# REMEX Board

Generated by Doxygen 1.8.18

# Chapter 1

# HOW TO CLONE THIS REPOSITORY INTO CODE COMPOSER

Step 1: In code compser create a new empty project by clicking on File>New>CCS Project and clicking "Empty Project"

Step 2: Set the project to target the MSP430 FR2355 and name the project Remex

Step 3: Using git initialize a new repository in the folder that was just created using `git init`

Step 4: Set the remote of the git repository to this gitlab repo by using the command `git remote add origin git@gitlab.cs.mtech.edu:ee-projects/nasa-rmc-software-group/motor_board2.0.git`

Step 5: Pull any updates in from the master branch into your repository using `git pull origin master`

## 1.1 HOW TO USE THE PROJECT

The project is designed to be adaptable to any system on the robot. The different hardware interfaces are designed such that they can be easily used by each system without the need for modification. The rest of this document will demonstrate how to use the code in this repository to develop firmware to control new robotic systems with the Remex board.

### 1.1.1 Remex

The `remex.c` and `remex.h` files are the primary files for the user to edit to utilize the board. The two primary functions that should be used are the `init` function and the `loop` function. The `init` function is called once after the board has turned on and has been set to 8MHz processing. The `loop` function is then continuously called afterwards.

The `remex.h` file has also defined the default register map that the I2C master uses to interface with the board. This register map along with the provided I2C slave functions `recieve_cb`, `transmit_cb`, `start_condition_cb`, and `stop_condition_cb` implement a simple I2C register map implementation that will allow an I2C master to read and write data to the slave board. This also allows the board to execute commands from the master device with the `process_cmd` function.

### 1.1.2 Hardware IO

There are two kinds of hardware I/O this board can use. First is the QEI encoder interface and the second type is the limit switch interface. The board can support up to two channels of QEI at once, or one channel of QEI and 2 limit switches at the same time. It is important to note that the second channel of QEI uses the same pins as is used for the limit switches so both cannot be initialized at the same time.

#### 1.1.2.1 QEI Interface

The QEI is defined in the `hardwareIO.h` file. There is only one function `init_encoders(int*, int*)` to initialize the QEI hardware interrupts. It takes two parameters. Each parameter is a pointer to an integer which will store the count of clicks. Interrupts in the `hardwareIO.c` will increment and decrement passed in variable. Either of the parameters can be NULL (0L). If either of the parameters is NULL the encoder related to that parameter will not be initialized

#### 1.1.2.2 Limit switches

Limit switches are also defined in the `hardwareIO.h` file. The `init_switches` the function will initialize hardware interrupts to watch the limit switch pins on a HI/LO edge trigger. The function also configures the pins with pull up resistors to make them active low.

```
init_switches(void(*switch_a)(int), void(*switch_b)(int));
```

The two parameters the function takes are function pointers that will be invoked when the interrupt for a limit switch is triggered. This allows the user to determine the behavior the board has in response to switches being triggered. If either function is passed in as NULL (0L) then the corresponding switch is not initialized.

### 1.1.3 ADC

This board has 3 channels of 12 bit ADC to sample different signals. The first two channels A and B are dedicated to the on board current sensors on the PCB. The third channel, C, is an auxiliary ADC channel that can be used for any purpose. There are two functions to utilize the ADC on the board.

```
void setup_ADC(void (*cba)(int), void (*cbb)(int), void (*cbc)(int));
```

The `setup_ADC` function will initialize the hardware to sample on the ADC pins of the board. The function takes 3 function pointers as parameters. When a sample from the ADC has been processed the corresponding function pointer will be called with the value of the sample passed in to the first parameter.

```
void read_ADC(int channel);
```

`read_ADC` trigger the ADC to start reading a new sample. When the ADC is finished processing the sample it will fire the appropriate function pointer with the value of the sample. The first parameter of the function determines which of the 3 ADC channels should be sampled. Constants for each channel are provided in the `adc.h` header file.

### 1.1.4 PWM

PWM is used to manage the motor controllers that the Remex board is connected to. The Remex board is able to manage two motor controllers at once and therefore has to channels of PWM. All PWM functions can be found in the `pwm.h` header file.

```
void init_PWM_A();
```

`init_PWM_A` will initialize the timer hardware for PWM control. There is a equivalent `init_PWM_B` which will initialize the second channel of PWM.

```
void set_PWM_A(const int in);
```

Setting the outputted PWM signal is as simple as calling the `set_PWM_A` function passing an integer between 2000 and 4000. 2000 being full reverse on continuous rotation devices, and minimum angle on positional devices. 4000 is the full forward on continuous rotation devices and maximum angle on positional devices. Any value outside of this range of 2000 to 4000 will be capped to either 4000 if `in > 4000` or 2000 if `in < 2000`. There is an equivalent

### 1.1.5 I2C

The Remex board recieves commands over a serial i2c bus connected through the USB mini port. The Remex board has hardware to deal with i2c and this hardware can be initialized with the function `i2c_slave_init`.

```
void i2c_slave_init(
        void (*stt_cb)(void),
        void (*stp_cb)(void),
        void (*rx_cb)(unsigned char in),
        void (*tx_cb)(unsigned volatile int *out),
        unsigned char slave_addr);
```

`i2c_slave_init` will set up the hardware to behave as an I2C slave with an address of `slave_addr`. The first four parameters of the `i2c_slave_init` function are pointers to functions that will be called at the different points of the I2C protocol. The first pointer `stt_cb` is called when the microcontroller detects the start condition. The `stp_cb` is called when the stop condition is detected. The `rx_cb` is called when the microcontroller recieves a byte of data over the i2c bus and the recieved byte is passed in as the first parameter. When the microcontroller is requested by the master to send data the `tx_cb` is called and a pointer to the byte to send data is provided. Users of the `tx_cb` function will be able to send a byte in the `tx_cb` function by setting `out` equal to the data to be sent.

### 1.1.6 Main

The `main.c` should not need to be modified much by the user. The main function will set the internal clock speed of the processor to 8MHz and is responsible for calling the `init` function and the `loop` function from `remex.h`.

# Chapter 2

# File Index

## 2.1 File List

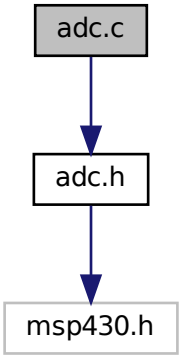
Here is a list of all files with brief descriptions:

# Chapter 3

# File Documentation

## 3.1 adc.c File Reference

`#include <adc.h>`
Include dependency graph for adc.c:




### Functions

- void setup_ADC (void(∗cba)(int), void(∗cbb)(int), void(∗cbc)(int))
- void read_adc (int channel)
- __interrupt void ADC10_ISR (void)

### 3.1.1 Function Documentation

#### 3.1.1.1 ADC10_ISR()

```
__interrupt void ADC10_ISR (
             void  )
```

#### 3.1.1.2 read_adc()

```
void read_adc (
             int channel )
```

read_adc will read the input channel A and call adc_callbackA with the result of the call

**Parameters**

| | | |
|---|---|---|
| in | *channel* | is the channel that should be read. Constants for channels to read are defined as constants. |

#### 3.1.1.3 setup_ADC()

```
void setup_ADC (
             void(*)(int) cba,
             void(*)(int) cbb,
             void(*)(int) cbc )
```

over_current.c

Created on: Aug 22, 2020 Author: Justin Bak Code Owner: Justin Bak

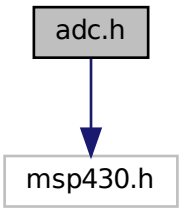
ADC on pins:

#### 3.1.1.4 PIN Channel

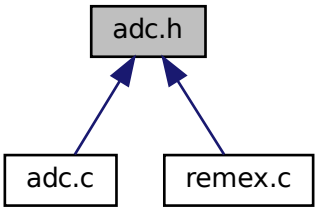
1.0 A 1.1 B 5.3 C

## 3.2 adc.h File Reference

`#include <msp430.h>`
Include dependency graph for adc.h:




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




### Macros

- #define CHANNEL_A ADCINCH_0
- #define CHANNEL_B ADCINCH_1
- #define CHANNEL_C ADCINCH_11

### Functions

- void read_adc (int channel)
- void setup_ADC (void(∗cba)(int), void(∗cbb)(int), void(∗cbc)(int))

### Variables

- void(∗ adc_callbackA )(int)

  *adc call back functions that will be executed when when an adc is read*
- void(∗ adc_callbackB )(int)
- void(∗ adc_callbackC )(int)

### 3.2.1 Macro Definition Documentation

#### 3.2.1.1 CHANNEL_A

```
#define CHANNEL_A ADCINCH_0
```

over_current.h

Created on: Aug 22, 2020 Author: Justin Bak Code Owner: Justin Bak Constants to be used in read_adc to select channels.

#### 3.2.1.2 CHANNEL_B

```
#define CHANNEL_B ADCINCH_1
```

#### 3.2.1.3 CHANNEL_C

```
#define CHANNEL_C ADCINCH_11
```

### 3.2.2 Function Documentation

#### 3.2.2.1 read_adc()

```
void read_adc (
            int channel )
```

read_adc will read the input channel A and call adc_callbackA with the result of the call

**Parameters**

| | | |
|---|---|---|
| in | *channel* | is the channel that should be read. Constants for channels to read are defined as constants. |

#### 3.2.2.2 setup_ADC()

```
void setup_ADC (
            void(*)(int) cba,
```

```
            void(*)(int) cbb,
            void(*)(int) cbc )
```

setup_ADC will map the call back functions and initilize GPIO for reading ADC on 3 channels.

**Parameters**

| | | |
|---|---|---|
| in | *cba* | function pointer which will be called by the ADC ISR when a sample on channel A is ready |
| in | *cbb* | function pointer which will be called by the ADC ISR when a sample on channel B is ready |
| in | *cbc* | function pointer which will be called by the ADC ISR when a sample on channel C is ready |

over_current.c

Created on: Aug 22, 2020 Author: Justin Bak Code Owner: Justin Bak

ADC on pins:

#### 3.2.2.3 PIN Channel

1.0 A 1.1 B 5.3 C

### 3.2.3 Variable Documentation

#### 3.2.3.1 adc_callbackA

```
void(* adc_callbackA) (int)
```

adc call back functions that will be executed when when an adc is read

#### 3.2.3.2 adc_callbackB

```
void(* adc_callbackB) (int)
```

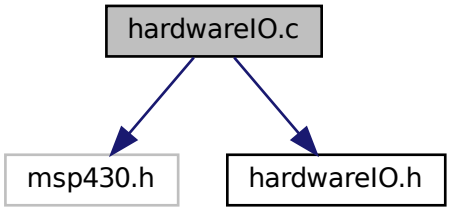
#### 3.2.3.3 adc_callbackC

```
void(* adc_callbackC) (int)
```

## 3.3 hardwareIO.c File Reference

```
#include <msp430.h>
#include "hardwareIO.h"
```

Include dependency graph for hardwareIO.c:




### Functions

- void init_encoders (int *count_a, int *count_b)
- __interrupt void Port_2 (void)

### Variables

- int * __count_a = 0
- int * __count_b = 0
- void(* switch_a_cb )(int)=0

### 3.3.1 Function Documentation

#### 3.3.1.1 init_encoders()

```
void init_encoders (
            int * count_a,
            int * count_b )
```

init_encoders will initialize hardware to read QEI encoders.

**Parameters**

| | | |
|---|---|---|
| in | *count_a* | integer pointer that should be updated from hardware interrupts for first encoder if pointer is 0 then do not initialize encoder. |
| in | *count_b* | integer pointer that should be updated from hardware interrupts for secondary encoder if pointer is 0 then do not initialize encoder. |

#### 3.3.1.2 Port_2()

```
__interrupt void Port_2 (
             void  )
```

### 3.3.2 Variable Documentation

#### 3.3.2.1 __count_a

```
int* __count_a = 0
```

#### 3.3.2.2 __count_b

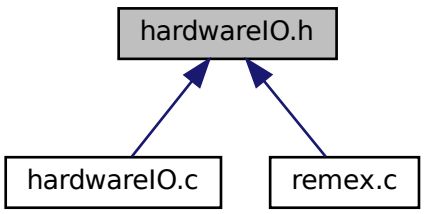
```
int * __count_b = 0
```

#### 3.3.2.3 switch_a_cb

```
void(* switch_a_cb) (int)=0
```

## 3.4 hardwareIO.h File Reference

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



## Functions

- void init_encoders (int ∗count_a, int ∗count_b)
- void init_switches (void(∗switch_a)(int), void(∗switch_b)(int))

### 3.4.1 Function Documentation

#### 3.4.1.1 init_encoders()

```
void init_encoders (
            int * count_a,
            int * count_b )
```

init_encoders will initialize hardware to read QEI encoders.

**Parameters**

| | | |
|---|---|---|
| in | *count_a* | integer pointer that should be updated from hardware interrupts for first encoder if pointer is 0 then do not initialize encoder. |
| in | *count_b* | integer pointer that should be updated from hardware interrupts for secondary encoder if pointer is 0 then do not initialize encoder. |

#### 3.4.1.2 init_switches()

```
void init_switches (
            void(*)(int) switch_a,
            void(*)(int) switch_b )
```

init_switches will initialize hardware interrupts to read limit switches or buttons.

**Parameters**

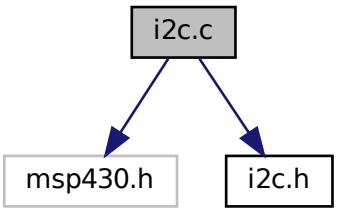
| | | |
|---|---|---|
| in | *switch_a* | is the callback function that will be invoked when the limit switch connected to pin 2.4 on the micro-controller is triggered. |
| in | *switch_b* | is the callback function that will be invoked when the limit switch connected to pin 2.5 on the micro-controller is triggered. |

if either parameter is null do not initialize hardware for the associated pin.

## 3.5 i2c.c File Reference

```
#include <msp430.h>
```

```
#include "i2c.h"
```
Include dependency graph for i2c.c:




## Functions

- void i2c_slave_init (void(∗stt_cb)(void), void(∗stp_cb)(void), void(∗rx_cb)(unsigned char in), void(∗tx_↩ cb)(unsigned volatile int ∗out), unsigned char slave_addr)
- __interrupt void USCIB0_ISR (void)

### 3.5.1 Function Documentation

#### 3.5.1.1 i2c_slave_init()

```
void i2c_slave_init (
            void(*)(void) stt_cb,
            void(*)(void) stp_cb,
            void(*)(unsigned char in) rx_cb,
            void(*)(unsigned volatile int *out) tx_cb,
            unsigned char slave_addr )
```

i2c_slave_init will initialize the i2c bus and set the device address.

**Parameters**

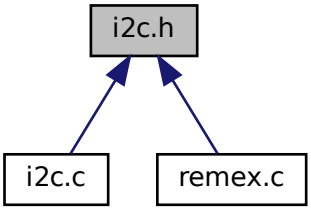
| | | |
|---|---|---|
| in | *stt_cb* | function pointer to be called when a i2c start condition is detected |
| in | *stp_cb* | function pointer to be called when a i2c stop condition is detected |
| in | *rx_cb* | function pointer to be called when i2c byte is received from master |
| in | *tx_cb* | function pointer to be called to send i2c byte to master |
| in | *slave_addr* | is the desired device address. |

#### 3.5.1.2 USCIB0_ISR()

```
__interrupt void USCIB0_ISR (
            void  )
```

## 3.6 i2c.h File Reference

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




### Macros

- #define SDA_PIN BIT2
- #define SCL_PIN BIT3

### Functions

- void i2c_slave_init (void(∗stt_cb)(void), void(∗stp_cb)(void), void(∗rx_cb)(unsigned char in), void(∗tx_↩ cb)(unsigned volatile int ∗out), unsigned char slave_addr)

### Variables

- void(∗ receive_func )(const unsigned char in)
- void(∗ transmit_func )(unsigned volatile int ∗out)
- void(∗ start_func )(void)
- void(∗ stop_func )(void)

### 3.6.1 Macro Definition Documentation

#### 3.6.1.1 SCL_PIN

```
#define SCL_PIN BIT3
```

#### 3.6.1.2 SDA_PIN

```
#define SDA_PIN BIT2
```

### 3.6.2 Function Documentation

#### 3.6.2.1 i2c_slave_init()

```
void i2c_slave_init (
            void(*)(void) stt_cb,
            void(*)(void) stp_cb,
            void(*)(unsigned char in) rx_cb,
            void(*)(unsigned volatile int *out) tx_cb,
            unsigned char slave_addr )
```

i2c_slave_init will initialize the i2c bus and set the device address.

**Parameters**

| | | |
|---|---|---|
| in | *stt_cb* | function pointer to be called when a i2c start condition is detected |
| in | *stp_cb* | function pointer to be called when a i2c stop condition is detected |
| in | *rx_cb* | function pointer to be called when i2c byte is received from master |
| in | *tx_cb* | function pointer to be called to send i2c byte to master |
| in | *slave_addr* | is the desired device address. |

### 3.6.3 Variable Documentation

#### 3.6.3.1 receive_func

```
void(* receive_func) (const unsigned char in)
```

#### 3.6.3.2 start_func

```
void(* start_func) (void)
```

#### 3.6.3.3 stop_func

```
void(* stop_func) (void)
```

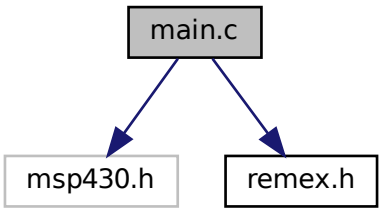
#### 3.6.3.4 transmit_func

```
void(* transmit_func) (unsigned volatile int *out)
```

## 3.7 main.c File Reference

```
#include <msp430.h>
#include "remex.h"
```

Include dependency graph for main.c:




### Macros

- #define ever (;;)

### Functions

- int main (void)

### 3.7.1 Macro Definition Documentation

#### 3.7.1.1 ever

```
#define ever (;;)
```

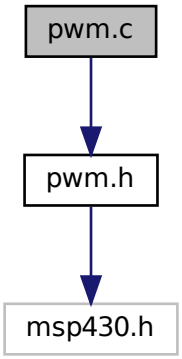
### 3.7.2 Function Documentation

#### 3.7.2.1 main()

```
int main (
            void  )
```

## 3.8 pwm.c File Reference

```
#include "pwm.h"
```
Include dependency graph for pwm.c:




### Functions

- void init_PWM_A ()
- void init_PWM_B ()
- void set_PWM_A (const int in)
- void set_PWM_B (const int in)
- void init_clk_src ()

### 3.8.1 Function Documentation

#### 3.8.1.1 init_clk_src()

```
void init_clk_src ( )
```

init_clk_src Sets the clock source of the PWM, the period of the clock, and the count mode of the clock

#### 3.8.1.2 init_PWM_A()

```
void init_PWM_A ( )
```

init_PWM_A Initializes PWM code for motor A(pin and timer settings)

#### 3.8.1.3 init_PWM_B()

```
void init_PWM_B ( )
```

init_PWM_B Initializes PWM code for motor B (pin and timer settings)

#### 3.8.1.4 set_PWM_A()

```
void set_PWM_A (
            const int in )
```

set_PWM_out_A Sets PWM to certain speed ranging from full speed backwards(2000) to full speed forwards(4000) for motor A

**Parameters**

| | | |
|---|---|---|
| in | *in* | value >=2000 and <=4000 |

#### 3.8.1.5 set_PWM_B()

```
void set_PWM_B (
            const int in )
```

set_PWM_out_B Sets PWM to certain speed ranging from full speed backwards(2000) to full speed forwards(4000) for motor B

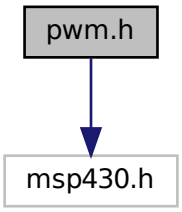
**Parameters**

| | | |
|---|---|---|
| in | *in* | value >=2000 and <=4000 |

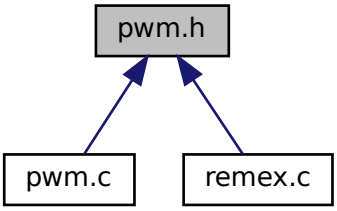
## 3.9 pwm.h File Reference

```
#include <msp430.h>
```

Include dependency graph for pwm.h:




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




### Macros

- #define PWMPINA BIT6
- #define PWMPINB BIT7
- #define MAX 4000
- #define MIN 2000

### Functions

- void init_PWM_A ()
- void init_PWM_B ()
- void set_PWM_A (const int in)
- void set_PWM_B (const int in)
- void init_clk_src ()

### 3.9.1 Macro Definition Documentation

#### 3.9.1.1 MAX

```
#define MAX 4000
```

#### 3.9.1.2 MIN

```
#define MIN 2000
```

#### 3.9.1.3 PWMPINA

```
#define PWMPINA BIT6
```

pwm.h

Created on: Oct 26, 2020 Author: Andrew McLinden, Marcus Frisbee

#### 3.9.1.4 PWMPINB

```
#define PWMPINB BIT7
```

### 3.9.2 Function Documentation

#### 3.9.2.1 init_clk_src()

```
void init_clk_src ( )
```

init_clk_src Sets the clock source of the PWM, the period of the clock, and the count mode of the clock

#### 3.9.2.2 init_PWM_A()

```
void init_PWM_A ( )
```

init_PWM_A Initializes PWM code for motor A(pin and timer settings)

#### 3.9.2.3 init_PWM_B()

```
void init_PWM_B ( )
```

init_PWM_B Initializes PWM code for motor B (pin and timer settings)

#### 3.9.2.4 set_PWM_A()

```
void set_PWM_A (
            const int in )
```

set_PWM_out_A Sets PWM to certain speed ranging from full speed backwards(2000) to full speed forwards(4000) for motor A

**Parameters**

| | | |
|---|---|---|
| in | *in* | value >=2000 and <=4000 |

#### 3.9.2.5 set_PWM_B()

```
void set_PWM_B (
            const int in )
```

set_PWM_out_B Sets PWM to certain speed ranging from full speed backwards(2000) to full speed forwards(4000) for motor B

**Parameters**

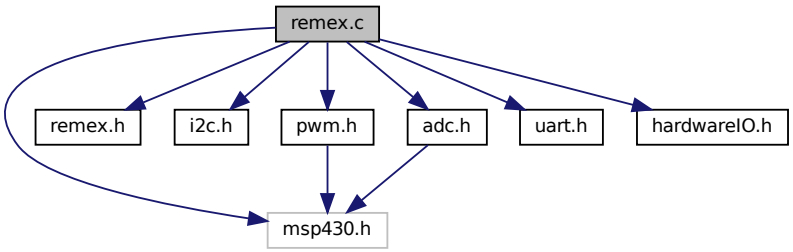
| | | |
|---|---|---|
| in | *in* | value >=2000 and <=4000 |

## 3.10 README.md File Reference

## 3.11 remex.c File Reference

```
#include <msp430.h>
#include "remex.h"
#include "i2c.h"
#include "pwm.h"
#include "adc.h"
#include "uart.h"
#include "hardwareIO.h"
```

Include dependency graph for remex.c:




### Functions

- void adc_channel_a (int current)

- void init (void)
- void loop (void)
- void clear_registers (void)
- void receive_cb (const unsigned char in)
- void transmit_cb (unsigned volatile int *out)
- void start_condition_cb (void)
- void stop_condition_cb (void)
- void process_cmd (unsigned char cmd)
- void int2str (int inval, char *str_out)

## Variables

- unsigned char regmap [REGMAP_SIZE] = { 0x00 }
- int encoder = 0
- char enc_str [4]
- unsigned char reg = UNKNOWN_REG
- enum i2c_states state = start

### 3.11.1 Function Documentation

#### 3.11.1.1 adc_channel_a()

```
void adc_channel_a (
            int current )
```

adc_channel_a is the call back function when the adc has finished it's conversion and has a value.

#### 3.11.1.2 clear_registers()

```
void clear_registers (
            void  )
```

clear_registers is a helper function that sets all the registers in the register map to zero.

#### 3.11.1.3 init()

```
void init (
            void  )
```

init is called once, and initializes the registers and libraries for the board.

#### 3.11.1.4 int2str()

```
void int2str (
            int inval,
            char * str_out )
```

int2str will convert an integer inval and store into string str_out

**Parameters**

| | | |
|---|---|---|
| in | *inval* | is the integer value to be converted to a string |
| out | *str_out* | is the output string where the string value of the integer will be written. |

#### 3.11.1.5 loop()

```
void loop (
            void  )
```

loop is called repeatedly as long as the board is powered.

#### 3.11.1.6 process_cmd()

```
void process_cmd (
            unsigned char cmd )
```

process_cmd is a function that is called when a command byte is received from the master.

**Parameters**

| | | |
|---|---|---|
| in | *cmd* | is the byte received from the master. |

#### 3.11.1.7 receive_cb()

```
void receive_cb (
            const unsigned char in )
```

receive_cb is a call back function that is called when the master writes a byte.

**Parameters**

| | | |
|---|---|---|
| in | *in* | is the byte received from the master. |

#### 3.11.1.8 start_condition_cb()

```
void start_condition_cb (
            void  )
```

start_condition_cb is a callback function for when a i2c start condition is detected from the master.

#### 3.11.1.9 stop_condition_cb()

```
void stop_condition_cb (
            void  )
```

stop_condition_cb is a callback function for when a i2c stop condition is detected from the master.

#### 3.11.1.10 transmit_cb()

```
void transmit_cb (
            unsigned volatile int * out )
```

transmit_cb is a callback function that is called when the master wants to read a byte from the slave.

**Parameters**

| | | |
|---|---|---|
| out | *the* | pointer to the transmit buffer to send a byte. |

### 3.11.2 Variable Documentation

#### 3.11.2.1 enc_str

```
char enc_str[4]
```

#### 3.11.2.2 encoder

```
int encoder = 0
```

#### 3.11.2.3 reg

```
unsigned char reg = UNKNOWN_REG
```

#### 3.11.2.4 regmap

```
unsigned char regmap[REGMAP_SIZE] = { 0x00 }
```

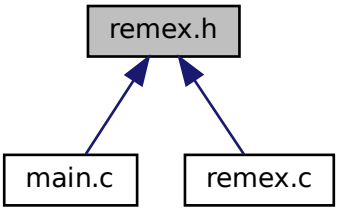
remex.c

#### 3.11.2.5 state

```
enum i2c_states state = start
```

## 3.12 remex.h File Reference

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




### Macros

- #define SLAVE_ADDR 0x48
- #define UNKNOWN_REG 0xFF
- #define UNKNOWN_PAR 0xFF
- #define command_reg 0xCC

  *Read / Write Registers.*
- #define mode_reg 0x00
- #define max_current_H 0x01
- #define max_current_L 0x02
- #define current_duration_H 0x03
- #define current_duration_L 0x04
- #define des_speed_a_H 0x05
- #define des_speed_a_L 0x06
- #define des_speed_b_H 0x07
- #define des_speed_b_L 0x08
- #define des_pos_a_H 0x09
- #define des_pos_a_L 0x0A
- #define des_pos_b_H 0x0B
- #define des_pos_b_L 0x0C
- #define READONLY 0x20
- #define position_a_H 0x20

  *Read only Registers.*
- #define position_a_L 0x21
- #define position_b_H 0x22
- #define position_b_L 0x23
- #define ADC_A_H 0x24
- #define ADC_A_L 0x25
- #define ADC_B_H 0x26

- #define ADC_B_L 0x27
- #define ADC_C_H 0x28
- #define ADC_C_L 0x29
- #define switch_states 0x2A
- #define remex_state 0x2F
- #define REGMAP_SIZE 0x2F

## Enumerations

- enum i2c_states { start, reg_set, stop, cmd_byte }

  *state machine states*

## Functions

- void adc_channel_a (int current)
- void adc_channel_b (int current)
- void adc_channel_c (int current)
- void clear_registers (void)
- void process_cmd (unsigned char cmd)
- void receive_cb (const unsigned char in)
- void transmit_cb (unsigned volatile int *out)
- void start_condition_cb (void)
- void stop_condition_cb (void)
- void init (void)
- void loop (void)
- void int2str (int inval, char *str_out)

### 3.12.1 Macro Definition Documentation

#### 3.12.1.1 ADC_A_H

```
#define ADC_A_H 0x24
```

#### 3.12.1.2 ADC_A_L

```
#define ADC_A_L 0x25
```

#### 3.12.1.3 ADC_B_H

```
#define ADC_B_H 0x26
```

#### 3.12.1.4 ADC_B_L

```
#define ADC_B_L 0x27
```

#### 3.12.1.5 ADC_C_H

```
#define ADC_C_H 0x28
```

#### 3.12.1.6 ADC_C_L

```
#define ADC_C_L 0x29
```

#### 3.12.1.7 command_reg

```
#define command_reg 0xCC
```

Read / Write Registers.

#### 3.12.1.8 current_duration_H

```
#define current_duration_H 0x03
```

#### 3.12.1.9 current_duration_L

```
#define current_duration_L 0x04
```

#### 3.12.1.10 des_pos_a_H

```
#define des_pos_a_H 0x09
```

#### 3.12.1.11 des_pos_a_L

```
#define des_pos_a_L 0x0A
```

#### 3.12.1.12 des_pos_b_H

```
#define des_pos_b_H 0x0B
```

#### 3.12.1.13 des_pos_b_L

```
#define des_pos_b_L 0x0C
```

#### 3.12.1.14 des_speed_a_H

```
#define des_speed_a_H 0x05
```

#### 3.12.1.15 des_speed_a_L

```
#define des_speed_a_L 0x06
```

#### 3.12.1.16 des_speed_b_H

```
#define des_speed_b_H 0x07
```

#### 3.12.1.17 des_speed_b_L

```
#define des_speed_b_L 0x08
```

#### 3.12.1.18 max_current_H

```
#define max_current_H 0x01
```

#### 3.12.1.19 max_current_L

```
#define max_current_L 0x02
```

#### 3.12.1.20 mode_reg

```
#define mode_reg 0x00
```

#### 3.12.1.21 position_a_H

```
#define position_a_H 0x20
```

Read only Registers.

#### 3.12.1.22 position_a_L

```
#define position_a_L 0x21
```

#### 3.12.1.23 position_b_H

```
#define position_b_H 0x22
```

#### 3.12.1.24 position_b_L

```
#define position_b_L 0x23
```

#### 3.12.1.25 READONLY

```
#define READONLY 0x20
```

#### 3.12.1.26 REGMAP_SIZE

```
#define REGMAP_SIZE 0x2F
```

#### 3.12.1.27 remex_state

```
#define remex_state 0x2F
```

#### 3.12.1.28 SLAVE_ADDR

```
#define SLAVE_ADDR 0x48
```

#### 3.12.1.29 switch_states

```
#define switch_states 0x2A
```

#### 3.12.1.30 UNKNOWN_PAR

```
#define UNKNOWN_PAR 0xFF
```

#### 3.12.1.31 UNKNOWN_REG

```
#define UNKNOWN_REG 0xFF
```

### 3.12.2 Enumeration Type Documentation

#### 3.12.2.1 i2c_states

```
enum i2c_states
```

state machine states

**Enumerator**

| | |
|---|---|
| start | |
| reg_set | |
| stop | |
| cmd_byte | |

### 3.12.3 Function Documentation

#### 3.12.3.1 adc_channel_a()

```
void adc_channel_a (
            int current )
```

adc_channel_a is the call back function when the adc has finished it's conversion and has a value.

#### 3.12.3.2 adc_channel_b()

```
void adc_channel_b (
            int current )
```

adc_channel_b is the call back function when the adc has finished it's conversion and has a value.

#### 3.12.3.3 adc_channel_c()

```
void adc_channel_c (
            int current )
```

adc_channel_c is the call back function when the adc has finished it's conversion and has a value.

#### 3.12.3.4 clear_registers()

```
void clear_registers (
            void  )
```

clear_registers is a helper function that sets all the registers in the register map to zero.

#### 3.12.3.5 init()

```
void init (
            void  )
```

init is called once, and initializes the registers and libraries for the board.

#### 3.12.3.6 int2str()

```
void int2str (
            int inval,
            char * str_out )
```

int2str will convert an integer inval and store into string str_out

**Parameters**

| | | |
|---|---|---|
| in | *inval* | is the integer value to be converted to a string |
| out | *str_out* | is the output string where the string value of the integer will be written. |

#### 3.12.3.7 loop()

```
void loop (
            void  )
```

loop is called repeatedly as long as the board is powered.

#### 3.12.3.8 process_cmd()

```
void process_cmd (
            unsigned char cmd )
```

process_cmd is a function that is called when a command byte is received from the master.

**Parameters**

| | | |
|---|---|---|
| in | *cmd* | is the byte received from the master. |

#### 3.12.3.9 receive_cb()

```
void receive_cb (
            const unsigned char in )
```

receive_cb is a call back function that is called when the master writes a byte.

**Parameters**

| | | |
|---|---|---|
| in | *in* | is the byte received from the master. |

#### 3.12.3.10 start_condition_cb()

```
void start_condition_cb (
            void  )
```

start_condition_cb is a callback function for when a i2c start condition is detected from the master.

#### 3.12.3.11 stop_condition_cb()

```
void stop_condition_cb (
            void  )
```

stop_condition_cb is a callback function for when a i2c stop condition is detected from the master.

#### 3.12.3.12 transmit_cb()

```
void transmit_cb (
            unsigned volatile int * out )
```

transmit_cb is a callback function that is called when the master wants to read a byte from the slave.

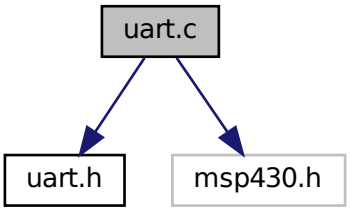
**Parameters**

| | | |
|---|---|---|
| out | *the* | pointer to the transmit buffer to send a byte. |

## 3.13 uart.c File Reference

```
#include "uart.h"
#include <msp430.h>
```

Include dependency graph for uart.c:




### Functions

- void init_UART ()
- void putc (char in)
- void puts (char *in)
- void putsn (char *in, int n)
- __interrupt void USCI_A1_ISR (void)

### 3.13.1 Function Documentation

#### 3.13.1.1 init_UART()

```
void init_UART ( )
```

Initialize UART with baud rate of 9600 on pins 4.2 and 4.3

#### 3.13.1.2 putc()

```
void putc (
            char in )
```

putc will output a single character in to the UART bus

**Parameters**

| | | |
|---|---|---|
| in | *in* | is the character to send to the UART bus. |

#### 3.13.1.3 puts()

```
void puts (
            char * in )
```

puts will output a string of characters to the UART bus

**Parameters**

| | | |
|---|---|---|
| in | *in* | is a null terminated string of characters to send to the UART bus. |

#### 3.13.1.4 putsn()

```
void putsn (
            char * in,
            int len )
```

putsn will output a string of characters to the UART bus

**Parameters**

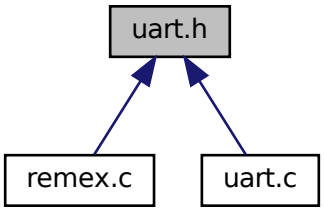
| | | |
|---|---|---|
| in | *in* | is a string of characters to send to the UART bus. |
| | *[len]* | is the length of the string that will be sent over the UART bus. |

#### 3.13.1.5 USCI_A1_ISR()

```
__interrupt void USCI_A1_ISR (
              void  )
```

## 3.14 uart.h File Reference

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




### Functions

- void init_UART ()
- void putc (char in)
- void puts (char *in)
- void putsn (char *in, int len)

### 3.14.1 Function Documentation

#### 3.14.1.1 init_UART()

```
void init_UART ( )
```

Initialize UART with baud rate of 9600 on pins 4.2 and 4.3

#### 3.14.1.2 putc()

```
void putc (
              char in )
```

putc will output a single character in to the UART bus

**Parameters**

| | | |
|---|---|---|
| in | *in* | is the character to send to the UART bus. |

#### 3.14.1.3 puts()

```
void puts (
            char * in )
```

puts will output a string of characters to the UART bus

**Parameters**

| | | |
|---|---|---|
| in | *in* | is a null terminated string of characters to send to the UART bus. |

#### 3.14.1.4 putsn()

```
void putsn (
            char * in,
            int len )
```

putsn will output a string of characters to the UART bus

**Parameters**

| | | |
|---|---|---|
| in | *in* | is a string of characters to send to the UART bus. |
| | *[len]* | is the length of the string that will be sent over the UART bus. |

# Index