INTERNET OF THINGS
DEPARTMENT OF COMPUTER ENGINEERING
SANTA CLARA UNIVERSITY, CALIFORNIA
BEHNAM DEZFOULI, PHD

Internet of Things

Lab 6

Advanced RTOS I/O Features

Inter-Integrated Circuit (I^2C)

I2C

- The device contains two I2C masters called `WICED_I2C_1` and `WICED_I2C_2`
- The OLED display and the PSoC on the shield connect to `WICED_I2C_2`

❑ **Example: We are interested in implementing the following application:**

- When the button on the base board is pressed, send a character over the I2C bus to the shield board
- This is used by the processor on the shield to toggle through the four LEDs on the shield

I2C

- As with other peripherals, you need to initialize the block using the initialization function
- However, in this case, the parameter you pass it is not the name of the block, but a structure of the type `wiced_i2c_device_t`
- That structure contains information about the I2C slave that you are going to communicate with

- We need to use the following facilities:

```
wiced_result_t wiced_i2c_init( const wiced_i2c_device_t* device );
```

Initializes an I2C interface: Prepares an I2C hardware interface for communication as a **master**

param [in] `device`: **The device for which the i2c port should be initialized**
return `WICED_SUCCESS`: on success
return `WICED_ERROR`: if an error occurred during initialization

I2C

- For example, the following could be used to initialize I2C block 2 (i.e., `WICED_I2C_2`) to connect to a slave at address 0x08 with a speed of 100 kHz (standard speed)

```
const wiced_i2c_device_t i2cDevice = {
        .port = WICED_I2C_2,
        .address = I2C_ADDRESS,
        .address_width = I2C_ADDRESS_WIDTH_7BIT,
        .speed_mode = I2C_STANDARD_SPEED_MODE
};
```

I2C

```
wiced_result_t wiced_i2c_init_tx_message
        ( wiced_i2c_message_t* message, const void* tx_buffer,
              uint16_t tx_buffer_length,
                     uint16_t retries, wiced_bool_t disable_dma );
```

Initialize the `wiced_i2c_message_t` structure for i2c tx transaction

param[out] **message:** Pointer to a message structure, this should be a valid pointer

param[in] **tx_buffer:** Pointer to a tx buffer that is already allocated

param[in] **tx_buffer_length:** Number of bytes to transmit

param[in] **retries:** The number of times to attempt send a message in case it can't not be sent

param[in] **disable_dma:** If true, disables the dma for current tx transaction. You may find it useful to switch off dma for short tx messages.

return **WICED_SUCCESS:** message structure was initialized properly

return **WICED_BADARG:** one of the arguments is given incorrectly

I2C

- There is a dedicated read function called `wiced_i2c_read` and a dedicated write function called `wiced_i2c_write`

- There is also a function called `wiced_i2c_transfer` which can do a read, a write, or both (see below)

```
wiced_result_t wiced_i2c_transfer
        ( const wiced_i2c_device_t* device,
               wiced_i2c_message_t* message,
                      uint16_t number_of_messages );
```

Transmits and/or receives data over an I2C interface

```
    param[in]  device: The i2c device to communicate with
    param[in]  message: A pointer to a message (or an array of
    messages) to be transmitted/received
    param[in]  number_of_messages : The number of messages to
    transfer. [1 .. N] messages
    return     WICED_SUCCESS: on success.
    return     WICED_ERROR: if an error occurred during message
    transfer
```

I2C

- **I2C slave address = 0x42**
- Standard Speed (100kHz)
- EZI2C register access
  - The first byte written is the register offset
  - All reads start at the previous write offset
  - The register map is as follows:

Offset	Description	Detail
0x00–0x03	DAC value	This value is used to set the DAC (Digital to Analog Converter) output voltage
0x04	**LED Values**	**4 least significant bits control CSLED3-CSLED0**
0x05	**LED Control Register**	**Set bit 1 in this register to allow the LED Values register to control the LEDs instead of the CapSense (CS) buttons**
0x06	Button Status	Captures status of the CapSense buttons, Proximity sensor, and Mechanical buttons. The bits are: Unused, MB1, MB0, Prox, CS3, CS2, CS1, CS0
0x07–0x0A	Temperature	Floating point temperature measurement from the thermistor
0x0B–0x0E	Humidity	Floating point humidity measurement
0x0F–0x12	Ambient Light	Floating point ambient light measurement
0x13–0x16	Potentiometer	Floating point potentiometer voltage measurement

Software Development using WICED Studio

Example: I2C-Write (`i2cwrite`)
Part: 1/3

```
#include "wiced.h"

#define I2C_ADDRESS  (0x42) //I2C slave address
#define RETRIES      (1)
#define DISABLE_DMA  (WICED_TRUE)
#define NUM_MESSAGES (1)

/* I2C register locations */
#define CONTROL_REG (0x05) //offset: LED control
#define LED_REG     (0x04) //offset: LED values

volatile wiced_bool_t buttonPress = WICED_FALSE;

/* Interrupt service routine for the button */
void button_isr(void* arg)
{
    buttonPress = WICED_TRUE;
}

/* Main application */
void application_start( )
{
    wiced_init();/* Initialize the WICED device */

    wiced_gpio_input_irq_enable(WICED_SH_MB1, IRQ_TRIGGER_FALLING_EDGE,
                                button_isr, NULL); /* Setup interrupt */
```

See the register map table

Software Development using WICED Studio

Example: : I2C-Write (`i2cwrite`)
Part: 2/3

```
/* Main application */
void application_start( )
{
    wiced_init();  /* Initialize the WICED device */

    wiced_gpio_input_irq_enable(WICED_SH_MB1, IRQ_TRIGGER_FALLING_EDGE,
                        button_isr, NULL); /* Setup interrupt */

    /* Setup I2C master */
    const wiced_i2c_device_t i2cDevice = {
        .port = WICED_I2C_2, //The slave address is 0x42
        .address = I2C_ADDRESS, //The address of the device on the I2C bus
        .address_width = I2C_ADDRESS_WIDTH_7BIT,
        .speed_mode = I2C_STANDARD_SPEED_MODE
    };

    wiced_i2c_init(&i2cDevice); //Initializes an I2C interface

    /* Setup transmit buffer and message */
    /* We will always write an offset and then a single value,
     * so we need 2 bytes in the buffer */
    uint8_t tx_buffer[] = {0, 0};
    wiced_i2c_message_t msg;
    wiced_i2c_init_tx_message(&msg, tx_buffer, sizeof(tx_buffer),
                                        RETRIES, DISABLE_DMA);
```

Set up I2C master

Prepare I2C message:

Now `msg` reflects the data in `tx_buffer`

Software Development using WICED Studio

Example: : I2C-Write (`i2cwrite`)
Part: 3/3

```
 /* Write a value of 0x01 to the control register to enable control of the
  * CapSense LEDs over I2C */
    tx_buffer[0] = CONTROL_REG;
    tx_buffer[1] = 0x01;
    wiced_i2c_transfer(&i2cDevice, &msg, NUM_MESSAGES);

    tx_buffer[0] = LED_REG; /* Set offset for the LED register (0x04) */

    while ( 1 )
    {
         if(buttonPress)
         {
                 tx_buffer[1] = tx_buffer[1] << 1; /* Shift to the next LED */
                 if (tx_buffer[1] > 0x08) /* Reset after turning on LED3 */
                 {
                         tx_buffer[1] = 0x01;
                 }
                 /* Send new I2C data */
                 wiced_i2c_transfer(&i2cDevice, &msg, NUM_MESSAGES);

                 buttonPress = WICED_FALSE; // Reset flag for next button press
         }
    }
}
```

We set bit 1 in LED control register 0x05 to allow the LED values register (i.e., 0x04) to control the LEDs

Modify and send I2C message

Deliverable

- **Task 6-1. Develop the following program:**
  - The four LEDs on the shield blink sequentially from left to right
  - The duration of each LED's turn on time is 200ms
  - When WICED_SH_MB1 is pressed, the direction of LED blinking is changed

Software Development using WICED Studio

I2C

❑ **Example:** We are interested in implementing the following application:

- **When the button on the base board is pressed, I2C is used to read the temperature, humidity, light, and PWM values from the analog co-processor on the shield board**
- **The values are printed to the UART**

- We need to set the offset to 0x07 to read the temperature
- We can do this just once and it will stay set for all future reads
- With an offset of **0x07** (see next table) you can read 16 bytes to get the **temperature**, **humidity**, ambient **light**, and **potentiometer** values (4 bytes each)

I2C

- **I2C slave address = 0x42**
- Standard Speed (100kHz)
- EZI2C register access
  - The first byte written is the register offset
  - All reads start at the previous write offset
  - The register map is as follows:

Offset	Description	Detail
0x00–0x03	DAC value	This value is used to set the DAC (Digital to Analog Converter) output voltage
0x04	LED Values	4 least significant bits control CSLED3-CSLED0
0x05	LED Control Register	Set bit 1 in this register to allow the LED Values register to control the LEDs instead of the CapSense (CS) buttons
0x06	Button Status	Captures status of the CapSense buttons, Proximity sensor, and Mechanical buttons. The bits are: Unused, MB1, MB0, Prox, CS3, CS2, CS1, CS0
0x07–0x0A	**Temperature**	**Floating point temperature measurement from the thermistor**
0x0B–0x0E	**Humidity**	**Floating point humidity measurement**
0x0F–0x12	**Ambient Light**	**Floating point ambient light measurement**
0x13–0x16	**Potentiometer**	**Floating point potentiometer voltage measurement**

Software Development using WICED Studio

Example: I2C-Read (i2cread)
Part: 1/3

```
#include "wiced.h"

#define I2C_ADDRESS (0x42)
#define RETRIES (1)
#define DISABLE_DMA (WICED_TRUE)
#define NUM_MESSAGES (1)

#define TEMPERATURE_REG  0x07

volatile wiced_bool_t buttonPress = WICED_FALSE;


/* Interrupt service routine for the button */
void button_isr(void* arg)
{
        buttonPress = WICED_TRUE;
}


/* Main application */
void application_start( )
{
    wiced_init();/* Initialize the WICED device */

    wiced_gpio_input_irq_enable(WICED_SH_MB1, IRQ_TRIGGER_FALLING_EDGE,
                                   button_isr, NULL); /* Setup interrupt */
```

See the register map table (this is the offset of temperature register)

Software Development using WICED Studio

Example: I2C-Read (`i2cread`)
Part: 2/3

```
    /* Setup I2C master */
     const wiced_i2c_device_t i2cDevice = {
          .port = WICED_I2C_2,
          .address = I2C_ADDRESS,
          .address_width = I2C_ADDRESS_WIDTH_7BIT,
          .speed_mode = I2C_STANDARD_SPEED_MODE
     };

     wiced_i2c_init(&i2cDevice);

     /* Tx buffer is used to set the offset */
     uint8_t tx_buffer[] = {TEMPERATURE_REG};
     wiced_i2c_message_t setOffset;
     wiced_i2c_init_tx_message(&setOffset, tx_buffer,
                                         sizeof(tx_buffer), RETRIES, DISABLE_DMA);



     /* Initialize offset */
     wiced_i2c_transfer(&i2cDevice, &setOffset, NUM_MESSAGES);
```

Preparing tx message for setting offset
Now `setOffset` reflects the data in `tx_buffer`

Setting offset to 0x07

Software Development using WICED Studio

```
   /* Rx buffer is used to get temperature, humidity, light, and POT data –
    * 4 bytes each */
    struct {
        float temp;
        float humidity;
        float light;
        float pot;
    } rx_buffer;

    wiced_i2c_message_t msg;
    wiced_i2c_init_rx_message(&msg, &rx_buffer, sizeof(rx_buffer), RETRIES,
                                                                   DISABLE_DMA);

    while ( 1 ) {
        if(buttonPress) {
                /* Get new data from I2C */
                wiced_i2c_transfer(&i2cDevice, &msg, NUM_MESSAGES);

                WPRINT_APP_INFO(("Temperature: %.1f\t Humidity: %.1f\t Light:
                          %.1f\t POT: %.1f\n", rx_buffer.temp
                                  rx_buffer.humidity, rx_buffer.light,
                                                           rx_buffer.pot));

                /* Reset flag for next button press */
                buttonPress = WICED_FALSE;
        }
    }
}
```

```
Output: Temperature: 28.5  Humidity: 48.9   Light: 103.0   POT: 1.0
```

Preparing rx message

Now `msg` reflects the data in `rx_buffer`

Here the `wiced_i2c_transfer` function is used to read data

Example: I2C-Read (`i2cread`)
Part: 3/3

PWM

PWM

❑ **Example:** We are interested in implementing an application that enables us to control the brightness of LED

- We need to configure a PWM to drive `WICED_SH_LED1` on the shield board instead of using the GPIO functions

- As `WICED_SH_LED1` is by default initialized to be controlled by GPIO driver, we need to call `wiced_gpio_deinit(…)` so that the PWM can drive the pin

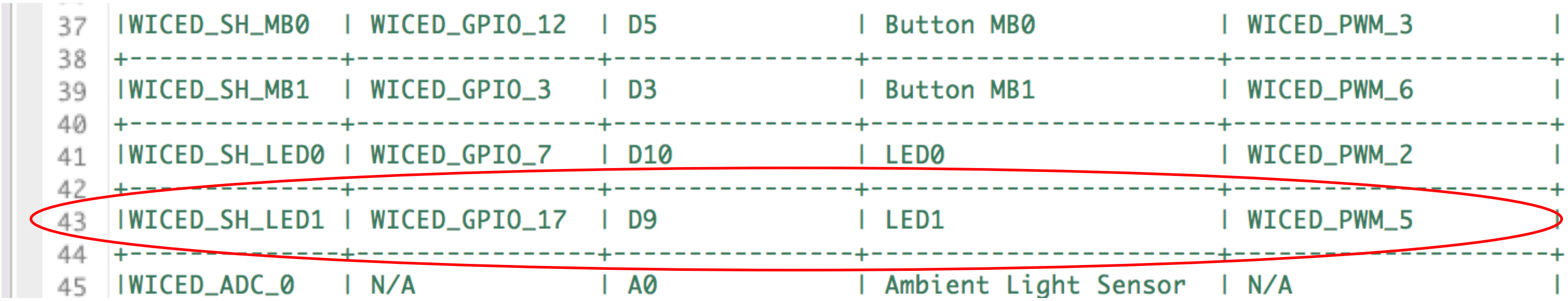
```
wiced_result_t wiced_gpio_deinit( wiced_gpio_t gpio );
```

De-initializes a GPIO pin

```
 *
 * @param[in] gpio    : The gpio pin which should be de-initialized
 *
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred with any step
 */
```

PWM

- `WICED_SH_LED1` is connected to `WICED_GPIO_17` so you need to find out which PWM is connected to that pin (look in the platform files, i.e., `platform.h`)



```
|WICED_SH_MB0   | WICED_GPIO_12   | D5               | Button MB0            | WICED_PWM_3          |
+---------------+-----------------+------------------+-----------------------+----------------------+
|WICED_SH_MB1   | WICED_GPIO_3    | D3               | Button MB1            | WICED_PWM_6          |
+---------------+-----------------+------------------+-----------------------+----------------------+
|WICED_SH_LED0  | WICED_GPIO_7    | D10              | LED0                  | WICED_PWM_2          |
+---------------+-----------------+------------------+-----------------------+----------------------+
|WICED_SH_LED1  | WICED_GPIO_17   | D9               | LED1                  | WICED_PWM_5          |
+---------------+-----------------+------------------+-----------------------+----------------------+
|WICED_ADC_0    | N/A             | A0               | Ambient Light Sensor  | N/A                  |
```

PWM

Initializes a PWM pin

```
 * Does not start the PWM output (use @ref wiced_pwm_start).
 * @param[in] pwm        : The PWM interface which should be initialized
 * @param[in] frequency  : Output signal frequency in Hertz
 * @param[in] duty_cycle : Duty cycle of signal as a floating-point percentage
(0.0 to 100.0)
 *
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred with any step
 */
wiced_result_t wiced_pwm_init( wiced_pwm_t pwm, uint32_t frequency,
                                                          float duty_cycle );
```

Starts PWM output on a PWM interface

```
 * @param[in] pwm        : The PWM interface which should be started
 *
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred with any step
 */
wiced_result_t wiced_pwm_start( wiced_pwm_t pwm );
```

Software Development using WICED Studio

Example: PWM-LED (`pwm_led`)
Part: 1/1

```c
#include "wiced.h"

#define  PWM_PIN  WICED_PWM_5

void application_start( )
{
    float duty_cycle = 0.0;

    wiced_init();/* Initialize the WICED device */

    // Need to de-init the GPIO if it is already set to drive the LED
    wiced_gpio_deinit(WICED_SH_LED1);

    while ( 1 )
    {
        wiced_pwm_init(PWM_PIN, 1000, duty_cycle);
        wiced_pwm_start(PWM_PIN);
        duty_cycle += 1.0;

        if(duty_cycle > 100.0)
        {
            duty_cycle = 0.0;
        }
        wiced_rtos_delay_milliseconds( 20 );
    }
}
```

Every 20ms we change the duty cycle of the 1KHz pulse generated

Deliverable

- Task 6-2. Develop the following code:
    - The brightness of an LED is controlled by light intensity
    - Move your hand above the shield board and show how lighr intensity is changed