iom361_r2_api

Generated by Doxygen 1.12.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 ioreg_t Struct Reference

`#include <iom361_r2.h>`

Data Fields

- uint32_t **switches**
- uint32_t **leds**
- uint32_t **rgbled**
- uint32_t **temperature**
- uint32_t **humidity**
- uint32_t **reserved_1**
- uint32_t **reserved_2**
- uint32_t **reserved_3**

3.1.1 Detailed Description

3.1.1.1 Register formats:

o switches[31:0]: One bit per switch starting w/ bit[0] (rightmost, LSB). Number of switches is specified in iom361_initialize(). Max of 32 switches. A switch is on for every bit that is 1

o leds[31:0]: One bit per LED starting with bit[0] (rightmost, LSB. Number of LEDS is specified in iom361_initialize(). Max of 32 LEDS. An LED is on (lit) for every bit that is 1. Contents of LED register is displayed on every write to the register. Format is 'o' for every lit LED. '_' for every dark LED.

o rgb_led[31:0]: Control register for RGB LED. Formatted as follows:

```
- bits[31:31]:  Enable - true if RGB outputs are enabled
- bits[30:24]: *reserved*
- bits[23:16]: 8-bit duty cycle for Red segment
- bits[15:8]:  8-bit duty cycle for Green segment
- bits[7:0]:   8-bit duty cycle for Blue segment
```

o temperature[31:0]: Temperature in degrees C. iom361 emulates an AHT0 temperature/humidity sensor. Temperature is 24-bit number that can be converted to a float with the following formula: Temp(degrees C) = (ST/2**20) * 200 - 50 where ST is the value in the register.

o humidity[31:0]: Relative humidity in %. iom361 emulates an AHT0 temperature/humidity sensor. Humidity is 24-bit number that can be converted to a float with the following formula: Rel Humidity(%) = (SRH/2**20) * 100 where SRH is the value in the register.

o reserved_1[31:0]: Reserved for future use. Can be written and read

o reserved_2[31:0]: Reserved for future use. Can be written and read

o reserved_3[31:0]: Reserved for future use. Can be written and read

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

- iom361_r2.h

Chapter 4

File Documentation

4.1 float_rndm.h

```
00001
00008  #ifndef _FLOAT_RNDM_H
00009  #define _FLOAT_RNDM_H
00010
00011  // function prototypes
00012  double positive_float_rand_in_range(double pos_a, double pos_b);
00013  double float_rand_in_range(double a, double b);
00014
00015  #endif
```

4.2 iom361_r2.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "float_rndm.h"
#include "iom361_r2.h"
```

Functions

- uint32_t * iom361_initialize (int num_switches, int num_leds, int *rtn_code)
- uint32_t iom361_readReg (uint32_t *base, uint32_t offset, int *rtn_code)
- uint32_t iom361_writeReg (uint32_t *base, int offset, uint32_t value, int *rtn_code)
- void _iom361_setSwitches (uint32_t value)
- void _iom361_setSensor1 (float new_temp, float new_humid)
- void _iom361_setSensor1_rndm (float temp_low, float temp_hi, float humid_low, float humid_hi)

4.2.1 Detailed Description

iom361.c - Source file for ECE 361 I/O module emulator

Author

: Roy Kravitz (roy.kravitz@pdx.edu)

Date

: 05-Nov-2033

Version

: 2.0

This is the source code for the ECE 361 I/O module emulation. The I/O module emulates a memory-mapped I/O system with a number of "typical" peripheral registers.

This version uses an array of uint3t_ instead of a struct. More accurate way to model memory mapped I/O registers

4.2.2 Function Documentation

4.2.2.1 _iom361_setSensor1()

```
void _iom361_setSensor1 (
             float new_temp,
             float new_humid)
```

_iom361_setSensor1 () - sets the temperature and humidity for Sensor 1

Used to set the temperature and humidity for the emulated AHT20 sensor. The sensor returns 20-bit unsigned values for the temperature and humidity. Fortunately you can set temp to 0 - 100 degrees C and humidity to 0 - 99% RH as floats and the function will calculate the value written to the register

Parameters

new_temp	new temperature value in degrees C. Specified as a float. conversion to register value is done in the function
new_humid	new humidity value Specified as float. conversion to a register value is done in the function

4.2.2.2 _iom361_setSensor1_rndm()

```
void _iom361_setSensor1_rndm (
             float temp_low,
             float temp_hi,
             float humid_low,
             float humid_hi)
```

_iom361_setSensor1_rndm () - sets the temperature and humidity for Sensor 1

Used to set the temperature and humidity for the emulated AHT20 sensor. The sensor returns 20-bit unsigned values for the temperature and humidity. This function is able to set the temperature and humidity values to random numbers within the specified range

Parameters

temp_low	low temperature for range in degrees C. Specified as a float. conversion to register value is done in the function
temp_hi	high temperature for range in degrees C. Specified as a float. conversion to register value is done in the function
humid_low	low relative humidity in range Specified as float. conversion to a register value is done in the function
humid_hi	high relative humidity in range Specified as float. conversion to a register value is done in the function

Note

Uses srand(time(NULL) to initialize rand(). This is done when iom361 is initialized.

4.2.2.3 _iom361_setSwitches()

```
void _iom361_setSwitches (
            uint32_t value)
```

_iom361_setSwitches () - sets the value of the switch register

Used to set the value of the switch I/O register location. The driver for the emulator keeps track of the base address so it doesn't need to be a parameter

Parameters

value	value for the switch register. Not all 32-bits may be switches. The number of switches is set when iom361 is initialized.

4.2.2.4 iom361_initialize()

```
uint32_t * iom361_initialize (
            int num_switches,
            int num_leds,
            int * rtn_code)
```

iom361_initialize() - initializes the ECE 361 I/O module

Initializes the I/O module emulator. Function returns a pointer to the base of the I/O register block. Returns NULL if the function fails. Updates rtn_code if the function succeeds (0) or fails ($>$ 0)

Parameters

num_switches	the number of switches (up to 32) in iom361
num_leds	the number of leds (up to 32) in iom361
$*$*rtn_code*	a pointer to the return code. Will be 0 for success, a different number if the call fails.

Returns

a pointer to the base of the I/O register block. NULL if function fails

4.2.2.5 iom361_readReg()

```
uint32_t iom361_readReg (
            uint32_t * base,
            uint32_t offset,
            int * rtn_code)
```

iom361_readReg() - returns the value of an I/O register

reads/returns the value of the I/O register at base + offset. Updates rtn_code if the function succeeds (0) or fails (> 0)

Parameters

base	address of the base of the I/O memory block
offset	offset into I/O memory block. All registers are 32-bits wide
**rtn_code*	a pointer to the return code. Will be 0 for success, a different number if the call fails.

Returns

the contents of the specified I/O register

4.2.2.6 iom361_writeReg()

```
uint32_t iom361_writeReg (
            uint32_t * base,
            int offset,
            uint32_t value,
            int * rtn_code)
```

iom361_writeReg() - writes a 32-bit value to an I/O register

writes a new value into the I/O register at base + offset. Updates rtn_code if the function succeeds (0) or fails (> 0)

Parameters

base	address of the base of the I/O memory block
offset	offset into I/O memory block. All registers are 32-bits wide
**rtn_code*	a pointer to the return code. Will be 0 for success, a different number if the call fails.

Returns

the contents of the specified I/O register (does a read)

4.3 iom361_r2.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
```

Data Structures

- struct ioreg_t

Macros

- #define **NUM_IO_REGS** 8

Typedefs

- typedef struct ioreg_t * **ioreg_ptr_t**

Enumerations

- enum {
  SWITCHES_REG = 0x00 , **LEDS_REG** = 0x04 , **RGB_LED_REG** = 0x08 , **TEMP_REG** = 0x0C ,
  HUMID_REG = 0x10 , **RSVD1_REG** = 0x14 , **RSVD2_REG** = 0x18 , **RSVD3_REG** = 0x1C }

Functions

- uint32_t * iom361_initialize (int num_switches, int num_leds, int *rtn_code)
- uint32_t iom361_readReg (uint32_t *base, uint32_t offset, int *rtn_code)
- uint32_t iom361_writeReg (uint32_t *base, int offset, uint32_t value, int *rtn_code)
- void _iom361_setSwitches (uint32_t value)
- void _iom361_setSensor1 (float new_temp, float new_humid)
- void _iom361_setSensor1_rndm (float temp_low, float temp_hi, float humid_low, float humid_hi)

4.3.1 Detailed Description

iom361_r2.h - Header file for ECE 361 I/O module emulator

Author

: Roy Kravitz (roy.kravitz@pdx.edu)

Date

: 05-Nov-2023

Version

: 2.0

This is the header file for the ECE 361 I/O module emulation. The I/O module emulates a memory-mapped I/O system with a number of "typical" peripheral registers.

4.3.2 Function Documentation

4.3.2.1 _iom361_setSensor1()

```
void _iom361_setSensor1 (
            float new_temp,
            float new_humid)
```

_iom361_setSensor1 () - sets the temperature and humidity for Sensor 1

Used to set the temperature and humidity for the emulated AHT20 sensor. The sensor returns 20-bit unsigned values for the temperature and humidity. Fortunately you can set temp to 0 - 100 degrees C and humidity to 0 - 99% RH as floats and the function will calculate the value written to the register

Parameters

new_temp	new temperature value in degrees C. Specified as a float. conversion to register value is done in the function
new_humid	new humidity value Specified as float. conversion to a register value is done in the function

4.3.2.2 _iom361_setSensor1_rndm()

```
void _iom361_setSensor1_rndm (
             float temp_low,
             float temp_hi,
             float humid_low,
             float humid_hi)
```

_iom361_setSensor1_rndm () - sets the temperature and humidity for Sensor 1

Used to set the temperature and humidity for the emulated AHT20 sensor. The sensor returns 20-bit unsigned values for the temperature and humidity. This function is able to set the temperature and humidity values to random numbers within the specified range

Parameters

temp_low	low temperature for range in degrees C. Specified as a float. conversion to register value is done in the function
temp_hi	high temperature for range in degrees C. Specified as a float. conversion to register value is done in the function
humid_low	low relative humidity in range Specified as float. conversion to a register value is done in the function
humid_hi	high relative humidity in range Specified as float. conversion to a register value is done in the function

Note

Uses srand(time(NULL) to initialize rand(). This is done when iom361 is initialized.

4.3.2.3 _iom361_setSwitches()

```
void _iom361_setSwitches (
             uint32_t value)
```

_iom361_setSwitches () - sets the value of the switch register

Used to set the value of the switch I/O register location. The driver for the emulator keeps track of the base address so it doesn't need to be a parameter

Parameters

value	value for the switch register. Not all 32-bits may be switches. The number of switches is set when iom361 is initialized.

4.3.2.4 iom361_initialize()

```
uint32_t * iom361_initialize (
            int num_switches,
            int num_leds,
            int * rtn_code)
```

iom361_initialize() - initializes the ECE 361 I/O module

Initializes the I/O module emulator. Function returns a pointer to the base of the I/O register block. Returns NULL if the function fails. Updates rtn_code if the function succeeds (0) or fails ($>$ 0)

Parameters

num_switches	the number of switches (up to 32) in iom361
num_leds	the number of leds (up to 32) in iom361
$*$*rtn_code*	a pointer to the return code. Will be 0 for success, a different number if the call fails.

Returns

a pointer to the base of the I/O register block. NULL if function fails

4.3.2.5 iom361_readReg()

```
uint32_t iom361_readReg (
            uint32_t * base,
            uint32_t offset,
            int * rtn_code)
```

iom361_readReg() - returns the value of an I/O register

reads/returns the value of the I/O register at base + offset. Updates rtn_code if the function succeeds (0) or fails ($>$ 0)

Parameters

base	address of the base of the I/O memory block
offset	offset into I/O memory block. All registers are 32-bits wide
$*$*rtn_code*	a pointer to the return code. Will be 0 for success, a different number if the call fails.

Returns

the contents of the specified I/O register

4.3.2.6 iom361_writeReg()

```
uint32_t iom361_writeReg (
            uint32_t * base,
            int offset,
            uint32_t value,
            int * rtn_code)
```

iom361_writeReg() - writes a 32-bit value to an I/O register

writes a new value into the I/O register at base + offset. Updates rtn_code if the function succeeds (0) or fails ($>$ 0)

Parameters

base	address of the base of the I/O memory block
offset	offset into I/O memory block. All registers are 32-bits wide
**rtn_code*	a pointer to the return code. Will be 0 for success, a different number if the call fails.

Returns

the contents of the specified I/O register (does a read)

4.4 iom361_r2.h

Go to the documentation of this file.

```
00001
00056  #ifndef _IOM361_H
00057  #define _IOM361_H
00058
00059  #include <stdint.h>
00060  #include <stdbool.h>
00061
00062  // define the I/O register map
00063  typedef struct {
00064      uint32_t   switches;
00065      uint32_t   leds;
00066      uint32_t   rgbled;
00067      uint32_t   temperature;
00068      uint32_t   humidity;
00069      uint32_t   reserved_1;
00070      uint32_t   reserved_2;
00071      uint32_t   reserved_3;
00072  } ioreg_t, *ioreg_ptr_t;
00073
00074  // typedefs and enums
00075  enum {
00076      SWITCHES_REG    = 0x00,
00077      LEDS_REG        = 0x04,
00078      RGB_LED_REG     = 0x08,
00079      TEMP_REG        = 0x0C,
00080      HUMID_REG       = 0x10,
00081      RSVD1_REG       = 0x14,
00082      RSVD2_REG       = 0x18,
00083      RSVD3_REG       = 0x1C
00084  };
00085
00086  // define constants
00087   #define NUM_IO_REGS   8       // There are 8 IO registers in the I/O map
00088
00089  /*
00090   * API functions.  These are low level functions that read/write the
00091   * I/O registers directly.  You can use them to build higher level
00092   * functionality in your own code, but it doesn't get much more basic
00093   * than this.
00094   */
00095
00110 uint32_t* iom361_initialize(int num_switches, int num_leds, int* rtn_code);
00111
00112
00125 uint32_t iom361_readReg(uint32_t* base, uint32_t offset, int* rtn_code);
00126
00127
00141 uint32_t iom361_writeReg(uint32_t* base, int offset, uint32_t value, int* rtn_code);
00142
00143
00144 /* These functions are used for testing.  They set a specific register to a value.  For
00145  * example, there is a function to write a new value to the switch register.  The same
00146  * for the temp/humidity sensor.  I added these functions because we are emulating
00147  * memory-mapped I/O...there is no "real" hardware at the other end.
00148  *
00149  * The function names start w/ a _ to differentiate them from what would normally be
00150  * the API.
00151  */
00152
00163 void _iom361_setSwitches(uint32_t value);
00164
00165
```

```
00179 void _iom361_setSensor1(float new_temp, float new_humid);
00180
00200 void _iom361_setSensor1_rndm(float temp_low, float temp_hi,
00201     float humid_low, float humid_hi);
00202
00203 #endif
00204
00205
00206
00207
00208
00209
```

4.5 test_iom361_r2.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <errno.h>
#include "iom361_r2.h"
```

Macros

- #define **TEMP_RANGE_LOW** 42.0
- #define **TEMP_RANGE_HI** 52.0
- #define **HUMID_RANGE_LOW** 72.6
- #define **HUMID_RANGE_HI** 87.3

Functions

- int **main** ()

Variables

- uint32_t * **io_base**

4.5.1 Detailed Description

test_iom361_r2 - verifies the functionality of the ECE 361 I/O emulator

Author

: Roy Kravitz (roy.kravitz@pdx.edu)

Date

: 05-Nov-2033

Version

: 2.0

Test program for the ECE 361 I/O emulator. Fairly basic:

```
- initializes the I/O emulator
- reads all of the registers and display initial values
- changes switches and writes them to LEDs
- changes temp and humidity and displays new values
```

Index