
3.3.2.2	setupButtons(__8bitreg_t *DDxn, __8bitreg_t *PORTxn, __8bitreg_t *PINxbits, BOOL enablePullup)	15
3.3.3	Variable Documentation	15
3.3.3.1	b4	15
3.3.3.2	b5	16
3.3.3.3	b6	16
3.3.3.4	b7	16

3.3.3.5	PINybits	16
3.4	DAC.c File Reference	16
3.4.1	Detailed Description	16
3.4.2	Function Documentation	16
3.4.2.1	setDAC(int DACn, int SPIVal)	16
3.5	definitions.c File Reference	17
3.5.1	Detailed Description	17
3.5.2	Function Documentation	17
3.5.2.1	abs(int a)	17
3.5.2.2	betweenTwoVals(int value, int lower, int upper)	18
3.6	encoder.c File Reference	18
3.6.1	Detailed Description	18
3.6.2	Function Documentation	19
3.6.2.1	singleByteWrite(unsigned char op_code, unsigned char data, int joint)	19
3.6.2.2	slaveDeselect(int joint)	19
3.6.2.3	slaveSelect(int joint)	19
3.7	lab1.c File Reference	19
3.7.1	Detailed Description	20
3.7.2	Function Documentation	20
3.7.2.1	ADCToSerial(int channel)	20
3.7.2.2	ADCToSerialPart7(int channel)	21
3.7.2.3	initPart2and7Timer()	21
3.7.2.4	setFrequencyForPostScale(int Frequency)	21
3.7.2.5	signalGeneratorMain(int channel)	21
3.7.2.6	waitForButton7()	22
3.7.2.7	waitForChar(char c)	22
3.8	led.c File Reference	22
3.8.1	Detailed Description	22
3.8.2	Function Documentation	23
3.8.2.1	setupLEDs(__8bitreg_t *DDxn, __8bitreg_t *PORTxn)	23

3.9	main.c File Reference	23
3.9.1	Detailed Description	23
3.10	motors.c File Reference	24
3.10.1	Detailed Description	24
3.10.2	Function Documentation	24
3.10.2.1	driveLink(int link, int dir)	24
3.10.2.2	gotoAngles(int lowerTheta, int upperTheta)	25
3.10.2.3	gotoXY(int x, int y)	25
3.11	Periph.c File Reference	25
3.11.1	Detailed Description	26
3.11.2	Macro Definition Documentation	26
3.11.2.1	ENC_CLR_CMD	26
3.11.2.2	ENC_CNTR	26
3.11.2.3	ENC_RD_CMD	26
3.11.3	Function Documentation	26
3.11.3.1	encCount(int chan)	26
3.11.3.2	enclnit(int chan)	27
3.11.3.3	getAccel(int axis)	27
3.11.3.4	IRDist(int chan)	27
3.11.3.5	resetEncCount(int chan)	27
3.12	PID.c File Reference	28
3.12.1	Detailed Description	28
3.12.2	Function Documentation	28
3.12.2.1	calcPID(char link, int setPoint, int actPos)	28
3.12.2.2	setConst(char link, float Kp, float Ki, float Kd)	29
3.13	pot.c File Reference	29
3.13.1	Detailed Description	29
3.13.2	Function Documentation	29
3.13.2.1	potAngle(int pot)	29
3.13.2.2	potVolts(int pot)	30
3.14	SPI.c File Reference	30
3.14.1	Detailed Description	30
3.14.2	Function Documentation	31
3.14.2.1	spiTransceive(BYTE data)	31

# Chapter 1

## Todo List

### Member **b7**

- make a nice array or something
- make for rest of buttons

### Member **clearADC** (int channel)

Create the corresponding function to clear the last ADC calculation register and disconnect the input to the ADC if desired.

### Member **degreesPerJoint2Val**

- change these to constants

### Member **gotoXY** (int x, int y)

Use kinematic equations to move the end effector to the desired position.

### Member **inPosition** (int theta1, int theta2)

- pass in the threshold to use

### Member **IRDist** (int chan)

Make a function that is able to get the ADC value of the IR sensor.

### Member **setFrequencyForPostScale** (int Frequency)

- make more generic

### Member **upperAngle**

- get rid of these globals

### Member **waitForButton7** ()

- make more generic

### Member **y\_coord**

- get rid of these globals





## Chapter 2

# File Index

### 2.1 File List

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

<a href="#">ADC.c</a>	ADC setup function definitions . . . . .	5
<a href="#">arm.c</a>	Arm library . . . . .	8
<a href="#">button.c</a>	Button setup function . . . . .	14
<a href="#">DAC.c</a>	DAC library . . . . .	16
<a href="#">definitions.c</a>	Defines for our team . . . . .	17
<a href="#">encoder.c</a>	Encoder library . . . . .	18
<a href="#">lab1.c</a>	Lab1 functions . . . . .	19
<a href="#">led.c</a>	LED setup functions . . . . .	22
<a href="#">main.c</a>	Main loop running on board. Runs Lab1 functions . . . . .	23
<a href="#">motors.c</a>	Motor driving functions for the arm . . . . .	24
<a href="#">Periph.c</a>	Peripheral library for encoder and accel . . . . .	25
<a href="#">PID.c</a>	The source file for PID constants and calculations . . . . .	28
<a href="#">pot.c</a>	The potentiometer functions . . . . .	29
<a href="#">SPI.c</a>	Arm library . . . . .	30



## Chapter 3

# File Documentation

### 3.1 ADC.c File Reference

ADC setup function definitions.

```
#include "RBELib/RBELib.h"
#include "include/definitions.h"
```

#### Functions

- [ISR](#) (ADC\_vect)  
*ISR for updating ADC readings Runs when ADC reading completes based on configured ADC settings.*
- void [initADC](#) (int channel)  
*Initializes the ADC and makes one channel active.*
- void [clearADC](#) (int channel)  
*Disables ADC functionality and clears any saved values (globals).*
- unsigned short [getADC](#) (int channel)  
*Get the analog value from the configured channel.*
- void [changeADC](#) (int channel)  
*Change the channel the ADC is sampling if using interrupts.*

#### Variables

- volatile unsigned short [adch](#)
- volatile unsigned short [adcl](#)
- volatile unsigned short [readNewChannel](#)

### 3.1.1 Detailed Description

ADC setup function definitions.

This has everything needed to configure the ADC for 10 bit operation and reading the values.

Author

cpbove@wpi.edu

Date

28-Jan-2016

Version

1.0

### 3.1.2 Function Documentation

#### 3.1.2.1 void changeADC ( int *channel* )

Change the channel the ADC is sampling if using interrupts.

Parameters

<i>channel</i>	The ADC channel to switch to.
----------------	-------------------------------

#### 3.1.2.2 void clearADC ( int *channel* )

Disables ADC functionality and clears any saved values (globals).

This still needs to be tested.

Parameters

<i>channel</i>	The ADC channel to disable.
----------------	-----------------------------

**Todo** Create the corresponding function to clear the last ADC calculation register and disconnect the input to the ADC if desired.

#### 3.1.2.3 unsigned short getADC ( int *channel* )

Get the analog value from the configured channel.

#### Parameters

<i>channel</i>	The ADC channel to run a conversion on.
----------------	---

#### Returns

adcVal The 10 bit value returned by the ADC conversion.

#### 3.1.2.4 void initADC ( int *channel* )

Initializes the ADC and makes one channel active.

Enables ADC to use interrupts

#### Parameters

<i>channel</i>	The ADC channel to initialize
----------------	-------------------------------

#### 3.1.2.5 ISR ( ADC\_vect )

ISR for updating ADC readings Runs when ADC reading completes based on configured ADC settings.

#### Parameters

<i>ADC_vect</i>	Interrupt vector for ADC on chip
-----------------	----------------------------------

### 3.1.3 Variable Documentation

#### 3.1.3.1 adch

for storing the adch register value after an interrupt

#### 3.1.3.2 adcl

for storing the adch register value after an interrupt

#### 3.1.3.3 readNewChannel

used to make sure we read the new channel after we switch channels

## 3.2 arm.c File Reference

arm library

```
#include "include/arm.h"
#include "include/definitions.h"
#include "include/button.h"
#include "math.h"
```

### Functions

- [ISR \(TIMER0\\_COMPA\\_vect\)](#)  
*Timer ISR that runs at 100Hz.*
- void [initArm \(\)](#)  
*initialize the arm variables*
- void [setupTimer \(\)](#)  
*sets a 100Hz timer up on Timer 0*
- void [serviceArm \(\)](#)  
*runs functions critical to arm operation. Call as often as possible.*
- void [printHeaderForJointAngle \(\)](#)  
*prints the header for streaming joint angles*
- void [printHeaderForLogging \(\)](#)  
*prints the header for the main logging of step responses*
- float [getJointAngle](#) (int joint)  
*gets the current, calibrated joint angle of the passed joint number*
- void [printJointAngle](#) (int joint)  
*prints adcval, pot mV, and joint angle of the passed joint number*
- void [printLogLineJoint2](#) (int setPoint)  
*prints time, setpoint, jointAngle, PID output, and motor current*
- void [printJointAnglesAndPos](#) ()  
*streams joint angles in degrees and (x,y) tip positions in mm*
- int [getCurrent](#) (int joint)  
*gets motor current of specified joint*
- int [convertVoltsToDACVal](#) (float volts)  
*converts a voltage value to a DAC value*
- void [setJointAngles](#) (int lowerJoint, int upperJoint)  
*updates globals with new desired ones*
- void [controlArmWithButtons](#) ()  
*changes desired joint angle of arm based on button presses for testing*
- void [controlPositionWithButtons](#) ()  
*changes desired joint angles of arm based on button presses*
- void [controlJoint1WithButtons](#) ()  
*Lab3 - control motor drive voltages with buttons.*
- void [drawTriangleWithButtons](#) ()  
*state machine for drawing triangles.*
- float [getTimeSeconds](#) ()  
*gets the time in seconds*
- void [calcXY](#) ()  
*updates 2 globals with the current xy in mm of the end effector*
- BOOL [inPosition](#) (int theta1, int theta2)  
*check if the arm is in the angle position specified*
- float [getGs](#) (int axis)  
*runs getAccel and converts to G's*

## Variables

- float [degreesPerJoint1Val](#)
- float [degreesPerJoint2Val](#)
- float [x\\_coord](#)
- float [y\\_coord](#)
- int [lowerAngle](#)
- int [upperAngle](#)
- volatile BOOL [servicePID](#)
- volatile unsigned long [timerCount](#)

### 3.2.1 Detailed Description

arm library

#### Author

[cpbove@wpi.edu](mailto:cpbove@wpi.edu)

#### Date

30-Jan-2016

#### Version

1.0

### 3.2.2 Function Documentation

#### 3.2.2.1 void controlArmWithButtons ( )

changes desired joint angle of arm based on button presses for testing

##### Note

This does not run the PID control, just sets setpoint

#### 3.2.2.2 void controlJoint1WithButtons ( )

Lab3 - control motor drive voltages with buttons.

Sets first link to 0, -3, +3, or plus 6 Volts button 4 sets voltage to 0 button 5 sets voltage to -3 button 6 sets voltage to +3 button 7 sets voltage to +6

#### 3.2.2.3 void controlPositionWithButtons ( )

changes desired joint angles of arm based on button presses

##### Note

This does not run the PID control, just sets setpoint

#### 3.2.2.4 int convertVoltsToDACVal ( float volts )

converts a voltage value to a DAC value

**Parameters**

<i>volts</i>	to convert to DAC -7.2 to 7.2 volts
--------------	-------------------------------------

**Returns**

DAC value 0-4095

**3.2.2.5 void drawTriangleWithButtons ( )**

state machine for drawing triangles.

button 4 draws a prescribed triangle button 5 just moves to a pose button 6 records a motion button 7 plays back the motion

**Note**

This does not run the PID control, just sets setpoint

**3.2.2.6 int getCurrent ( int *joint* )**

gets motor current of specified joint

**Parameters**

<i>joint</i>	The joint to get the current for
--------------	----------------------------------

**Returns**

current in mA

**3.2.2.7 float getGs ( int *axis* )**

runs getAccel and converts to G's

**Parameters**

<i>axis</i>	0-2 for x,y,z axis to get g's on
-------------	----------------------------------

**Returns**

-3.0 to 3.0 g's of force



### 3.2.2.8 float getJointAngle ( int *joint* )

gets the current, calibrated joint angle of the passed joint number

**Parameters**

<i>joint</i>	1 or 2 of the joint to get the angle for
--------------	--

**Returns**

angle of joint in degrees (generally 0 to 180)

**3.2.2.9 float getTimeSeconds ( )**

gets the time in seconds

**Returns**

time in seconds

**3.2.2.10 BOOL inPosition ( int *theta1*, int *theta2* )**

check if the arm is in the angle position specified

**Parameters**

<i>theta1</i>	angle of first joint
<i>theta2</i>	angle of second joint

**Returns**

TRUE if arm is between

**Todo** pass in the threshold to use

**3.2.2.11 ISR ( TIMER0\_COMPA\_vect )**

Timer ISR that runs at 100Hz.

set flags for servicing at fixed intervals.

**Parameters**

<i>TIMER0_COMPA_vect</i>	Interrupt vector for timer0 vector on AVR
--------------------------	---

**3.2.2.12 void printJointAngle ( int *joint* )**

prints adcval,pot mV, and joint angle of the passed joint number

## Parameters

<i>joint</i>	1 or 2 of the joint to print data for
--------------	---------------------------------------

3.2.2.13 void printLogLineJoint2 ( int *setPoint* )

prints time, setpoint, jointAngle, PID output, and motor current

## Parameters

<i>setPoint</i>	the current command to the controller
-----------------	---------------------------------------

3.2.2.14 void setJointAngles ( int *lowerJoint*, int *upperJoint* )

updates globals with new desired ones

## Parameters

<i>lowerJoint</i>	position for the lower joint 1
<i>upperJoint</i>	position for the upper joint 2

## 3.2.3 Variable Documentation

## 3.2.3.1 degreesPerJoint1Val

for storing the degrees per adc value for joint 1

## 3.2.3.2 degreesPerJoint2Val

for storing the degrees per adc value for joint 2

**Todo** change these to constants

## 3.2.3.3 lowerAngle

for storing the current lowerAngle of the arm

## 3.2.3.4 servicePID

flag - TRUE if PID controller needs to be serviced, FALSE otherwise

### 3.2.3.5 timerCount

for keeping time. increments in 0.01 seconds

### 3.2.3.6 upperAngle

for storing the current upperAngle of the arm

**Todo** get rid of these globals

### 3.2.3.7 x\_coord

for storing the current x coordinate of the arm

### 3.2.3.8 y\_coord

for storing the current y coordinate of the arm

**Todo** get rid of these globals

## 3.3 button.c File Reference

button setup function

```
#include "RBELib/RBELib.h"
#include "include/button.h"
```

### Functions

- void [setupButtons](#) ([\\_\\_8bitreg\\_t](#) \*DDxn, [\\_\\_8bitreg\\_t](#) \*PORTxn, [\\_\\_8bitreg\\_t](#) \*PINxbits, BOOL enablePullup)  
*sets registers to configure the buttons as inputs*
- void [serviceButtons](#) ()  
*runs functions critical to button operation. Call as often as possible.*
- unsigned char [lastButtonPressed](#) ()  
*determines what button was pressed last*

### Variables

- [\\_\\_8bitreg\\_t](#) \* [PINybits](#)
- unsigned char [b4](#)
- unsigned char [b5](#)
- unsigned char [b6](#)
- unsigned char [b7](#)

### 3.3.1 Detailed Description

button setup function

To read buttons, check `PINxbits._Py`

Author

[cpbove@wpi.edu](mailto:cpbove@wpi.edu)

Date

28-Jan-2016

Version

1.0

### 3.3.2 Function Documentation

#### 3.3.2.1 unsigned char lastButtonPressed ( )

determines what button was pressed last

Returns

number of last button pressed

#### 3.3.2.2 void setupButtons ( \_\_8bitreg\_t \* DDxn, \_\_8bitreg\_t \* PORTxn, \_\_8bitreg\_t \* PINxbits, BOOL enablePullup )

sets registers to configure the buttons as inputs

Parameters

<i>*DDxn</i>	Pointer to the digital port (DDxn) that buttons are connected to
<i>*PORTxn</i>	Pointer to the PORTxn register for the buttons
<i>*PINxbits</i>	Pointer to the PINx register for the buttons
<i>enablePullUp</i>	Set true to enable pull up resistors on the buttons

### 3.3.3 Variable Documentation

#### 3.3.3.1 b4

storing last state of button 4

### 3.3.3.2 b5

storing last state of button 5

### 3.3.3.3 b6

storing last state of button 6

### 3.3.3.4 b7

storing last state of button 7

**Todo** make a nice array or something  
make for rest of buttons

### 3.3.3.5 \* PINybits

pointer to the PINbits the buttons are connected to

## 3.4 DAC.c File Reference

DAC library.

```
#include "RBELib/RBELib.h"
```

### Functions

- void `setDAC` (int DACn, int SPIVal)  
*Set the DAC to the given value on the chosen channel.*

### 3.4.1 Detailed Description

DAC library.

Author

`cpbove@wpi.edu`

Date

30-Jan-2016

Version

1.0

### 3.4.2 Function Documentation

#### 3.4.2.1 void `setDAC` ( int *DACn*, int *SPIVal* )

Set the DAC to the given value on the chosen channel.

## Parameters

<i>DACn</i>	The channel (0-3) that you want to set.
<i>SPIVal</i>	The value you want to set it to.

## 3.5 definitions.c File Reference

defines for our team

```
#include "RBELib/RBELib.h"
#include "include/definitions.h"
```

## Functions

- int [abs](#) (int a)  
*take absolute value of passed integer*
- BOOL [betweenTwoVals](#) (int value, int lower, int upper)  
*check if passed value is between lower and upper bounds*

### 3.5.1 Detailed Description

defines for our team

some basic useful functions that our team uses.

## Author

[cpbove@wpi.edu](mailto:cpbove@wpi.edu)

## Date

3-Feb-2016

## Version

1.0

### 3.5.2 Function Documentation

#### 3.5.2.1 int abs ( int a )

take absolute value of passed integer

**Parameters**

<i>a</i>	The number to take the absolute value of
----------	--

**3.5.2.2 BOOL betweenTwoVals ( int *value*, int *lower*, int *upper* )**

check if passed value is between lower and upper bounds

**Parameters**

<i>value</i>	the number to check with
<i>lower</i>	the lower bound to check against
<i>upper</i>	the upper bound to check against

**Returns**

true if value is between upper and lower numbers

**3.6 encoder.c File Reference**

encoder library

```
#include "include/encoder.h"
#include "include/definitions.h"
```

**Functions**

- void [singleByteWrite](#) (unsigned char op\_code, unsigned char data, int joint)  
*write a single byte to encoder*
- void [slaveSelect](#) (int joint)  
*sets the chip select low for the passed joint*
- void [slaveDeselect](#) (int joint)  
*sets the chip select high for the passed joint*
- void [waitForTransmissionEnd](#) ()  
*holds processor until transmission has finished*

**3.6.1 Detailed Description**

encoder library

**Author**

LSI Computer Systems, modified by [cpbove@wpi.edu](mailto:cpbove@wpi.edu)

**Date**

16-Feb-2016

**Version**

1.0



### 3.6.2 Function Documentation

#### 3.6.2.1 void singleByteWrite ( unsigned char *op\_code*, unsigned char *data*, int *joint* )

write a single byte to encoder

Parameters

<i>op_code</i>	one of the defined operation codes
<i>data</i>	data to write to the encoder
<i>joint</i>	1 or 2 for the encoder joint to read

#### 3.6.2.2 void slaveDeselect ( int *joint* )

sets the chip select high for the passed joint

Parameters

<i>joint</i>	1 or 2 to deselect
--------------	--------------------

#### 3.6.2.3 void slaveSelect ( int *joint* )

sets the chip select low for the passed joint

Parameters

<i>joint</i>	1 or 2 to select
--------------	------------------

## 3.7 lab1.c File Reference

Lab1 functions.

```
#include "RBELib/RBELib.h"
#include "include/definitions.h"
#include "include/lab1.h"
```

### Functions

- void [initPart2and7Timer](#) ()  
*ISR to increment global timer0Count, used for timestamps on Part 2 and 7 Also toggles an LED each run.*
- void [initSignalGeneratorTimer](#) ()  
*Configures Timer1 for Part 4 of lab(signal generation), prints header for logging.*
- void [outputSetup](#) ()

*Configures outputs for generating signals on pins.*

- void [waitForButton7](#) ()  
*Holds processor until button 7 is pressed.*
- void [waitForChar](#) (char c)  
*Holds the processor until the right character is received from USART.*
- void [ADCToSerial](#) (int channel)  
*Streams timestamp, adcval, potmv, and potangle to PC w/ USART Transmits timestamps, ADC value in counts, pot output in millivolts, and pot angle in degrees over the serial port to the screen in the Terminal.*
- void [ADCToSerialPart7](#) (int channel)  
*Streams timestamp and ADC values to the serial port for 1 second Transmits timestamps, ADC value in counts in degrees over the serial port to the screen in the Terminal.*
- void [setFrequencyForPostScale](#) (int Frequency)  
*Configures postscaler in ISR based on inputted frequency.*
- void [signalGeneratorMain](#) (int channel)  
*Generates signals w/particular duty cycles set by pot.*
- void [triangleSignalGengerator](#) ()

## Variables

- volatile unsigned long **timer0Count**
- volatile unsigned long **timer2Count**
- volatile unsigned **timer2CountSub**
- unsigned **postScale**

### 3.7.1 Detailed Description

Lab1 functions.

This includes all the functions needed for Lab1.

#### Authors

[cpbove@wpi.edu](mailto:cpbove@wpi.edu) [jlhonicker@wpi.edu](mailto:jlhonicker@wpi.edu) [dmmurray@wpi.edu](mailto:dmmurray@wpi.edu)

#### Date

28-Jan-2016

#### Version

1.0

### 3.7.2 Function Documentation

#### 3.7.2.1 void ADCToSerial ( int channel )

Streams timestamp, adcval, potmv, and potangle to PC w/ USART Transmits timestamps, ADC value in counts, pot output in millivolts, and pot angle in degrees over the serial port to the screen in the Terminal.

## Parameters

<i>channel</i>	The ADC channel to read from
----------------	------------------------------

3.7.2.2 void ADCToSerialPart7 ( int *channel* )

Streams timestamp and ADC values to the serial port for 1 second Transmits timestamps, ADC value in counts in degrees over the serial port to the screen in the Terminal.

## Parameters

<i>channel</i>	The ADC channel to read
----------------	-------------------------

## 3.7.2.3 void initPart2and7Timer ( )

ISR to increment global timer0Count, used for timestamps on Part 2 and 7 Also toggles an LED each run.

## Parameters

<i>TIMER0_COMPA_vect</i>	Vector for Timer0 ISR ISR to increment global timer2Count, used for signal generation
--------------------------	---

Uses a sub counter to get a slower rate for some of the signal generations.

## Parameters

<i>TIMER2_COMPA_vect</i>	Vector for Timer2 ISR Configures Timer0 for Parts 2 and 7 of lab, prints header for logging
--------------------------	---

3.7.2.4 void setFrequencyForPostScale ( int *Frequency* )

Configures postscaler in ISR based on inputted frequency.

Only handles a few frequencies: 1,20,100

**Todo** make more generic

## Parameters

<i>frequency</i>	The frequency to change to (1, 20, or 100)
------------------	--

3.7.2.5 void signalGeneratorMain ( int *channel* )

Generates signals w/particular duty cycles set by pot.

Also reads buttons 5-7 to determine which frequency to output

#### Parameters

<i>channel</i>	The ADC channel to read the pot from
----------------	--------------------------------------

#### 3.7.2.6 void waitForButton7 ( )

Holds processor until button 7 is pressed.

**Todo** make more generic

#### 3.7.2.7 void waitForChar ( char c )

Holds the processor until the right character is received from USART.

#### Parameters

<i>c</i>	The character to wait for.
----------	----------------------------

## 3.8 led.c File Reference

LED setup functions.

```
#include "include/led.h"
```

### Functions

- void [setupLEDs](#) (\_\_8bitreg\_t \*DDxn, \_\_8bitreg\_t \*PORTxn)  
*configures registers as outputs for the LEDs*

#### 3.8.1 Detailed Description

LED setup functions.

#### Author

[cpbove@wpi.edu](mailto:cpbove@wpi.edu)

#### Date

28-Jan-2016

#### Version

1.0

## 3.8.2 Function Documentation

### 3.8.2.1 void setupLEDs ( \_\_8bitreg\_t \* DDxn, \_\_8bitreg\_t \* PORTxn )

configures registers as outputs for the LEDs

Sets all the ports as outputs and then puts them all low to prevent them from turning on unintentionally.

#### Parameters

*DDxn	Pointer to the digital outputs (DDxn) that buttons will be connected to
*PORTxn	Pointer to the PORTxn that actually writes the outputs
enablePullUp	Set true to enable pull ups on the buttons

## 3.9 main.c File Reference

Main loop running on board. Runs Lab1 functions.

```
#include "RBELib/RBELib.h"
#include "include/definitions.h"
#include "include/button.h"
#include "include/lab1.h"
#include "include/led.h"
#include "include/arm.h"
```

### Functions

- int [main](#) (void)  
*main loop for AVR chip*

### 3.9.1 Detailed Description

Main loop running on board. Runs Lab1 functions.

This code runs a collection of Lab 1 functions to complete the lab. Some functions need to be uncommented or commented depending on which section they complete. This should be more streamlined in the future.

#### Author

[cpbove@wpi.edu](mailto:cpbove@wpi.edu)

#### Date

28-Jan-2016

#### Version

1.0

## 3.10 motors.c File Reference

Motor driving functions for the arm.

```
#include "RBELib/RBELib.h"
#include "include/definitions.h"
#include "include/arm.h"
```

### Functions

- void [stopMotors](#) ()  
*Helper function to stop the motors on the arm.*
- void [gotoAngles](#) (int lowerTheta, int upperTheta)  
*Drive the arm to a desired angle.*
- void [gotoXY](#) (int x, int y)  
*Drive the end effector of the arm to a desired X and Y position in the workspace.*
- void [driveLink](#) (int link, int dir)  
*Drive a link (upper or lower) in a desired direction.*
- void [homePos](#) ()  
*Drive the arm to a "home" position using the potentiometers. This should be called before using the encoders and just goes to a default position. Once this has been called once, you can initialize/clear the encoders.*

### 3.10.1 Detailed Description

Motor driving functions for the arm.

#### Author

Chris Bove

#### Date

Feb 3, 2016

### 3.10.2 Function Documentation

#### 3.10.2.1 void [driveLink](#) ( int *link*, int *dir* )

Drive a link (upper or lower) in a desired direction.

#### Parameters

<i>link</i>	Which link to control.
<i>dir</i>	Which way to drive the link. Between -2048 and +2048

## 3.10.2.2 void gotoAngles ( int lowerTheta, int upperTheta )

Drive the arm to a desired angle.

## Parameters

<i>lowerTheta</i>	The desired angle for the lower link.
<i>upperTheta</i>	The desired angle for the upper link.

## 3.10.2.3 void gotoXY ( int x, int y )

Drive the end effector of the arm to a desired X and Y position in the workspace.

## Parameters

<i>x</i>	The desired x position for the end effector.
<i>y</i>	The desired y position for the end effector.

**Todo** Use kinematic equations to move the end effector to the desired position.

## 3.11 Periph.c File Reference

peripheral library for encoder and accel

```
#include "RBELib/RBELib.h"
#include "include/encoder.h"
```

### Macros

- #define `ENC_CNTR` 0x20
- #define `ENC_CLR_CMD` 0x00
- #define `ENC_RD_CMD` 0x40

### Functions

- signed int `getAccel` (int axis)  
*Find the acceleration in the given axis (X, Y, Z).*
- int `IRDist` (int chan)  
*Read an IR sensor and calculate the distance of the block.*
- void `enclnit` (int chan)  
*Initialize the encoders with the desired settings.*
- void `resetEncCount` (int chan)  
*Reset the current count of the encoder ticks.*
- signed long `encCount` (int chan)  
*Finds the current count of one of the encoders.*

### 3.11.1 Detailed Description

peripheral library for encoder and accel

#### Author

[cpbove@wpi.edu](mailto:cpbove@wpi.edu) with help from Joe St. Germain

#### Date

15-02-2016

#### Version

1.0

### 3.11.2 Macro Definition Documentation

#### 3.11.2.1 `#define ENC_CLR_CMD 0x00`

hex value for clearing the encoder count

#### 3.11.2.2 `#define ENC_CNTR 0x20`

hex value for addressing the CNTR register on the encoder

#### 3.11.2.3 `#define ENC_RD_CMD 0x40`

hex value command for reading the encoder count

### 3.11.3 Function Documentation

#### 3.11.3.1 `signed long encCount ( int chan )`

Finds the current count of one of the encoders.

#### Parameters

<i>chan</i>	Channel of the encoder (change: Joint 1 or 2)
-------------	---

#### Returns

count The current count of the encoder.



### 3.11.3.2 void enclnit ( int *chan* )

Initialize the encoders with the desired settings.

#### Parameters

<i>chan</i>	Channel to initialize (change: Joint 1 or 2)
-------------	--

#### Note

leaves the encoder in Byte 4, Quad X1 default state

### 3.11.3.3 signed int getAccel ( int *axis* )

Find the acceleration in the given axis (X, Y, Z).

#### Parameters

<i>axis</i>	The axis that you want to get the measurement of.
-------------	---

#### Returns

gVal Value of acceleration.

### 3.11.3.4 int IRDist ( int *chan* )

Read an IR sensor and calculate the distance of the block.

#### Parameters

<i>chan</i>	The port that the IR sensor is on.
-------------	------------------------------------

#### Returns

value The distance the block is from the sensor.

**Todo** Make a function that is able to get the ADC value of the IR sensor.

### 3.11.3.5 void resetEncCount ( int *chan* )

Reset the current count of the encoder ticks.

#### Parameters

<i>chan</i>	The channel to clear. (change: Joint 1 or 2)
-------------	--

## 3.12 PID.c File Reference

The source file for PID constants and calculations.

```
#include "RBELib/RBELib.h"
#include "include/definitions.h"
#include "math.h"
```

### Functions

- void [setConst](#) (char link, float Kp, float Ki, float Kd)  
*Sets the Kp, Ki, and Kd values for 1 link.*
- signed int [calcPID](#) (char link, int setPoint, int actPos)  
*Calculate the PID value.*

### Variables

- pidConst [pidConsts](#)  
*Declaration for use in other files.*

### 3.12.1 Detailed Description

The source file for PID constants and calculations.

Sets the PID constants and calculate the PID value.

#### Author

Chris Bove

#### Date

2-3-16

### 3.12.2 Function Documentation

#### 3.12.2.1 signed int calcPID ( char link, int setPoint, int actPos )

Calculate the PID value.

#### Parameters

<i>link</i>	Which link to calculate the error for (Use 'U' and 'L').
<i>setPoint</i>	The desired position of the link.
<i>actPos</i>	The current position of the link.

#### 3.12.2.2 void setConst ( char *link*, float *Kp*, float *Ki*, float *Kd* )

Sets the Kp, Ki, and Kd values for 1 link.

##### Parameters

<i>link</i>	The link you want to set the values for (2 or 3).
<i>Kp</i>	Proportional value.
<i>Ki</i>	Integral value.
<i>Kd</i>	Derivative value.

## 3.13 pot.c File Reference

The potentiometer functions.

```
#include "RBELib/RBELib.h"
```

### Functions

- int [potAngle](#) (int pot)  
*Find the angle of the given potentiometer.*
- int [potVolts](#) (int pot)  
*Find the voltage value of the given potentiometer.*

### 3.13.1 Detailed Description

The potentiometer functions.

Use these functions to read the values from the pots.

#### Author

Chris Bove

#### Date

2-3-16

### 3.13.2 Function Documentation

#### 3.13.2.1 int potAngle ( int *pot* )

Find the angle of the given potentiometer.

**Parameters**

<i>pot</i>	The pot to check.
------------	-------------------

**Returns**

angle Angle of the potentiometer.

**3.13.2.2 int potVolts ( int *pot* )**

Find the voltage value of the given potentiometer.

**Parameters**

<i>pot</i>	The pot to get the value of.
------------	------------------------------

**Returns**

volts Voltage of potentiometer.

## 3.14 SPI.c File Reference

arm library

```
#include "RBELib/RBELib.h"
#include "include/definitions.h"
```

**Functions**

- void [initSPI](#) ()  
*Initializes the SPI bus for communication with all of your SPI devices.*
- unsigned char [spiTransceive](#) (BYTE data)  
*Send and receive a byte out of the MOSI line.*

### 3.14.1 Detailed Description

arm library

**Author**

[cpbove@wpi.edu](mailto:cpbove@wpi.edu)

**Date**

30-Jan-2016

**Version**

1.0

### 3.14.2 Function Documentation

#### 3.14.2.1 unsigned char spiTransceive ( BYTE *data* )

Send and receive a byte out of the MOSI line.

Please note that even if you do not want to receive any data back from a SPI device, the SPI standard requires you still receive something back even if it is junk data.

##### Parameters

<i>data</i>	The byte to send down the SPI bus.
-------------	------------------------------------

##### Returns

value The byte shifted in during transmit



# Index

- ADC.c, [5](#)
  - adch, [7](#)
  - adcl, [7](#)
  - changeADC, [6](#)
  - clearADC, [6](#)
  - getADC, [6](#)
  - ISR, [7](#)
  - initADC, [7](#)
  - readNewChannel, [7](#)
- ADCToSerial
  - lab1.c, [20](#)
- ADCToSerialPart7
  - lab1.c, [21](#)
- abs
  - definitions.c, [17](#)
- adch
  - ADC.c, [7](#)
- adcl
  - ADC.c, [7](#)
- arm.c, [8](#)
  - controlArmWithButtons, [9](#)
  - controlJoint1WithButtons, [9](#)
  - controlPositionWithButtons, [9](#)
  - convertVoltsToDACVal, [9](#)
  - degreesPerJoint1Val, [13](#)
  - degreesPerJoint2Val, [13](#)
  - drawTriangleWithButtons, [10](#)
  - getCurrent, [10](#)
  - getGs, [10](#)
  - getJointAngle, [10](#)
  - getTimeSeconds, [12](#)
  - ISR, [12](#)
  - inPosition, [12](#)
  - lowerAngle, [13](#)
  - printJointAngle, [12](#)
  - printLogLineJoint2, [13](#)
  - servicePID, [13](#)
  - setJointAngles, [13](#)
  - timerCount, [13](#)
  - upperAngle, [14](#)
  - x\_coord, [14](#)
  - y\_coord, [14](#)
- b4
  - button.c, [15](#)
- b5
  - button.c, [15](#)
- b6
  - button.c, [16](#)
- b7
  - button.c, [16](#)
- betweenTwoVals
  - definitions.c, [18](#)
- button.c, [14](#)
  - b4, [15](#)
  - b5, [15](#)
  - b6, [16](#)
  - b7, [16](#)
  - lastButtonPressed, [15](#)
  - PINybits, [16](#)
  - setupButtons, [15](#)
- calcPID
  - PID.c, [28](#)
- changeADC
  - ADC.c, [6](#)
- clearADC
  - ADC.c, [6](#)
- controlArmWithButtons
  - arm.c, [9](#)
- controlJoint1WithButtons
  - arm.c, [9](#)
- controlPositionWithButtons
  - arm.c, [9](#)
- convertVoltsToDACVal
  - arm.c, [9](#)
- DAC.c, [16](#)
  - setDAC, [16](#)
- definitions.c, [17](#)
  - abs, [17](#)
  - betweenTwoVals, [18](#)
- degreesPerJoint1Val
  - arm.c, [13](#)
- degreesPerJoint2Val
  - arm.c, [13](#)
- drawTriangleWithButtons
  - arm.c, [10](#)
- driveLink
  - motors.c, [24](#)
- ENC\_CLR\_CMD
  - Periph.c, [26](#)
- ENC\_CNTR
  - Periph.c, [26](#)
- ENC\_RD\_CMD
  - Periph.c, [26](#)
- encCount
  - Periph.c, [26](#)
- enclnit

- Periph.c, 26
- encoder.c, 18
  - singleByteWrite, 19
  - slaveDeselect, 19
  - slaveSelect, 19
- getADC
  - ADC.c, 6
- getAccel
  - Periph.c, 27
- getCurrent
  - arm.c, 10
- getGs
  - arm.c, 10
- getJointAngle
  - arm.c, 10
- getTimeSeconds
  - arm.c, 12
- gotoAngles
  - motors.c, 24
- gotoXY
  - motors.c, 25
- IRDist
  - Periph.c, 27
- ISR
  - ADC.c, 7
  - arm.c, 12
- inPosition
  - arm.c, 12
- initADC
  - ADC.c, 7
- initPart2and7Timer
  - lab1.c, 21
- lab1.c, 19
  - ADCToSerial, 20
  - ADCToSerialPart7, 21
  - initPart2and7Timer, 21
  - setFrequencyForPostScale, 21
  - signalGeneratorMain, 21
  - waitForButton7, 22
  - waitForChar, 22
- lastButtonPressed
  - button.c, 15
- led.c, 22
  - setupLEDs, 23
- lowerAngle
  - arm.c, 13
- main.c, 23
- motors.c, 24
  - driveLink, 24
  - gotoAngles, 24
  - gotoXY, 25
- PID.c, 28
  - calcPID, 28
  - setConst, 28
- PINybits
  - button.c, 16
- Periph.c, 25
  - ENC\_CLR\_CMD, 26
  - ENC\_CNTR, 26
  - ENC\_RD\_CMD, 26
  - encCount, 26
  - enclnit, 26
  - getAccel, 27
  - IRDist, 27
  - resetEncCount, 27
- pot.c, 29
  - potAngle, 29
  - potVolts, 30
- potAngle
  - pot.c, 29
- potVolts
  - pot.c, 30
- printJointAngle
  - arm.c, 12
- printLogLineJoint2
  - arm.c, 13
- readNewChannel
  - ADC.c, 7
- resetEncCount
  - Periph.c, 27
- SPI.c, 30
  - spiTransceive, 31
- servicePID
  - arm.c, 13
- setConst
  - PID.c, 28
- setDAC
  - DAC.c, 16
- setFrequencyForPostScale
  - lab1.c, 21
- setJointAngles
  - arm.c, 13
- setupButtons
  - button.c, 15
- setupLEDs
  - led.c, 23
- signalGeneratorMain
  - lab1.c, 21
- singleByteWrite
  - encoder.c, 19
- slaveDeselect
  - encoder.c, 19
- slaveSelect
  - encoder.c, 19
- spiTransceive
  - SPI.c, 31
- timerCount
  - arm.c, 13
- upperAngle



---

arm.c, [14](#)

waitForButton7  
lab1.c, [22](#)

waitForChar  
lab1.c, [22](#)

x\_coord  
arm.c, [14](#)

y\_coord  
arm.c, [14](#)