Department Of Electronic and Telecommunication Engineering-University of Moratuwa

EN2160-Electronic Design Realization
Design Documentation-Exercise Mate

Discussion Group O

Group 39

Wickramasinghe.M.M.M-210707L
Rajapaksha.N.N-210504L

Contents

1 General Information

1.1 Marketing Needs

The inspiration for Exercise Mate, a comprehensive exercise tracking and guidance system, stemmed from discussions with Professor Jayasinghe. The system aims to assist users in performing exercises correctly and tracking their progress efficiently.

1.2 Existing Product Analysis

Solutions from industry leaders like Fitbit, Garmin, and Apple were analyzed through YouTube videos, product brochures, and academic papers. Key features such as heart rate monitoring, GPS tracking, and personalized workout plans were highlighted.

Figure 1: Existing Products

1.3 User Profile

Exercise Mate targets a diverse range of users including fitness enthusiasts, athletes, and individuals seeking a guided exercise regimen. Additionally, it caters to professionals from the medical industry, gymnasiums, the sports industry, and corporate wellness programs. This comprehensive approach ensures that Exercise Mate can provide tailored solutions and benefits to a wide array of users, promoting health and fitness across various sectors.

1.3.1 Need List

- Accurate counting of different exercises.
- Simple user interface to select exercise type and start/stop counting.
- Lightweight, portable design that doesn't interfere with workouts.
- Long battery life with less weight to last through extended workout sessions.
- Sweat/water resistance for use during intense training.
- Affordability for mass market appeal.

1.4 Market Segment

The market segment for Exercise Mate includes fitness enthusiasts, athletes, and individuals seeking a guided exercise regimen, as well as professionals from the medical industry, gymnasiums, the sports industry, and corporate wellness programs. This diverse segment encompasses both individual consumers and institutional users, ranging from those pursuing personal fitness goals to organizations aiming to enhance employee wellness and sports performance. By addressing the needs of these varied groups, Exercise Mate positions itself as a versatile solution in the health and fitness market.

1.5 Specifications

The specifications of the device are as follows:

- **Control Capabilities:** The device can control various connected exercise equipment and provides guidance through visual and audio cues. It offers real-time feedback on form and movement during exercises.
- **Automatic Functionality:** When the automatic function is enabled, Exercise Mate operates flawlessly, executing predefined workout routines with precision and efficiency. It adjusts exercises based on user performance and preferences.
- **Platform Compatibility:** The device is designed to be compatible with anyplace to be used.
- **User Interface:** Exercise Mate features an intuitive user interface with a separate display module.
- **Durability:** Built with high-quality, durable materials, Exercise Mate is designed to withstand intense workout environments and long-term use, ensuring reliability and longevity.

2 Electronics Design

2.1 Conceptual Block Diagram

The selected design features a hand-worn device that tracks motion using accelerometers and a gyroscope sensor. First, the user selects the exercise type from the available options using the control button. As the exercise is performed, data from the accelerometer and gyroscope is processed by the microcontroller to identify the motion. This allows for accurate tracking of the exercise count for the selected exercise type. The real-time count is transmitted to a central device via an RF antenna. The stationary central device receives the data and displays the count of the selected exercise type in real-time. This approach ensures that exercises can be effectively tracked in our chosen design.

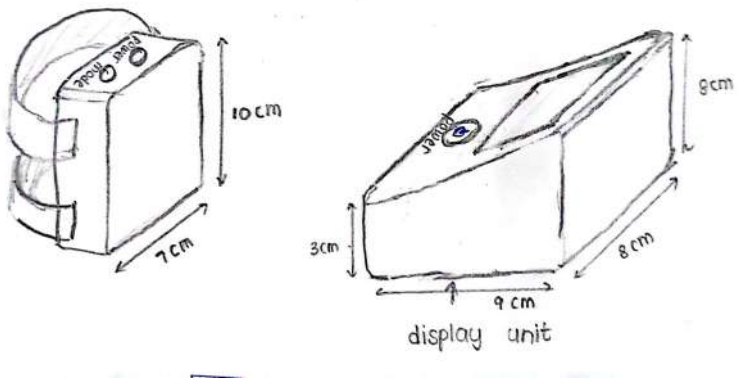


Figure 2: Wearable devices with separate display - Enclosure

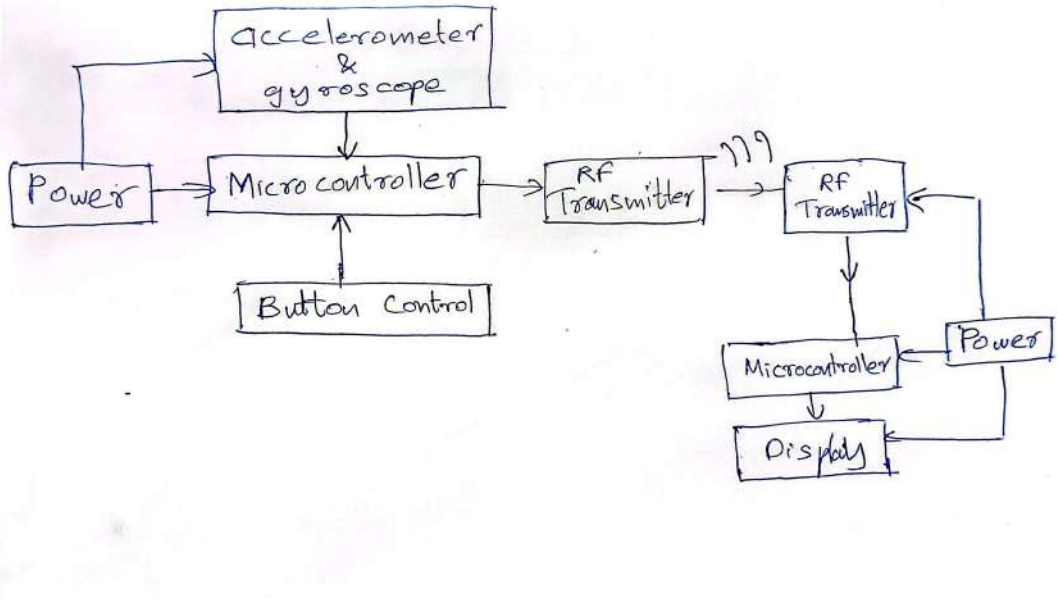


Figure 3: Wearable devices with separate display - Function Block Diagram

2.2 Sub-Assembly Design

- **Power supply**: The power sub-assembly system provides the necessary electrical power to all components of the device.
- **Accelerometer and Gyroscope**: These sensors track movement and orientation, providing crucial data for motion detection.
- **Microcontroller Unit**: This is the central processing unit of the device, responsible for data acquisition, processing, and control tasks, coordinating the operation of all other components.
- **Display Unit**: The display unit presents information to the user, allowing for real-time exercises tracking.
- **RF Antenna**: The RF antenna is used for wireless communication, enabling the device to send and receive data over radio frequencies.

2.3 Component Selection

- **LM1117MPX Power Regulators**: The LM1117MPX-3.3 and LM1117MPX-5[5] were selected to regulate power for the microcontroller and display unit, respectively. These regulators offer a low dropout voltage of 1.2 V at 800 mA, ensuring efficient power regulation even under high load conditions[5]. Additionally, they feature current limiting and thermal shutdown for reliable and safe operation. The requirement of a minimum 10-µF tantalum capacitor at the output enhances transient response and stability, making these regulators ideal for dynamic environments[5].
- **MPU6050 Sensor**: The MPU6050[3] was chosen as it combines a 3-axis gyroscope, 3-axis accelerometer, and Digital Motion Processor™ (DMP) into a single compact device. This integration simplifies the motion tracking system by reducing the need for multiple discrete components, thereby lowering design complexity and cost[3]. The dedicated I2C sensor bus supports the addition of an external 3-axis compass, providing a complete 9-axis MotionFusion™ output for enhanced motion tracking accuracy[3] . The sensor's programmable full-scale ranges for both the gyroscope ($\pm$250 to $\pm$2000°/sec) and accelerometer ($\pm$2g to $\pm$16g) ensure precise tracking of various motion types[3].
- **Atmel ATmega328P Microcontroller**: The Atmel ATmega328P [2] was chosen for its high performance and low power consumption, featuring an advanced RISC architecture that delivers up to 20 MIPS throughput at 20 MHz. It offers versatile memory options, including 32K bytes of flash memory, 1K bytes of EEPROM, and 2K bytes of SRAM, providing ample storage for program and data[2]. The microcontroller also includes a rich set of peripherals such as multiple timer/counters, PWM channels, ADCs, USART, SPI, and I2C interfaces, which support a wide range of functionalities[2]. Additionally, its power-saving modes and robust operational capabilities across a wide voltage range (1.8 - 5.5V) and temperature range (-40°C to 85°C) make it a reliable choice for wearable devices[2].

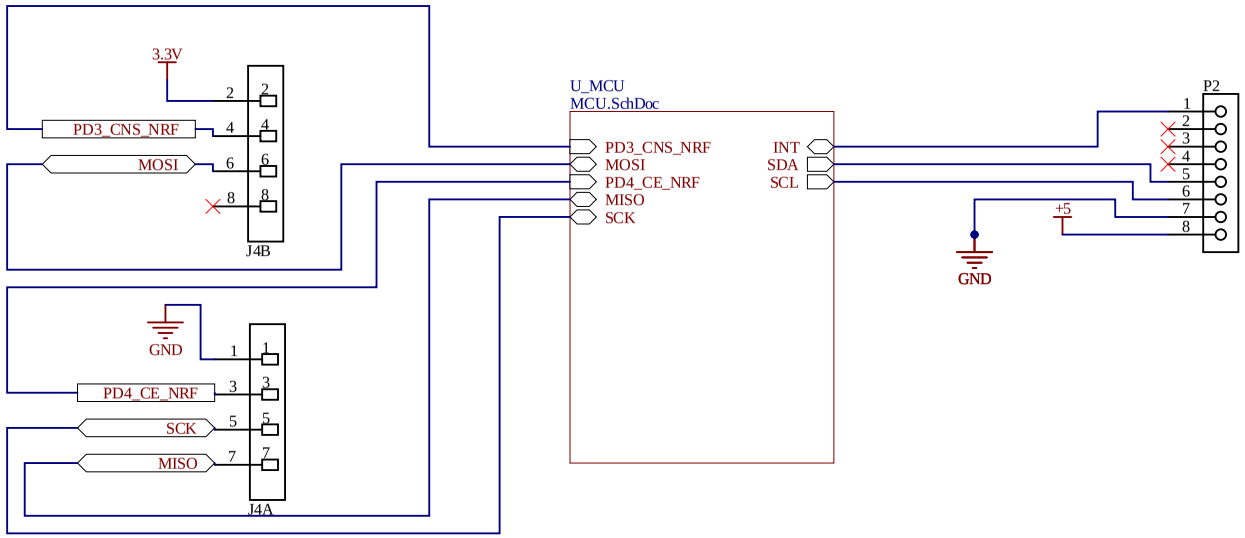
- **SSD1306 OLED Display Unit**: The SSD1306 OLED display unit[4] was selected for its efficient and compact design, which integrates a controller for a 128x64 dot-matrix graphic display system. This single-chip solution reduces the number of external components needed and lowers power consumption, which is critical for portable applications. The SSD1306's built-in contrast control, display RAM, and oscillator further simplify the design. Its compatibility with multiple communication interfaces (Parallel, I2C, SPI) allows for flexible integration with various microcontrollers[4]. The 256-step brightness control ensures clear and adjustable display visibility, making it suitable for real-time data visualization in our project[4].

- **NRF24L01 Antenna**: The NRF24L01[1] was chosen for its efficient and low-power radio transceiver capabilities in the 2.4 - 2.5 GHz ISM band. This transceiver includes a fully integrated frequency synthesizer, power amplifier, crystal oscillator, demodulator, modulator, and Enhanced ShockBurst™ protocol engine, providing a comprehensive and reliable wireless communication solution[1]. Its programmable output power and frequency channels, combined with low current consumption (9.0mA at -6dBm output and 12.3mA in RX mode), make it an excellent choice for energy-efficient wireless data transmission. The built-in power-down and standby modes further enhance power savings, ensuring prolonged battery life for our wearable device[1].

2.4 Final Circuit Diagrams

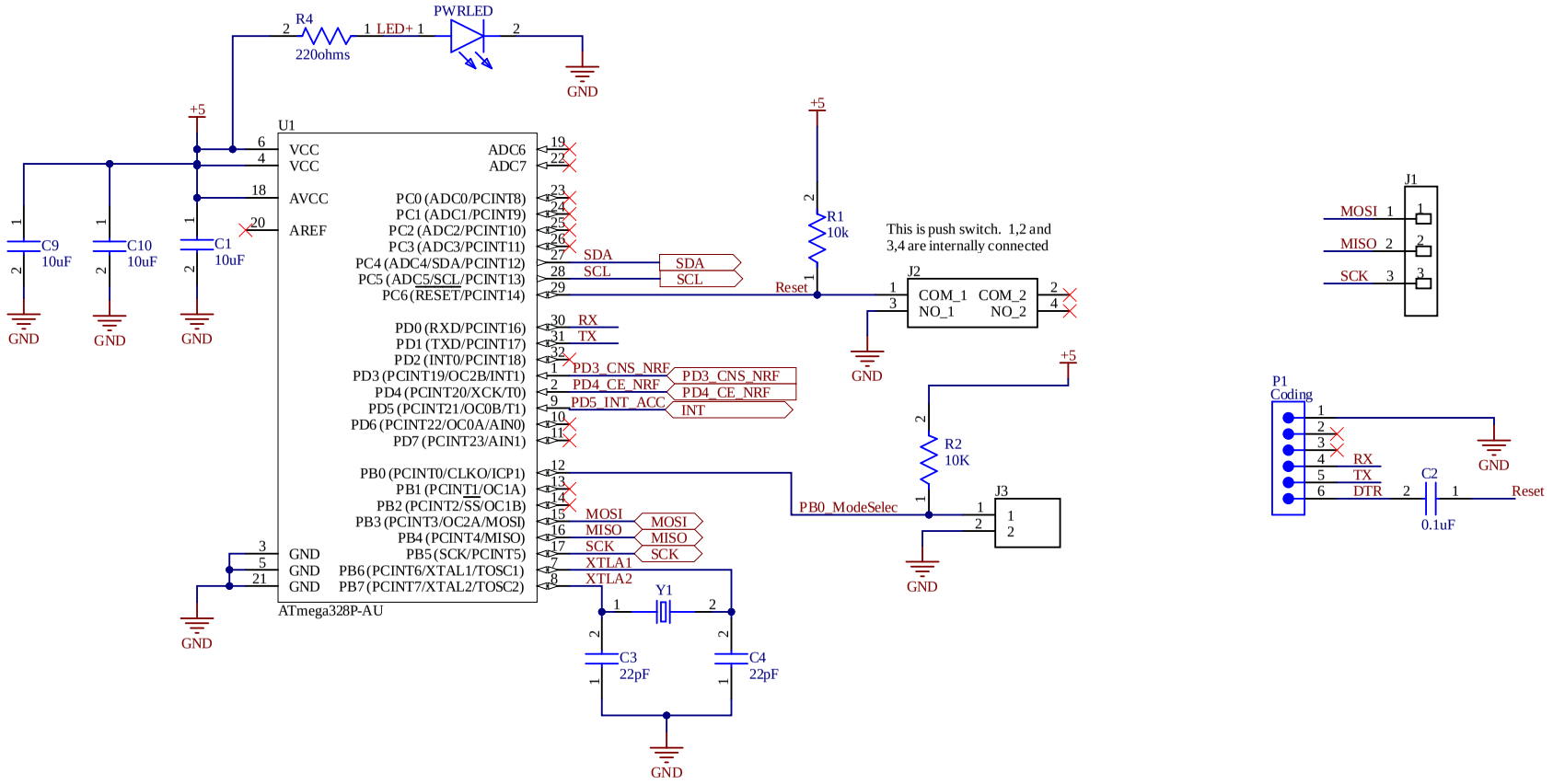
2.4.1 Final Schematic Design

The schematic design for our electronic device project was created using Altium Designer software. The design focuses on surface-mounted components (SMD) to ensure the final product is compact and efficient. Throughout the design process, we adhered to the industry standards outlined in guidelines for drawing schematics [6]. These standards guided us in component selection, circuit layout, and ensuring the overall reliability and functionality of the device.

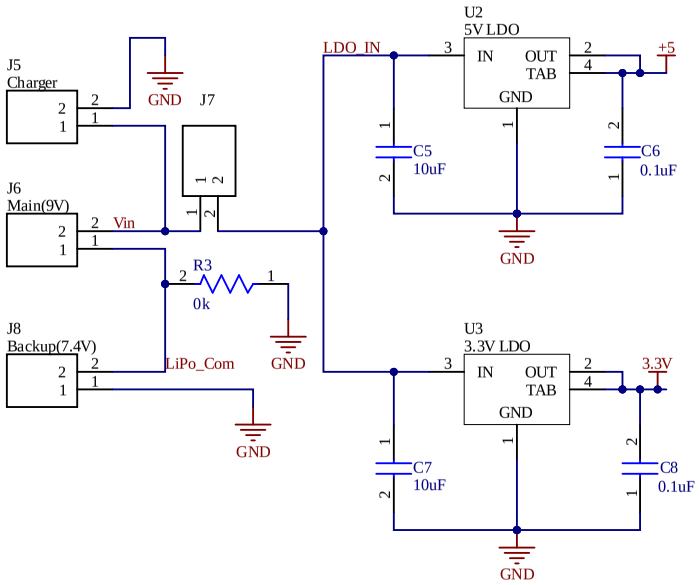


Designator
Power Reg.SchDoc

Title	Top-level circuit	
Size A4	Number 1	Revision 2
Date: 6/23/2024		Sheet 1 of 3
File: D:\Altium PCB\..\top-level.SchDoc		Drawn By: Rajapaksha N.N.



Title	Microcontroller Unit	
Size: A4	Number: 2	Revision: 2
Date: 6/23/2024		Sheet 2 of 3
File: D:\Altium PCB\..\MCU.SchDoc		Drawn By: Rajapaksha N.N.



Title	Power regulator		
Size A4	Number 3		Revision 2
Date:	6/23/2024	Sheet 3 of 3	
File:	D:\Altium PCB\..\Power Reg.SchDoc	Drawn By:	Rajapaksha N.N.

2.5 Layout Information

For the printed circuit board (PCB) design, we also utilized Altium Designer software. The decision to use surface-mounted components (SMD) was made to keep the PCB compact and suitable for modern electronic applications. The PCB is a four-layer board, which helps in managing the complexity of the circuit while maintaining a small footprint. The dimensions and layer overview of the PCB are detailed below, illustrating the careful planning and layout that went into this design to ensure reliability and performance.

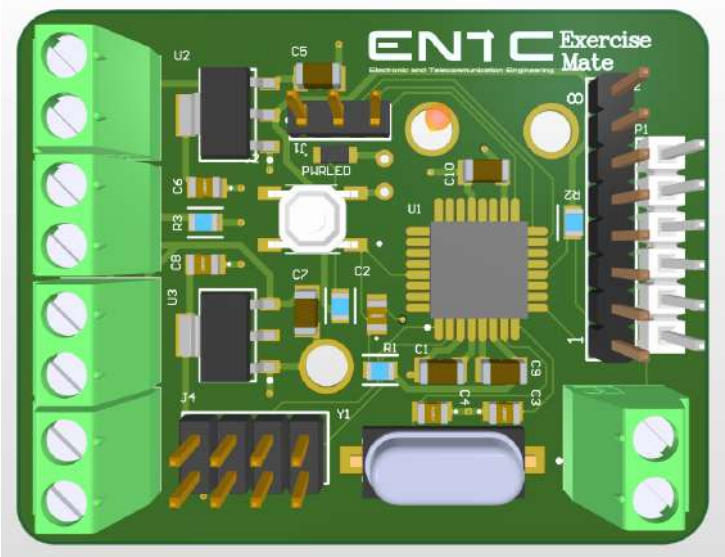
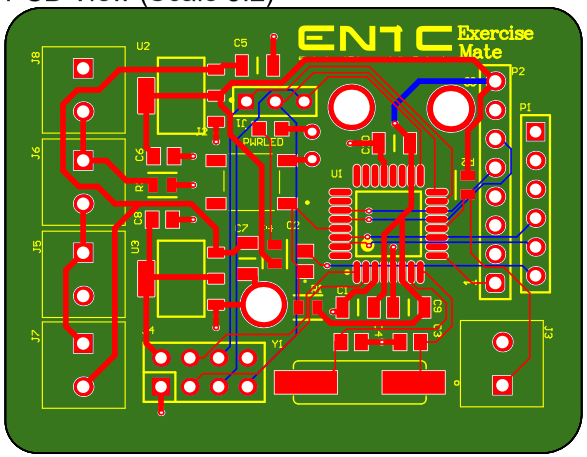
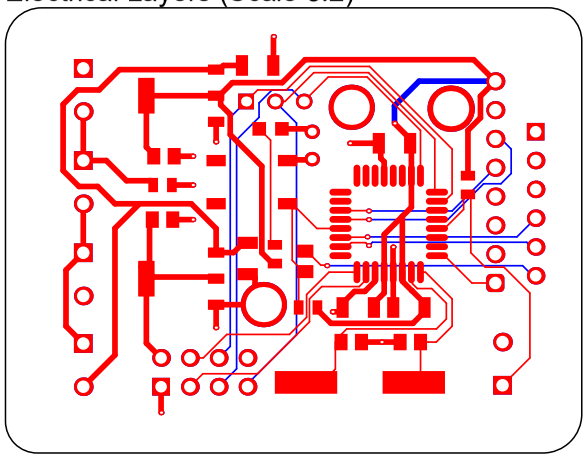


Figure 4: 3D view of the PCB

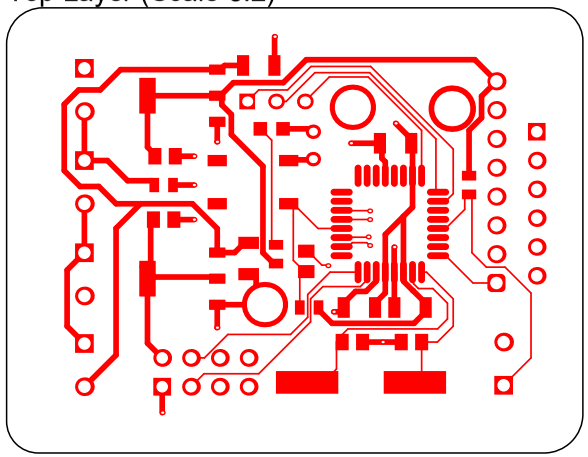
PCB View (Scale 3:2)



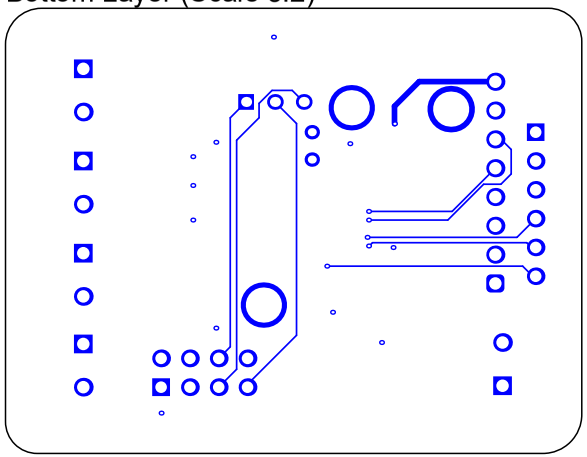
Electrical Layers (Scale 3:2)



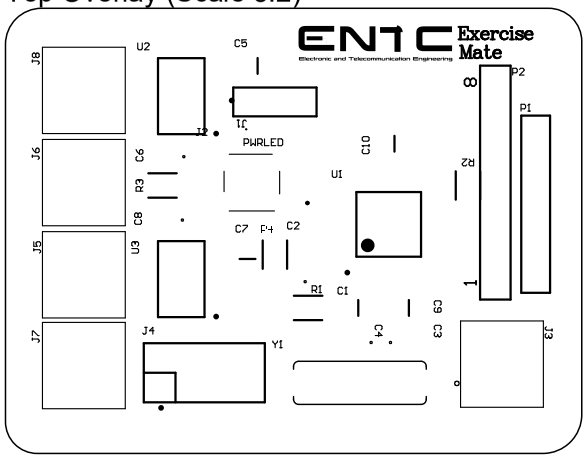
Top Layer (Scale 3:2)



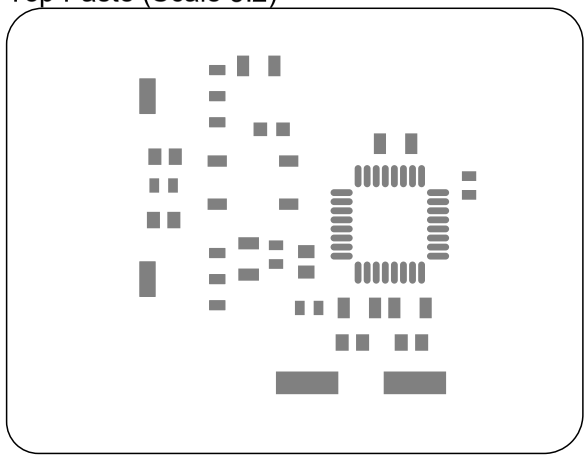
Bottom Layer (Scale 3:2)



Top Overlay (Scale 3:2)



Top Paste (Scale 3:2)



Title	PCB layer overview		
Size	A4	Number 2	Revision 2
Date	22/06/2024	Sheet 2 of 2	
File	D:\Altium PCB\exercise_mate\	Drawn by	Rajapaksha N.N.

2.6 Printed PCB

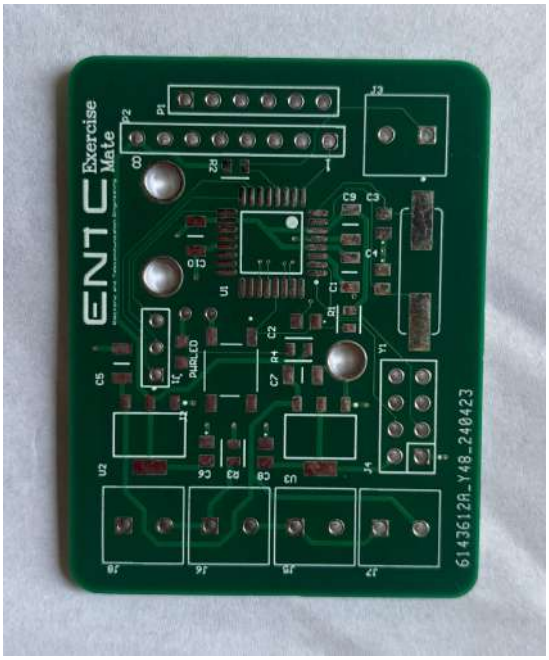


Figure 5: Bare PCB Top

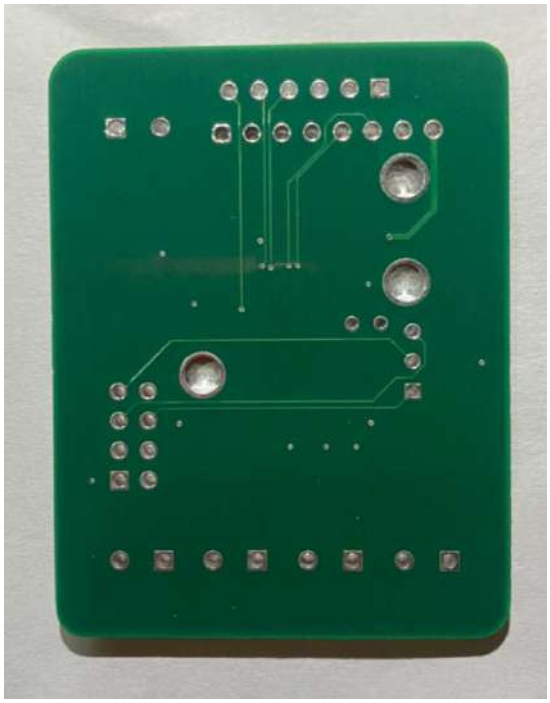


Figure 6: Bare PCB Bottom

2.7 Soldered PCB

Figure 7: Soldered PCB

Figure 8: Soldered PCB

2.8 Circuit Testing

Extensive testing was conducted to ensure the circuit's reliability and accuracy. Issues were identified and resolved, ensuring the system's robustness.

2.8.1 PCB testing

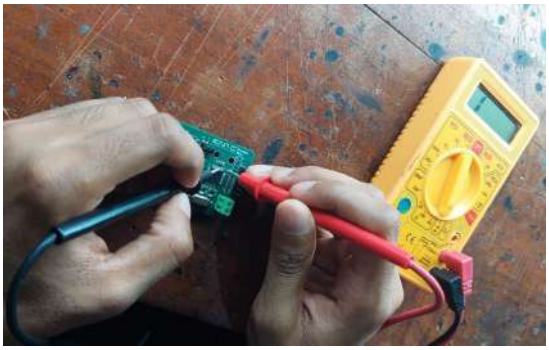


Figure 9: PCB conduction testing

Figure 10: PCB testing

3 Industrial Design

3.1 Design of Mechanical Sub-assemblies

The system's housing was designed to be ergonomic and durable. It includes compartments for sensors, the micro-controller, and the display unit.

3.2 Specifications

- **Dimensions**: Compact and lightweight.
- **Materials**: Durable plastic components.
- **Finish**: Smooth finish for comfort and aesthetics.

3.3 Selection Criteria for Various Parts

Parts were selected based on durability, cost, and ease of assembly.

3.4 Rough Sketches

Initial sketches provided a blueprint for the design, which was refined through iterations.

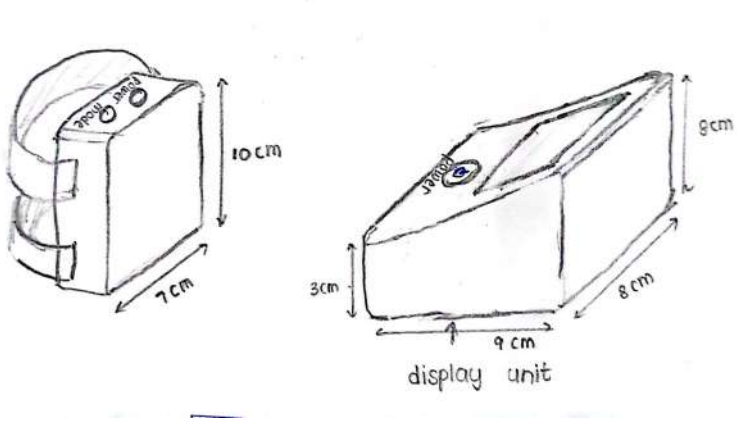
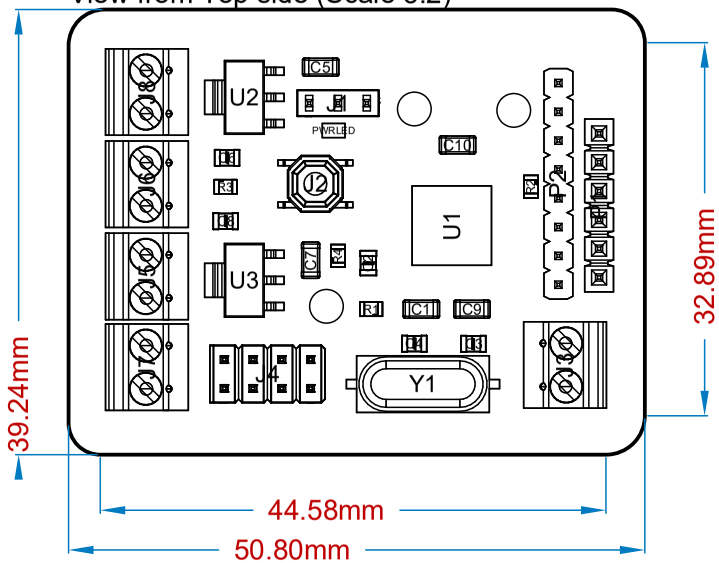


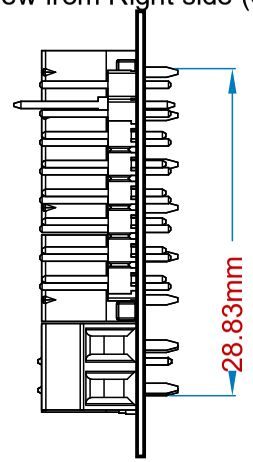
Figure 11: Wearable devices with separate display - Enclosure

3.5 PCB Dimensions Finalized

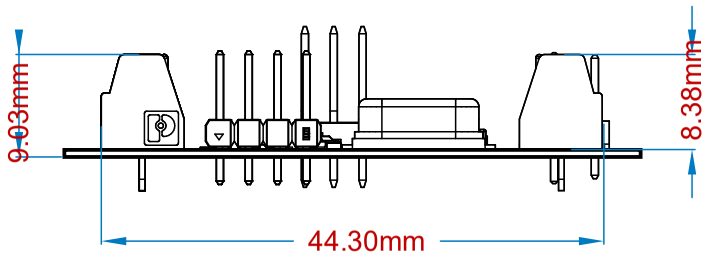
View from Top side (Scale 3:2)



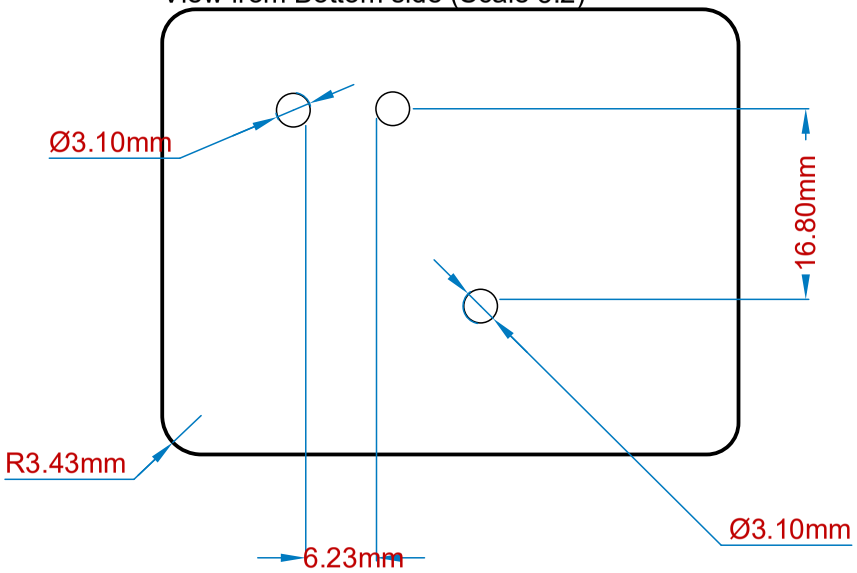
View from Right side (Scale 3:2)



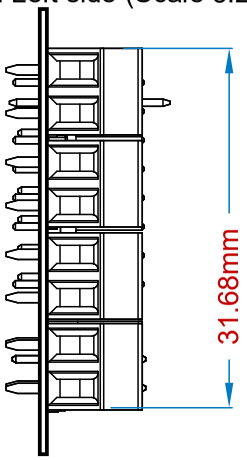
View from Front side (Scale 3:2)



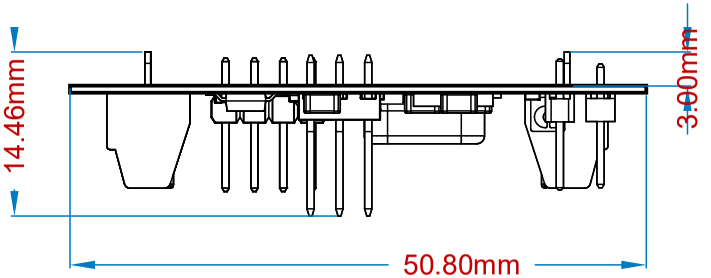
View from Bottom side (Scale 3:2)



View from Left side (Scale 3:2)



View from Back side (Scale 3:2)



Title	PCB dimensions		
Size	A4	Number 1	Revision 2
Date	22/06/2024	Sheet 1 of 2	
File	D:\Altium PCB\exercise_mate\	Drawn by	Rajapaksha N.N.

3.6 Solidworks Design

3.6.1 Hand Wearable Device

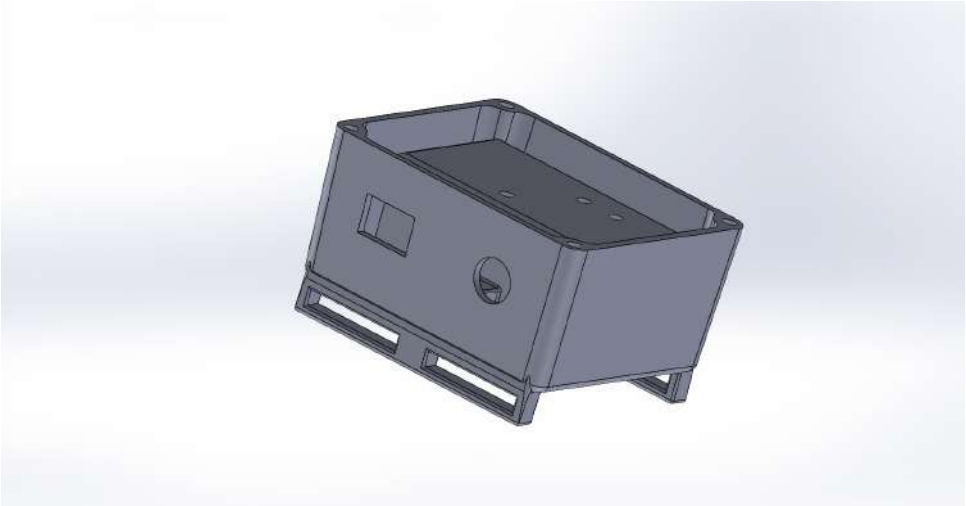


Figure 12: Bottom part of the design

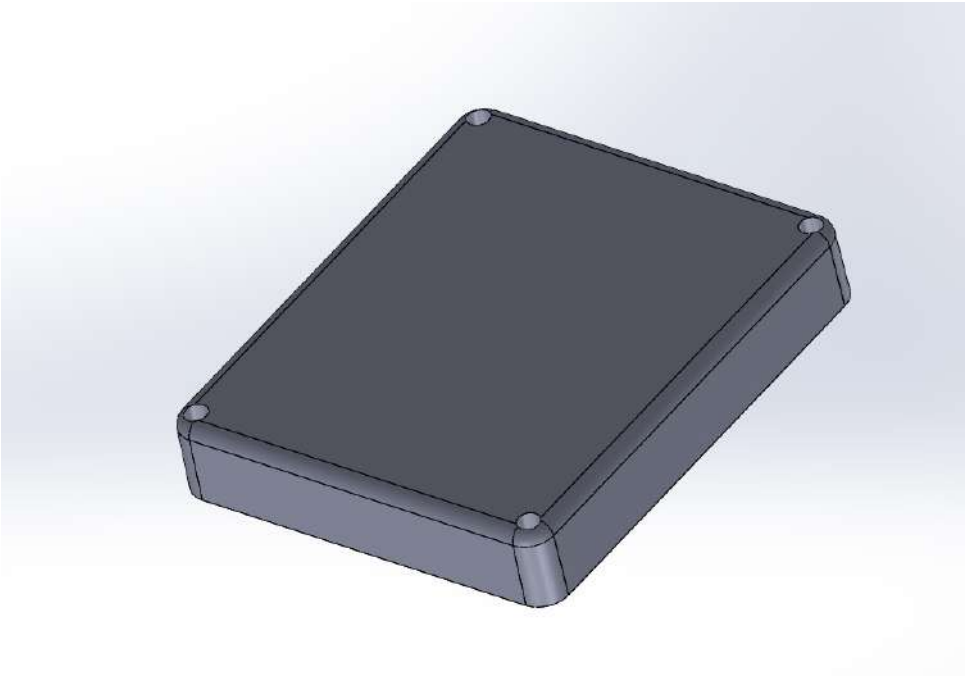


Figure 13: Top part of the design

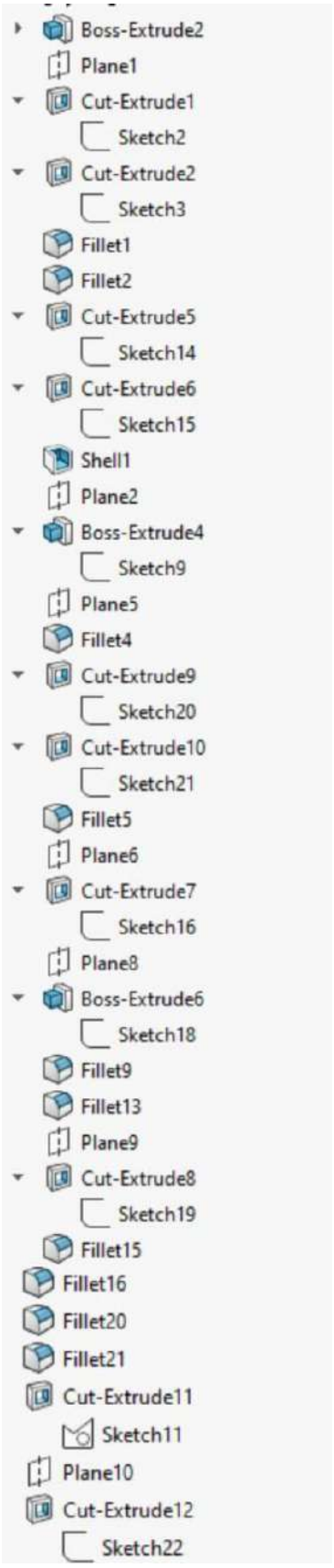


Figure 14: Design tree of the bottom part of the design

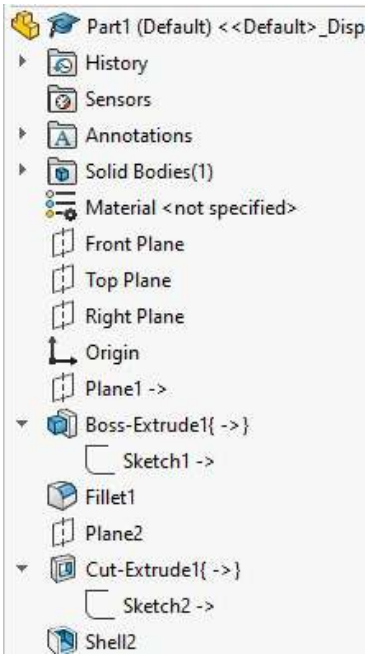


Figure 15: Design tree of the top part of the design

3.6.2 Central Device

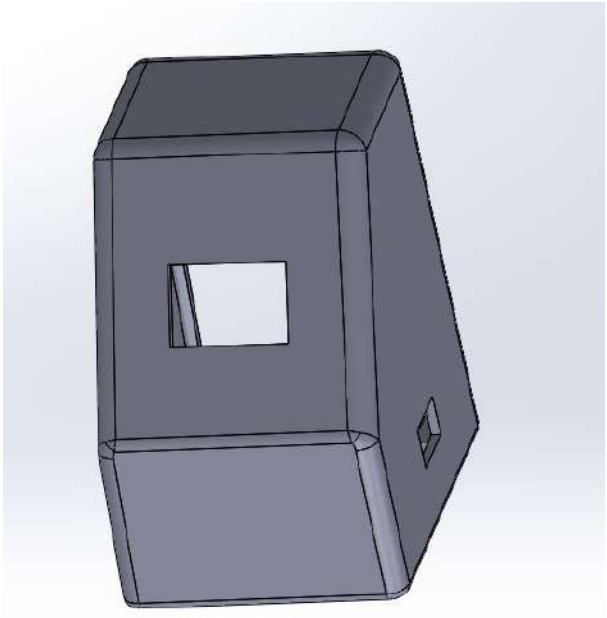


Figure 16: Bottom part of the design

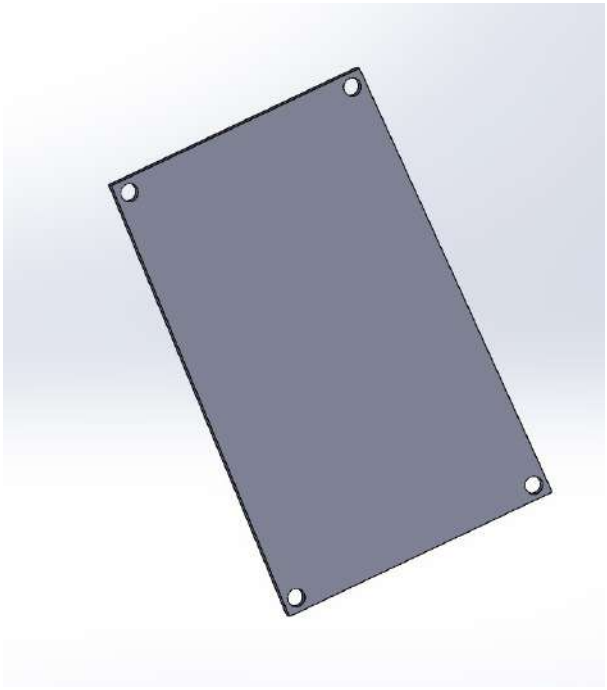


Figure 17: Top part of the design

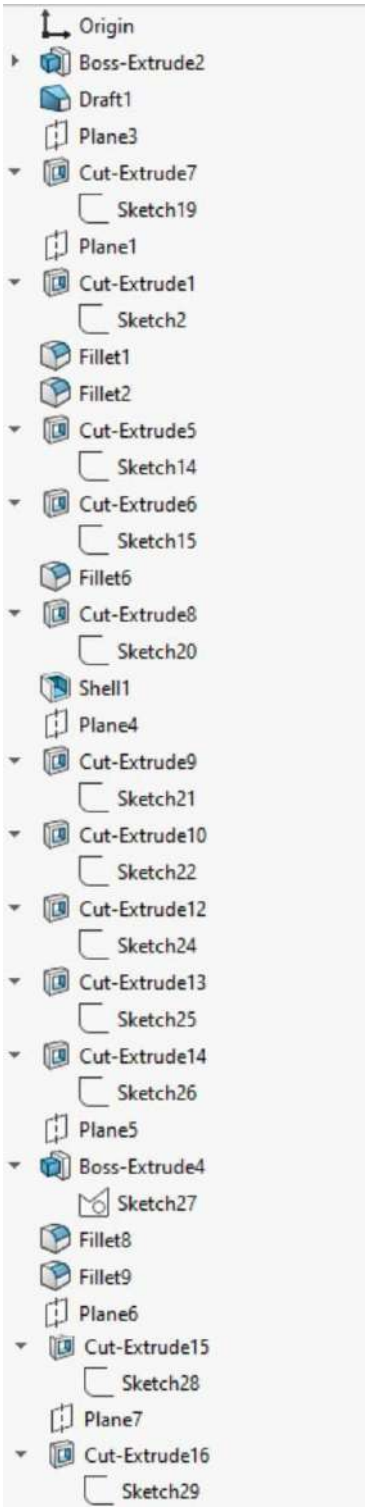


Figure 18: Design tree of the bottom part of the design

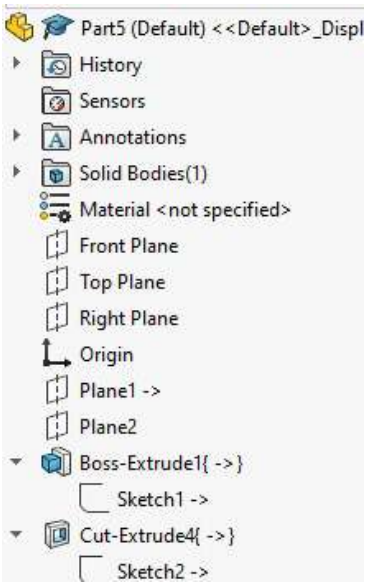


Figure 19: Design tree of the top part of the design

3.6.3 Battery area closing part

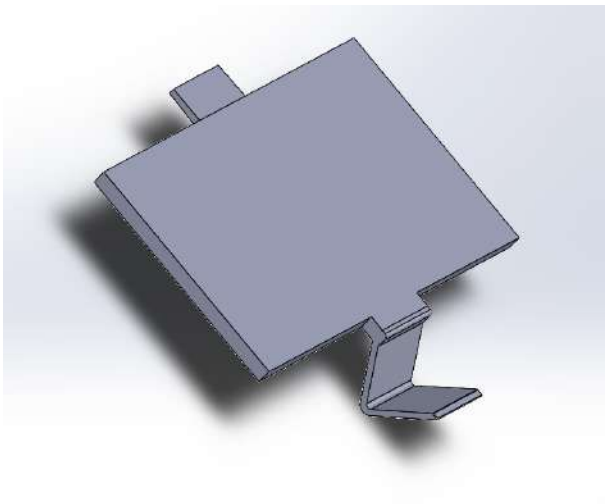


Figure 20: Battery area closing part

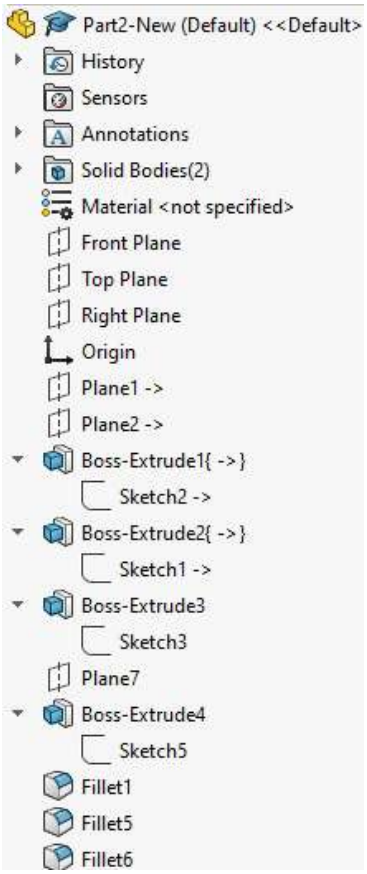


Figure 21: Design tree of the Battery area closing part

3.7 Physically Build parts

3.7.1 Hand Wearable Device

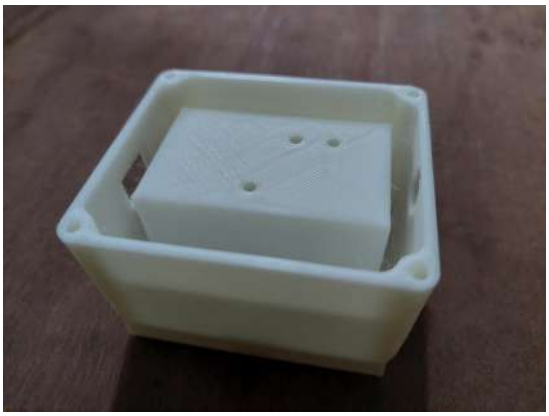


Figure 22: Bottom of the device

Figure 23: Top of the Device

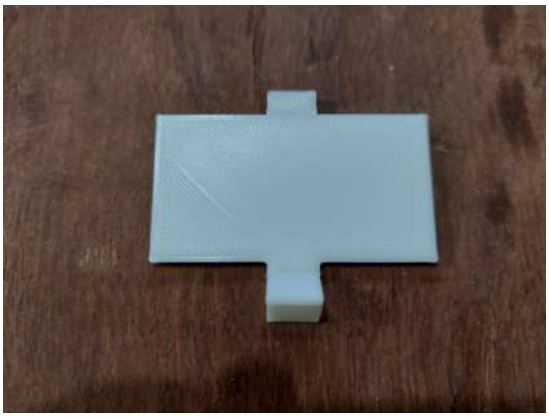


Figure 24: Battery closing Part

Figure 25: Enclosure Full

3.7.2 Display Unit

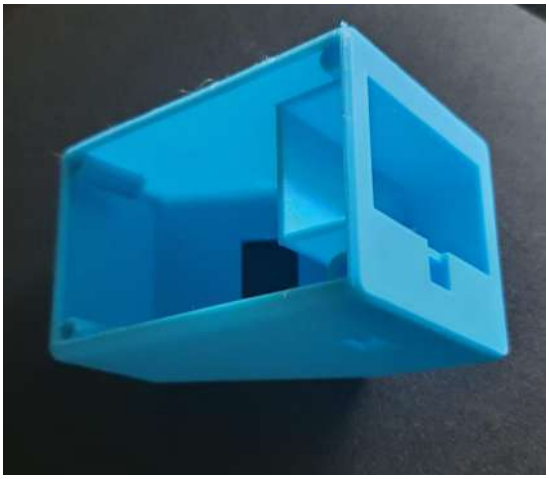


Figure 26: Bottom of the device

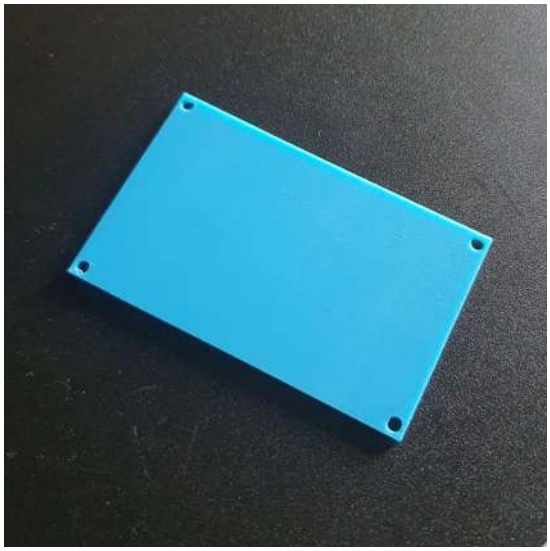


Figure 27: Top of the Device

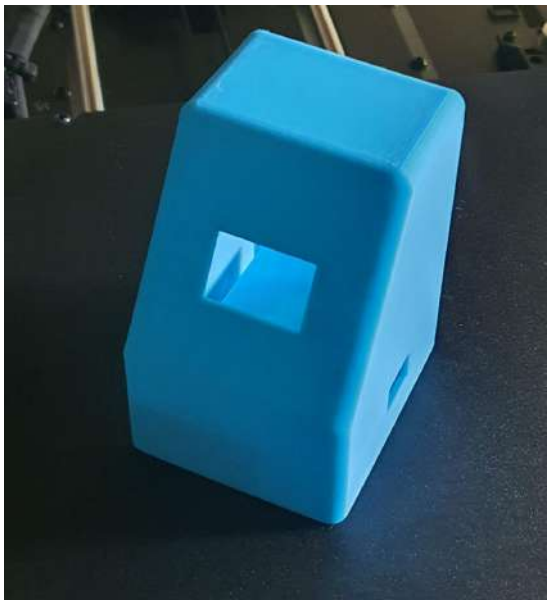
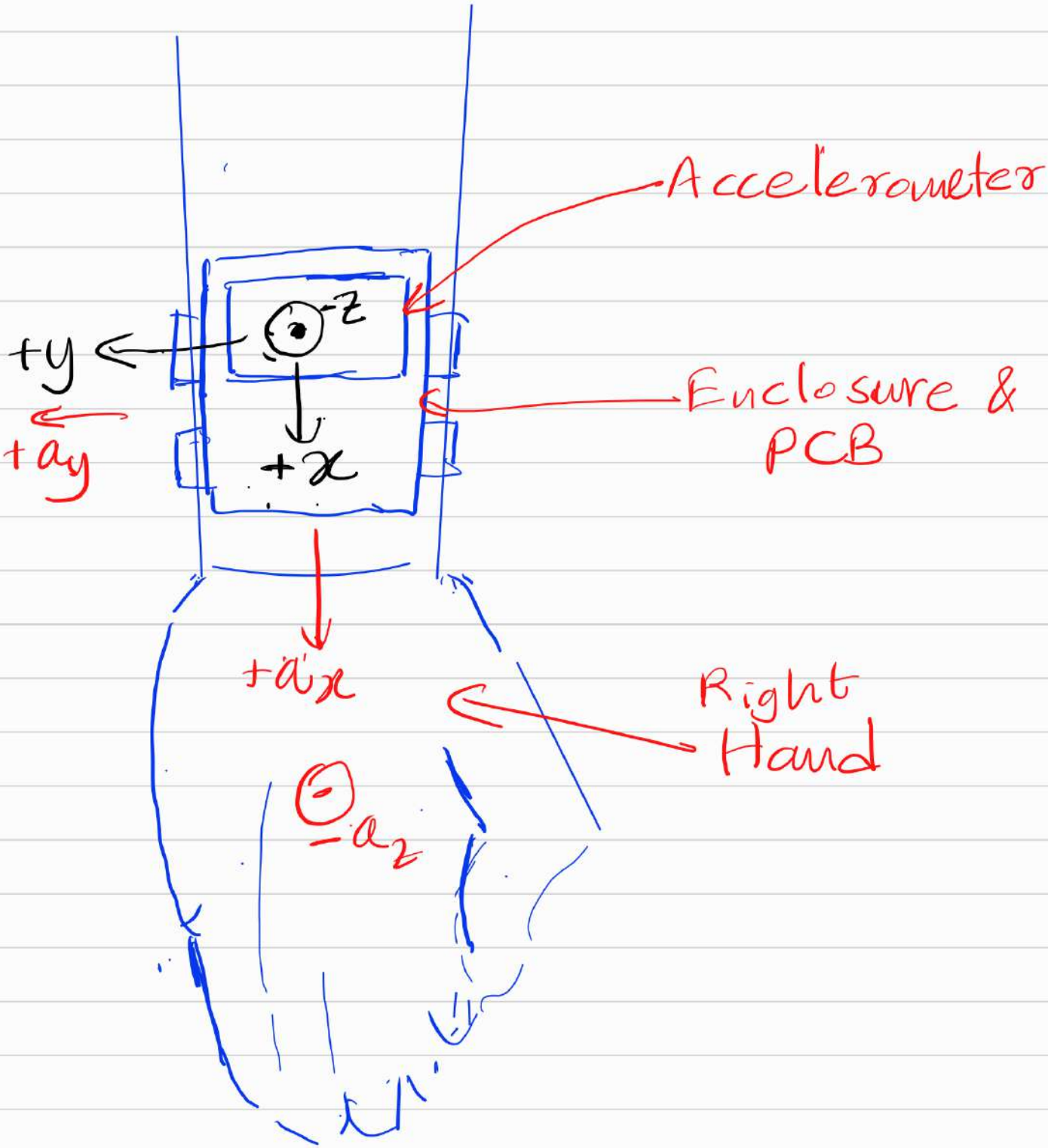


Figure 28: Enclosure Full

4 Embedded Software Implementation

4.1 Methodology

Orientation of Accelerometer



$+x, +y, -z \Rightarrow$ directions relative to hand

$+a_x, +a_y, -a_z \Rightarrow$ relative acceleration directions

Initialy, we tried to count exercises from detecting direction of the velocities. velocities can be calculated from,

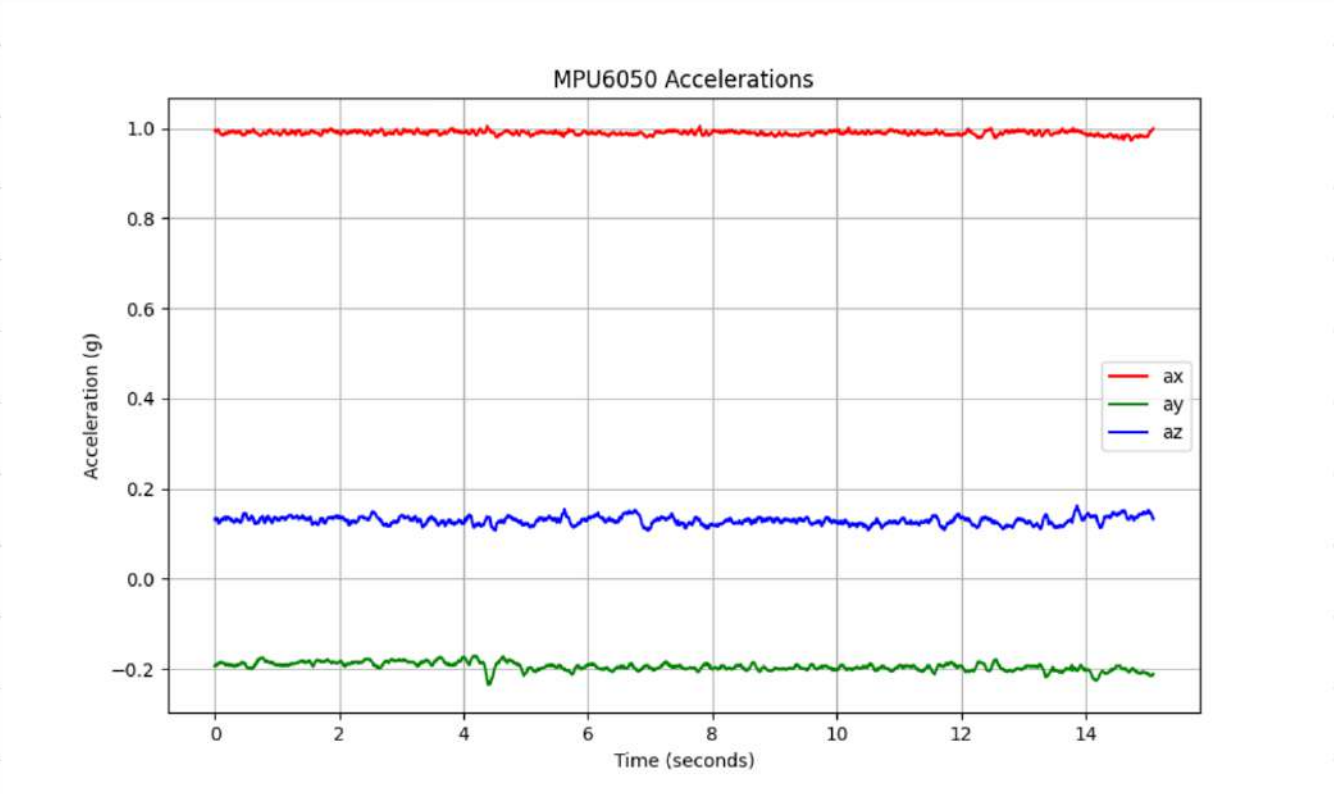
$$\therefore v_i(t_n) = \sum_{k=0}^{n} a_i(t_k) \cdot \Delta t \qquad i = x, y, z$$

Δt - sampling interval

But since the gravity is added to the acceleration components as a disturbance, correct velocities can't be calculated directly.

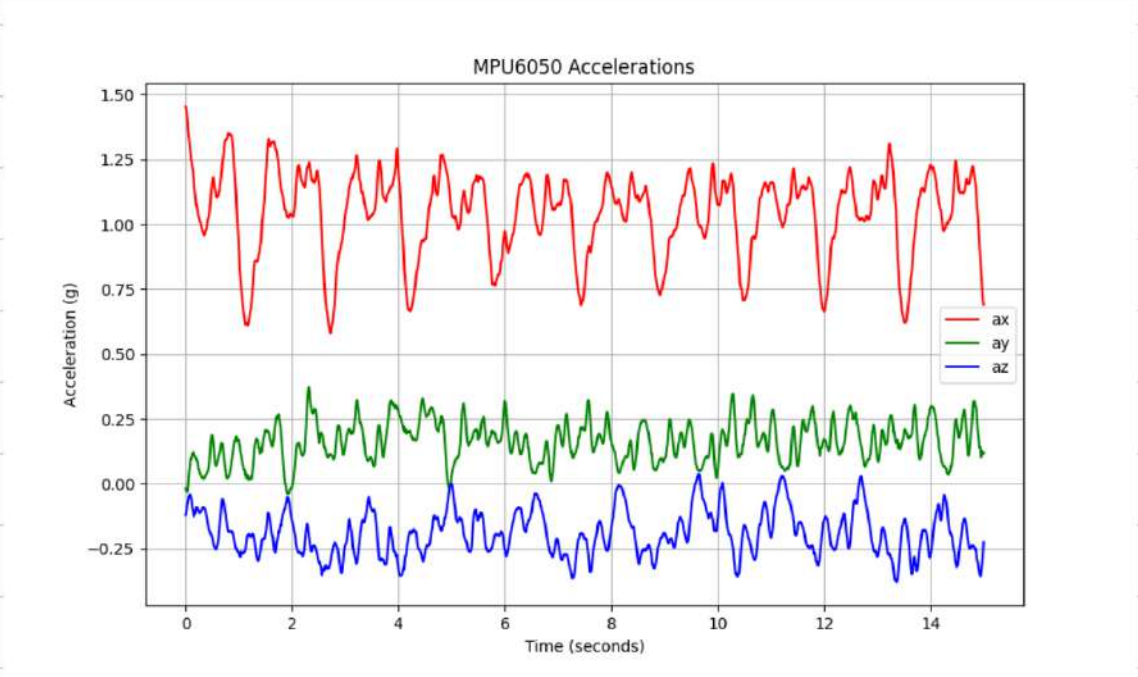
Therefore, in our final design we count exercises by detecting periodic cycles in the acceleration readings.

Steady state reading in above orientation



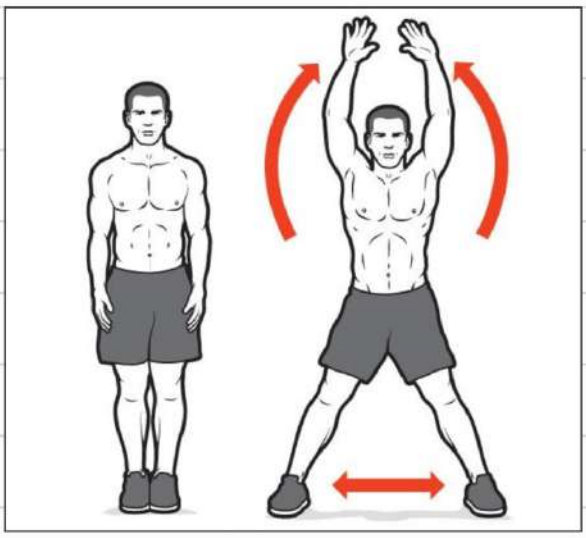
Selected exercise modes for now.

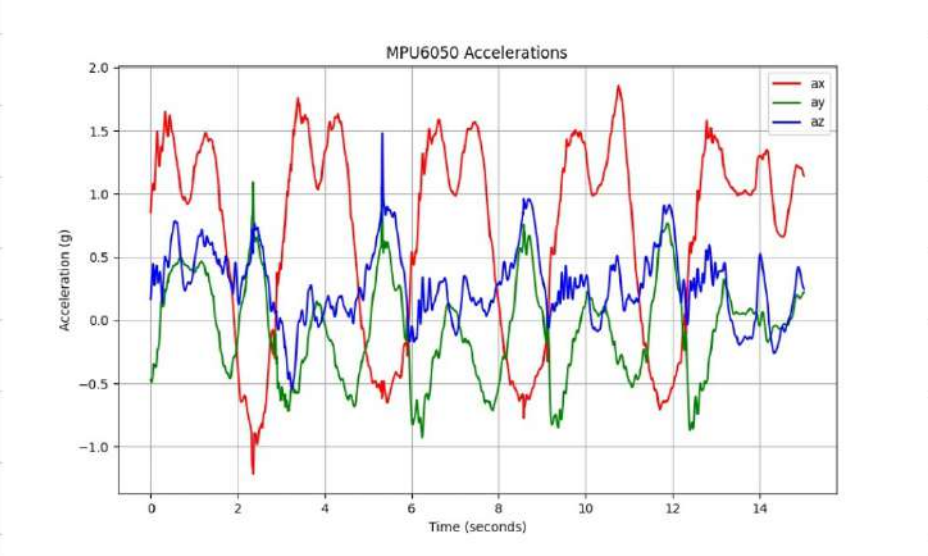
1) Walking



If a_x go beyond positive_threshold = +1.1g,
count += 1
If a_x fall beyond negative_threshold = +0.8g,
count += 1

2) jumping jacks

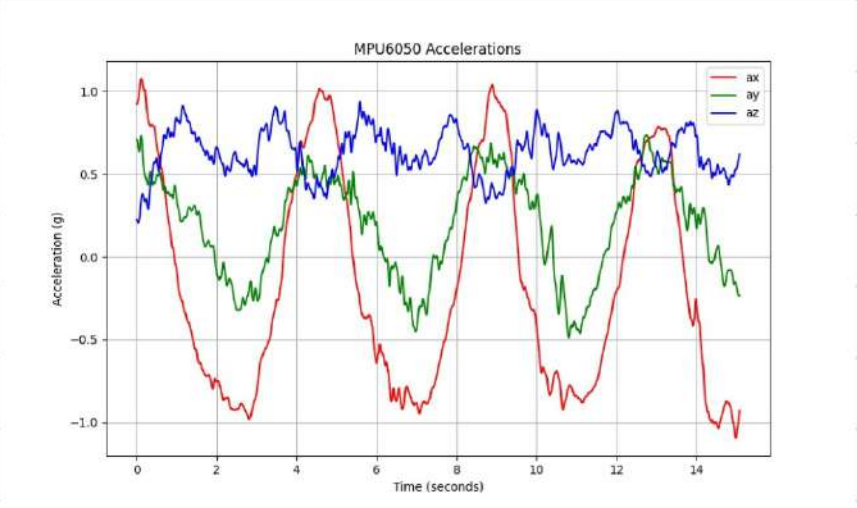




If ($a_y < 0$) ;

count = count + 1

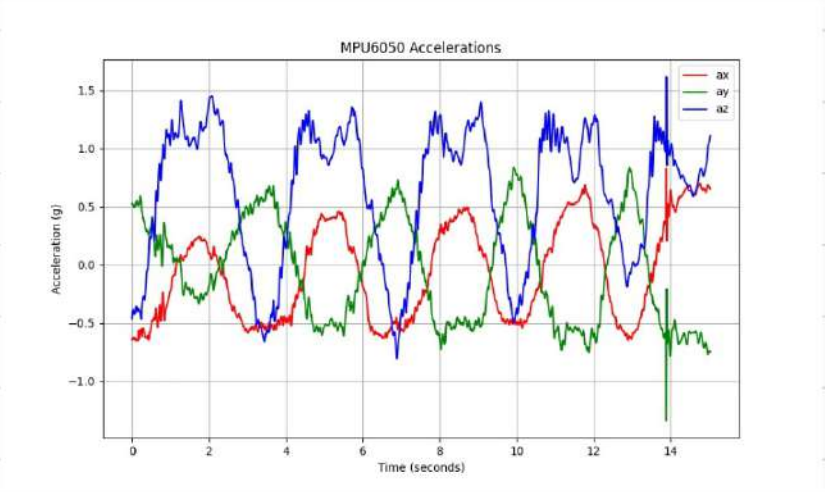
3) side shoulder stretch



If (a_z goes beyond possitive_threshold = +0.5g and less than possitive_threshold = +1g) ;

count = count + 1

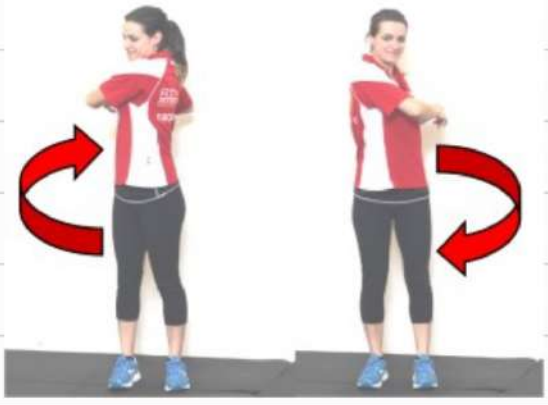
4) overhead shoulder stretch

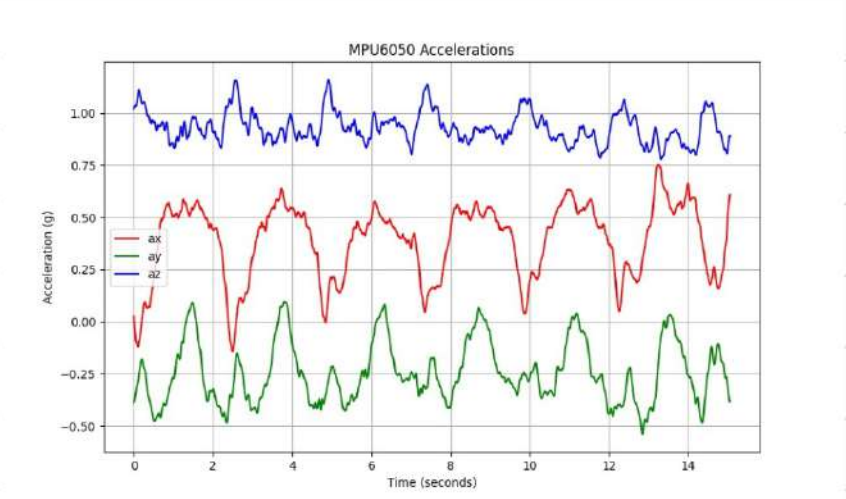


If (a_y goes beyond possitive_threshold $= +0.5g$ and a_z lesser than possitive_threshold $= 0g$) ;

count = count + 1

5) torso twist





If (a_x goes beyond possitive_threshold $= +0.5g$) ;

count = count + 1

* When taking the threshold values we only consider two people data. For a more accurate threshold value we need to get a larger data set and need to normalize the data.

4.2 ExerciseMate code

4.2.1 transmitter code

```
1
2  #include <avr/io.h>
3  #include <avr/interrupt.h>
4  #include <stdbool.h>
5  #include <string.h>
6  #include <math.h>
7  #include "i2c.h"
8  #include "mpu6050.h"
9  #include "nrf24l01.h"
10 #include <util/delay.h>
11
12 // Define threshold constants
13 #define UPPER_THRESHOLD 1.1
14 #define LOWER_THRESHOLD 0.8
15 #define DROP_THRESHOLD 0.8
16
17 // Global variables
18 volatile uint8_t exercise_mode = 0; // 0-4 for 5 exercise modes
19 volatile uint16_t exercise_count = 0;
20 volatile bool rf_interrupt = false;
21 volatile bool transmit_data = false;
22 int16_t accel_buff[3]; // Buffer to store accelerometer data
23
24 // Setup the nRF24L01 module
25 nRF24L01 *setup_rf(void) {
26     nRF24L01 *rf = nRF24L01_init();
27     rf->ss.port = &PORTB;
28     rf->ss.pin = PB2;
29     rf->ce.port = &PORTB;
30     rf->ce.pin = PB1;
31     rf->sck.port = &PORTB;
32     rf->sck.pin = PB5;
33     rf->mosi.port = &PORTB;
34     rf->mosi.pin = PB3;
35     rf->miso.port = &PORTB;
36     rf->miso.pin = PB4;
37     EICRA |= _BV(ISC01); // Interrupt on falling edge of INT0 (PD2)
38     EIMSK |= _BV(INT0);  // Enable INT0
39     nRF24L01_begin(rf);
40     return rf;
41 }
42
43 // Setup the MPU6050 accelerometer
44 void setup_mpu6050(void) {
```

```
    i2c_init();
    mpu6050_init();
}

// Setup the button input
void setup_button(void) {
    DDRB &= ~_BV(PB0); // Set PB0 as input
    PORTB |= _BV(PB0); // Enable pull-up resistor on PB0
}

// Variables to store previous accelerometer values
double ax_prev = -20, ay_prev = -20, az_prev = -20;

// Walking mode state enumeration
typedef enum {
    BELOW,
    EXCEEDED
} State;

State state = BELOW; // Initial state

int main(void) {
    sei(); // Enable global interrupts
    nRF24L01 *rf = setup_rf(); // Initialize RF module
    setup_mpu6050(); // Initialize MPU6050
    setup_button(); // Initialize button

    while (1) {
        // Check if the button is pressed
        if (bit_is_clear(PINB, PB0)) {
            _delay_ms(50); // Debounce delay
            exercise_mode = (exercise_mode + 1) % 5; // Cycle through exercise modes
            exercise_count = 0; // Reset exercise count
            transmit_data = true; // Flag to transmit data
        }

        // Transmit data if flagged
        if (transmit_data) {
            transmit_data = false;
            nRF24L01Message msg;
            msg.data[0] = exercise_mode;
            msg.data[1] = exercise_count >> 8;
            msg.data[2] = exercise_count & 0xFF;
            msg.length = 3;
            nRF24L01_transmit(rf, (uint8_t[5]){0x01, 0x01, 0x01, 0x01, 0x01}, &msg);
        }

        // Check RF interrupt flag
```

```
93          if (rf_interrupt) {
94              rf_interrupt = false;
95              int success = nRF24L01_transmit_success(rf);
96              if (success != 0)
97                  nRF24L01_flush_transmit_message(rf);
98          }
99
100         // Read accelerometer data
101         double ax, ay, az;
102         mpu6050_read_accel_ALL(accel_buff);
103
104         // Convert accelerometer data to g
105         ax = accel_buff[0] * 2.0 / 32768.0;
106         ay = accel_buff[1] * 2.0 / 32768.0;
107         az = accel_buff[2] * 2.0 / 32768.0;
108
109         // Exercise counting logic based on exercise mode
110         if (exercise_mode == 0) {
111             switch (state) {
112                 case BELOW:
113                     if (ax > UPPER_THRESHOLD) {
114                         exercise_count++;
115                         state = EXCEEDED;
116                     }
117                     break;
118                 case EXCEEDED:
119                     if (ax < DROP_THRESHOLD) {
120                         state = BELOW;
121                     }
122                     break;
123             }
124         } else if (exercise_mode == 1) {
125             switch (state) {
126                 case BELOW:
127                     if (ay <= 0) {
128                         exercise_count++;
129                         state = EXCEEDED;
130                     }
131                     break;
132                 case EXCEEDED:
133                     if (ay > 0.5) {
134                         state = BELOW;
135                     }
136                     break;
137             }
138         } else if (exercise_mode == 2) {
139             switch (state) {
140                 case BELOW:
```

```
                    if (az >= 0.5) {
                        exercise_count++;
                        state = EXCEEDED;
                    }
                    break;
                case EXCEEDED:
                    if (az < 0) {
                        state = BELOW;
                    }
                    break;
            }
        } else if (exercise_mode == 3) {
            switch (state) {
                case BELOW:
                    if (ay >= 0.5 && az <= 0) {
                        exercise_count++;
                        state = EXCEEDED;
                    }
                    break;
                case EXCEEDED:
                    if (az > 0.25) {
                        state = BELOW;
                    }
                    break;
            }
        } else if (exercise_mode == 4) {
            switch (state) {
                case BELOW:
                    if (ax >= 0.5) {
                        exercise_count++;
                        state = EXCEEDED;
                    }
                    break;
                case EXCEEDED:
                    if (ax < 0.25) {
                        state = BELOW;
                    }
                    break;
            }
        }

        // Update previous accelerometer values
        ax_prev = ax;
        ay_prev = ay;
        az_prev = az;
    }

    return 0;
```

```
}

// nRF24L01 interrupt handler
ISR(INT0_vect) {
    rf_interrupt = true;
}


```

4.2.2 receiver code

```

#include <avr/io.h>           // AVR standard I/O definitions
#include <avr/interrupt.h>    // AVR interrupt handling
#include <stdbool.h>          // Standard boolean definitions
#include <string.h>           // String manipulation functions
#include <util/delay.h>       // Delay functions
#include "nrf24l01.h"         // NRF24L01 module header
#include "lcd.h"              // display header

volatile bool rf_interrupt = false;  // Flag to indicate NRF24L01 interrupt

// Function to set up NRF24L01 module
nRF24L01 *setup_rf(void) {
    nRF24L01 *rf = nRF24L01_init();  // Initialize NRF24L01 module
    // Set up pins for SPI communication
    rf->ss.port = &PORTB;
    rf->ss.pin = PB2;
    rf->ce.port = &PORTB;
    rf->ce.pin = PB1;
    rf->sck.port = &PORTB;
    rf->sck.pin = PB5;
    rf->mosi.port = &PORTB;
    rf->mosi.pin = PB3;
    rf->miso.port = &PORTB;
    rf->miso.pin = PB4;
    EICRA |= _BV(ISC01);    // Configure interrupt on falling edge of INT0 (PD2)
    EIMSK |= _BV(INT0);     // Enable INT0 interrupt
    nRF24L01_begin(rf);     // Begin NRF24L01 module
    return rf;              // Return pointer to NRF24L01 instance
}

// Function to set up SSD1306 OLED display
void setup_display(void) {
    ssd1306_init();                // Initialize SSD1306 display
    ssd1306_clear_display();       // Clear display
```

```
    ssd1306_set_cursor(0, 0);      // Set cursor to top-left corner
    ssd1306_print_string("Select exercise mode:");  // Print initial message
}

// Function to get the exercise mode string
const char* get_exercise_mode_string(uint8_t mode) {
    switch (mode) {
        case 0: return "Walking";
        case 1: return "Jumping jacks";
        case 2: return "Side shoulder stretch";
        case 3: return "Side shoulder stretch";
        case 4: return "Overhead shoulder stretch";
        default: return "Unknown mode";
    }
}

// Main function
int main(void) {
    uint8_t address[5] = {0x01, 0x01, 0x01, 0x01, 0x01};  // Address for NRF24L01
    ↪ communication
    sei();  // Enable global interrupts
    nRF24L01 *rf = setup_rf();  // Set up NRF24L01 module
    setup_display();  // Set up SSD1306 display
    nRF24L01_listen(rf, 0, address);  // Start listening on NRF24L01 with given
    ↪ address

    while (1) {
        if (rf_interrupt) {  // Check if NRF24L01 interrupt occurred
            rf_interrupt = false;  // Clear interrupt flag

            if (nRF24L01_data_received(rf)) {  // Check if data received
                nRF24L01Message msg;  // NRF24L01 message container
                nRF24L01_read_received_data(rf, &msg);  // Read received data
                uint8_t exercise_mode = msg.data[0];  // Extract exercise mode from
                ↪ message
                uint16_t exercise_count = (msg.data[1] << 8) | msg.data[2];  //
                ↪ Extract exercise count

                // Print exercise mode and count on SSD1306 display
                ssd1306_set_cursor(0, 16);
                ssd1306_print_string("Exercise mode: ");
                ssd1306_print_string(get_exercise_mode_string(exercise_mode));

                ssd1306_set_cursor(0, 32);
                ssd1306_print_string("Exercise count: ");
                ssd1306_print_int(exercise_count);

                ssd1306_update_display();  // Update SSD1306 display
```

```
            }
        }
    }

    return 0;  // Return statement (never actually reached)
}

// Interrupt Service Routine (ISR) for INT0 vector (PD2)
ISR(INT0_vect) {
    rf_interrupt = true;  // Set interrupt flag indicating data received from
    ↪ NRF24L01
}
```

All included libraries such as "i2c.h", "mpu6050.h", "nrf24l01.h", and "lcd.h" are open source and have been modified as necessary to suit our application. Implementations of these header files, apart from built-in or AVR libraries, are provided below.

4.3 source code for MPU sensor

4.3.1 i2c.c

```
/*************************************************************************
* Title:    I2C master library using hardware TWI interface
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: twimaster.c,v 1.3 2005/07/02 11:14:21 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device with hardware TWI
* Usage:    API compatible with I2C Software Library i2cmaster.h
*************************************************************************/
#include <inttypes.h>
#include <compat/twi.h>

#include "i2c.h"


/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 4000000UL
#endif

/* I2C clock in Hz */
#define SCL_CLOCK  100000L


/*************************************************************************
```

```
 Initialization of the I2C bus interface. Need to be called only once
*************************************************************************/
void i2c_init(void)
{
  /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */

  TWSR = 0;                         /* no prescaler */
  TWBR = ((F_CPU/SCL_CLOCK)-16)/2;  /* must be > 10 for stable operation */

}/* i2c_init */


/*************************************************************************
  Issues a start condition and sends address and transfer direction.
  return 0 = device accessible, 1= failed to access device
*************************************************************************/
unsigned char i2c_start(unsigned char address)
{
    uint8_t   twst;

        // send START condition
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

        // send device address
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wail until transmission completed and ACK/NACK has been received
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

        return 0;

}/* i2c_start */


/*************************************************************************
 Issues a start condition and sends address and transfer direction.
```

```
 If device is busy, use ack polling to wait until device is ready

 Input:   address and transfer direction of I2C device
*****************************************************************************/
void i2c_start_wait(unsigned char address)
{
    uint8_t   twst;


    while ( 1 )
    {
            // send START condition
            TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

            // wait until transmission completed
            while(!(TWCR & (1<<TWINT)));

            // check value of TWI Status Register. Mask prescaler bits.
            twst = TW_STATUS & 0xF8;
            if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

            // send device address
            TWDR = address;
            TWCR = (1<<TWINT) | (1<<TWEN);

            // wail until transmission completed
            while(!(TWCR & (1<<TWINT)));

            // check value of TWI Status Register. Mask prescaler bits.
            twst = TW_STATUS & 0xF8;
            if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
            {
                /* device busy, send stop condition to terminate write operation */
                TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

                // wait until stop condition is executed and bus released
                while(TWCR & (1<<TWSTO));

                continue;
            }
            //if( twst != TW_MT_SLA_ACK) return 1;
            break;
     }

}/* i2c_start_wait */


/*****************************************************************************
```

```
 Issues a repeated start condition and sends address and transfer direction

 Input:   address and transfer direction of I2C device

 Return:  0 device accessible
          1 failed to access device
*************************************************************************/
unsigned char i2c_rep_start(unsigned char address)
{
    return i2c_start( address );

}/* i2c_rep_start */


/*************************************************************************
 Terminates the data transfer and releases the I2C bus
*************************************************************************/
void i2c_stop(void)
{
    /* send stop condition */
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

        // wait until stop condition is executed and bus released
        while(TWCR & (1<<TWSTO));

}/* i2c_stop */


/*************************************************************************
  Send one byte to I2C device

  Input:    byte to be transfered
  Return:   0 write successful
            1 write failed
*************************************************************************/
unsigned char i2c_write( unsigned char data )
{
    uint8_t   twst;

        // send data to the previously addressed device
        TWDR = data;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits
        twst = TW_STATUS & 0xF8;
```

```
169         if( twst != TW_MT_DATA_ACK) return 1;
170         return 0;
171 
172 }/* i2c_write */
173 
174 
175 /*************************************************************************
176  Read one byte from the I2C device, request more data from device
177 
178  Return:  byte read from I2C device
179 *************************************************************************/
180 unsigned char i2c_readAck(void)
181 {
182         TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
183         while(!(TWCR & (1<<TWINT)));
184 
185     return TWDR;
186 
187 }/* i2c_readAck */
188 
189 
190 /*************************************************************************
191  Read one byte from the I2C device, read is followed by a stop condition
192 
193  Return:  byte read from I2C device
194 *************************************************************************/
195 unsigned char i2c_readNak(void)
196 {
197         TWCR = (1<<TWINT) | (1<<TWEN);
198         while(!(TWCR & (1<<TWINT)));
199 
200     return TWDR;
201 
202 }/* i2c_readNak */
203 
204 
205 
206 
207 
208 // read one byte from dev, stored in value, return 1 for error
209 void i2c_read_byte(uint8_t dev_addr, uint8_t reg_addr, uint8_t* data){
210 
211         i2c_start_wait(dev_addr+I2C_WRITE);         //start i2c to write register
        ↪  address
212         i2c_write(reg_addr);                        //write address of register to
        ↪  read
213         i2c_rep_start(dev_addr+I2C_READ);        //restart i2c to start reading
214         *data = i2c_readNak();
```

```
215         i2c_stop();
216
217 }
218
219
220
221 // write one byte to dev
222 void i2c_write_byte(uint8_t dev_addr, uint8_t reg_addr, uint8_t data){
223
224         i2c_start_wait(dev_addr+I2C_WRITE);
225          i2c_write(reg_addr);
226         i2c_write(data);
227         i2c_stop();
228
229 }
230
231
232 // read multiple bytes from dev
233 void i2c_read_bytes(uint8_t dev_addr, uint8_t first_reg_addr, uint8_t length,
    ↪  uint8_t* data){
234         i2c_start_wait(dev_addr+I2C_WRITE);         //start i2c to write register
        ↪  address
235         i2c_write(first_reg_addr);                 //write address of register to read
236         i2c_rep_start(dev_addr+I2C_READ);         //restart i2c to start reading
237
238         uint8_t i;
239         for(i=0; i<length-1; i++){
240                 *(data+i) = i2c_readAck();
241         }
242         *(data+i) = i2c_readNak();
243         i2c_stop();
244 }
245
246
247
248 // write multiple bytes to dev
249 void i2c_write_bytes(uint8_t dev_addr, uint8_t reg_addr, uint8_t length, uint8_t*
    ↪  data){
250
251         i2c_start_wait(dev_addr+I2C_WRITE);
252         i2c_write(reg_addr);
253
254         uint8_t i;
255         for(i=0; i<length; i++){
256                 i2c_write( data[i] );
257         }
258
259         i2c_stop();
```

```
260
261 }
262
```

4.3.2 i2c.h

```
1
2  #ifndef _I2CMASTER_H
3  #define _I2CMASTER_H   1
4  /*************************************************************************
5  * Title:    C include file for the I2C master interface
6  *           (i2cmaster.S or twimaster.c)
7  * Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
8  * File:     $Id: i2cmaster.h,v 1.10 2005/03/06 22:39:57 Peter Exp $
9  * Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
10 * Target:   any AVR device
11 * Usage:    see Doxygen manual
12 *************************************************************************/
13
14 #ifdef DOXYGEN
15 /**
16  @defgroup pfleury_ic2master I2C Master library
17  @code #include <i2cmaster.h> @endcode
18
19  @brief I2C (TWI) Master Software Library
20
21  Basic routines for communicating with I2C slave devices. This single master
22  implementation is limited to one bus master on the I2C bus.
23
24  This I2c library is implemented as a compact assembler software implementation of
     ↪  the I2C protocol
25  which runs on any AVR (i2cmaster.S) and as a TWI hardware interface for all AVR with
     ↪  built-in TWI hardware (twimaster.c).
26  Since the API for these two implementations is exactly the same, an application can
     ↪  be linked either against the
27  software I2C implementation or the hardware I2C implementation.
28
29  Use 4.7k pull-up resistor on the SDA and SCL pin.
30
31  Adapt the SCL and SDA port and pin definitions and eventually the delay routine in
     ↪  the module
32  i2cmaster.S to your target when using the software I2C implementation !
33
34  Adjust the  CPU clock frequence F_CPU in twimaster.c or in the Makfile when using
     ↪  the TWI hardware implementaion.
35
```

```
36  @note
37     The module i2cmaster.S is based on the Atmel Application Note AVR300, corrected
       ↪  and adapted
38     to GNU assembler and AVR-GCC C call interface.
39     Replaced the incorrect quarter period delays found in AVR300 with
40     half period delays.
41
42  @author Peter Fleury pfleury@gmx.ch  http://jump.to/fleury
43
44  @par API Usage Example
45   The following code shows typical usage of this library, see example
       ↪  test_i2cmaster.c
46
47  @code
48
49  #include <i2cmaster.h>
50
51
52  #define Dev24C02  0xA2      // device address of EEPROM 24C02, see datasheet
53
54  int main(void)
55  {
56      unsigned char ret;
57
58      i2c_init();                             // initialize I2C library
59
60      // write 0x75 to EEPROM address 5 (Byte Write)
61      i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode
62      i2c_write(0x05);                        // write address = 5
63      i2c_write(0x75);                        // write value 0x75 to EEPROM
64      i2c_stop();                             // set stop conditon = release bus
65
66
67      // read previously written value back from EEPROM address 5
68      i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode
69
70      i2c_write(0x05);                        // write address = 5
71      i2c_rep_start(Dev24C02+I2C_READ);       // set device address and read mode
72
73      ret = i2c_readNak();                    // read one byte from EEPROM
74      i2c_stop();
75
76      for(;;);
77  }
78  @endcode
79
80 */
81 #endif /* DOXYGEN */
```

```
82
83 /**@{*/
84
85 #if (__GNUC__ * 100 + __GNUC_MINOR__) < 304
86 #error "This library requires AVR-GCC 3.4 or later, update to newer AVR-GCC compiler
   ↪ !"
87 #endif
88
89 #include <avr/io.h>
90
91 /** defines the data direction (reading from I2C device) in
   ↪ i2c_start(),i2c_rep_start() */
92 #define I2C_READ    1
93
94 /** defines the data direction (writing to I2C device) in i2c_start(),i2c_rep_start()
   ↪ */
95 #define I2C_WRITE   0
96
97
98 /**
99  @brief initialize the I2C master interace. Need to be called only once
100  @param  void
101  @return none
102  */
103 extern void i2c_init(void);
104
105
106 /**
107  @brief Terminates the data transfer and releases the I2C bus
108  @param void
109  @return none
110  */
111 extern void i2c_stop(void);
112
113
114 /**
115  @brief Issues a start condition and sends address and transfer direction
116
117  @param    addr address and transfer direction of I2C device
118  @retval   0   device accessible
119  @retval   1   failed to access device
120  */
121 extern unsigned char i2c_start(unsigned char addr);
122
123
124 /**
125  @brief Issues a repeated start condition and sends address and transfer direction
126
```

```
127  @param   addr address and transfer direction of I2C device
128  @retval  0 device accessible
129  @retval  1 failed to access device
130  */
131 extern unsigned char i2c_rep_start(unsigned char addr);
132 
133 
134 /**
135  @brief Issues a start condition and sends address and transfer direction
136 
137  If device is busy, use ack polling to wait until device ready
138  @param    addr address and transfer direction of I2C device
139  @return   none
140  */
141 extern void i2c_start_wait(unsigned char addr);
142 
143 
144 /**
145  @brief Send one byte to I2C device
146  @param    data  byte to be transfered
147  @retval   0 write successful
148  @retval   1 write failed
149  */
150 extern unsigned char i2c_write(unsigned char data);
151 
152 
153 /**
154  @brief    read one byte from the I2C device, request more data from device
155  @return   byte read from I2C device
156  */
157 extern unsigned char i2c_readAck(void);
158 
159 /**
160  @brief    read one byte from the I2C device, read is followed by a stop condition
161  @return   byte read from I2C device
162  */
163 extern unsigned char i2c_readNak(void);
164 
165 /**
166  @brief    read one byte from the I2C device
167 
168  Implemented as a macro, which calls either i2c_readAck or i2c_readNak
169 
170  @param    ack 1 send ack, request more data from device<br>
171                0 send nak, read is followed by a stop condition
172  @return   byte read from I2C device
173  */
174 extern unsigned char i2c_read(unsigned char ack);
```

```
175
176
177 /*************
178  Functions that I added
179 *************/
180
181 void i2c_read_byte(uint8_t dev_addr, uint8_t reg_addr, uint8_t* data);
182
183 void i2c_write_byte(uint8_t dev_addr, uint8_t reg_addr, uint8_t data);
184
185 void i2c_read_bytes(uint8_t dev_addr, uint8_t first_reg_addr, uint8_t length,
    ↪  uint8_t* data);
186
187 void i2c_write_bytes(uint8_t dev_addr, uint8_t reg_addr, uint8_t length, uint8_t*
    ↪  data);
188
189
190 #define i2c_read(ack)  (ack) ? i2c_readAck() : i2c_readNak();
191
192
193 /**@}*/
194 #endif
195
```

4.3.3 i2c.h

```
1  #ifndef _I2CMASTER_H
2  #define _I2CMASTER_H   1
3  /*************************************************************************
4  * Title:    C include file for the I2C master interface
5  *           (i2cmaster.S or twimaster.c)
6  * Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
7  * File:     $Id: i2cmaster.h,v 1.10 2005/03/06 22:39:57 Peter Exp $
8  * Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
9  * Target:   any AVR device
10 * Usage:    see Doxygen manual
11 *************************************************************************/
12
13 #ifdef DOXYGEN
14 /**
15  @defgroup pfleury_ic2master I2C Master library
16  @code #include <i2cmaster.h> @endcode
17
18  @brief I2C (TWI) Master Software Library
19
20  Basic routines for communicating with I2C slave devices. This single master
```

```
21 implementation is limited to one bus master on the I2C bus.
22
23 This I2c library is implemented as a compact assembler software implementation of
   ↪ the I2C protocol
24 which runs on any AVR (i2cmaster.S) and as a TWI hardware interface for all AVR with
   ↪ built-in TWI hardware (twimaster.c).
25 Since the API for these two implementations is exactly the same, an application can
   ↪ be linked either against the
26 software I2C implementation or the hardware I2C implementation.
27
28 Use 4.7k pull-up resistor on the SDA and SCL pin.
29
30 Adapt the SCL and SDA port and pin definitions and eventually the delay routine in
   ↪ the module
31 i2cmaster.S to your target when using the software I2C implementation !
32
33 Adjust the  CPU clock frequence F_CPU in twimaster.c or in the Makfile when using
   ↪ the TWI hardware implementaion.
34
35 @note
36    The module i2cmaster.S is based on the Atmel Application Note AVR300, corrected
      ↪ and adapted
37    to GNU assembler and AVR-GCC C call interface.
38    Replaced the incorrect quarter period delays found in AVR300 with
39    half period delays.
40
41 @author Peter Fleury pfleury@gmx.ch  http://jump.to/fleury
42
43 @par API Usage Example
44  The following code shows typical usage of this library, see example
    ↪ test_i2cmaster.c
45
46 @code
47
48 #include <i2cmaster.h>
49
50
51 #define Dev24C02  0xA2      // device address of EEPROM 24C02, see datasheet
52
53 int main(void)
54 {
55     unsigned char ret;
56
57     i2c_init();                             // initialize I2C library
58
59     // write 0x75 to EEPROM address 5 (Byte Write)
60     i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode
61     i2c_write(0x05);                        // write address = 5
```

```
62      i2c_write(0x75);                        // write value 0x75 to EEPROM
63      i2c_stop();                             // set stop conditon = release bus
64
65
66      // read previously written value back from EEPROM address 5
67      i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode
68
69      i2c_write(0x05);                        // write address = 5
70      i2c_rep_start(Dev24C02+I2C_READ);       // set device address and read mode
71
72      ret = i2c_readNak();                    // read one byte from EEPROM
73      i2c_stop();
74
75      for(;;);
76  }
77  @endcode
78
79 */
80 #endif /* DOXYGEN */
81
82 /**@{*/
83
84 #if (__GNUC__ * 100 + __GNUC_MINOR__) < 304
85 #error "This library requires AVR-GCC 3.4 or later, update to newer AVR-GCC compiler
   ↪  !"
86 #endif
87
88 #include <avr/io.h>
89
90 /** defines the data direction (reading from I2C device) in
   ↪  i2c_start(),i2c_rep_start() */
91 #define I2C_READ    1
92
93 /** defines the data direction (writing to I2C device) in i2c_start(),i2c_rep_start()
   ↪  */
94 #define I2C_WRITE   0
95
96
97 /**
98  @brief initialize the I2C master interace. Need to be called only once
99  @param  void
100  @return none
101  */
102 extern void i2c_init(void);
103
104
105 /**
106  @brief Terminates the data transfer and releases the I2C bus
```

```
107  @param void
108  @return none
109  */
110 extern void i2c_stop(void);
111 
112 
113 /**
114  @brief Issues a start condition and sends address and transfer direction
115 
116  @param    addr address and transfer direction of I2C device
117  @retval   0   device accessible
118  @retval   1   failed to access device
119  */
120 extern unsigned char i2c_start(unsigned char addr);
121 
122 
123 /**
124  @brief Issues a repeated start condition and sends address and transfer direction
125 
126  @param   addr address and transfer direction of I2C device
127  @retval  0 device accessible
128  @retval  1 failed to access device
129  */
130 extern unsigned char i2c_rep_start(unsigned char addr);
131 
132 
133 /**
134  @brief Issues a start condition and sends address and transfer direction
135 
136  If device is busy, use ack polling to wait until device ready
137  @param    addr address and transfer direction of I2C device
138  @return   none
139  */
140 extern void i2c_start_wait(unsigned char addr);
141 
142 
143 /**
144  @brief Send one byte to I2C device
145  @param    data  byte to be transfered
146  @retval   0 write successful
147  @retval   1 write failed
148  */
149 extern unsigned char i2c_write(unsigned char data);
150 
151 
152 /**
153  @brief    read one byte from the I2C device, request more data from device
154  @return   byte read from I2C device
```

```
 */
extern unsigned char i2c_readAck(void);

/**
 @brief    read one byte from the I2C device, read is followed by a stop condition
 @return   byte read from I2C device
 */
extern unsigned char i2c_readNak(void);

/**
 @brief    read one byte from the I2C device

 Implemented as a macro, which calls either i2c_readAck or i2c_readNak

 @param    ack 1 send ack, request more data from device<br>
               0 send nak, read is followed by a stop condition
 @return   byte read from I2C device
 */
extern unsigned char i2c_read(unsigned char ack);


/*************
 Functions that I added
*************/

void i2c_read_byte(uint8_t dev_addr, uint8_t reg_addr, uint8_t* data);

void i2c_write_byte(uint8_t dev_addr, uint8_t reg_addr, uint8_t data);

void i2c_read_bytes(uint8_t dev_addr, uint8_t first_reg_addr, uint8_t length,
↪  uint8_t* data);

void i2c_write_bytes(uint8_t dev_addr, uint8_t reg_addr, uint8_t length, uint8_t*
↪  data);


#define i2c_read(ack)  (ack) ? i2c_readAck() : i2c_readNak();


/**@}*/
#endif
```

4.3.4 mpu6050.c

```

```

```c
#include <inttypes.h>
#include <stdint.h>
#include "i2c.h"
#include "mpu6050_reg.h"
#include "mpu6050.h"


//start mpu6050 over I2C
//return 0x68(device address with AD0 low),
//return 0 if error
uint8_t mpu6050_start(void){

        uint8_t res;
        res = i2c_start(MPU6050_ADDRESS+I2C_WRITE);
        if(~res){
                i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_WHO_AM_I, &res);

                if(res==0x68){
                        return res;
                }else{
                        return 0;
                }
        }else{
                return 0;
        }
}



//configure important settings in mpu6050
//subject to change app(ilcation) by app
void mpu6050_init(void){

        i2c_write_byte(MPU6050_ADDRESS, MPU6050_RA_PWR_MGMT_1, 0x00); //exit sleep
        ↪  mode
        i2c_write_byte(MPU6050_ADDRESS, MPU6050_RA_CONFIG, 0x01); // LPF, bandwidth =
        ↪  184(accel) and 188(gyro)
        i2c_write_byte(MPU6050_ADDRESS, MPU6050_RA_GYRO_CONFIG, 1<<4); // gyro ADC
        ↪  scale: 1000 deg/s
        i2c_write_byte(MPU6050_ADDRESS, MPU6050_RA_ACCEL_CONFIG, 0x00); //accel ADC
        ↪  scale: 2 g

        i2c_write_byte(MPU6050_ADDRESS, MPU6050_RA_INT_ENABLE, 0x00); //enable data
        ↪  ready interrupt
        i2c_write_byte(MPU6050_ADDRESS, MPU6050_RA_SIGNAL_PATH_RESET, 0x00); //don't
        ↪  reset signal path

}
```

```
45
46
47
48 // read gyro X, Y, Z all at once, high- & low-8-bits combined
49 // return int16_t (signed) in buff
50 //buff must have at least 3 available places
51 //no error handling for too small buff
52 void mpu6050_read_gyro_ALL(int16_t * buff){
53
54         uint8_t tmp[2];
55
56         mpu6050_read_gyro_X(tmp);
57         buff[0] = (tmp[0]<<8)|(tmp[1]);
58         mpu6050_read_gyro_Y(tmp);
59         buff[1] = (tmp[0]<<8)|(tmp[1]);
60         mpu6050_read_gyro_Z(tmp);
61         buff[2] = (tmp[0]<<8)|(tmp[1]);
62 }
63
64
65 // read accel X, Y, Z all at once, high- & low-8-bits combined
66 // return int16_t (signed) in buff
67 //buff must have at least 3 available places
68 //no error handling for too small buff
69 void mpu6050_read_accel_ALL(int16_t * buff){
70
71         uint8_t tmp[2];
72
73         mpu6050_read_accel_X(tmp);
74         buff[0] = (tmp[0]<<8)|(tmp[1]);
75         mpu6050_read_accel_Y(tmp);
76         buff[1] = (tmp[0]<<8)|(tmp[1]);
77         mpu6050_read_accel_Z(tmp);
78         buff[2] = (tmp[0]<<8)|(tmp[1]);
79 }
80
81
82
83
84
85 //read gyro X, high- & low-8-bits separated, high first
86 //buff must have at least 2 available places
87 //no error handling for too small buff
88 void mpu6050_read_gyro_X(uint8_t * buff){
89
90         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_GYRO_XOUT_H, buff);
91         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_GYRO_XOUT_L, buff+1);
92 }
```

```
93
94  //read gyro Y, high- & low-8-bits separated, high first
95  //buff must have at least 2 available places
96  //no error handling for too small buff
97  void mpu6050_read_gyro_Y(uint8_t * buff){
98
99          i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_GYRO_YOUT_H, buff);
100         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_GYRO_YOUT_L, buff+1);
101 }
102
103 //read gyro Z, high- & low-8-bits separated, high first
104 //buff must have at least 2 available places
105 //no error handling for too small buff
106 void mpu6050_read_gyro_Z(uint8_t * buff){
107
108         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_GYRO_ZOUT_H, buff);
109         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_GYRO_ZOUT_L, buff+1);
110 }
111
112
113
114 //read accel X, high- & low-8-bits separated, high first
115 //buff must have at least 2 available places
116 //no error handling for too small buff
117 void mpu6050_read_accel_X(uint8_t * buff){
118
119         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_ACCEL_XOUT_H, buff);
120         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_ACCEL_XOUT_L, buff+1);
121 }
122
123 //read accel Y, high- & low-8-bits separated, high first
124 //buff must have at least 2 available places
125 //no error handling for too small buff
126 void mpu6050_read_accel_Y(uint8_t * buff){
127
128         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_ACCEL_YOUT_H, buff);
129         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_ACCEL_YOUT_L, buff+1);
130 }
131
132 //read accel Z, high- & low-8-bits separated, high first
133 //buff must have at least 2 available places
134 //no error handling for too small buff
135 void mpu6050_read_accel_Z(uint8_t * buff){
136
137         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_ACCEL_ZOUT_H, buff);
138         i2c_read_byte(MPU6050_ADDRESS, MPU6050_RA_ACCEL_ZOUT_L, buff+1);
139 }
```

```
140
```

4.3.5 mpu6050.h

```
1  /*
2          useful functions to manipulate mpu6050
3  */
4
5  #ifndef MPU6050
6  #define MPU6050
7
8
9  //start mpu6050 over I2C
10 //return 0x68(device address with AD0 low),
11 //return 0 if error
12 uint8_t mpu6050_start(void);
13
14
15 //configure important settings in mpu6050
16 //subject to change app(ilcation) by app
17 void mpu6050_init(void);
18
19
20 // read gyro/accel X, Y, Z all at once, high- & low-8-bits combined
21 // return int16_t (signed) in buff
22 // buff must have at least 3 available places
23 // data sequence: (buff)-->X, (buff+1)-->Y, (buff+2)-->Z
24 // no error handling for too small buff
25 void mpu6050_read_gyro_ALL(int16_t * buff);
26 void mpu6050_read_accel_ALL(int16_t * buff);
27
28
29 //read gyro/accel X, Y, Z, high- & low-8-bits separated, high first
30 //buff must have at least 2 available places
31 //no error handling for too small buff
32 void mpu6050_read_gyro_X(uint8_t * buff);
33 void mpu6050_read_gyro_Y(uint8_t * buff);
34 void mpu6050_read_gyro_Z(uint8_t * buff);
35 void mpu6050_read_accel_X(uint8_t * buff);
36 void mpu6050_read_accel_Y(uint8_t * buff);
37 void mpu6050_read_accel_Z(uint8_t * buff);
38
39
40
41 #endif
```

```

```

4.3.6 mpu6050_reg.h

```
/*
        define mpu6050 register addresses here
*/
#ifndef MPU6050_REG
#define MPU6050_REG

#define MPU6050_ADDRESS 0b11010010 // Address with end write bit
#define MPU6050_RA_XG_OFFS_TC 0x00 //[7] PWR_MODE, [6:1] XG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_YG_OFFS_TC 0x01 //[7] PWR_MODE, [6:1] YG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_ZG_OFFS_TC 0x02 //[7] PWR_MODE, [6:1] ZG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_X_FINE_GAIN 0x03 //[7:0] X_FINE_GAIN
#define MPU6050_RA_Y_FINE_GAIN 0x04 //[7:0] Y_FINE_GAIN
#define MPU6050_RA_Z_FINE_GAIN 0x05 //[7:0] Z_FINE_GAIN
#define MPU6050_RA_XA_OFFS_H 0x06 //[15:0] XA_OFFS
#define MPU6050_RA_XA_OFFS_L_TC 0x07
#define MPU6050_RA_YA_OFFS_H 0x08 //[15:0] YA_OFFS
#define MPU6050_RA_YA_OFFS_L_TC 0x09
#define MPU6050_RA_ZA_OFFS_H 0x0A //[15:0] ZA_OFFS
#define MPU6050_RA_ZA_OFFS_L_TC 0x0B
#define MPU6050_RA_XG_OFFS_USRH 0x13 //[15:0] XG_OFFS_USR
#define MPU6050_RA_XG_OFFS_USRL 0x14
#define MPU6050_RA_YG_OFFS_USRH 0x15 //[15:0] YG_OFFS_USR
#define MPU6050_RA_YG_OFFS_USRL 0x16
#define MPU6050_RA_ZG_OFFS_USRH 0x17 //[15:0] ZG_OFFS_USR
#define MPU6050_RA_ZG_OFFS_USRL 0x18
#define MPU6050_RA_SMPLRT_DIV 0x19
#define MPU6050_RA_CONFIG 0x1A
#define MPU6050_RA_GYRO_CONFIG 0x1B
#define MPU6050_RA_ACCEL_CONFIG 0x1C
#define MPU6050_RA_FF_THR 0x1D
#define MPU6050_RA_FF_DUR 0x1E
#define MPU6050_RA_MOT_THR 0x1F
#define MPU6050_RA_MOT_DUR 0x20
#define MPU6050_RA_ZRMOT_THR 0x21
#define MPU6050_RA_ZRMOT_DUR 0x22
#define MPU6050_RA_FIFO_EN 0x23
#define MPU6050_RA_I2C_MST_CTRL 0x24
#define MPU6050_RA_I2C_SLV0_ADDR 0x25
#define MPU6050_RA_I2C_SLV0_REG 0x26
#define MPU6050_RA_I2C_SLV0_CTRL 0x27
#define MPU6050_RA_I2C_SLV1_ADDR 0x28
#define MPU6050_RA_I2C_SLV1_REG 0x29
```

```
43 #define MPU6050_RA_I2C_SLV1_CTRL 0x2A
44 #define MPU6050_RA_I2C_SLV2_ADDR 0x2B
45 #define MPU6050_RA_I2C_SLV2_REG 0x2C
46 #define MPU6050_RA_I2C_SLV2_CTRL 0x2D
47 #define MPU6050_RA_I2C_SLV3_ADDR 0x2E
48 #define MPU6050_RA_I2C_SLV3_REG 0x2F
49 #define MPU6050_RA_I2C_SLV3_CTRL 0x30
50 #define MPU6050_RA_I2C_SLV4_ADDR 0x31
51 #define MPU6050_RA_I2C_SLV4_REG 0x32
52 #define MPU6050_RA_I2C_SLV4_DO 0x33
53 #define MPU6050_RA_I2C_SLV4_CTRL 0x34
54 #define MPU6050_RA_I2C_SLV4_DI 0x35
55 #define MPU6050_RA_I2C_MST_STATUS 0x36
56 #define MPU6050_RA_INT_PIN_CFG 0x37
57 #define MPU6050_RA_INT_ENABLE 0x38
58 #define MPU6050_RA_DMP_INT_STATUS 0x39
59 #define MPU6050_RA_INT_STATUS 0x3A
60 #define MPU6050_RA_ACCEL_XOUT_H 0x3B
61 #define MPU6050_RA_ACCEL_XOUT_L 0x3C
62 #define MPU6050_RA_ACCEL_YOUT_H 0x3D
63 #define MPU6050_RA_ACCEL_YOUT_L 0x3E
64 #define MPU6050_RA_ACCEL_ZOUT_H 0x3F
65 #define MPU6050_RA_ACCEL_ZOUT_L 0x40
66 #define MPU6050_RA_TEMP_OUT_H 0x41
67 #define MPU6050_RA_TEMP_OUT_L 0x42
68 #define MPU6050_RA_GYRO_XOUT_H 0x43
69 #define MPU6050_RA_GYRO_XOUT_L 0x44
70 #define MPU6050_RA_GYRO_YOUT_H 0x45
71 #define MPU6050_RA_GYRO_YOUT_L 0x46
72 #define MPU6050_RA_GYRO_ZOUT_H 0x47
73 #define MPU6050_RA_GYRO_ZOUT_L 0x48
74 #define MPU6050_RA_EXT_SENS_DATA_00 0x49
75 #define MPU6050_RA_EXT_SENS_DATA_01 0x4A
76 #define MPU6050_RA_EXT_SENS_DATA_02 0x4B
77 #define MPU6050_RA_EXT_SENS_DATA_03 0x4C
78 #define MPU6050_RA_EXT_SENS_DATA_04 0x4D
79 #define MPU6050_RA_EXT_SENS_DATA_05 0x4E
80 #define MPU6050_RA_EXT_SENS_DATA_06 0x4F
81 #define MPU6050_RA_EXT_SENS_DATA_07 0x50
82 #define MPU6050_RA_EXT_SENS_DATA_08 0x51
83 #define MPU6050_RA_EXT_SENS_DATA_09 0x52
84 #define MPU6050_RA_EXT_SENS_DATA_10 0x53
85 #define MPU6050_RA_EXT_SENS_DATA_11 0x54
86 #define MPU6050_RA_EXT_SENS_DATA_12 0x55
87 #define MPU6050_RA_EXT_SENS_DATA_13 0x56
88 #define MPU6050_RA_EXT_SENS_DATA_14 0x57
89 #define MPU6050_RA_EXT_SENS_DATA_15 0x58
90 #define MPU6050_RA_EXT_SENS_DATA_16 0x59
```

```
#define MPU6050_RA_EXT_SENS_DATA_17 0x5A
#define MPU6050_RA_EXT_SENS_DATA_18 0x5B
#define MPU6050_RA_EXT_SENS_DATA_19 0x5C
#define MPU6050_RA_EXT_SENS_DATA_20 0x5D
#define MPU6050_RA_EXT_SENS_DATA_21 0x5E
#define MPU6050_RA_EXT_SENS_DATA_22 0x5F
#define MPU6050_RA_EXT_SENS_DATA_23 0x60
#define MPU6050_RA_MOT_DETECT_STATUS 0x61
#define MPU6050_RA_I2C_SLV0_DO 0x63
#define MPU6050_RA_I2C_SLV1_DO 0x64
#define MPU6050_RA_I2C_SLV2_DO 0x65
#define MPU6050_RA_I2C_SLV3_DO 0x66
#define MPU6050_RA_I2C_MST_DELAY_CTRL 0x67
#define MPU6050_RA_SIGNAL_PATH_RESET 0x68
#define MPU6050_RA_MOT_DETECT_CTRL 0x69
#define MPU6050_RA_USER_CTRL 0x6A
#define MPU6050_RA_PWR_MGMT_1 0x6B
#define MPU6050_RA_PWR_MGMT_2 0x6C
#define MPU6050_RA_BANK_SEL 0x6D
#define MPU6050_RA_MEM_START_ADDR 0x6E
#define MPU6050_RA_MEM_R_W 0x6F
#define MPU6050_RA_DMP_CFG_1 0x70
#define MPU6050_RA_DMP_CFG_2 0x71
#define MPU6050_RA_FIFO_COUNTH 0x72
#define MPU6050_RA_FIFO_COUNTL 0x73
#define MPU6050_RA_FIFO_R_W 0x74
#define MPU6050_RA_WHO_AM_I 0x75


#endif
```

4.3.7 uart.c

```


#define F_CPU 16000000UL

#include "uart.h"


// initialize UART
void uart_init() {

  UBRR0H = UBRRH_VALUE;
  UBRR0L = UBRRL_VALUE;

```

```
14    UCSR0A &= ~(_BV(U2X0));
15
16    UCSR0C = _BV(UCSZ01) | _BV(UCSZ00); // 8-bit data
17    UCSR0B = _BV(RXEN0) | _BV(TXEN0) | _BV(RXCIE0);   // Enable RX and TX
18
19
20  }
21
22  void uart_putchar(char data) {
23      while ( !( UCSR0A & (1<<UDRE0)) );
24      UDR0 = data;
25  }
26
27  char uart_getchar(void) {
28      while ( !(UCSR0A & (1<<RXC0)) ) ;
29      return UDR0;
30  }
31
32  void uart_putstring(char * s) {
33    int i = 0;
34    while(s[i]!='\0'){
35      uart_putchar(s[i]);
36      ++i;
37    };
38
39  }
40
41  void uart_putint16(int16_t data) {
42          uint8_t a = data & 0xFF;
43          uart_putchar(a);
44          a = data >> 8;
45          uart_putchar(a);
46
47  }
48
49
50  void uart_putdouble(double data){
51          char * a = &data;
52          uart_putchar(*(a++));
53          uart_putchar(*(a++));
54          uart_putchar(*(a++));
55          uart_putchar(*a);
56
57  }
58
```

4.3.8 uart.h

```
1
2  #include <inttypes.h>
3  #include <stdint.h>
4  #include <avr/io.h>
5  #include <util/setbaud.h>
6
7
8  // initialize UART
9  void uart_init();
10
11 void uart_putchar(char data);
12
13 char uart_getchar(void);
14
15 void uart_putstring(char * s);
16
17 //send int16 as 2 chars
18 //care needed on PC side
19 void uart_putint16(int16_t data);
20
21 //send double as 8 chars
22 //care needed on PC side
23 void uart_putdouble(double data);
24
```

4.4 source code for nRF24L01 module

4.4.1 nrf24l01-mnemonics.h

```
1
2  /**
3   *  From the specifications available in the download section of
4   *  http://www.nordicsemi.com/kor/Products/2.4GHz-RF/nRF24L01P
5   */
6
7  #ifndef _NRF24L01_MNEMONICS_H
8  #define _NRF24L01_MNEMONICS_H
9
10
11 // SPI Commands
12
13 #define R_REGISTER          0x00 // 000A AAAA
14 #define W_REGISTER          0x20 // 001A AAAA
15 #define R_RX_PAYLOAD        0x61 // 0110 0001
```

```
16 #define W_TX_PAYLOAD         0xA0 // 1010 0000
17 #define FLUSH_TX             0xE1 // 1110 0001
18 #define FLUSH_RX             0xE2 // 1110 0010
19 #define REUSE_TX_PL          0xE3 // 1110 0011
20 #define R_RX_PL_WID          0x60 // 0110 0000
21 #define W_ACK_PAYLOAD        0xA8 // 1010 1PPP
22 #define W_TX_PAYLOAD_NOACK   0xB0 // 1011 0000
23 #define NOP                  0xFF // 1111 1111
24
25
26 // Register Map
27
28 #define CONFIG      0x00
29 #define MASK_RX_DR  6
30 #define MASK_TX_DS  5
31 #define MASK_MAX_RT 4
32 #define EN_CRC      3
33 #define CRCO        2
34 #define PWR_UP      1
35 #define PRIM_RX     0
36
37 #define EN_AA       0x01
38 #define ENAA_P5     5
39 #define ENAA_P4     4
40 #define ENAA_P3     3
41 #define ENAA_P2     2
42 #define ENAA_P1     1
43 #define ENAA_P0     0
44
45 #define EN_RXADDR   0x02
46 #define ERX_P5      5
47 #define ERX_P4      4
48 #define ERX_P3      3
49 #define ERX_P2      2
50 #define ERX_P1      1
51 #define ERX_P0      0
52
53 #define SETUP_AW    0x03
54 #define AW          0
55
56 #define SETUP_RETR  0x04
57 #define ARD         4
58 #define ARC         0
59
60 #define RF_CH       0x05
61
62 #define RF_SETUP    0x06
63 #define CONT_WAVE   7
```

```
64 #define RF_DR_LOW   5
65 #define PLL_LOCK    4
66 #define RF_DR_HIGH  3
67 #define RF_PWR      1
68
69 #define STATUS              0x07
70 #define RX_DR               6
71 #define TX_DS               5
72 #define MAX_RT              4
73 #define RX_P_NO_MASK        0x0E
74 #define STATUS_TX_FULL      0
75
76 #define OBSERVE_TX  0x08
77 #define PLOS_CNT    4
78 #define ARC_CNT     0
79
80 #define RPD         0x09
81
82 #define RX_ADDR_P0  0x0A
83 #define RX_ADDR_P1  0x0B
84 #define RX_ADDR_P2  0x0C
85 #define RX_ADDR_P3  0x0D
86 #define RX_ADDR_P4  0x0E
87 #define RX_ADDR_P5  0x0F
88
89 #define TX_ADDR     0x10
90
91 #define RX_PW_P0    0x11
92 #define RX_PW_P1    0x12
93 #define RX_PW_P2    0x13
94 #define RX_PW_P3    0x14
95 #define RX_PW_P4    0x15
96 #define RX_PW_P5    0x16
97
98 #define FIFO_STATUS         0x17
99 #define TX_REUSE            6
100 #define FIFO_TX_FULL        5
101 #define TX_EMPTY            4
102 #define RX_FULL             1
103 #define RX_EMPTY            0
104
105 #define DYNPD       0x1C
106 #define DPL_P5      5
107 #define DPL_P4      4
108 #define DPL_P3      3
109 #define DPL_P2      2
110 #define DPL_P1      1
111 #define DPL_P0      0
```

```
112
113 #define FEATURE     0x1D
114 #define EN_DPL      2
115 #define EN_ACK_PAY  1
116 #define EN_DYN_ACK  0
117
118 #endif
119
```

4.4.2 nrf24l01.c

```
1
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <avr/io.h>
5  #include <util/delay.h>
6  #include <string.h>
7  #include "nrf24l01.h"
8  #include "nrf24l01-mnemonics.h"
9
10
11 static void copy_address(uint8_t *source, uint8_t *destination);
12 inline static void set_as_output(gpio_pin pin);
13 inline static void set_high(gpio_pin pin);
14 inline static void set_low(gpio_pin pin);
15 static void spi_init(nRF24L01 *rf);
16 static uint8_t spi_transfer(uint8_t data);
17
18
19 nRF24L01 *nRF24L01_init(void) {
20     nRF24L01 *rf = malloc(sizeof(nRF24L01));
21     memset(rf, 0, sizeof(nRF24L01));
22     return rf;
23 }
24
25 void nRF24L01_begin(nRF24L01 *rf) {
26     set_as_output(rf->ss);
27     set_as_output(rf->ce);
28
29     set_high(rf->ss);
30     set_low(rf->ce);
31
32     spi_init(rf);
33
34     nRF24L01_send_command(rf, FLUSH_RX, NULL, 0);
35     nRF24L01_send_command(rf, FLUSH_TX, NULL, 0);
```

```
36      nRF24L01_clear_interrupts(rf);
37
38      uint8_t data;
39      data = _BV(EN_CRC) | _BV(CRCO) | _BV(PWR_UP) | _BV(PRIM_RX);
40      nRF24L01_write_register(rf, CONFIG, &data, 1);
41
42      // enable Auto Acknowlegde on all pipes
43      data = _BV(ENAA_P0) | _BV(ENAA_P1) | _BV(ENAA_P2)
44           | _BV(ENAA_P3) | _BV(ENAA_P4) | _BV(ENAA_P5);
45      nRF24L01_write_register(rf, EN_AA, &data, 1);
46
47      // enable Dynamic Payload on al pipes
48      data = _BV(DPL_P0) | _BV(DPL_P1) | _BV(DPL_P2)
49           | _BV(DPL_P3) | _BV(DPL_P4) | _BV(DPL_P5);
50      nRF24L01_write_register(rf, DYNPD, &data, 1);
51
52      // enable Dynamic Payload (global)
53      data = _BV(EN_DPL);
54      nRF24L01_write_register(rf, FEATURE, &data, 1);
55
56      // disable all rx addresses
57      data = 0;
58      nRF24L01_write_register(rf, EN_RXADDR, &data, 1);
59  }
60
61  uint8_t nRF24L01_send_command(nRF24L01 *rf, uint8_t command, void *data,
62      size_t length) {
63      set_low(rf->ss);
64
65      rf->status = spi_transfer(command);
66      for (unsigned int i = 0; i < length; i++)
67          ((uint8_t*)data)[i] = spi_transfer(((uint8_t*)data)[i]);
68
69      set_high(rf->ss);
70
71      return rf->status;
72  }
73
74  uint8_t nRF24L01_write_register(nRF24L01 *rf, uint8_t reg_address, void *data,
75      size_t length) {
76      return nRF24L01_send_command(rf, W_REGISTER | reg_address, data, length);
77  }
78
79  uint8_t nRF24L01_read_register(nRF24L01 *rf, uint8_t reg_address, void *data,
80      size_t length) {
81      return nRF24L01_send_command(rf, R_REGISTER | reg_address, data, length);
82  }
83
```

```
84  uint8_t nRF24L01_no_op(nRF24L01 *rf) {
85      return nRF24L01_send_command(rf, NOP, NULL, 0);
86  }
87
88  uint8_t nRF24L01_update_status(nRF24L01 *rf) {
89      return nRF24L01_no_op(rf);
90  }
91
92  uint8_t nRF24L01_get_status(nRF24L01 *rf) {
93      return rf->status;
94  }
95
96  bool nRF24L01_data_received(nRF24L01 *rf) {
97      set_low(rf->ce);
98      nRF24L01_update_status(rf);
99      return nRF24L01_pipe_number_received(rf) >= 0;
100 }
101
102 void nRF24L01_listen(nRF24L01 *rf, int pipe, uint8_t *address) {
103     uint8_t addr[5];
104     copy_address(address, addr);
105
106     nRF24L01_write_register(rf, RX_ADDR_P0 + pipe, addr, 5);
107
108     uint8_t current_pipes;
109     nRF24L01_read_register(rf, EN_RXADDR, &current_pipes, 1);
110     current_pipes |= _BV(pipe);
111     nRF24L01_write_register(rf, EN_RXADDR, &current_pipes, 1);
112
113     set_high(rf->ce);
114 }
115
116 bool nRF24L01_read_received_data(nRF24L01 *rf, nRF24L01Message *message) {
117     message->pipe_number = nRF24L01_pipe_number_received(rf);
118     nRF24L01_clear_receive_interrupt(rf);
119     if (message->pipe_number < 0) {
120         message->length = 0;
121         return false;
122     }
123
124     nRF24L01_read_register(rf, R_RX_PL_WID, &message->length, 1);
125
126     if (message->length > 0) {
127         nRF24L01_send_command(rf, R_RX_PAYLOAD, &message->data,
128             message->length);
129     }
130
131     return true;
```

```
}

int nRF24L01_pipe_number_received(nRF24L01 *rf) {
    int pipe_number = (rf->status & RX_P_NO_MASK) >> 1;
    return pipe_number <= 5 ? pipe_number : -1;
}

void nRF24L01_transmit(nRF24L01 *rf, void *address, nRF24L01Message *msg) {
    nRF24L01_clear_transmit_interrupts(rf);
    uint8_t addr[5];
    copy_address((uint8_t *)address, addr);
    nRF24L01_write_register(rf, TX_ADDR, addr, 5);
    copy_address((uint8_t *)address, addr);
    nRF24L01_write_register(rf, RX_ADDR_P0, addr, 5);
    nRF24L01_send_command(rf, W_TX_PAYLOAD, &msg->data, msg->length);
    uint8_t config;
    nRF24L01_read_register(rf, CONFIG, &config, 1);
    config &= ~_BV(PRIM_RX);
    nRF24L01_write_register(rf, CONFIG, &config, 1);
    set_high(rf->ce);
}

int nRF24L01_transmit_success(nRF24L01 *rf) {
    set_low(rf->ce);
    nRF24L01_update_status(rf);
    int success;
    if (rf->status & _BV(TX_DS)) success = 0;
    else if (rf->status & _BV(MAX_RT)) success = -1;
    else success = -2;
    nRF24L01_clear_transmit_interrupts(rf);
    uint8_t config;
    nRF24L01_read_register(rf, CONFIG, &config, 1);
    config |= _BV(PRIM_RX);
    nRF24L01_write_register(rf, CONFIG, &config, 1);
    return success;
}

void nRF24L01_flush_transmit_message(nRF24L01 *rf) {
    nRF24L01_send_command(rf, FLUSH_TX, NULL, 0);
}

void nRF24L01_retry_transmit(nRF24L01 *rf) {
    uint8_t config;
    nRF24L01_read_register(rf, CONFIG, &config, 1);
    config &= ~_BV(PRIM_RX);
    nRF24L01_write_register(rf, CONFIG, &config, 1);
    set_high(rf->ce);
}
```

```
180
181 void nRF24L01_clear_interrupts(nRF24L01 *rf) {
182     uint8_t data = _BV(RX_DR) | _BV(TX_DS) | _BV(MAX_RT);
183     nRF24L01_write_register(rf, STATUS, &data, 1);
184 }
185
186 void nRF24L01_clear_transmit_interrupts(nRF24L01 *rf) {
187     uint8_t data = _BV(TX_DS) | _BV(MAX_RT);
188     nRF24L01_write_register(rf, STATUS, &data, 1);
189 }
190
191 void nRF24L01_clear_receive_interrupt(nRF24L01 *rf) {
192     uint8_t data = _BV(RX_DR) | rf->status;
193     nRF24L01_write_register(rf, STATUS, &data, 1);
194 }
195
196 static void copy_address(uint8_t *source, uint8_t *destination) {
197     for (int i = 0; i < 5; i++)
198         destination[i] = source[i];
199 }
200
201 inline static void set_as_output(gpio_pin pin) {
202     volatile uint8_t *ddr = pin.port - 1;
203     *ddr |= _BV(pin.pin);
204 }
205
206 inline static void set_as_input(gpio_pin pin) {
207     volatile uint8_t *ddr = pin.port - 1;
208     *ddr &= ~_BV(pin.pin);
209 }
210
211 inline static void set_high(gpio_pin pin) {
212     *pin.port |= _BV(pin.pin);
213 }
214
215 inline static void set_low(gpio_pin pin) {
216     *pin.port &= ~_BV(pin.pin);
217 }
218
219 static void spi_init(nRF24L01 *rf) {
220     // set as master
221     SPCR |= _BV(MSTR);
222     // enable SPI
223     SPCR |= _BV(SPE);
224     // MISO pin automatically overrides to input
225     set_as_output(rf->sck);
226     set_as_output(rf->mosi);
227     set_as_input(rf->miso);
```

```
228     // SPI mode 0: Clock Polarity CPOL = 0, Clock Phase CPHA = 0
229     SPCR &= ~_BV(CPOL);
230     SPCR &= ~_BV(CPHA);
231     // Clock 2X speed
232     SPCR &= ~_BV(SPR0);
233     SPCR &= ~_BV(SPR1);
234     SPSR |= _BV(SPI2X);
235     // most significant first (MSB)
236     SPCR &= ~_BV(DORD);
237 }
238 
239 static uint8_t spi_transfer(uint8_t data) {
240     SPDR = data;
241     while (!(SPSR & _BV(SPIF)));
242     return SPDR;
243 }
244 
```

4.4.3 nrf24l01.h

```
1  #include <stdlib.h>
2  #include <stdint.h>
3  #include <stdbool.h>
4  
5  #ifndef _NRF24L01_H
6  #define _NRF24L01_H
7  
8  
9  typedef struct {
10     int pipe_number;
11     uint8_t data[32];
12     uint8_t length;
13 } nRF24L01Message;
14 
15 typedef struct {
16     volatile uint8_t *port;
17     uint8_t pin;
18 } gpio_pin;
19 
20 typedef struct {
21     gpio_pin ss; // slave select
22     gpio_pin ce; // chip enabled
23     gpio_pin sck; // serial clock
24     gpio_pin mosi; // master out slave in
25     gpio_pin miso; // master in slave out
26     uint8_t status;
```

```
27 } nRF24L01;
28
29
30 nRF24L01 *nRF24L01_init(void);
31 void nRF24L01_begin(nRF24L01 *rf);
32 uint8_t nRF24L01_send_command(nRF24L01 *rf, uint8_t command, void *data,
33     size_t length);
34 uint8_t nRF24L01_write_register(nRF24L01 *rf, uint8_t reg_address, void *data,
35     size_t length);
36 uint8_t nRF24L01_read_register(nRF24L01 *rf, uint8_t regAddress, void *data,
37     size_t length);
38 uint8_t nRF24L01_no_op(nRF24L01 *rf);
39 uint8_t nRF24L01_update_status(nRF24L01 *rf);
40 uint8_t nRF24L01_get_status(nRF24L01 *rf);
41 void nRF24L01_listen(nRF24L01 *rf, int pipe, uint8_t *address);
42 bool nRF24L01_data_received(nRF24L01 *rf);
43 bool nRF24L01_read_received_data(nRF24L01 *rf, nRF24L01Message *message);
44 int nRF24L01_pipe_number_received(nRF24L01 *rf);
45 void nRF24L01_transmit(nRF24L01 *rf, void *address, nRF24L01Message *msg);
46 int nRF24L01_transmit_success(nRF24L01 *rf);
47 void nRF24L01_flush_transmit_message(nRF24L01 *rf);
48 void nRF24L01_retry_transmit(nRF24L01 *rf);
49 void nRF24L01_clear_interrupts(nRF24L01 *rf);
50 void nRF24L01_clear_transmit_interrupts(nRF24L01 *rf);
51 void nRF24L01_clear_receive_interrupt(nRF24L01 *rf);
52
53 #endif
54
```

4.5 source code for SSD1306 OLED display

4.5.1 font.c

```
1  /*
2   *  font.c
3   *  i2c
4   *
5   *  Created by Michael Köhler on 16.09.18.
6   *  Copyright 2018 Skie-Systems. All rights reserved.
7   *
8   */
9  #include "font.h"
10
11 const char ssd1306oled_font[][6] PROGMEM = {
12 {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // sp
13 {0x00, 0x00, 0x00, 0x2f, 0x00, 0x00}, // !
```

```
{0x00, 0x00, 0x07, 0x00, 0x07, 0x00}, // "
{0x00, 0x14, 0x7f, 0x14, 0x7f, 0x14}, // #
{0x00, 0x24, 0x2a, 0x7f, 0x2a, 0x12}, // $
{0x00, 0x62, 0x64, 0x08, 0x13, 0x23}, // %
{0x00, 0x36, 0x49, 0x55, 0x22, 0x50}, // &
{0x00, 0x00, 0x05, 0x03, 0x00, 0x00}, // '
{0x00, 0x00, 0x1c, 0x22, 0x41, 0x00}, // (
{0x00, 0x00, 0x41, 0x22, 0x1c, 0x00}, // )
{0x00, 0x14, 0x08, 0x3E, 0x08, 0x14}, // *
{0x00, 0x08, 0x08, 0x3E, 0x08, 0x08}, // +
{0x00, 0x00, 0x00, 0xA0, 0x60, 0x00}, // ,
{0x00, 0x08, 0x08, 0x08, 0x08, 0x08}, // -
{0x00, 0x00, 0x60, 0x60, 0x00, 0x00}, // .
{0x00, 0x20, 0x10, 0x08, 0x04, 0x02}, // /
{0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E}, // 0
{0x00, 0x00, 0x42, 0x7F, 0x40, 0x00}, // 1
{0x00, 0x42, 0x61, 0x51, 0x49, 0x46}, // 2
{0x00, 0x21, 0x41, 0x45, 0x4B, 0x31}, // 3
{0x00, 0x18, 0x14, 0x12, 0x7F, 0x10}, // 4
{0x00, 0x27, 0x45, 0x45, 0x45, 0x39}, // 5
{0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30}, // 6
{0x00, 0x01, 0x71, 0x09, 0x05, 0x03}, // 7
{0x00, 0x36, 0x49, 0x49, 0x49, 0x36}, // 8
{0x00, 0x06, 0x49, 0x49, 0x29, 0x1E}, // 9
{0x00, 0x00, 0x36, 0x36, 0x00, 0x00}, // :
{0x00, 0x00, 0x56, 0x36, 0x00, 0x00}, // ;
{0x00, 0x08, 0x14, 0x22, 0x41, 0x00}, // <
{0x00, 0x14, 0x14, 0x14, 0x14, 0x14}, // =
{0x00, 0x00, 0x41, 0x22, 0x14, 0x08}, // >
{0x00, 0x02, 0x01, 0x51, 0x09, 0x06}, // ?
{0x00, 0x32, 0x49, 0x59, 0x51, 0x3E}, // @
{0x00, 0x7C, 0x12, 0x11, 0x12, 0x7C}, // A
{0x00, 0x7F, 0x49, 0x49, 0x49, 0x36}, // B
{0x00, 0x3E, 0x41, 0x41, 0x41, 0x22}, // C
{0x00, 0x7F, 0x41, 0x41, 0x22, 0x1C}, // D
{0x00, 0x7F, 0x49, 0x49, 0x49, 0x41}, // E
{0x00, 0x7F, 0x09, 0x09, 0x09, 0x01}, // F
{0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A}, // G
{0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F}, // H
{0x00, 0x00, 0x41, 0x7F, 0x41, 0x00}, // I
{0x00, 0x20, 0x40, 0x41, 0x3F, 0x01}, // J
{0x00, 0x7F, 0x08, 0x14, 0x22, 0x41}, // K
{0x00, 0x7F, 0x40, 0x40, 0x40, 0x40}, // L
{0x00, 0x7F, 0x02, 0x0C, 0x02, 0x7F}, // M
{0x00, 0x7F, 0x04, 0x08, 0x10, 0x7F}, // N
{0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E}, // O
{0x00, 0x7F, 0x09, 0x09, 0x09, 0x06}, // P
{0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E}, // Q
```

```
{0x00, 0x7F, 0x09, 0x19, 0x29, 0x46}, // R
{0x00, 0x46, 0x49, 0x49, 0x49, 0x31}, // S
{0x00, 0x01, 0x01, 0x7F, 0x01, 0x01}, // T
{0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F}, // U
{0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F}, // V
{0x00, 0x3F, 0x40, 0x38, 0x40, 0x3F}, // W
{0x00, 0x63, 0x14, 0x08, 0x14, 0x63}, // X
{0x00, 0x07, 0x08, 0x70, 0x08, 0x07}, // Y
{0x00, 0x61, 0x51, 0x49, 0x45, 0x43}, // Z
{0x00, 0x00, 0x7F, 0x41, 0x41, 0x00}, // [
{0x00, 0x55, 0x2A, 0x55, 0x2A, 0x55}, // backslash
{0x00, 0x00, 0x41, 0x41, 0x7F, 0x00}, // ]
{0x00, 0x04, 0x02, 0x01, 0x02, 0x04}, // ^
{0x00, 0x40, 0x40, 0x40, 0x40, 0x40}, // _
{0x00, 0x00, 0x01, 0x02, 0x04, 0x00}, // '
{0x00, 0x20, 0x54, 0x54, 0x54, 0x78}, // a
{0x00, 0x7F, 0x48, 0x44, 0x44, 0x38}, // b
{0x00, 0x38, 0x44, 0x44, 0x44, 0x20}, // c
{0x00, 0x38, 0x44, 0x44, 0x48, 0x7F}, // d
{0x00, 0x38, 0x54, 0x54, 0x54, 0x18}, // e
{0x00, 0x08, 0x7E, 0x09, 0x01, 0x02}, // f
{0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C}, // g
{0x00, 0x7F, 0x08, 0x04, 0x04, 0x78}, // h
{0x00, 0x00, 0x44, 0x7D, 0x40, 0x00}, // i
{0x00, 0x40, 0x80, 0x84, 0x7D, 0x00}, // j
{0x00, 0x7F, 0x10, 0x28, 0x44, 0x00}, // k
{0x00, 0x00, 0x41, 0x7F, 0x40, 0x00}, // l
{0x00, 0x7C, 0x04, 0x18, 0x04, 0x78}, // m
{0x00, 0x7C, 0x08, 0x04, 0x04, 0x78}, // n
{0x00, 0x38, 0x44, 0x44, 0x44, 0x38}, // o
{0x00, 0xFC, 0x24, 0x24, 0x24, 0x18}, // p
{0x00, 0x18, 0x24, 0x24, 0x18, 0xFC}, // q
{0x00, 0x7C, 0x08, 0x04, 0x04, 0x08}, // r
{0x00, 0x48, 0x54, 0x54, 0x54, 0x20}, // s
{0x00, 0x04, 0x3F, 0x44, 0x40, 0x20}, // t
{0x00, 0x3C, 0x40, 0x40, 0x20, 0x7C}, // u
{0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C}, // v
{0x00, 0x3C, 0x40, 0x30, 0x40, 0x3C}, // w
{0x00, 0x44, 0x28, 0x10, 0x28, 0x44}, // x
{0x00, 0x1C, 0xA0, 0xA0, 0xA0, 0x7C}, // y
{0x00, 0x44, 0x64, 0x54, 0x4C, 0x44}, // z
{0x00, 0x00, 0x08, 0x77, 0x41, 0x00}, // {
{0x00, 0x00, 0x00, 0x63, 0x00, 0x00}, // ¦
{0x00, 0x00, 0x41, 0x77, 0x08, 0x00}, // }
{0x00, 0x08, 0x04, 0x08, 0x08, 0x04}, // ~
/* end of normal char-set */
/* put your own signs/chars here, edit special_char too */
/* be sure that your first special char stand here */
```

```
{0x00, 0x3A, 0x40, 0x40, 0x20, 0x7A}, // ü, !!! Important: this must be
↪  special_char[0] !!!
{0x00, 0x3D, 0x40, 0x40, 0x40, 0x3D}, // Ü
{0x00, 0x21, 0x54, 0x54, 0x54, 0x79}, // ä
{0x00, 0x7D, 0x12, 0x11, 0x12, 0x7D}, // Ä
{0x00, 0x39, 0x44, 0x44, 0x44, 0x39}, // ö
{0x00, 0x3D, 0x42, 0x42, 0x42, 0x3D}, // Ö
{0x00, 0x02, 0x05, 0x02, 0x00, 0x00}, // °
{0x00, 0x7E, 0x01, 0x49, 0x55, 0x73}, // ß
{0x00, 0x7C, 0x10, 0x10, 0x08, 0x1C}, // µ
{0x00, 0x30, 0x48, 0x20, 0x48, 0x30}, //
{0x00, 0x5C, 0x62, 0x02, 0x62, 0x5C} //
};
const char special_char[][2] PROGMEM = {
    // define position of special char in font
    // {special char, position in font}
    // be sure that last element of this
    // array are {0xff, 0xff} and first element
    // are {first special char, first element after normal char-set in font}
    {'ü', 95},  // special_char[0]
    {'Ü', 96},
    {'ä', 97},
    {'Ä', 98},
    {'ö', 99},
    {'Ö', 100},
    {'°', 101},
    {'ß', 102},
    {'µ', 103},
    {'', 104},
    {'', 105},
    {0xff, 0xff} // end of table special_char
};

```

4.5.2 font.h

```
/*
 *  font.h
 *  i2c
 *
 *  Created by Michael Köhler on 13.09.18.
 *  Copyright 2018 Skie-Systems. All rights reserved.
 *
 */
#ifndef _font_h_
#define _font_h_
```

```
11 #include <avr/pgmspace.h>
12
13 extern const char ssd1306oled_font[][6] PROGMEM;
14 extern const char special_char[][2] PROGMEM;
15
16 #endif
17
```

4.5.3 i2c.c

```
1
2 //
3 //  i2c.c
4 //  i2c
5 //
6 //  Created by Michael Köhler on 09.10.17.
7 //
8 //
9
10 #include "i2c.h"
11
12 #if defined (__AVR_ATmega328__) || defined(__AVR_ATmega328P__) || \
13 defined(__AVR_ATmega168P__) || defined(__AVR_ATmega168PA__) || \
14 defined(__AVR_ATmega88P__) || \
15 defined(__AVR_ATmega8__) || \
16 defined(__AVR_ATmega48P__) || \
17 defined(__AVR_ATmega1284P__) || \
18 defined (__AVR_ATmega324A__) || defined (__AVR_ATmega324P__) || defined
   ↪  (__AVR_ATmega324PA__) || \
19 defined (__AVR_ATmega644__) || defined (__AVR_ATmega644A__) || defined
   ↪  (__AVR_ATmega644P__) || defined (__AVR_ATmega644PA__) || \
20 defined (__AVR_ATmega1284P__) || \
21 defined (__AVR_ATmega2560__)
22 #if PSC_I2C != 1 && PSC_I2C != 4 && PSC_I2C != 16 && PSC_I2C != 64
23 #error "Wrong prescaler for TWI !"
24 #elif SET_TWBR < 0 || SET_TWBR > 255
25 #error "TWBR out of range, change PSC_I2C or F_I2C !"
26 #endif
27
28 uint8_t I2C_ErrorCode;
29 /************************************************
30  Public Function: i2c_init
31
32  Purpose: Initialise TWI/I2C interface
33
34  Input Parameter: none
```

```
35
36  Return Value: none
37  ********************************************/
38 void i2c_init(void){
39     // set clock
40     switch (PSC_I2C) {
41         case 4:
42             TWSR = 0x1;
43             break;
44         case 16:
45             TWSR = 0x2;
46             break;
47         case 64:
48             TWSR = 0x3;
49             break;
50         default:
51             TWSR = 0x00;
52             break;
53     }
54     TWBR = (uint8_t)SET_TWBR;
55     // enable
56     TWCR = (1 << TWEN);
57 }
58 /*********************************************
59  Public Function: i2c_start
60
61  Purpose: Start TWI/I2C interface
62
63  Input Parameter:
64  - uint8_t i2c_addr: Adress of reciever
65
66  Return Value: none
67  ********************************************/
68 void i2c_start(uint8_t i2c_addr){
69     // i2c start
70     TWCR = (1 << TWINT)|(1 << TWSTA)|(1 << TWEN);
71         uint16_t timeout = F_CPU/F_I2C*2.0;
72     while((TWCR & (1 << TWINT)) == 0 &&
73                 timeout !=0){
74                 timeout--;
75                 if(timeout == 0){
76                         I2C_ErrorCode |= (1 << I2C_START);
77                         return;
78                 }
79         };
80     // send adress
81     TWDR = i2c_addr;
82     TWCR = (1 << TWINT)|( 1 << TWEN);
```

```
    timeout = F_CPU/F_I2C*2.0;
    while((TWCR & (1 << TWINT)) == 0 &&
                  timeout !=0){
                timeout--;
                if(timeout == 0){
                        I2C_ErrorCode |= (1 << I2C_SENDADRESS);
                        return;
                }
        };
}
/*********************************************
 Public Function: i2c_stop

 Purpose: Stop TWI/I2C interface

 Input Parameter: none

 Return Value: none
 *********************************************/
void i2c_stop(void){
    // i2c stop
    TWCR = (1 << TWINT)|(1 << TWSTO)|(1 << TWEN);
}
/*********************************************
 Public Function: i2c_byte

 Purpose: Send byte at TWI/I2C interface

 Input Parameter:
 - uint8_t byte: Byte to send to reciever

 Return Value: none
 *********************************************/
void i2c_byte(uint8_t byte){
    TWDR = byte;
    TWCR = (1 << TWINT)|( 1 << TWEN);
    uint16_t timeout = F_CPU/F_I2C*2.0;
    while((TWCR & (1 << TWINT)) == 0 &&
                  timeout !=0){
                timeout--;
                if(timeout == 0){
                        I2C_ErrorCode |= (1 << I2C_BYTE);
                        return;
                }
        };
}
/*********************************************
 Public Function: i2c_readAck
```

```
131
132  Purpose: read acknowledge from TWI/I2C Interface
133
134  Input Parameter: none
135
136  Return Value: uint8_t
137   - TWDR: recieved value at TWI/I2C-Interface, 0 at timeout
138   - 0:    Error at read
139  ********************************************/
140 uint8_t i2c_readAck(void){
141     TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);
142     uint16_t timeout = F_CPU/F_I2C*2.0;
143     while((TWCR & (1 << TWINT)) == 0 &&
144                   timeout !=0){
145                 timeout--;
146                 if(timeout == 0){
147                         I2C_ErrorCode |= (1 << I2C_READACK);
148                         return 0;
149                 }
150         };
151     return TWDR;
152 }
153
154  /*********************************************
155  Public Function: i2c_readNAck
156
157  Purpose: read non-acknowledge from TWI/I2C Interface
158
159  Input Parameter: none
160
161  Return Value: uint8_t
162   - TWDR: recieved value at TWI/I2C-Interface
163   - 0:    Error at read
164  *********************************************/
165 uint8_t i2c_readNAck(void){
166     TWCR = (1<<TWINT)|(1<<TWEN);
167     uint16_t timeout = F_CPU/F_I2C*2.0;
168     while((TWCR & (1 << TWINT)) == 0 &&
169                   timeout !=0){
170                 timeout--;
171                 if(timeout == 0){
172                         I2C_ErrorCode |= (1 << I2C_READNACK);
173             return 0;
174                 }
175         };
176     return TWDR;
177 }
178 #else
```

```
179 #error "Micorcontroller not supported now!"
180 #endif
181
```

4.5.4 i2c.h

```
1
2 //
3 //  i2c.h
4 //  i2c
5 //
6 //  Created by Michael Köhler on 09.10.17.
7 //
8 //
9
10 #ifndef i2c_h
11 #define i2c_h
12
13 #ifdef __cplusplus
14 extern "C" {
15 #endif
16
17 /* TODO: setup i2c/twi */
18 #define F_I2C                       100000UL// clock i2c
19 #define PSC_I2C                       1               // prescaler i2c
20 #define SET_TWBR                (F_CPU/F_I2C-16UL)/(PSC_I2C*2UL)
21
22 #include <stdio.h>
23 #include <avr/io.h>
24
25 extern uint8_t I2C_ErrorCode;               // variable for communication error at
   ↪ twi
26                                                                  // ckeck it
                                                                    ↪ in your
                                                                    ↪ code
27                                                                  // 0 means no
                                                                    ↪ error
28                                                                  // define
                                                                    ↪ bits in
                                                                    ↪ I2C-ErrorCode:
29 #define I2C_START               0                       // bit 0: timeout
   ↪ start-condition
30 #define I2C_SENDADRESS        1                       // bit 0: timeout
   ↪ device-adress
31 #define I2C_BYTE               2                       // bit 0: timeout
   ↪ byte-transmission
```

```
32 #define I2C_READACK                3                        // bit 0: timeout read
   ↪  acknowledge
33 #define I2C_READNACK        4                         // bit 0: timeout read
   ↪  nacknowledge
34
35 void i2c_init(void);                                 // init hw-i2c
36 void i2c_start(uint8_t i2c_addr);        // send i2c_start_condition
37 void i2c_stop(void);                                 // send i2c_stop_condition
38 void i2c_byte(uint8_t byte);                // send data_byte
39
40 uint8_t i2c_readAck(void);           // read byte with ACK
41 uint8_t i2c_readNAck(void);          // read byte with NACK
42
43 #ifdef __cplusplus
44 }
45 #endif
46
47 #endif /* i2c_h */
48
```

4.5.5 lcd.c

```
1
2  /*
3   * This file is part of lcd library for ssd1306/ssd1309/sh1106 oled-display.
4   *
5   * lcd library for ssd1306/ssd1309/sh1106 oled-display is free software: you can
    ↪  redistribute it and/or modify
6   * it under the terms of the GNU General Public License as published by
7   * the Free Software Foundation, either version 3 of the License, or any later
    ↪  version.
8   *
9   * lcd library for ssd1306/ssd1309/sh1106 oled-display is distributed in the hope
    ↪  that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with Foobar.  If not, see <http://www.gnu.org/licenses/>.
16  *
17  * Diese Datei ist Teil von lcd library for ssd1306/ssd1309/sh1106 oled-display.
18  *
19  * lcd library for ssd1306/ssd1309/sh1106 oled-display ist Freie Software: Sie können
    ↪  es unter den Bedingungen
20  * der GNU General Public License, wie von der Free Software Foundation,
```

```
 * Version 3 der Lizenz oder jeder späteren
 * veröffentlichten Version, weiterverbreiten und/oder modifizieren.
 *
 * lcd library for ssd1306/ssd1309/sh1106 oled-display wird in der Hoffnung, dass es
 ↪  nützlich sein wird, aber
 * OHNE JEDE GEWÄHRLEISTUNG, bereitgestellt; sogar ohne die implizite
 * Gewährleistung der MARKTFÄHIGKEIT oder EIGNUNG FÜR EINEN BESTIMMTEN ZWECK.
 * Siehe die GNU General Public License für weitere Details.
 *
 * Sie sollten eine Kopie der GNU General Public License zusammen mit diesem
 * Programm erhalten haben. Wenn nicht, siehe <http://www.gnu.org/licenses/>.
 *
 *  lcd.h
 *
 *  Created by Michael Köhler on 22.12.16.
 *  Copyright 2016 Skie-Systems. All rights reserved.
 *
 *  at GRAPHICMODE lib needs static SRAM for display:
 *  DISPLAY-WIDTH * DISPLAY-HEIGHT + 2 bytes
 *
 *  at TEXTMODE lib need static SRAM for display:
 *  2 bytes (cursorPosition)
 */

#include "lcd.h"
#include "font.h"
#include <string.h>

#if defined SPI
#include <util/delay.h>
#endif

static struct {
    uint8_t x;
    uint8_t y;
} cursorPosition;

static uint8_t charMode = NORMALSIZE;
#if defined GRAPHICMODE
#include <stdlib.h>
static uint8_t displayBuffer[DISPLAY_HEIGHT/8][DISPLAY_WIDTH];
#elif defined TEXTMODE
#else
#error "No valid displaymode! Refer lcd.h"
#endif


const uint8_t init_sequence [] PROGMEM = {    // Initialization Sequence
```

```
68      LCD_DISP_OFF,    // Display OFF (sleep mode)
69      0x20, 0b00,      // Set Memory Addressing Mode
70      // 00=Horizontal Addressing Mode; 01=Vertical Addressing Mode;
71      // 10=Page Addressing Mode (RESET); 11=Invalid
72      0xB0,            // Set Page Start Address for Page Addressing Mode, 0-7
73      0xC8,            // Set COM Output Scan Direction
74      0x00,            // --set low column address
75      0x10,            // --set high column address
76      0x40,            // --set start line address
77      0x81, 0x3F,      // Set contrast control register
78      0xA1,            // Set Segment Re-map. A0=address mapped; A1=address 127 mapped.
79      0xA6,            // Set display mode. A6=Normal; A7=Inverse
80      0xA8, DISPLAY_HEIGHT-1, // Set multiplex ratio(1 to 64)
81      0xA4,            // Output RAM to Display
82                                      // 0xA4=Output follows RAM content;
                                        ↪  0xA5,Output ignores RAM content
83      0xD3, 0x00,      // Set display offset. 00 = no offset
84      0xD5,            // --set display clock divide ratio/oscillator frequency
85      0xF0,            // --set divide ratio
86      0xD9, 0x22,      // Set pre-charge period
87                       // Set com pins hardware configuration
88  #if DISPLAY_HEIGHT==64
89      0xDA, 0x12,
90  #elif DISPLAY_HEIGHT==32
91      0xDA, 0x02,
92  #endif
93      0xDB,            // --set vcomh
94      0x20,            // 0x20,0.77xVcc
95      0x8D, 0x14,      // Set DC-DC enable
96
97
98  };
99  #pragma mark LCD COMMUNICATION
100 void lcd_command(uint8_t cmd[], uint8_t size) {
101 #if defined I2C
102     i2c_start((LCD_I2C_ADR << 1) | 0);
103     i2c_byte(0x00);    // 0x00 for command, 0x40 for data
104     for (uint8_t i=0; i<size; i++) {
105         i2c_byte(cmd[i]);
106     }
107     i2c_stop();
108 #elif defined SPI
109         LCD_PORT &= ~(1 << CS_PIN);
110         LCD_PORT &= ~(1 << DC_PIN);
111         for (uint8_t i=0; i<size; i++) {
112         SPDR = cmd[i];
113         while(!(SPSR & (1<<SPIF)));
114     }
```

```
115     LCD_PORT |= (1 << CS_PIN);
116 #endif
117 }
118 void lcd_data(uint8_t data[], uint16_t size) {
119 #if defined I2C
120     i2c_start((LCD_I2C_ADR << 1) | 0);
121     i2c_byte(0x40);    // 0x00 for command, 0x40 for data
122     for (uint16_t i = 0; i<size; i++) {
123         i2c_byte(data[i]);
124     }
125     i2c_stop();
126 #elif defined SPI
127         LCD_PORT &= ~(1 << CS_PIN);
128         LCD_PORT |= (1 << DC_PIN);
129         for (uint16_t i = 0; i<size; i++) {
130         SPDR = data[i];
131         while(!(SPSR & (1<<SPIF)));
132     }
133     LCD_PORT |= (1 << CS_PIN);
134 #endif
135 }
136 #pragma mark -
137 #pragma mark GENERAL FUNCTIONS
138 void lcd_init(uint8_t dispAttr){
139 #if defined I2C
140     i2c_init();
141 #elif defined SPI
142         DDRB |= (1 << PB2)|(1 << PB3)|(1 << PB5);
143     SPCR = (1 << SPE)|(1<<MSTR)|(1<<SPR0);
144     LCD_DDR |= (1 << CS_PIN)|(1 << DC_PIN)|(1 << RES_PIN);
145     LCD_PORT |= (1 << CS_PIN)|(1 << DC_PIN)|(1 << RES_PIN);
146     LCD_PORT &= ~(1 << RES_PIN);
147     _delay_ms(10);
148     LCD_PORT |= (1 << RES_PIN);
149 #endif
150 
151     uint8_t commandSequence[sizeof(init_sequence)+1];
152     for (uint8_t i = 0; i < sizeof (init_sequence); i++) {
153         commandSequence[i] = (pgm_read_byte(&init_sequence[i]));
154     }
155     commandSequence[sizeof(init_sequence)]=(dispAttr);
156     lcd_command(commandSequence, sizeof(commandSequence));
157     lcd_clrscr();
158 }
159 void lcd_gotoxy(uint8_t x, uint8_t y){
160     x = x * sizeof(FONT[0]);
161     lcd_goto_xpix_y(x,y);
162 }
```

```
163 void lcd_goto_xpix_y(uint8_t x, uint8_t y){
164     if( x > (DISPLAY_WIDTH) || y > (DISPLAY_HEIGHT/8-1)) return;// out of display
165     cursorPosition.x=x;
166     cursorPosition.y=y;
167 #if defined (SSD1306) || defined (SSD1309)
168     uint8_t commandSequence[] = {0xb0+y, 0x21, x, 0x7f};
169 #elif defined SH1106
170     uint8_t commandSequence[] = {0xb0+y, 0x21, 0x00+((2+x) & (0x0f)), 0x10+( ((2+x) &
        ↪  (0xf0)) >> 4 ), 0x7f};
171 #endif
172     lcd_command(commandSequence, sizeof(commandSequence));
173 }
174 void lcd_clrscr(void){
175 #ifdef GRAPHICMODE
176     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
177         memset(displayBuffer[i], 0x00, sizeof(displayBuffer[i]));
178         lcd_gotoxy(0,i);
179         lcd_data(displayBuffer[i], sizeof(displayBuffer[i]));
180     }
181 #elif defined TEXTMODE
182     uint8_t displayBuffer[DISPLAY_WIDTH];
183     memset(displayBuffer, 0x00, sizeof(displayBuffer));
184     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
185         lcd_gotoxy(0,i);
186         lcd_data(displayBuffer, sizeof(displayBuffer));
187     }
188 #endif
189     lcd_home();
190 }
191 void lcd_home(void){
192     lcd_gotoxy(0, 0);
193 }
194 void lcd_invert(uint8_t invert){
195     uint8_t commandSequence[1];
196     if (invert != YES) {
197         commandSequence[0] = 0xA6;
198     } else {
199         commandSequence[0] = 0xA7;
200     }
201     lcd_command(commandSequence, 1);
202 }
203 void lcd_sleep(uint8_t sleep){
204     uint8_t commandSequence[1];
205     if (sleep != YES) {
206         commandSequence[0] = 0xAF;
207     } else {
208         commandSequence[0] = 0xAE;
209     }
```

```
210     lcd_command(commandSequence, 1);
211 }
212 void lcd_set_contrast(uint8_t contrast){
213     uint8_t commandSequence[2] = {0x81, contrast};
214     lcd_command(commandSequence, sizeof(commandSequence));
215 }
216 void lcd_putc(char c){
217     switch (c) {
218         case '\b':
219             // backspace
220             lcd_gotoxy(cursorPosition.x-charMode, cursorPosition.y);
221             lcd_putc(' ');
222             lcd_gotoxy(cursorPosition.x-charMode, cursorPosition.y);
223             break;
224         case '\t':
225             // tab
226             if( (cursorPosition.x+charMode*4) < (DISPLAY_WIDTH/
                ↪  sizeof(FONT[0])-charMode*4) ){
227                 lcd_gotoxy(cursorPosition.x+charMode*4, cursorPosition.y);
228             }else{
229                 lcd_gotoxy(DISPLAY_WIDTH/ sizeof(FONT[0]), cursorPosition.y);
230             }
231             break;
232         case '\n':
233             // linefeed
234             if(cursorPosition.y < (DISPLAY_HEIGHT/8-1)){
235                 lcd_gotoxy(cursorPosition.x, cursorPosition.y+charMode);
236             }
237             break;
238         case '\r':
239             // carrige return
240             lcd_gotoxy(0, cursorPosition.y);
241             break;
242         default:
243             // char doesn't fit in line
244             if( (cursorPosition.x >= DISPLAY_WIDTH-sizeof(FONT[0])) || (c < ' ') )
                ↪  break;
245             // mapping char
246             c -= ' ';
247             if (c >= pgm_read_byte(&special_char[0][1]) ) {
248                 char temp = c;
249                 c = 0xff;
250                 for (uint8_t i=0; pgm_read_byte(&special_char[i][1]) != 0xff; i++) {
251                     if ( pgm_read_byte(&special_char[i][0])-' ' == temp ) {
252                         c = pgm_read_byte(&special_char[i][1]);
253                         break;
254                     }
255                 }
```

```
256                 if ( c == 0xff ) break;
257             }
258             // print char at display
259 #ifdef GRAPHICMODE
260             if (charMode == DOUBLESIZE) {
261                 uint16_t doubleChar[sizeof(FONT[0])];
262                 uint8_t dChar;
263                 if ((cursorPosition.x+2*sizeof(FONT[0]))>DISPLAY_WIDTH) break;
264 
265                 for (uint8_t i=0; i < sizeof(FONT[0]); i++) {
266                     doubleChar[i] = 0;
267                     dChar = pgm_read_byte(&(FONT[(uint8_t)c][i]));
268                     for (uint8_t j=0; j<8; j++) {
269                         if ((dChar & (1 << j))) {
270                             doubleChar[i] |= (1 << (j*2));
271                             doubleChar[i] |= (1 << ((j*2)+1));
272                         }
273                     }
274                 }
275                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
276                 {
277                     // load bit-pattern from flash
278                     displayBuffer[cursorPosition.y+1][cursorPosition.x+(2*i)] =
                    ↪  doubleChar[i] >> 8;
279                     displayBuffer[cursorPosition.y+1][cursorPosition.x+(2*i)+1] =
                    ↪  doubleChar[i] >> 8;
280                     displayBuffer[cursorPosition.y][cursorPosition.x+(2*i)] =
                    ↪  doubleChar[i] & 0xff;
281                     displayBuffer[cursorPosition.y][cursorPosition.x+(2*i)+1] =
                    ↪  doubleChar[i] & 0xff;
282                 }
283                 cursorPosition.x += sizeof(FONT[0])*2;
284             } else {
285                     if ((cursorPosition.x+sizeof(FONT[0]))>DISPLAY_WIDTH) break;
286 
287                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
288                 {
289                     // load bit-pattern from flash
290                     displayBuffer[cursorPosition.y][cursorPosition.x+i]
                    ↪  =pgm_read_byte(&(FONT[(uint8_t)c][i]));
291                 }
292                 cursorPosition.x += sizeof(FONT[0]);
293             }
294 #elif defined TEXTMODE
295             if (charMode == DOUBLESIZE) {
296                 uint16_t doubleChar[sizeof(FONT[0])];
297                 uint8_t dChar;
298                 if ((cursorPosition.x+2*sizeof(FONT[0]))>DISPLAY_WIDTH) break;
```

```
299
300                 for (uint8_t i=0; i < sizeof(FONT[0]); i++) {
301                     doubleChar[i] = 0;
302                     dChar = pgm_read_byte(&(FONT[(uint8_t)c][i]));
303                     for (uint8_t j=0; j<8; j++) {
304                         if ((dChar & (1 << j))) {
305                             doubleChar[i] |= (1 << (j*2));
306                             doubleChar[i] |= (1 << ((j*2)+1));
307                         }
308                     }
309                 }
310                 uint8_t data[sizeof(FONT[0])*2];
311                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
312                 {
313                     // print font to ram, print 6 columns
314                     data[i<<1]=(doubleChar[i] & 0xff);
315                     data[(i<<1)+1]=(doubleChar[i] & 0xff);
316                 }
317                 lcd_data(data, sizeof(FONT[0])*2);
318
319 #if defined (SSD1306) || defined (SSD1309)
320                 uint8_t commandSequence[] = {0xb0+cursorPosition.y+1,
321                     0x21,
322                     cursorPosition.x,
323                     0x7f};
324 #elif defined SH1106
325                 uint8_t commandSequence[] = {0xb0+cursorPosition.y+1,
326                     0x21,
327                     0x00+((2+cursorPosition.x) & (0x0f)),
328                     0x10+( ((2+cursorPosition.x) & (0xf0)) >> 4 ),
329                     0x7f};
330 #endif
331                 lcd_command(commandSequence, sizeof(commandSequence));
332
333                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
334                 {
335                     // print font to ram, print 6 columns
336                     data[i<<1]=(doubleChar[i] >> 8);
337                     data[(i<<1)+1]=(doubleChar[i] >> 8);
338                 }
339                 lcd_data(data, sizeof(FONT[0])*2);
340
341                 commandSequence[0] = 0xb0+cursorPosition.y;
342 #if defined (SSD1306) || defined (SSD1309)
343                 commandSequence[2] = cursorPosition.x+(2*sizeof(FONT[0]));
344 #elif defined SH1106
345                 commandSequence[2] = 0x00+((2+cursorPosition.x+(2*sizeof(FONT[0]))) &
                    ↪ (0x0f));
```

```
346                 commandSequence[3] = 0x10+( ((2+cursorPosition.x+(2*sizeof(FONT[0])))
                    ↪  & (0xf0)) >> 4 );
347 #endif
348                 lcd_command(commandSequence, sizeof(commandSequence));
349                 cursorPosition.x += sizeof(FONT[0])*2;
350             } else {
351                 uint8_t data[sizeof(FONT[0])];
352                 if ((cursorPosition.x+sizeof(FONT[0]))>DISPLAY_WIDTH) break;
353 
354                     for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
355                 {
356                     // print font to ram, print 6 columns
357                     data[i]=(pgm_read_byte(&(FONT[(uint8_t)c][i])));
358                 }
359                 lcd_data(data, sizeof(FONT[0]));
360                 cursorPosition.x += sizeof(FONT[0]);
361             }
362 #endif
363             break;
364     }
365 
366 }
367 void lcd_charMode(uint8_t mode){
368     charMode = mode;
369 }
370 void lcd_flip(uint8_t flipping){
371         uint8_t command[2] = {0xC8, 0xA1};
372         switch(flipping){
373                 case 0:
374                         // normal mode default at init (needs to be reload data to
                            ↪  display)
375                         command[0] = 0xC8;
376                         command[1] = 0xA1;
377                         lcd_command(command, sizeof(command));
378                         break;
379                 case 1:
380                         // flip horizontal && vertical (needs to be reload data to
                            ↪  display)
381                         command[0] = 0xC0;
382                         command[1] = 0xA0;
383                         lcd_command(command, sizeof(command));
384                         break;
385                 case 2:
386                         // flip vertical (immediate without reload data to display)
387                         command[0] = 0xC0;
388                         lcd_command(command, sizeof(command));
389                         break;
390                 case 3:
```

```
                        // flip horizontal (needs to be reload data to display)
                        command[1] = 0xA0;
                        lcd_command(command, sizeof(command));
                default:
                        // do nothing
                        break;
        }
}
void lcd_puts(const char* s){
    while (*s) {
        lcd_putc(*s++);
    }
}
void lcd_puts_p(const char* progmem_s){
    register uint8_t c;
    while ((c = pgm_read_byte(progmem_s++))) {
        lcd_putc(c);
    }
}
#ifdef GRAPHICMODE
#pragma mark -
#pragma mark GRAPHIC FUNCTIONS
uint8_t lcd_drawPixel(uint8_t x, uint8_t y, uint8_t color){
    if( x > DISPLAY_WIDTH-1 || y > (DISPLAY_HEIGHT-1)) return 1; // out of Display

    if( color == WHITE){
        displayBuffer[(y / 8)][x] |= (1 << (y % 8));
    } else {
        displayBuffer[(y / 8)][x] &= ~(1 << (y % 8));
    }

    return 0;
}
uint8_t lcd_drawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t color){
        uint8_t result;

    int dx =  abs(x2-x1), sx = x1<x2 ? 1 : -1;
    int dy = -abs(y2-y1), sy = y1<y2 ? 1 : -1;
    int err = dx+dy, e2; /* error value e_xy */

    while(1){
        result = lcd_drawPixel(x1, y1, color);
        if (x1==x2 && y1==y2) break;
        e2 = 2*err;
        if (e2 > dy) { err += dy; x1 += sx; } /* e_xy+e_x > 0 */
        if (e2 < dx) { err += dx; y1 += sy; } /* e_xy+e_y < 0 */
    }

```

```
439     return result;
440 }
441 uint8_t lcd_drawRect(uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t
    ↪  color){
442     uint8_t result;
443 
444     result = lcd_drawLine(px1, py1, px2, py1, color);
445     result = lcd_drawLine(px2, py1, px2, py2, color);
446     result = lcd_drawLine(px2, py2, px1, py2, color);
447     result = lcd_drawLine(px1, py2, px1, py1, color);
448 
449     return result;
450 }
451 uint8_t lcd_fillRect(uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t
    ↪  color){
452     uint8_t result;
453 
454     if( px1 > px2){
455         uint8_t temp = px1;
456         px1 = px2;
457         px2 = temp;
458         temp = py1;
459         py1 = py2;
460         py2 = temp;
461     }
462     for (uint8_t i=0; i<=(py2-py1); i++){
463         result = lcd_drawLine(px1, py1+i, px2, py1+i, color);
464     }
465 
466     return result;
467 }
468 uint8_t lcd_drawCircle(uint8_t center_x, uint8_t center_y, uint8_t radius, uint8_t
    ↪  color){
469     uint8_t result;
470 
471     int16_t f = 1 - radius;
472     int16_t ddF_x = 1;
473     int16_t ddF_y = -2 * radius;
474     int16_t x = 0;
475     int16_t y = radius;
476 
477     result = lcd_drawPixel(center_x  , center_y+radius, color);
478     result = lcd_drawPixel(center_x  , center_y-radius, color);
479     result = lcd_drawPixel(center_x+radius, center_y  , color);
480     result = lcd_drawPixel(center_x-radius, center_y  , color);
481 
482     while (x<y) {
483         if (f >= 0) {
```

```
            y--;
            ddF_y += 2;
            f += ddF_y;
        }
        x++;
        ddF_x += 2;
        f += ddF_x;

        result = lcd_drawPixel(center_x + x, center_y + y, color);
        result = lcd_drawPixel(center_x - x, center_y + y, color);
        result = lcd_drawPixel(center_x + x, center_y - y, color);
        result = lcd_drawPixel(center_x - x, center_y - y, color);
        result = lcd_drawPixel(center_x + y, center_y + x, color);
        result = lcd_drawPixel(center_x - y, center_y + x, color);
        result = lcd_drawPixel(center_x + y, center_y - x, color);
        result = lcd_drawPixel(center_x - y, center_y - x, color);
    }
    return result;
}
uint8_t lcd_fillCircle(uint8_t center_x, uint8_t center_y, uint8_t radius, uint8_t
↪  color) {
    uint8_t result;
    for(uint8_t i=0; i<= radius;i++){
        result = lcd_drawCircle(center_x, center_y, i, color);
    }
    return result;
}
uint8_t lcd_drawBitmap(uint8_t x, uint8_t y, const uint8_t *picture, uint8_t width,
↪  uint8_t height, uint8_t color){
    uint8_t result,i,j, byteWidth = (width+7)/8;
    for (j = 0; j < height; j++) {
        for(i=0; i < width;i++){
            if(pgm_read_byte(picture + j * byteWidth + i / 8) & (128 >> (i & 7))){
                result = lcd_drawPixel(x+i, y+j, color);
            } else {
                result = lcd_drawPixel(x+i, y+j, !color);
            }
        }
    }
    return result;
}
void lcd_display() {
#if defined (SSD1306) || defined (SSD1309)
    lcd_gotoxy(0,0);
    lcd_data(&displayBuffer[0][0], DISPLAY_WIDTH*DISPLAY_HEIGHT/8);
#elif defined SH1106
    for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
        lcd_gotoxy(0,i);
```

```
530         lcd_data(displayBuffer[i], sizeof(displayBuffer[i]));
531     }
532 #endif
533 }
534 void lcd_clear_buffer() {
535     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
536         memset(displayBuffer[i], 0x00, sizeof(displayBuffer[i]));
537     }
538 }
539 uint8_t lcd_check_buffer(uint8_t x, uint8_t y) {
540     if( x > DISPLAY_WIDTH-1 || y > (DISPLAY_HEIGHT-1)) return 0; // out of Display
541     return displayBuffer[(y / (DISPLAY_HEIGHT/8))][x] & (1 << (y %
    ↪  (DISPLAY_HEIGHT/8)));
542 }
543 void lcd_display_block(uint8_t x, uint8_t line, uint8_t width) {
544     if (line > (DISPLAY_HEIGHT/8-1) || x > DISPLAY_WIDTH - 1){return;}
545     if (x + width > DISPLAY_WIDTH) { // no -1 here, x alone is width 1
546         width = DISPLAY_WIDTH - x;
547     }
548     lcd_goto_xpix_y(x,line);
549     lcd_data(&displayBuffer[line][x], width);
550 }
551 #endif
552
```

4.5.6 lcd.h

```
1
2 /*
3  * This file is part of lcd library for ssd1306/ssd1309/sh1106 oled-display.
4  *
5  * lcd library for ssd1306/ssd1309/sh1106 oled-display is free software: you can
    ↪  redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, either version 3 of the License, or any later
    ↪  version.
8  *
9  * lcd library for ssd1306/ssd1309/sh1106 oled-display is distributed in the hope
    ↪  that it will be useful,
10 * but WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12 * GNU General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with Foobar.  If not, see <http://www.gnu.org/licenses/>.
16 *
```

```
17  * Diese Datei ist Teil von lcd library for ssd1306/ssd1309/sh1106 oled-display.
18  *
19  * lcd library for ssd1306/ssd1309/sh1106 oled-display ist Freie Software: Sie können
    ↪  es unter den Bedingungen
20  * der GNU General Public License, wie von der Free Software Foundation,
21  * Version 3 der Lizenz oder jeder späteren
22  * veröffentlichten Version, weiterverbreiten und/oder modifizieren.
23  *
24  * lcd library for ssd1306/ssd1309/sh1106 oled-display wird in der Hoffnung, dass es
    ↪  nützlich sein wird, aber
25  * OHNE JEDE GEWÄHRLEISTUNG, bereitgestellt; sogar ohne die implizite
26  * Gewährleistung der MARKTFÄHIGKEIT oder EIGNUNG FÜR EINEN BESTIMMTEN ZWECK.
27  * Siehe die GNU General Public License für weitere Details.
28  *
29  * Sie sollten eine Kopie der GNU General Public License zusammen mit diesem
30  * Programm erhalten haben. Wenn nicht, siehe <http://www.gnu.org/licenses/>.
31  *
32  *  lcd.h
33  *
34  *  Created by Michael Köhler on 22.12.16.
35  *  Copyright 2016 Skie-Systems. All rights reserved.
36  *
37  *  at GRAPHICMODE lib needs SRAM for display
38  *  DISPLAY-WIDTH * DISPLAY-HEIGHT + 2 bytes
39  */
40
41 #ifndef LCD_H
42 #define LCD_H
43
44 #ifdef __cplusplus
45 extern "C" {
46 #endif
47
48 #if (__GNUC__ * 100 + __GNUC_MINOR__) < 303
49 #error "This library requires AVR-GCC 3.3 or later, update to newer AVR-GCC compiler
    ↪  !"
50 #endif
51
52 #include <inttypes.h>
53 #include <avr/pgmspace.h>
54
55         /* TODO: define bus */
56 #define I2C                         // I2C or SPI
57     /* TODO: define displaycontroller */
58 #define SH1106                  // or SSD1306
59     /* TODO: define displaymode */
60 #define TEXTMODE                // TEXTMODE for only text to display,
61     // GRAPHICMODE for text and graphic
```

```
62      /* TODO: define font */
63  #define FONT            ssd1306oled_font// set font here, refer font-name at
    ↪  font.h/font.c
64
65      /* TODO: define I2C-adress for display */
66
67      // using 7-bit-adress for lcd-library
68      // if you use your own library for twi check I2C-adress-handle
69  #define LCD_I2C_ADR         (0x78 >> 1)    // 7 bit slave-adress without r/w-bit
70      // r/w-bit are set/unset by library
71      // e.g. 8 bit slave-adress:
72      // 0x78 = adress 0x3C with cleared r/w-bit (write-mode)
73
74
75  #ifdef I2C
76  #include "i2c.h"        // library for I2C-communication
77      // if you want to use other lib for I2C
78      // edit i2c_xxx commands in this library
79      // i2c_start(), i2c_byte(), i2c_stop()
80
81  #elif defined SPI
82          // if you want to use your other lib/function for SPI replace SPI-commands
83  #define LCD_PORT        PORTB
84  #define LCD_DDR                DDRB
85  #define RES_PIN                PB0
86  #define DC_PIN                PB1
87  #define CS_PIN                PB2
88
89  #endif
90
91  #ifndef YES
92  #define YES        1
93  #endif
94
95  #define NORMALSIZE 1
96  #define DOUBLESIZE 2
97
98  #define LCD_DISP_OFF        0xAE
99  #define LCD_DISP_ON        0xAF
100
101 #define WHITE            0x01
102 #define BLACK            0x00
103
104 #define DISPLAY_WIDTH        128
105 #define DISPLAY_HEIGHT        64
106
107
108
```

```
109     void lcd_command(uint8_t cmd[], uint8_t size);    // transmit command to display
110     void lcd_data(uint8_t data[], uint16_t size);    // transmit data to display
111     void lcd_init(uint8_t dispAttr);
112     void lcd_home(void);                        // set cursor to 0,0
113     void lcd_invert(uint8_t invert);                // invert display
114     void lcd_sleep(uint8_t sleep);                    // display goto sleep (power
        ↪  off)
115     void lcd_set_contrast(uint8_t contrast);    // set contrast for display
116     void lcd_puts(const char* s);                    // print string, \n-terminated,
        ↪  from ram on screen (TEXTMODE)
117                                                 // or buffer (GRAPHICMODE)
118     void lcd_puts_p(const char* progmem_s);        // print string from flash on
        ↪  screen (TEXTMODE)
119                                                 // or buffer (GRAPHICMODE)
120
121     void lcd_clrscr(void);                        // clear screen (and buffer at
        ↪  GRFAICMODE)
122     void lcd_gotoxy(uint8_t x, uint8_t y);      // set curser at pos x, y. x means
        ↪  character,
123     // y means line (page, refer lcd manual)
124     void lcd_goto_xpix_y(uint8_t x, uint8_t y); // set curser at pos x, y. x means
        ↪  pixel,
125     // y means line (page, refer lcd manual)
126     void lcd_putc(char c);                        // print character on screen at
        ↪  TEXTMODE
127     // at GRAPHICMODE print character to buffer
128     void lcd_charMode(uint8_t mode);            // set size of chars
129     void lcd_flip(uint8_t flipping);                // flip display,
130                                                 // flipping == 0: no flip (normal
                                                    ↪  mode)
131                                                     // == 1: flip horizontal &
                                                        ↪  vertical
132                                                     // == 2: flip(mirrored) vertical
133                                                     // == 3: flip(mirrored)
                                                        ↪  horizontal
134 #if defined GRAPHICMODE
135     uint8_t lcd_drawPixel(uint8_t x, uint8_t y, uint8_t color);
136     uint8_t lcd_drawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t
        ↪  color);
137     uint8_t lcd_drawRect(uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t
        ↪  color);
138     uint8_t lcd_fillRect(uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t
        ↪  color);
139     uint8_t lcd_drawCircle(uint8_t center_x, uint8_t center_y, uint8_t radius,
        ↪  uint8_t color);
140     uint8_t lcd_fillCircle(uint8_t center_x, uint8_t center_y, uint8_t radius,
        ↪  uint8_t color);
141     uint8_t lcd_drawBitmap(uint8_t x, uint8_t y, const uint8_t picture[], uint8_t
        ↪  width, uint8_t height, uint8_t color);
```

```
142     void lcd_display(void);                         // copy buffer to display RAM
143     void lcd_clear_buffer(void);                   // clear display buffer
144     uint8_t lcd_check_buffer(uint8_t x, uint8_t y); // read a pixel value from the
        ↪  display buffer
145     void lcd_display_block(uint8_t x, uint8_t line, uint8_t width); // display (part
        ↪  of) a display line
146 #endif
147 
148 #ifdef __cplusplus
149 }
150 #endif
151 #endif /*  LCD_H  */
```

A Daily log entries

A.1 A.1 26 February - 3 March

A.1.1 A.1.1 Research Phase

- Conducted research on existing exercise tracking devices, focusing on accuracy, usability, and affordability.
- Evaluated industry standards and technologies used in similar projects.
- Decided to design a wearable exercise tracking device based on gathered insights and requirements.

A.2 A.2 4 March - 10 March

A.2.1 A.2.1 Conceptualization Phase

- Engaged in brainstorming sessions to generate innovative ideas and methods for the exercise tracking device.
- Developed multiple conceptual designs and evaluated them based on criteria such as efficiency, scalability, cost, and technical complexity.
- Analyzed and ranked each conceptual framework, selecting the most promising designs for further development.

A.3 A.3 11 March - 17 March

A.3.1 A.3.1 Design Evaluation Phase

- Conducted detailed evaluations of selected conceptual designs, considering factors like technical feasibility, scalability, and alignment with project objectives.
- Developed a structured assessment method to ensure objective evaluation.
- Selected the best conceptual design based on thorough analysis and consensus among team members.

A.4 A.4 18 March - 24 March

A.4.1 A.4.1 Sensor Mechanism Selection

- Analyzed various sensor mechanisms for efficiency, precision, and compatibility with the overall device design.
- Finalized the most suitable sensor mechanism through collaborative discussions and technical assessments.
- Developed a comprehensive project plan, outlining key milestones, tasks, and timelines.

- Defined design specifications, including technical requirements and performance benchmarks.

A.5 A.5 25 March - 31 March

A.5.1 A.5.1 Design Phase

- Commenced the design phase by creating a prototype of the exercise tracking device using Solidworks software.
- Visualized and validated key features and functionalities of the conceptual design through the prototype.
- Iteratively refined the prototype based on feedback and feasibility assessments to meet performance requirements and address limitations.

A.6 A.6 1 April - 7 April

A.6.1 A.6.1 Design Refinement

- Continued iterative refinement of the device's prototype.
- Incorporated feedback from initial reviews and conducted feasibility assessments to enhance the design.
- Ensured the prototype met all specified performance requirements and addressed any identified limitations.

A.7 A.7 8 April - 14 April

A.7.1 A.7.1 Design Finalizing

- Finalized the PCB design and the Solidworks Designs.

A.8 A.8 15 April - 21 April

A.8.1 A.8.1 PCB Printing

- Submitted the PCB for printing to JLCPCB.

A.9 A.9 22 April - 28 April

A.9.1 A.9.1 Final Design Adjustments

- Finalized the detailed design documentation for the device.
- Prepared for the upcoming evaluation by reviewing all project elements to ensure completeness and accuracy.
- Ensured that all design elements were aligned with stakeholder requirements and project goals.

A.10 A.10 29 April - 5 May

A.10.1 A.10.1 Review and Testing

- Created the prototype.
- Conducted internal reviews of the design documentation and prototype.
- Engaged in peer review sessions to gather additional feedback and make necessary adjustments.
- Tested the PCB designed for the device to ensure it met all operational specifications.

A.11 A.11 6 May - 12 May

A.11.1 A.11.1 Enclosure

- Finalized the enclosure design in Solidworks and 3D printed it.

A.12 A.12 13 May - 22 June

A.12.1 A.12.1 Final Report Preparation

- Compiled all project documentation, including detailed design reports, test results, and feedback from demonstrations.
- Finalized the project report, ensuring it included comprehensive details about the design process, challenges faced, and solutions implemented.
- Included the referred datasheets in design documentation and design methodology reports.
- Reviewed and revised all sections of the report to ensure accuracy, clarity, and completeness.

A.13 A.13 23 June - 7 July

A.13.1 A.13.1 Report Corrections and Finalization

- Reviewed feedback on the project report and made necessary corrections.
- Addressed any last-minute adjustments.
- Compiled final project documentation.

B Reviews

Checked by	Sign
Lakshan.W.D.T-210335T	
Gunawaradana W.M.T.V-210198A	
Mihiranga N.G.D.-210387D	
Morawakgoda M.K.I.G.-210391J	

C Previous Codes Used

C.1 Transmitter Code

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

RF24 radio(9, 10); // CE, CSN pins
const byte address[6] = "00001";

int exerciseCount = 0;
int exerciseMode = 1;
```

```
17
18 void setup() {
19   Serial.begin(9600);
20   display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
21   display.clearDisplay();
22   display.setTextSize(2);
23   display.setTextColor(WHITE);
24   display.setCursor(0,0);
25   display.display();
26
27   radio.begin();
28   radio.openReadingPipe(0, address);
29   radio.setPALevel(RF24_PA_MIN);
30   radio.startListening();
31 }
32
33 void loop() {
34   if (radio.available()) {
35     radio.read(&exerciseCount, sizeof(exerciseCount));
36
37     display.clearDisplay();
38     display.setCursor(0,0);
39     display.print("Mode: ");
40     display.print(exerciseMode);
41     display.setCursor(0, 20);
42     display.print("Count: ");
43     display.print(exerciseCount);
44     display.display();
45   }
46 }
47
48
```

C.2 Receiver Code

```
1 #include <SPI.h>
2 #include <nRF24L01.h>
3 #include <RF24.h>
4 #include <Adafruit_GFX.h>
5 #include <Adafruit_SSD1306.h>
6
7 #define SCREEN_WIDTH 128
8 #define SCREEN_HEIGHT 64
9 #define OLED_RESET    -1
10 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

```
11
12 RF24 radio(9, 10); // CE, CSN pins
13 const byte address[6] = "00001";
14
15 int exerciseCount = 0;
16 int exerciseMode = 1;
17
18 void setup() {
19   Serial.begin(9600);
20   display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
21   display.clearDisplay();
22   display.setTextSize(2);
23   display.setTextColor(WHITE);
24   display.setCursor(0,0);
25   display.display();
26
27   radio.begin();
28   radio.openReadingPipe(0, address);
29   radio.setPALevel(RF24_PA_MIN);
30   radio.startListening();
31 }
32
33 void loop() {
34   if (radio.available()) {
35     radio.read(&exerciseCount, sizeof(exerciseCount));
36
37     display.clearDisplay();
38     display.setCursor(0,0);
39     display.print("Mode: ");
40     display.print(exerciseMode);
41     display.setCursor(0, 20);
42     display.print("Count: ");
43     display.print(exerciseCount);
44     display.display();
45   }
46 }
47
```

References

[1] SparkFun Electronics. nrf24l01+ preliminary product specification v1.2. Technical report, 2006. URL: https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf.

[2] Atmel Inc. Atmega328 data sheet. Technical report, 2012. URL: https://www.mouser.com/pdfdocs/gravitech_atmega328_datasheet.pdf.

[3] InvenSense Inc. Mpu-6000 and mpu-6050 product specification. Technical report, 2015. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[4] Adafruit Industries. Ssd1306 - adafruit, 2024. URL: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf.

[5] ON Semiconductor. Lm1117 adjustable low dropout voltage regulator. Technical report, 2023. URL: https://www.onsemi.com/pdf/datasheet/lm1117-d.pdf.

[6] Tim J. Sobering. Guidelines for drawing schematics. Technical report, SDE Consulting, 2008. Accessed: 2024-06-22. URL: https://www.k-state.edu/edl/docs/pubs/technical-resources/Technote8.pdf.

Data Sheets

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

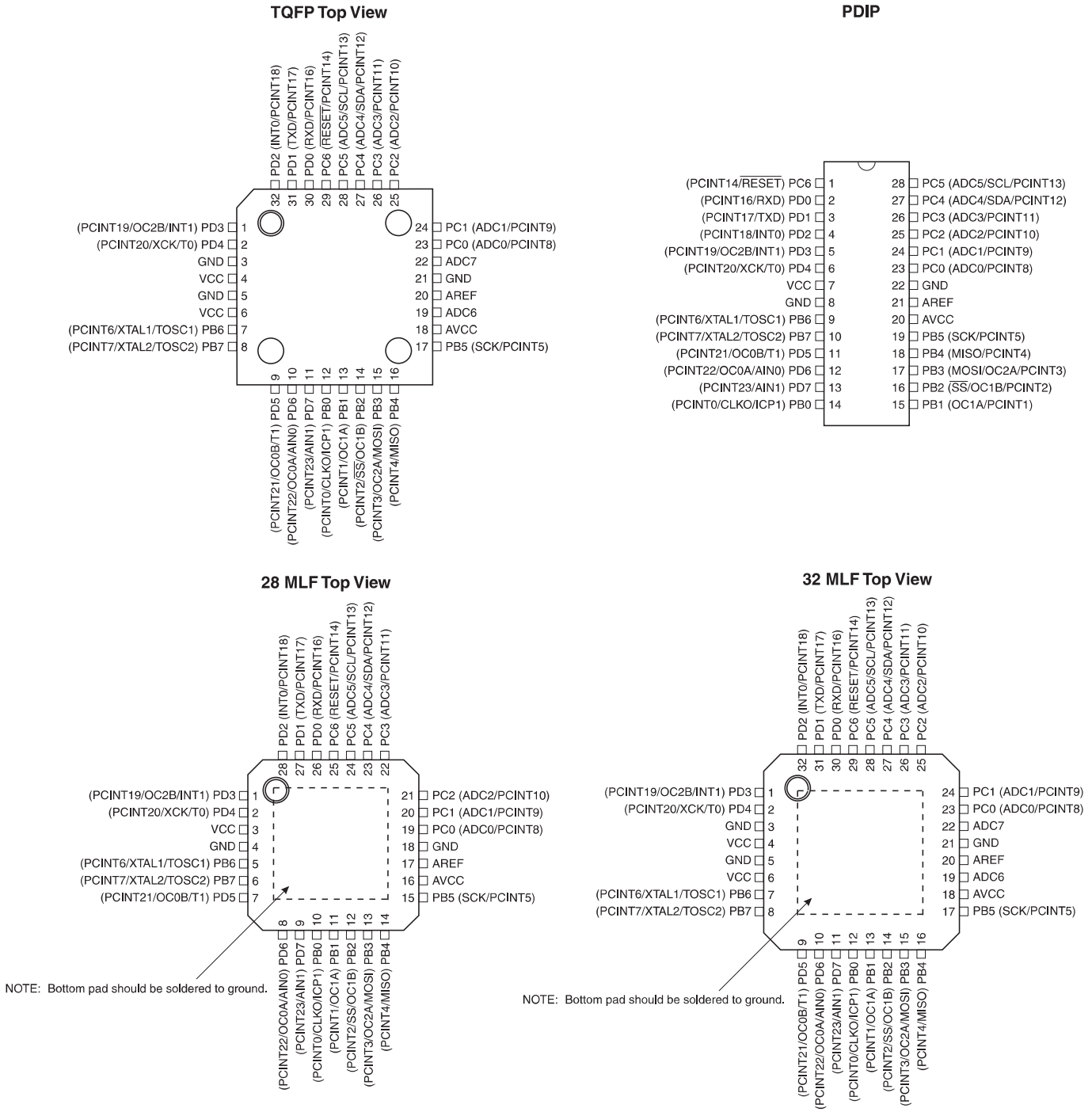
ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P

Summary

Rev. 8161CS–AVR–05/09

1. Pin Configurations

Figure 1-1. Pinout ATmega48PA/88PA/168PA/328P



1.1 Pin Descriptions

1.1.1 VCC

Digital supply voltage.

1.1.2 GND

Ground.

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in ”Alternate Functions of Port B” on page 76 and ”System Clock and Clock Options” on page 26.

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 28-3 on page 308. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in ”Alternate Functions of Port C” on page 79.

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in ”Alternate Functions of Port D” on page 82.

1.1.7 AV_{CC}

AV_{CC} is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC6..4 use digital supply voltage, V_{CC}.

1.1.8 AREF

AREF is the analog reference pin for the A/D Converter.

1.1.9 ADC7:6 (TQFP and QFN/MLF Package Only)

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

MPU-6000 and MPU-6050
Product Specification
Revision 3.4

6 Electrical Characteristics

6.1 Gyroscope Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		°/s	
	FS_SEL=1		±500		°/s	
	FS_SEL=2		±1000		°/s	
	FS_SEL=3		±2000		°/s	
Gyroscope ADC Word Length			16		bits	
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)	
	FS_SEL=1		65.5		LSB/(°/s)	
	FS_SEL=2		32.8		LSB/(°/s)	
	FS_SEL=3		16.4		LSB/(°/s)	
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%	
Sensitivity Scale Factor Variation Over Temperature			±2		%	
Nonlinearity	Best fit straight line; 25°C		0.2		%	
Cross-Axis Sensitivity			±2		%	
GYROSCOPE ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance	25°C		±20		°/s	
ZRO Variation Over Temperature	-40°C to +85°C		±20		°/s	
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.5V		4		°/s	
Linear Acceleration Sensitivity	Static		0.1		°/s/*g*	
SELF-TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	1
GYROSCOPE NOISE PERFORMANCE	**FS_SEL=0**					
Total RMS Noise	DLPFCFG=2 (100Hz)		0.05		°/s-rms	
Low-frequency RMS noise	Bandwidth 1Hz to10Hz		0.033		°/s-rms	
Rate Noise Spectral Density	At 10Hz		0.005		°/s/ √ Hz	
GYROSCOPE MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	
Y-Axis		27	30	33	kHz	
Z-Axis		24	27	30	kHz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		256	Hz	
OUTPUT DATA RATE						
	Programmable	4		8,000	Hz	
GYROSCOPE START-UP TIME	**DLPFCFG=0**					
ZRO Settling (from power-on)	to ±1°/s of Final		30		ms	

1. Please refer to the following document for further information on Self-Test: *MPU-6000/MPU-6050 Register Map and Descriptions*

6.2 Accelerometer Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0		±2		*g*	
	AFS_SEL=1		±4		*g*	
	AFS_SEL=2		±8		*g*	
	AFS_SEL=3		±16		*g*	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*	
	AFS_SEL=1		8,192		LSB/*g*	
	AFS_SEL=2		4,096		LSB/*g*	
	AFS_SEL=3		2,048		LSB/*g*	
Initial Calibration Tolerance			±3		%	
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		±0.02		%/°C	
Nonlinearity	Best Fit Straight Line		0.5		%	
Cross-Axis Sensitivity			±2		%	
ZERO-G OUTPUT						
Initial Calibration Tolerance	X and Y axes		±50		m*g*	1
	Z axis		±80		m*g*	
Zero-G Level Change vs. Temperature	X and Y axes, 0°C to +70°C		±35			
	Z axis, 0°C to +70°C		±60		m*g*	
SELF TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	2
NOISE PERFORMANCE						
Power Spectral Density	@10Hz, AFS_SEL=0 & ODR=1kHz		400		μ*g*/√Hz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		260	Hz	
OUTPUT DATA RATE						
	Programmable Range	4		1,000	Hz	
INTELLIGENCE FUNCTION INCREMENT			32		mg/LSB	

1. Typical zero-g initial calibration tolerance value after MSL3 preconditioning
2. Please refer to the following document for further information on Self-Test: *MPU-6000/MPU-6050 Register Map and Descriptions*

6.3 Electrical and Other Common Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	Units	Notes
TEMPERATURE SENSOR						
Range			-40 to +85		°C	
Sensitivity	Untrimmed		340		LSB/ºC	
Temperature Offset	35°C		-521		LSB	
Linearity	Best fit straight line (-40°C to +85°C)		±1		°C	
VDD POWER SUPPLY						
Operating Voltages		2.375		3.46	V	
Normal Operating Current	Gyroscope + Accelerometer + DMP		3.9		mA	
	Gyroscope + Accelerometer (DMP disabled)		3.8		mA	
	Gyroscope + DMP (Accelerometer disabled)		3.7		mA	
	Gyroscope only (DMP & Accelerometer disabled)		3.6		mA	
	Accelerometer only (DMP & Gyroscope disabled)		500		µA	
Accelerometer Low Power Mode Current	1.25 Hz update rate		10		µA	
	5 Hz update rate		20		µA	
	20 Hz update rate		70		µA	
	40 Hz update rate		140		µA	
Full-Chip Idle Mode Supply Current			5		µA	
Power Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value			100	ms	
VLOGIC REFERENCE VOLTAGE	MPU-6050 only					
Voltage Range	VLOGIC must be ≤VDD at all times	1.71		VDD	V	
Power Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value			3	ms	
Normal Operating Current			100		µA	
TEMPERATURE RANGE						
Specified Temperature Range	Performance parameters are not applicable beyond Specified Temperature Range	-40		+85	°C	

6.4 Electrical Specifications, Continued

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	Units	Notes
SERIAL INTERFACE						
SPI Operating Frequency, All Registers Read/Write	MPU-6000 only, Low Speed Characterization		100 ±10%		kHz	
	MPU-6000 only, High Speed Characterization		1 ±10%		MHz	
SPI Operating Frequency, Sensor and Interrupt Registers Read Only	MPU-6000 only		20 ±10%		MHz	
I^2C Operating Frequency	All registers, Fast-mode			400	kHz	
	All registers, Standard-mode			100	kHz	
I^2C ADDRESS	AD0 = 0		1101000			
	AD0 = 1		1101001			
DIGITAL INPUTS (SDI/SDA, AD0, SCLK/SCL, FSYNC, /CS, CLKIN)						
V_{IH}, High Level Input Voltage	MPU-6000	0.7*VDD			V	
	MPU-6050	0.7*VLOGIC			V	
V_{IL}, Low Level Input Voltage	MPU-6000			0.3*VDD	V	
	MPU-6050			0.3*VLOGIC	V	
C_I, Input Capacitance			< 5		pF	
DIGITAL OUTPUT (SDO, INT)						
V_{OH}, High Level Output Voltage	R_{LOAD}=1MΩ; MPU-6000	0.9*VDD			V	
	R_{LOAD}=1MΩ; MPU-6050	0.9*VLOGIC			V	
V_{OL1}, LOW-Level Output Voltage	R_{LOAD}=1MΩ; MPU-6000			0.1*VDD	V	
	R_{LOAD}=1MΩ; MPU-6050			0.1*VLOGIC	V	
$V_{OL.INT1}$, INT Low-Level Output Voltage	OPEN=1, 0.3mA sink Current			0.1	V	
Output Leakage Current	OPEN=1		100		nA	
t_{INT}, INT Pulse Width	LATCH_INT_EN=0		50		µs	

6.5 Electrical Specifications, Continued

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

Parameters	Conditions	Typical	Units	Notes
Primary I^2C I/O (SCL, SDA)				
V_{IL}, LOW-Level Input Voltage	MPU-6000	-0.5 to 0.3*VDD	V	
V_{IH}, HIGH-Level Input Voltage	MPU-6000	0.7*VDD to VDD + 0.5V	V	
V_{hys}, Hysteresis	MPU-6000	0.1*VDD	V	
VIL, LOW Level Input Voltage	MPU-6050	-0.5V to 0.3*VLOGIC	V	
VIH, HIGH-Level Input Voltage	MPU-6050	0.7*VLOGIC to VLOGIC + 0.5V	V	
Vhys, Hysteresis	MPU-6050	0.1*VLOGIC	V	
V_{OL1}, LOW-Level Output Voltage	3mA sink current	0 to 0.4	V	
I_{OL}, LOW-Level Output Current	V_{OL} = 0.4V	3	mA	
	V_{OL} = 0.6V	5	mA	
Output Leakage Current		100	nA	
t_{of}, Output Fall Time from V_{IHmax} to V_{ILmax}	C_b bus capacitance in pF	20+0.1C_b to 250	ns	
C_I, Capacitance for Each I/O pin		< 10	pF	
Auxiliary I^2C I/O (AUX_CL, AUX_DA)	**MPU-6050: *AUX_VDDIO*=0**			
V_{IL}, LOW-Level Input Voltage		-0.5V to 0.3*VLOGIC	V	
V_{IH}, HIGH-Level Input Voltage		0.7*VLOGIC to VLOGIC + 0.5V	V	
V_{hys}, Hysteresis		0.1*VLOGIC	V	
V_{OL1}, LOW-Level Output Voltage	VLOGIC > 2V; 1mA sink current	0 to 0.4	V	
V_{OL3}, LOW-Level Output Voltage	VLOGIC < 2V; 1mA sink current	0 to 0.2*VLOGIC	V	
I_{OL}, LOW-Level Output Current	V_{OL} = 0.4V	1	mA	
	V_{OL} = 0.6V	1	mA	
Output Leakage Current		100	nA	
t_{of}, Output Fall Time from V_{IHmax} to V_{ILmax}	C_b bus capacitance in pF	20+0.1C_b to 250	ns	
C_I, Capacitance for Each I/O pin		< 10	pF	

6.6 Electrical Specifications, Continued

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

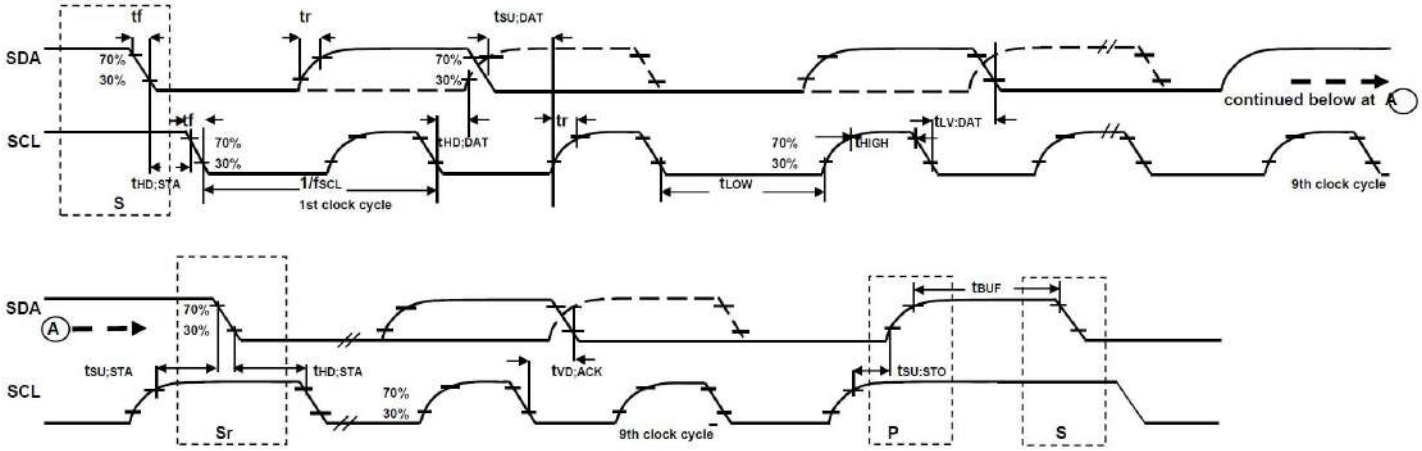
Parameters	Conditions	Min	Typical	Max	Units	Notes
INTERNAL CLOCK SOURCE	**CLK_SEL=0,1,2,3**					
Gyroscope Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	
Gyroscope Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	
Accelerometer Sample Rate			1		kHz	
Clock Frequency Initial Tolerance	CLK_SEL=0, 25°C	-5		+5	%	
	CLK_SEL=1,2,3; 25°C	-1		+1	%	
Frequency Variation over Temperature	CLK_SEL=0		-15 to +10		%	
	CLK_SEL=1,2,3		±1		%	
PLL Settling Time	CLK_SEL=1,2,3		1	10	ms	
EXTERNAL 32.768kHz CLOCK	**CLK_SEL=4**					
External Clock Frequency			32.768		kHz	
External Clock Allowable Jitter	Cycle-to-cycle rms		1 to 2		µs	
Gyroscope Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8.192		kHz	
Gyroscope Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1.024		kHz	
Accelerometer Sample Rate			1.024		kHz	
PLL Settling Time			1	10	ms	
EXTERNAL 19.2MHz CLOCK	**CLK_SEL=5**					
External Clock Frequency			19.2		MHz	
Gyroscope Sample Rate	Full programmable range	3.9		8000	Hz	
Gyroscope Sample Rate, Fast Mode	DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	
Gyroscope Sample Rate, Slow Mode	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	
Accelerometer Sample Rate			1		kHz	
PLL Settling Time			1	10	ms	

6.7 I²C Timing Characterization

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

Parameters	Conditions	Min	Typical	Max	Units	Notes
I²C TIMING	**I²C FAST-MODE**					
f_{SCL}, SCL Clock Frequency				400	kHz	
$t_{HD.STA}$, (Repeated) START Condition Hold Time		0.6			µs	
t_{LOW}, SCL Low Period		1.3			µs	
t_{HIGH}, SCL High Period		0.6			µs	
$t_{SU.STA}$, Repeated START Condition Setup Time		0.6			µs	
$t_{HD.DAT}$, SDA Data Hold Time		0			µs	
$t_{SU.DAT}$, SDA Data Setup Time		100			ns	
t_r, SDA and SCL Rise Time	C_b bus cap. from 10 to 400pF	20+0.1C_b		300	ns	
t_f, SDA and SCL Fall Time	C_b bus cap. from 10 to 400pF	20+0.1C_b		300	ns	
$t_{SU.STO}$, STOP Condition Setup Time		0.6			µs	
t_{BUF}, Bus Free Time Between STOP and START Condition		1.3			µs	
C_b, Capacitive Load for each Bus Line			< 400		pF	
$t_{VD.DAT}$, Data Valid Time				0.9	µs	
$t_{VD.ACK}$, Data Valid Acknowledge Time				0.9	µs	

Note: Timing Characteristics apply to both Primary and Auxiliary I²C Bus

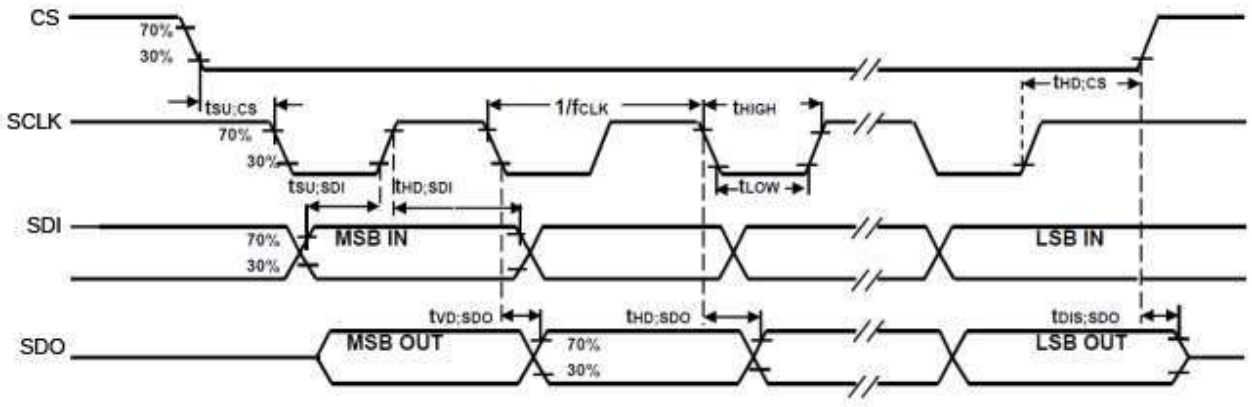


I²C Bus Timing Diagram

6.8 SPI Timing Characterization (MPU-6000 only)

Typical Operating Circuit of Section 7.2, VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C, unless otherwise noted.

Parameters	Conditions	Min	Typical	Max	Units	Notes
SPI TIMING						
f_{SCLK}, SCLK Clock Frequency				1	MHz	
t_{LOW}, SCLK Low Period		400			ns	
t_{HIGH}, SCLK High Period		400			ns	
$t_{SU.CS}$, CS Setup Time		8			ns	
$t_{HD.CS}$, CS Hold Time		500			ns	
$t_{SU.SDI}$, SDI Setup Time		11			ns	
$t_{HD.SDI}$, SDI Hold Time		7			ns	
$t_{VD.SDO}$, SDO Valid Time	C_{load} = 20pF			100	ns	
$t_{HD.SDO}$, SDO Hold Time	C_{load} = 20pF	4			ns	
$t_{DIS.SDO}$, SDO Output Disable Time				10	ns	



SPI Bus Timing Diagram

6.9 Absolute Maximum Ratings

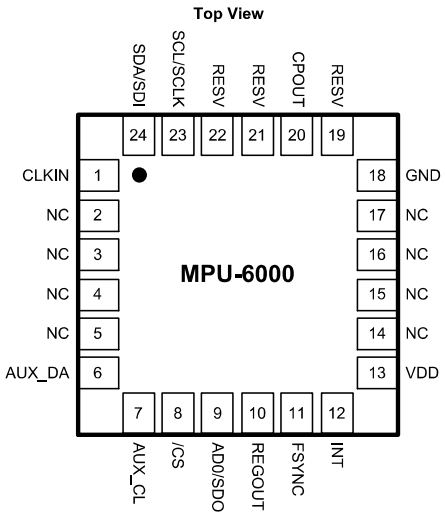
Stress above those listed as "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to the absolute maximum ratings conditions for extended periods may affect device reliability.

Parameter	Rating
Supply Voltage, VDD	-0.5V to +6V
VLOGIC Input Voltage Level (MPU-6050)	-0.5V to VDD + 0.5V
REGOUT	-0.5V to 2V
Input Voltage Level (CLKIN, AUX_DA, AD0, FSYNC, INT, SCL, SDA)	-0.5V to VDD + 0.5V
CPOUT (2.5V ≤ VDD ≤ 3.6V)	-0.5V to 30V
Acceleration (Any Axis, unpowered)	10,000g for 0.2ms
Operating Temperature Range	-40°C to +105°C
Storage Temperature Range	-40°C to +125°C
Electrostatic Discharge (ESD) Protection	2kV (HBM); 250V (MM)
Latch-up	JEDEC Class II (2),125°C ±100mA

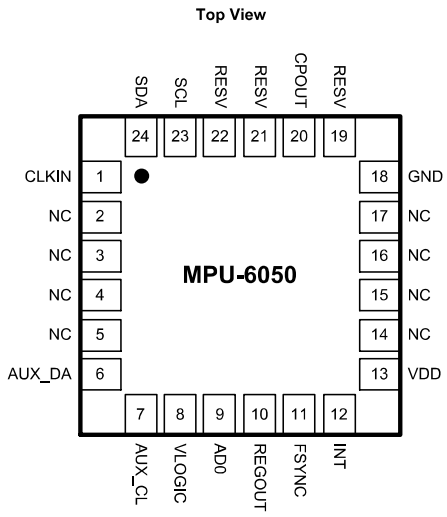
7 Applications Information

7.1 Pin Out and Signal Description

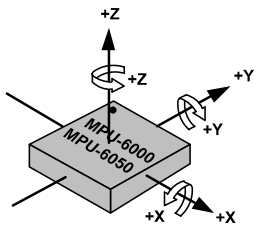
Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
1	Y	Y	CLKIN	Optional external reference clock input. Connect to GND if unused.
6	Y	Y	AUX_DA	I^2C master serial data, for connecting to external sensors
7	Y	Y	AUX_CL	I^2C Master serial clock, for connecting to external sensors
8	Y		/CS	SPI chip select (0=SPI mode)
8		Y	VLOGIC	Digital I/O supply voltage
9	Y		AD0 / SDO	I^2C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I^2C Slave Address LSB (AD0)
10	Y	Y	REGOUT	Regulator filter capacitor connection
11	Y	Y	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	Y	Y	INT	Interrupt digital output (totem pole or open-drain)
13	Y	Y	VDD	Power supply voltage and Digital I/O supply voltage
18	Y	Y	GND	Power supply ground
19, 21	Y	Y	RESV	Reserved. Do not connect.
20	Y	Y	CPOUT	Charge pump capacitor connection
22	Y	Y	RESV	Reserved. Do not connect.
23	Y		SCL / SCLK	I^2C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I^2C serial clock (SCL)
24	Y		SDA / SDI	I^2C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I^2C serial data (SDA)
2, 3, 4, 5, 14, 15, 16, 17	Y	Y	NC	Not internally connected. May be used for PCB trace routing.



QFN Package
24-pin, 4mm x 4mm x 0.9mm

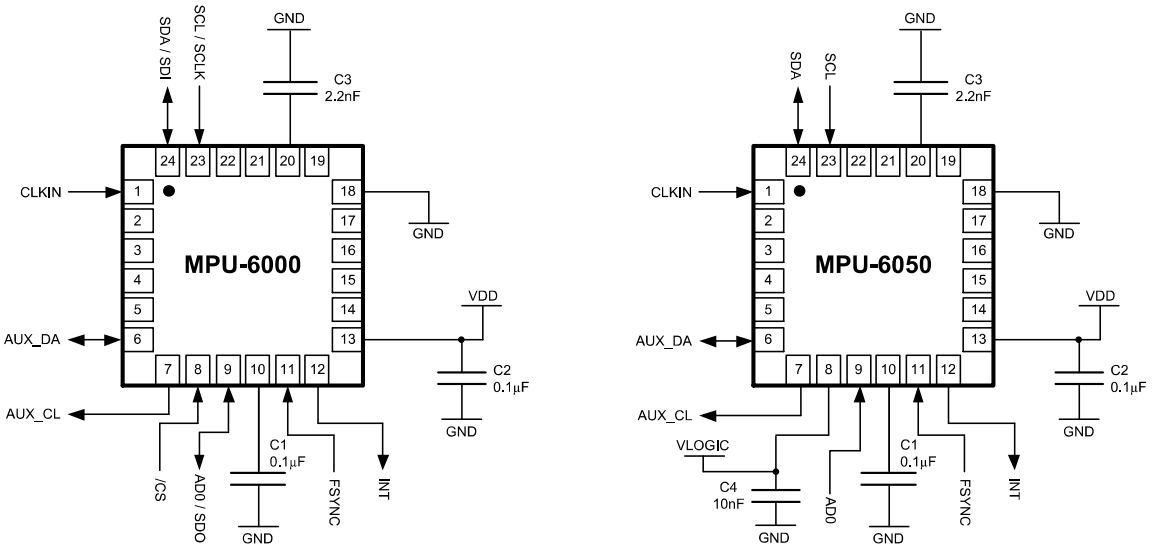


QFN Package
24-pin, 4mm x 4mm x 0.9mm



Orientation of Axes of Sensitivity and Polarity of Rotation

7.2 Typical Operating Circuit



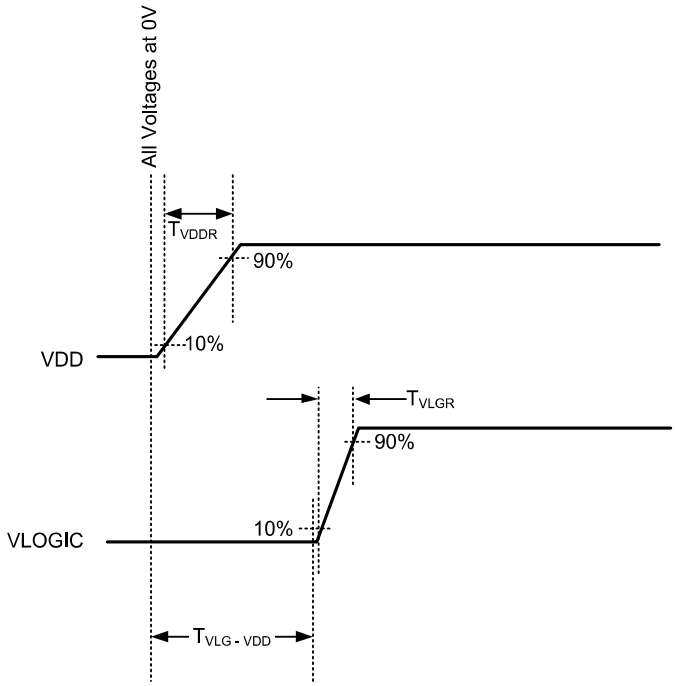
Typical Operating Circuits

7.3 Bill of Materials for External Components

Component	Label	Specification	Quantity
Regulator Filter Capacitor (Pin 10)	C1	Ceramic, X7R, 0.1µF ±10%, 2V	1
VDD Bypass Capacitor (Pin 13)	C2	Ceramic, X7R, 0.1µF ±10%, 4V	1
Charge Pump Capacitor (Pin 20)	C3	Ceramic, X7R, 2.2nF ±10%, 50V	1
VLOGIC Bypass Capacitor (Pin 8)	C4*	Ceramic, X7R, 10nF ±10%, 4V	1

* MPU-6050 Only.

7.4 Recommended Power-on Procedure



Power-Up Sequencing

1. VLOGIC amplitude must always be ≤VDD amplitude
2. T_{VDDR} is VDD rise time: Time for VDD to rise from 10% to 90% of its final value
3. T_{VDDR} is ≤100ms
4. T_{VLGR} is VLOGIC rise time: Time for VLOGIC to rise from 10% to 90% of its final value
5. T_{VLGR} is ≤3ms
6. $T_{VLG-VDD}$ is the delay from the start of VDD ramp to the start of VLOGIC rise
7. $T_{VLG-VDD}$ is ≥0
8. VDD and VLOGIC must be monotonic ramps

SSD1306

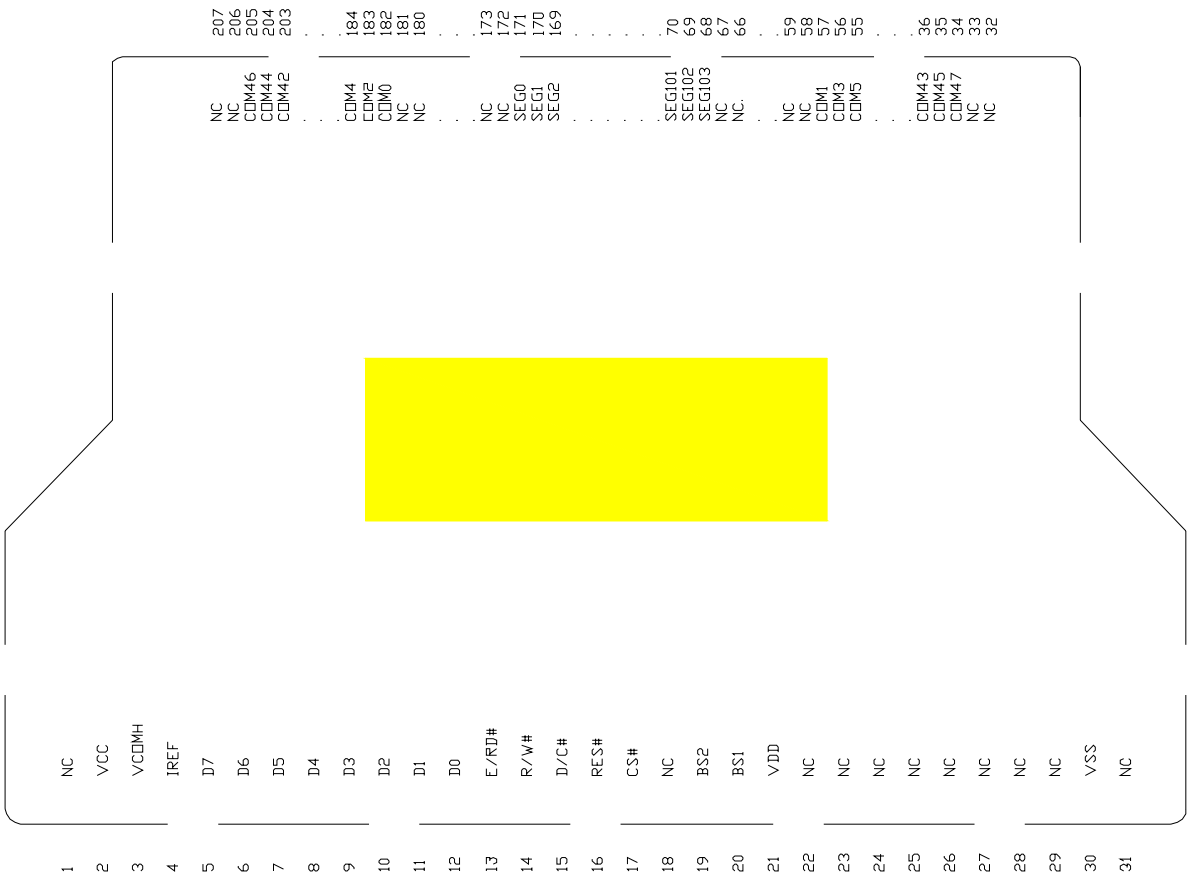
Advance Information

128 x 64 Dot Matrix
OLED/PLED Segment/Common Driver with Controller

6 PIN ARRANGEMENT

6.1 SSD1306TR1 pin assignment

Figure 6-1 : SSD1306TR1 Pin Assignment



Note:
(1) COM sequence (Split) is under command setting: DAh, 12h

11 MAXIMUM RATINGS

Table 11-1 : Maximum Ratings (Voltage Referenced to VSS)

Symbol	Parameter	Value	Unit
V_{DD}	Supply Voltage	-0.3 to +4	V
V_{CC}		0 to 16	V
V_{SEG}	SEG output voltage	0 to V_{CC}	V
V_{COM}	COM output voltage	0 to 0.9*V_{CC}	V
V_{in}	Input voltage	V_{SS}-0.3 to V_{DD}+0.3	V
T_A	Operating Temperature	-40 to +85	ºC
T_{stg}	Storage Temperature Range	-65 to +150	ºC

Maximum ratings are those values beyond which damages to the device may occur. Functional operation should be restricted to the limits in the Electrical Characteristics tables or Pin Description section

This device may be light sensitive. Caution should be taken to avoid exposure of this device to any light source during normal operation. This device is not radiation protected.

12 DC CHARACTERISTICS

Condition (Unless otherwise specified):

Voltage referenced to V_{SS}

V_{DD} = 1.65 to 3.3V

T_A = 25°C

Table 12-1 : DC Characteristics

Symbol	Parameter	Test Condition	Min	Typ	Max	Unit
V_{CC}	Operating Voltage	-	7	-	15	V
V_{DD}	Logic Supply Voltage	-	1.65	-	3.3	V
V_{OH}	High Logic Output Level	I_{OUT} = 100uA, 3.3MHz	0.9 x V_{DD}	-	-	V
V_{OL}	Low Logic Output Level	I_{OUT} = 100uA, 3.3MHz	-	-	0.1 x V_{DD}	V
V_{IH}	High Logic Input Level	-	0.8 x V_{DD}	-	-	V
V_{IL}	Low Logic Input Level	-	-	-	0.2 x V_{DD}	V
$I_{CC, SLEEP}$	$I_{CC,}$ Sleep mode Current	V_{DD} = 1.65V~3.3V, V_{CC} = 7V~15V Display OFF, No panel attached	-	-	10	uA
$I_{DD, SLEEP}$	$I_{DD,}$ Sleep mode Current	V_{DD} = 1.65V~3.3V, V_{CC} = 7V~15V Display OFF, No panel attached	-	-	10	uA
I_{CC}	V_{CC} Supply Current V_{DD} = 2.8V, V_{CC} = 12V, I_{REF} = 12.5uA No loading, Display ON, All ON	Contrast = FFh	-	430	780	uA
I_{DD}	V_{DD} Supply Current V_{DD} = 2.8V, V_{CC} = 12V, I_{REF} = 12.5uA No loading, Display ON, All ON		-	50	150	uA
I_{SEG}	Segment Output Current V_{DD}=2.8V, V_{CC}=12V, I_{REF}=12.5uA, Display ON.	Contrast=FFh	-	100	-	uA
		Contrast=AFh	-	69	-	
		Contrast=3Fh	-	25	-	
Dev	Segment output current uniformity	Dev = (I_{SEG} – I_{MID})/I_{MID} I_{MID} = (I_{MAX} + I_{MIN})/2 I_{SEG}[0:131] = Segment current at contrast = FFh	-3	-	+3	%
Adj. Dev	Adjacent pin output current uniformity (contrast = FF)	Adj Dev = (I[n]-I[n+1]) / (I[n]+I[n+1])	-2	-	+2	%

13 AC CHARACTERISTICS

Conditions:

Voltage referenced to V_{SS}
V_{DD}=1.65 to3.3V
T_A = 25°C

Table 13-1 : AC Characteristics

Symbol	Parameter	Test Condition	Min	Typ	Max	Unit
F_{OSC} (1)	Oscillation Frequency of Display Timing Generator	V_{DD} = 2.8V	333	370	407	kHz
F_{FRM}	Frame Frequency for 64 MUX Mode	128x64 Graphic Display Mode, Display ON, Internal Oscillator Enabled	-	F_{OSC} x 1/(DxKx64) (2)	-	Hz
RES#	Reset low pulse width		3	-	-	us

Note

(1) F_{OSC} stands for the frequency value of the internal oscillator and the value is measured when command D5h A[7:4] is in default value.

(2) D: divide ratio (default value = 1)
K: number of display clocks (default value = 54)
Please refer to Table 9-1 (Set Display Clock Divide Ratio/Oscillator Frequency, D5h) for detailed description

Table 13-2 : 6800-Series MCU Parallel Interface Timing Characteristics

(V_{DD} - V_{SS} = 1.65V to 3.3V, T_A = 25°C)

Symbol	Parameter	Min	Typ	Max	Unit
t_{cycle}	Clock Cycle Time	300	-	-	ns
t_{AS}	Address Setup Time	0	-	-	ns
t_{AH}	Address Hold Time	0	-	-	ns
t_{DSW}	Write Data Setup Time	40	-	-	ns
t_{DHW}	Write Data Hold Time	7	-	-	ns
t_{DHR}	Read Data Hold Time	20	-	-	ns
t_{OH}	Output Disable Time	-	-	70	ns
t_{ACC}	Access Time	-	-	140	ns
PW_{CSL}	Chip Select Low Pulse Width (read) Chip Select Low Pulse Width (write)	120 60	-	-	ns
PW_{CSH}	Chip Select High Pulse Width (read) Chip Select High Pulse Width (write)	60 60	-	-	ns
t_R	Rise Time	-	-	40	ns
t_F	Fall Time	-	-	40	ns

Figure 13-1 : 6800-series MCU parallel interface characteristics

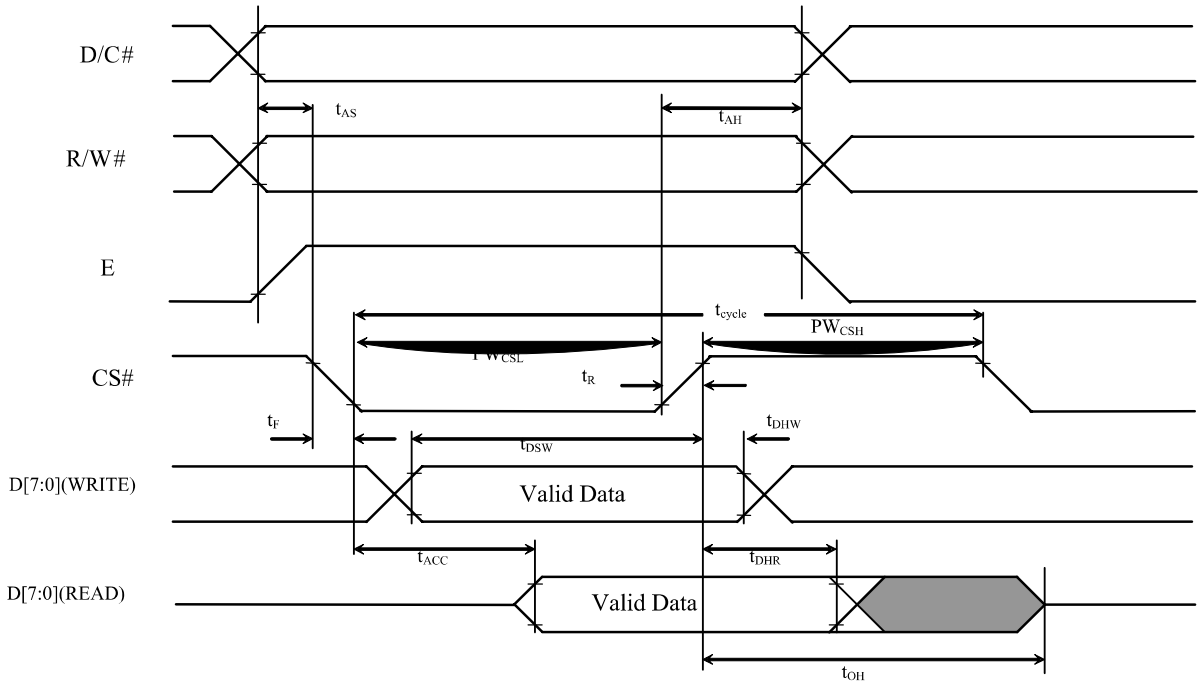


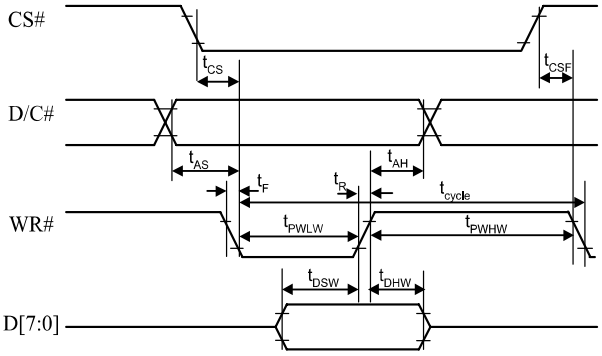
Table 13-3 : 8080-Series MCU Parallel Interface Timing Characteristics

(V_{DD} - V_{SS} = 1.65V to 3.3V, T_A = 25°C)

Symbol	Parameter	Min	Typ	Max	Unit
t_{cycle}	Clock Cycle Time	300	-	-	ns
t_{AS}	Address Setup Time	10	-	-	ns
t_{AH}	Address Hold Time	0	-	-	ns
t_{DSW}	Write Data Setup Time	40	-	-	ns
t_{DHW}	Write Data Hold Time	7	-	-	ns
t_{DHR}	Read Data Hold Time	20	-	-	ns
t_{OH}	Output Disable Time	-	-	70	ns
t_{ACC}	Access Time	-	-	140	ns
t_{PWLR}	Read Low Time	120	-	-	ns
t_{PWLW}	Write Low Time	60	-	-	ns
t_{PWHR}	Read High Time	60	-	-	ns
t_{PWHW}	Write High Time	60	-	-	ns
t_R	Rise Time	-	-	40	ns
t_F	Fall Time	-	-	40	ns
t_{CS}	Chip select setup time	0	-	-	ns
t_{CSH}	Chip select hold time to read signal	0	-	-	ns
t_{CSF}	Chip select hold time	20	-	-	ns

Figure 13-2 : 8080-series parallel interface characteristics

Write Cycle



Read cycle

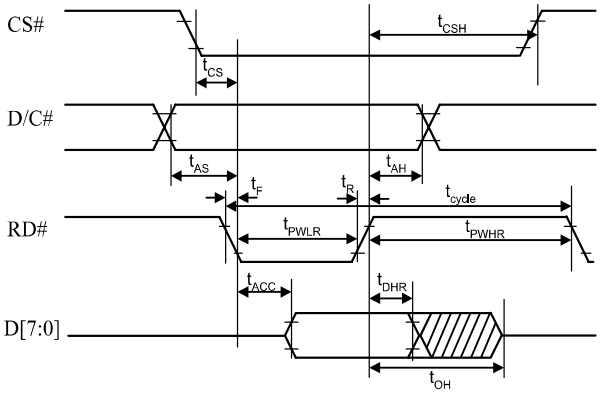


Table 13-4 : 4-wire Serial Interface Timing Characteristics

(V_{DD} - V_{SS} = 1.65V to 3.3V, T_A = 25°C)

Symbol	Parameter	Min	Typ	Max	Unit
t_{cycle}	Clock Cycle Time	100	-	-	ns
t_{AS}	Address Setup Time	15	-	-	ns
t_{AH}	Address Hold Time	15	-	-	ns
t_{CSS}	Chip Select Setup Time	20	-	-	ns
t_{CSH}	Chip Select Hold Time	10	-	-	ns
t_{DSW}	Write Data Setup Time	15	-	-	ns
t_{DHW}	Write Data Hold Time	15	-	-	ns
t_{CLKL}	Clock Low Time	20	-	-	ns
t_{CLKH}	Clock High Time	20	-	-	ns
t_R	Rise Time	-	-	40	ns
t_F	Fall Time	-	-	40	ns

Figure 13-3 : 4-wire Serial interface characteristics

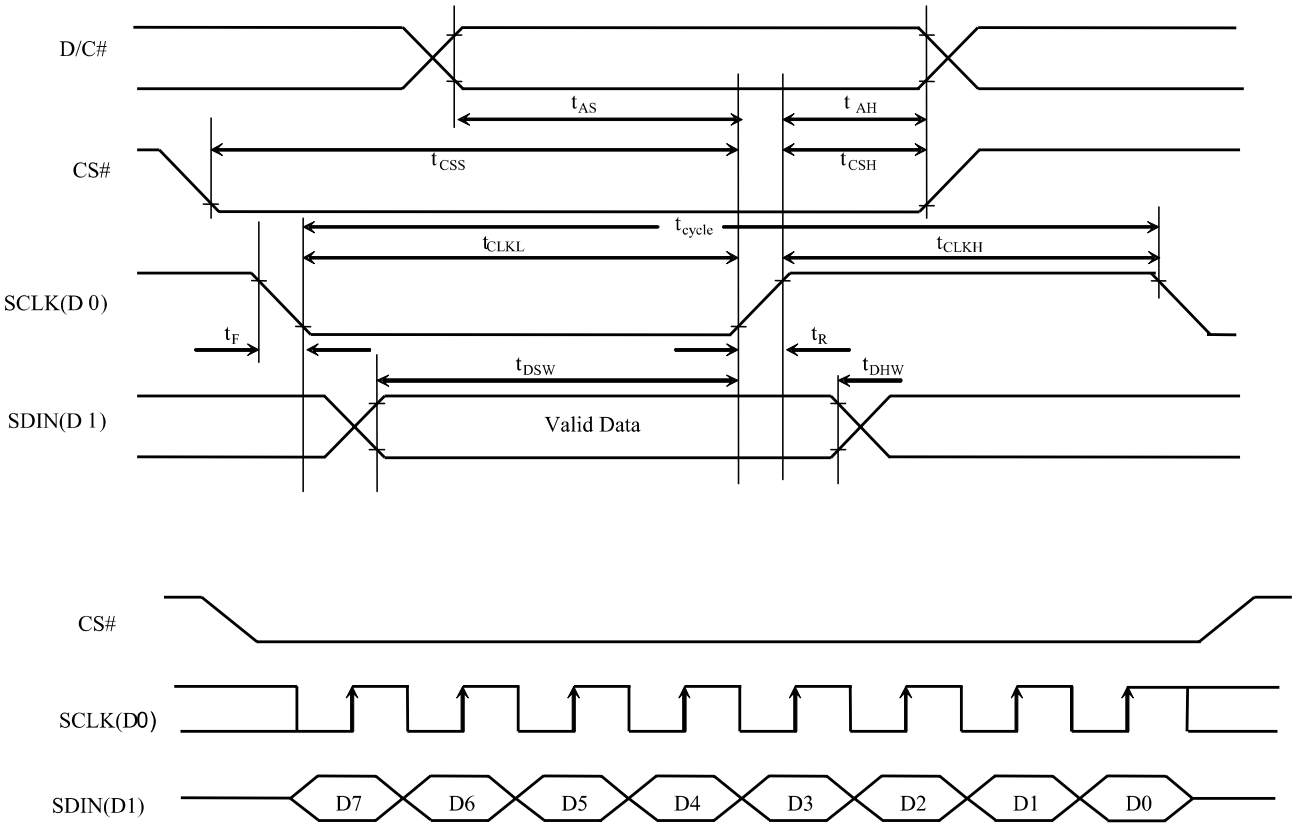
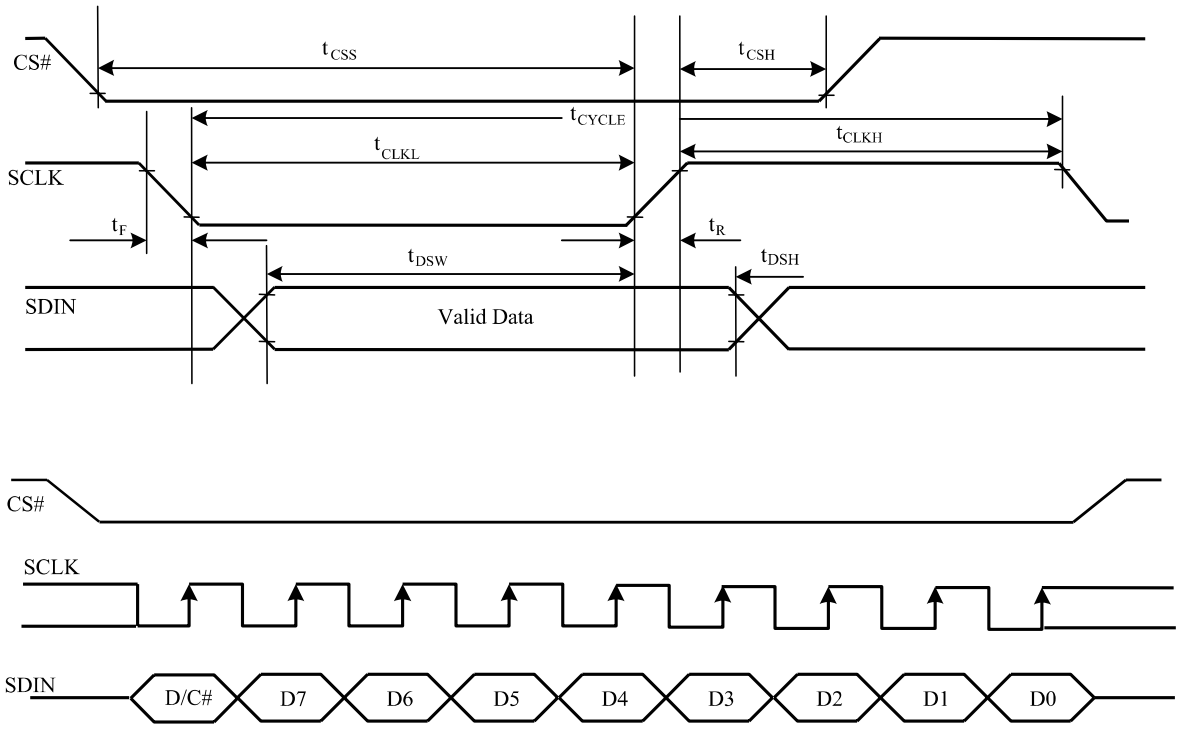


Table 13-5 : 3-wire Serial Interface Timing Characteristics

(V_{DD} - V_{SS} = 1.65V to 3.3V, T_A = 25°C)

Symbol	Parameter	Min	Typ	Max	Unit
t_{cycle}	Clock Cycle Time	100	-	-	ns
t_{CSS}	Chip Select Setup Time	20	-	-	ns
t_{CSH}	Chip Select Hold Time	10	-	-	ns
t_{DSW}	Write Data Setup Time	15	-	-	ns
t_{DHW}	Write Data Hold Time	15	-	-	ns
t_{CLKL}	Clock Low Time	20	-	-	ns
t_{CLKH}	Clock High Time	20	-	-	ns
t_R	Rise Time	-	-	40	ns
t_F	Fall Time	-	-	40	ns

Figure 13-4 : 3-wire Serial interface characteristics



Conditions:

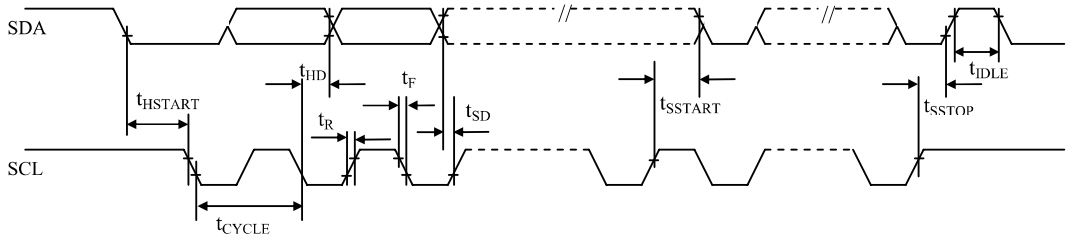
V_{DD} - V_{SS} = V_{DD} - V_{SS} = 1.65V to 3.3V

T_A = 25°C

Table 13-6 :I²C Interface Timing Characteristics

Symbol	Parameter	Min	Typ	Max	Unit
t_{cycle}	Clock Cycle Time	2.5	-	-	us
t_{HSTART}	Start condition Hold Time	0.6	-	-	us
t_{HD}	Data Hold Time (for "SDA_{OUT}" pin)	0	-	-	ns
	Data Hold Time (for "SDA_{IN}" pin)	300	-	-	ns
t_{SD}	Data Setup Time	100	-	-	ns
t_{SSTART}	Start condition Setup Time (Only relevant for a repeated Start condition)	0.6	-	-	us
t_{SSTOP}	Stop condition Setup Time	0.6	-	-	us
t_R	Rise Time for data and clock pin	-	-	300	ns
t_F	Fall Time for data and clock pin	-	-	300	ns
t_{IDLE}	Idle Time before a new transmission can start	1.3	-	-	us

Figure 13-5 : I²C interface Timing characteristics



800 mA Low-Dropout Linear Regulator

LM1117, LM1117I

SOT−223
CASE 318H

The LM1117 is a low dropout voltage regulator with a dropout of 1.2 V at 800 mA of load current. The LM1117 is available in an adjustable version, which can set the output voltage from 1.25 to 13.8 V with only two external resistors. In addition, it is available in five fixed voltages, 1.8 V, 2.5 V, 3.3 V, and 5 V.

The LM1117 offers current limiting and thermal shutdown. Its circuit is trimmed to assure output voltage accuracy to within +/−1%.

Features

- Available in 1.8 V, 2.5 V, 3.3 V, 5.0 V, and Adjustable Versions
- Space−Saving SOT−223 Package
- Current Limiting and Thermal Protection
- Output Current 800 mA
- Line Regulation 0.2% (Maximum)
- Load Regulation 0.4% (Maximum)
- Temperature Range: −40°C to 125°C
- These are Pb-Free Devices

Applications

- Post Regulator for Switching DC−DC Converter
- High Efficiency Linear Regulators
- Battery Chargers
- Portable Instrumentation
- Active SCSI Termination Regulation

PIN CONFIGURATION

SOT−223
(Top View)

Pin: 1. Adjust/Ground
2. Output
3. Input

Heatsink tab is connected to Pin 2.

ORDERING INFORMATION

See detailed ordering and shipping information on page 11 of this data sheet.

DEVICE MARKING INFORMATION

See general marking information in the device marking section on page 11 of this data sheet.

TYPICAL APPLICATIONS

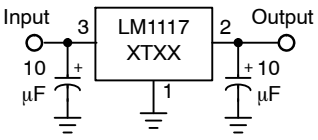


Figure 1. Fixed Output Regulator

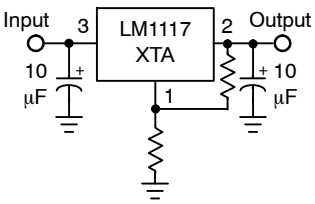


Figure 2. Adjustable Output Regulator

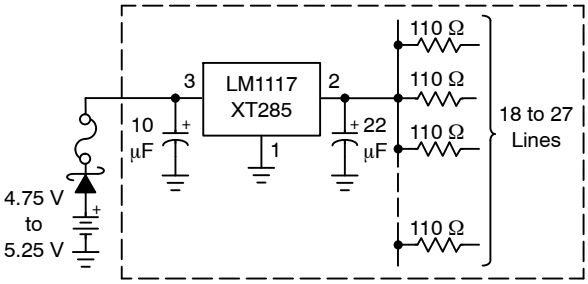


Figure 3. Active SCSI Bus Terminator

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Input Voltage (Note 1)	V_{in}	20	V
Output Short Circuit Duration (Notes 2 and 3)	–	Infinite	–
Power Dissipation and Thermal Characteristics Case 318H (SOT−223) Power Dissipation (Note 2) Thermal Resistance, Junction−to−Ambient, Minimum Size Pad Thermal Resistance, Junction−to−Case	 P_D $R_{\theta JA}$ $R_{\theta JC}$	 Internally Limited 160 15	 W °C/W °C/W
Maximum Die Junction Temperature Range	T_J	−55 to 150	°C
Storage Temperature Range	T_{stg}	−65 to 150	°C
Operating Ambient Temperature Range LM1117 LM1117I	T_A	 0 to +125 −40 to +125	°C

Stresses exceeding those listed in the Maximum Ratings table may damage the device. If any of these limits are exceeded, device functionality should not be assumed, damage may occur and reliability may be affected.

1. This device series contains ESD protection and exceeds the following tests:
   Human Body Model (HBM), Class 2, 2000 V
   Machine Model (MM), Class B, 200 V
   Charge Device Model (CDM), Class IV, 2000 V.
2. Internal thermal shutdown protection limits the die temperature to approximately 175°C. Proper heatsinking is required to prevent activation.
   The maximum package power dissipation is: $$P_D = \frac{T_{J(max)} - T_A}{R_{\theta JA}}$$
3. The regulator output current must not exceed 1.0 A with V_{in} greater than 12 V.

ELECTRICAL CHARACTERISTICS

(C_{in} = 10 μF, C_{out} = 10 μF, for typical value T_A = 25°C, for min and max values T_A is the operating ambient temperature range that applies unless otherwise noted.) (Note 4)

Characteristic	Symbol	Min	Typ	Max	Unit
Reference Voltage, Adjustable Output Devices (V_{in}–V_{out} = 2.0 V, I_{out} = 10 mA, T_A = 25°C) (V_{in}–V_{out} = 1.4 V to 10 V, I_{out} = 10 mA to 800 mA) (Note 4)	V_{ref}	 1.238 1.225	 1.25 –	 1.262 1.270	V
Output Voltage, Fixed Output Devices 1.8 V (V_{in} = 3.8 V, I_{out} = 10 mA, T_A = 25 °C) (V_{in} = 3.2 V to 11.8 V, I_{out} = 0 mA to 800 mA) (Note 4) 2.5 V (V_{in} = 4.5 V, I_{out} = 10 mA, T_A = 25 °C) (V_{in} = 3.9 V to 10 V, I_{out} = 0 mA to 800 mA,) (Note 4) 3.3 V (V_{in} = 5.3 V, I_{out} = 10 mA, T_A = 25 °C) (V_{in} = 4.75 V to 10 V, I_{out} = 0 mA to 800 mA) (Note 4) 5.0 V (V_{in} = 7.0 V, I_{out} = 10 mA, T_A = 25 °C) (V_{in} = 6.5 V to 12 V, I_{out} = 0 mA to 800 mA) (Note 4)	V_{out}	 1.782 1.755 2.475 2.450 3.267 3.235 4.950 4.900	 1.800 – 2.500 – 3.300 – 5.000 –	 1.818 1.845 2.525 2.550 3.333 3.365 5.050 5.100	V
Line Regulation (Note 5) Adjustable (V_{in} = 2.75 V to 16.25 V, I_{out} = 10 mA)	Reg_{line}	–	0.04	0.1	%
1.8 V (V_{in} = 3.2 V to 11.8 V, I_{out} = 0 mA) 2.5 V (V_{in} = 3.9 V to 10 V, I_{out} = 0 mA) 3.3 V (V_{in} = 4.75 V to 15 V, I_{out} = 0 mA) 5.0 V (V_{in} = 6.5 V to 15 V, I_{out} = 0 mA)		 – – – –	 0.4 0.5 0.8 0.9	 1.0 2.5 4.5 6.0	mV
Load Regulation (Note 5) Adjustable (I_{out} = 10 mA to 800 mA, V_{in} = 4.25 V)	Reg_{line}	–	0.2	0.4	%
1.8 V (I_{out} = 0 mA to 800 mA, V_{in} = 3.2 V) 2.5 V (I_{out} = 0 mA to 800 mA, V_{in} = 3.9 V) 3.3 V (I_{out} = 0 mA to 800 mA, V_{in} = 4.75 V) 5.0 V (I_{out} = 0 mA to 800 mA, V_{in} = 6.5 V)		 – – – –	 2.6 3.3 4.3 6.7	 6.0 7.5 10 15	mV
Dropout Voltage (Measured at V_{out} – 100 mV) (I_{out} = 100 mA) (I_{out} = 500 mA) (I_{out} = 800 mA)	V_{in}–V_{out}	 – – –	 0.95 1.01 1.07	 1.10 1.15 1.20	V
Output Current Limit (V_{in}–V_{out} = 5.0 V, T_A = 25°C, Note 6)	I_{out}	1000	1500	2200	mA
Minimum Required Load Current for Regulation, Adjustable Output Devices (V_{in} = 15 V)	$I_{L(min)}$	–	0.8	5.0	mA
Quiescent Current 1.8 V (V_{in} = 11.8 V) 2.5 V (V_{in} = 10 V) 3.3 V (V_{in} = 15 V) 5.0 V (V_{in} = 15 V)	I_Q	 – – – –	 4.2 5.2 6.0 6.0	 10 10 10 10	mA
Thermal Regulation (T_A = 25°C, 30 ms Pulse)		–	0.01	0.1	%/W
Ripple Rejection (V_{in}–V_{out} = 6.4 V, I_{out} = 500 mA, 10 V_{pp} 120 Hz Sinewave) Adjustable 1.8 V 2.5 V 3.3 V 5.0 V	RR	 67 66 62 60 57	 73 70 68 64 61	 – – – – –	dB
Adjustment Pin Current (V_{in} = 11.25 V, I_{out} = 800 mA)	I_{adj}	–	52	120	μA
Adjust Pin Current Change (V_{in}–V_{out} = 1.4 V to 10 V, I_{out} = 10 mA to 800 mA)	ΔI_{adj}	–	0.4	5.0	μA
Temperature Stability	S_T	–	0.5	–	%
Long Term Stability (T_A = 25°C, 1000 Hrs End Point Measurement)	S_t	–	0.3	–	%
RMS Output Noise (f = 10 Hz to 10 kHz)	N	–	0.003	–	%V_{out}

Product parametric performance is indicated in the Electrical Characteristics for the listed test conditions, unless otherwise noted. Product performance may not be indicated by the Electrical Characteristics if operated under different conditions.

4. LM1117: T_{low} = 0°C, T_{high} = 125°C
   LM1117I: T_{low} = –40°C, T_{high} = 125°C
5. Low duty cycle pulse techniques are used during testing to maintain the junction temperature as close to ambient as possible.
6. The regulator output current must not exceed 1.0 A with V_{in} greater than 12 V.

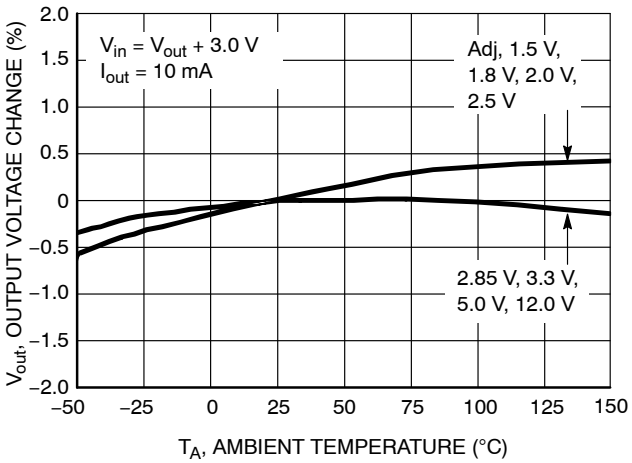


Figure 4. Output Voltage Change vs. Temperature

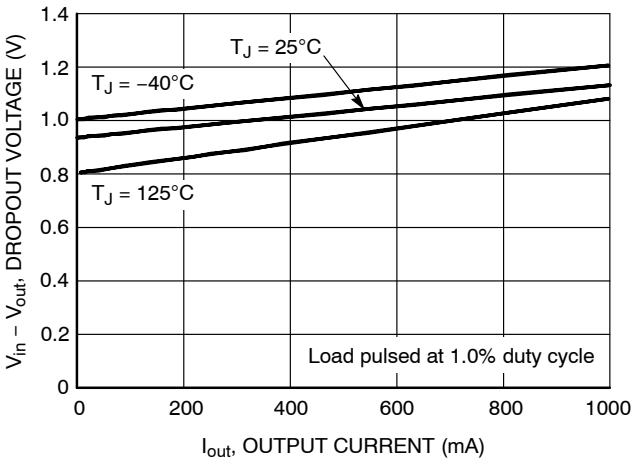


Figure 5. Dropout Voltage vs. Output Current

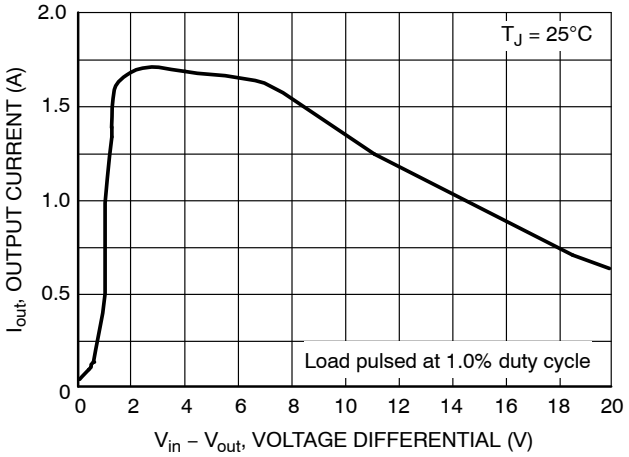


Figure 6. Output Short Circuit Current vs. Differential Voltage

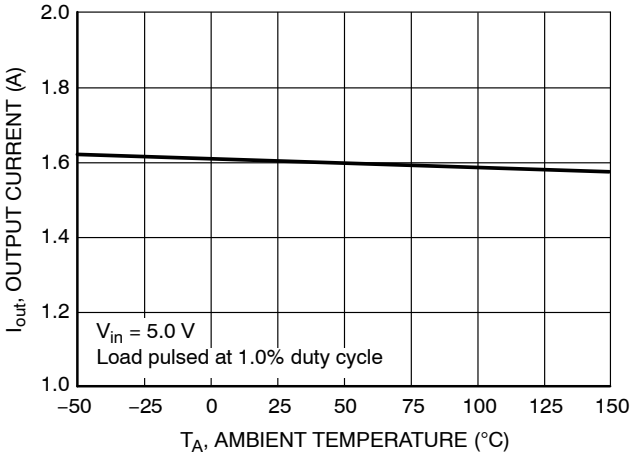


Figure 7. Output Short Circuit Current vs. Temperature

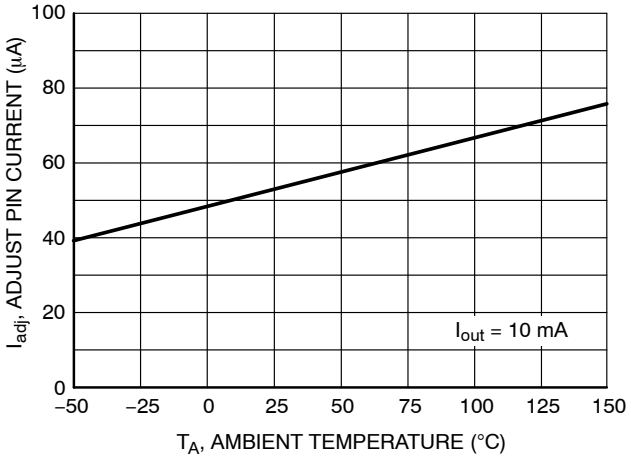


Figure 8. Adjust Pin Current vs. Temperature

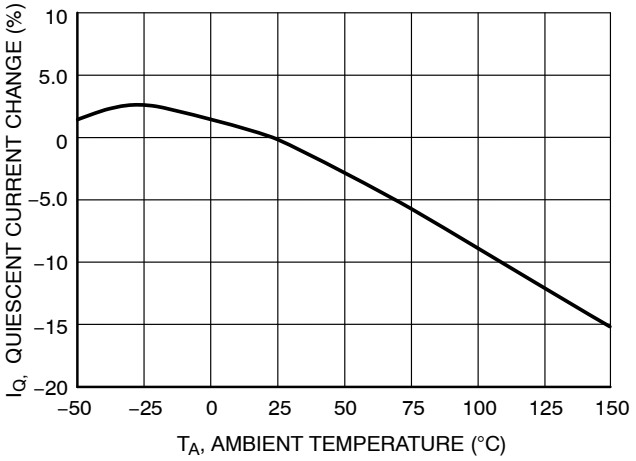


Figure 9. Quiescent Current Change vs. Temperature

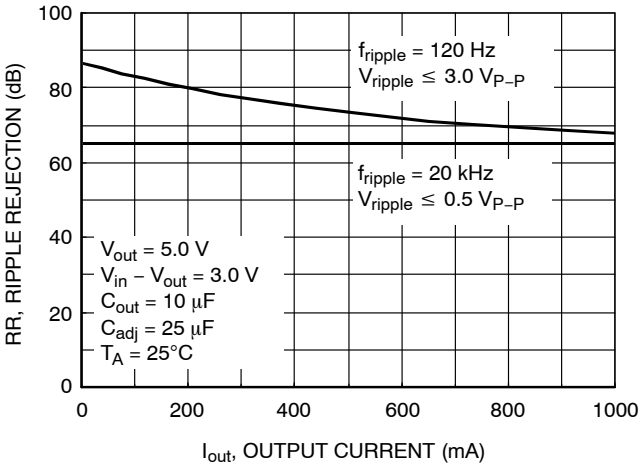


Figure 10. LM1117XTA Ripple Rejection vs. Output Current

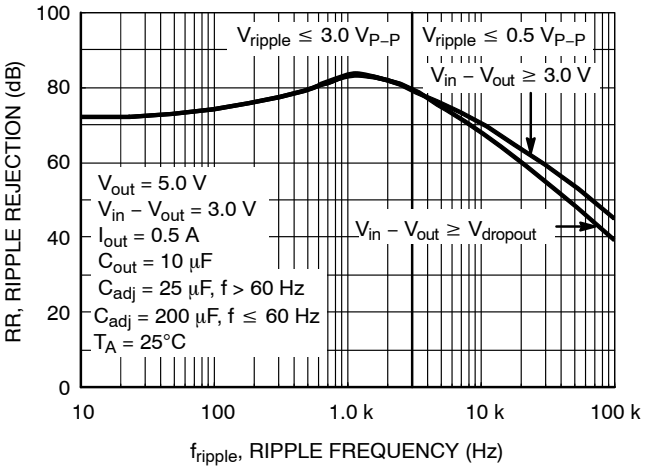


Figure 11. LM1117XTA Ripple Rejection vs. Frequency

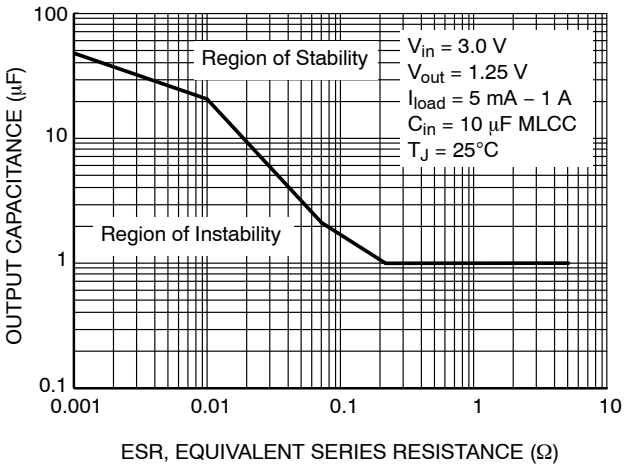


Figure 12. Output Capacitance vs. ESR

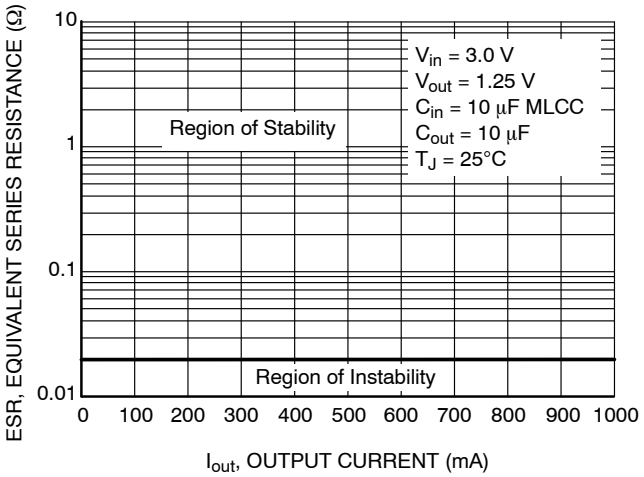


Figure 13. Typical ESR vs. Output Current

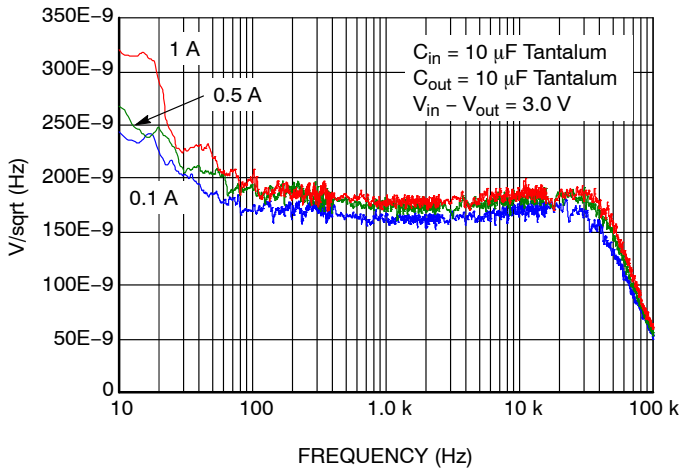


Figure 14. Output Spectral Noise Density vs. Frequency, V_{out} = 1V5

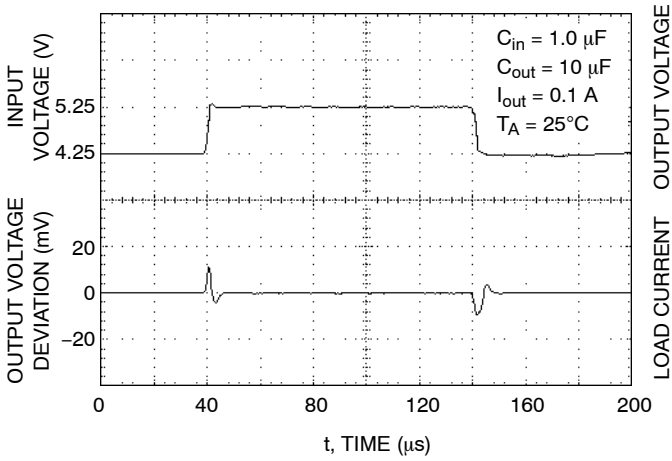


Figure 15. LM1117XT285 Line Transient Response

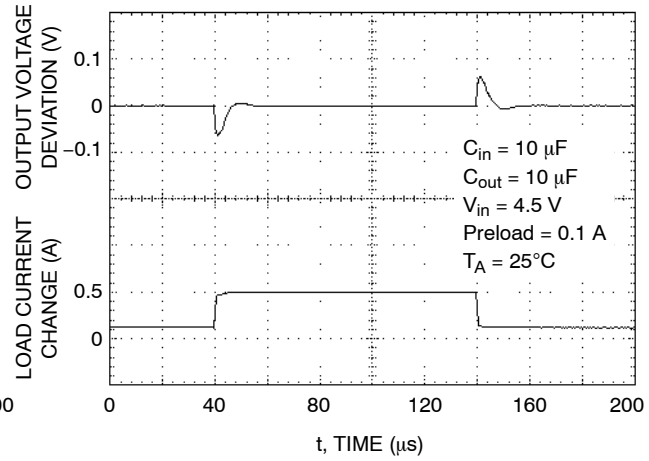


Figure 16. LM1117XT285 Load Transient Response

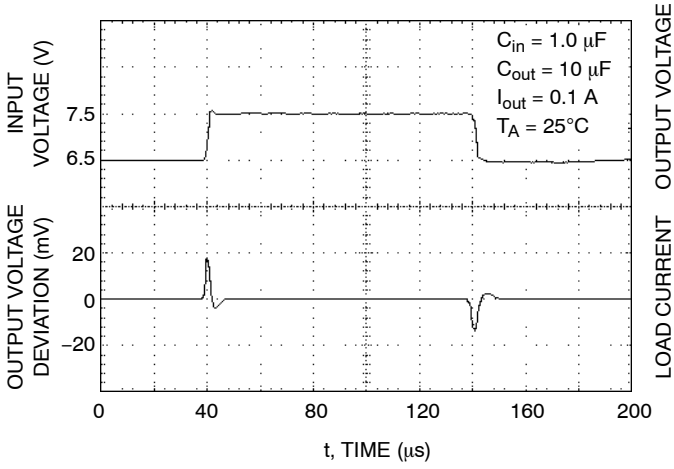


Figure 17. LM1117XT50 Line Transient Response

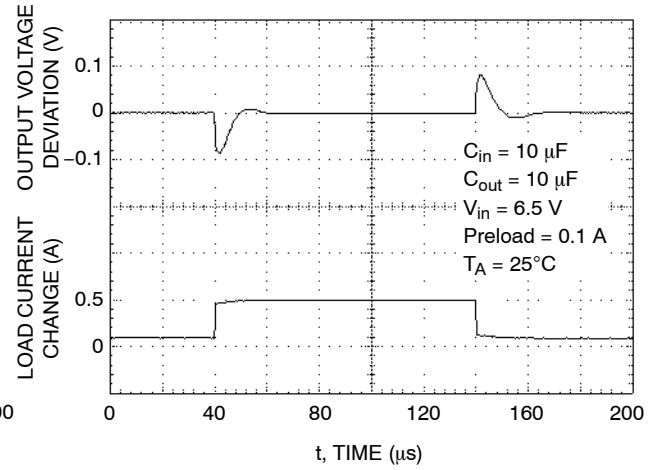


Figure 18. LM1117XT50 Load Transient Response

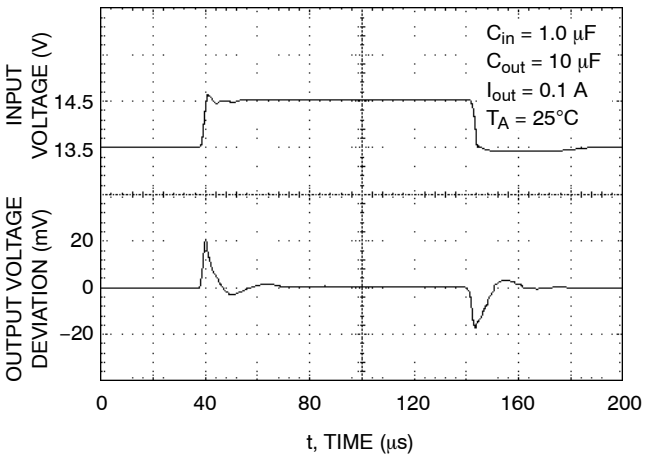


Figure 19. LM1117XT12 Line Transient Response

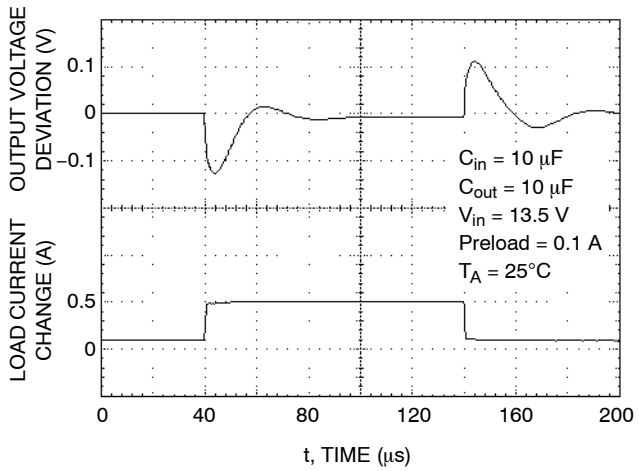


Figure 20. LM1117XT12 Load Transient Response

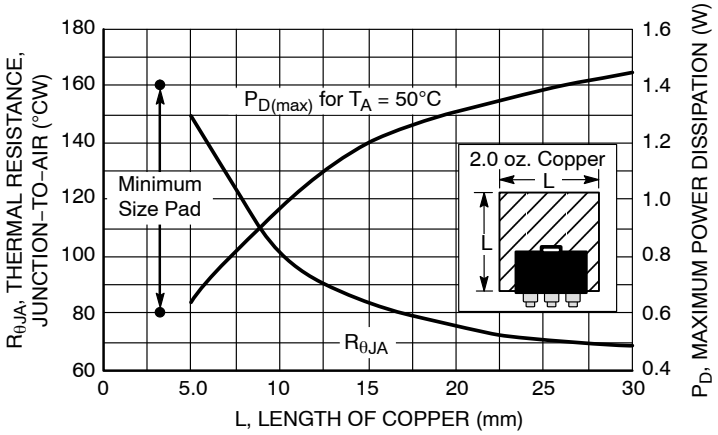


Figure 21. SOT−223 Thermal Resistance and Maximum Power Dissipation vs. P.C.B. Copper Length

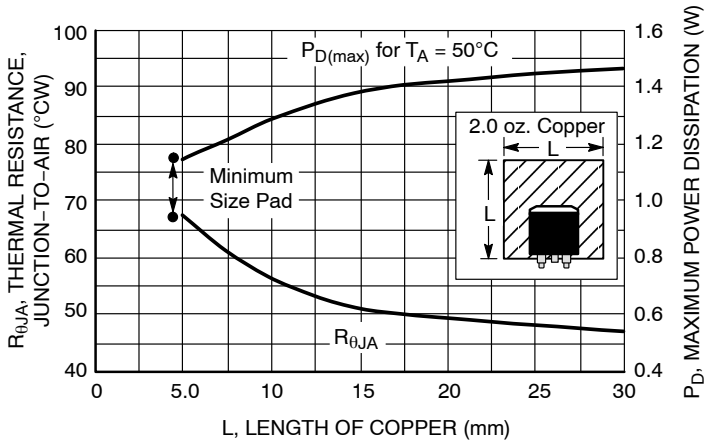


Figure 22. DPAK Thermal Resistance and Maximum Power Dissipation vs. P.C.B. Copper Length

APPLICATIONS INFORMATION

Introduction

The LM1117 features a significant reduction in dropout voltage along with enhanced output voltage accuracy and temperature stability when compared to older industry standard three−terminal adjustable regulators. These devices contain output current limiting, safe operating area compensation and thermal shutdown protection making them designer friendly for powering numerous consumer and industrial products. The LM1117 series is pin compatible with the older LM317 and its derivative device types.

Output Voltage

The typical application circuits for the fixed and adjustable output regulators are shown in Figures 23 and 24. The adjustable devices are floating voltage regulators. They develop and maintain the nominal 1.25 V reference voltage between the output and adjust pins. The reference voltage is programmed to a constant current source by resistor R1, and this current flows through R2 to ground to set the output voltage. The programmed current level is usually selected to be greater than the specified 5.0 mA minimum that is required for regulation. Since the adjust pin current, I_{adj}, is significantly lower and constant with respect to the programmed load current, it generates a small output voltage error that can usually be ignored. For the fixed output devices R1 and R2 are included within the device and the ground current I_{gnd}, ranges from 3.0 mA to 5.0 mA depending upon the output voltage.

External Capacitors

Input bypass capacitor C_{in} may be required for regulator stability if the device is located more than a few inches from the power source. This capacitor will reduce the circuit's sensitivity when powered from a complex source impedance and significantly enhance the output transient response. The input bypass capacitor should be mounted with the shortest possible track length directly across the regulator's input and ground terminals. A 10 μF ceramic or tantalum capacitor should be adequate for most applications.

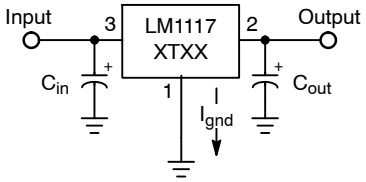
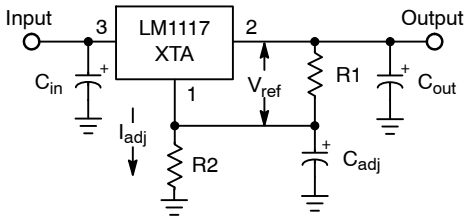


Figure 23. Fixed Output Regulator

Frequency compensation for the regulator is provided by capacitor C_{out} and its use is mandatory to ensure output stability. A minimum capacitance value of 4.7 μF with an equivalent series resistance (ESR) that is within the limits of 33 mΩ (typ) to 2.2 Ω is required. See Figures 12 and 13. The capacitor type can be ceramic, tantalum, or aluminum electrolytic as long as it meets the minimum capacitance value and ESR limits over the circuit's entire operating temperature range. Higher values of output capacitance can be used to enhance loop stability and transient response with the additional benefit of reducing output noise.



$$V_{out} = V_{ref}\left(1 + \frac{R2}{R1}\right) + I_{adj}\,R2$$

Figure 24. Adjustable Output Regulator

The output ripple will increase linearly for fixed and adjustable devices as the ratio of output voltage to the reference voltage increases. For example, with a 12 V regulator, the output ripple will increase by 12 V/1.25 V or 9.6 and the ripple rejection will decrease by 20 log of this ratio or 19.6 dB. The loss of ripple rejection can be restored to the values shown with the addition of bypass capacitor C_{adj}, shown in Figure 24. The reactance of C_{adj} at the ripple frequency must be less than the resistance of R1. The value of R1 can be selected to provide the minimum required load current to maintain regulation and is usually in the range of 100 Ω to 200 Ω.

$$C_{adj} > \frac{1}{2\,\pi\,f_{ripple}\,R1}$$

The minimum required capacitance can be calculated from the above formula. When using the device in an application that is powered from the AC line via a transformer and a full wave bridge, the value for C_{adj} is:

$$f_{ripple} = 120\ Hz,\ R1 = 120\ \Omega,\ \text{then}\ C_{adj} > 11.1\ \mu F$$

The value for C_{adj} is significantly reduced in applications where the input ripple frequency is high. If used as a post regulator in a switching converter under the following conditions:

$$f_{ripple} = 50\ kHz,\ R1 = 120\ \Omega,\ \text{then}\ C_{adj} > 0.027\ \mu F$$

Figures 10 and 11 shows the level of ripple rejection that is obtainable with the adjust pin properly bypassed.

Protection Diodes

The LM1117 family has two internal low impedance diode paths that normally do not require protection when used in the typical regulator applications. The first path connects between V_{out} and V_{in}, and it can withstand a peak surge current of about 15 A. Normal cycling of V_{in} cannot generate a current surge of this magnitude. Only when V_{in} is shorted or crowbarred to ground and C_{out} is greater than 50 μF, it becomes possible for device damage to occur. Under these conditions, diode D1 is required to protect the device. The second path connects between C_{adj} and V_{out}, and it can withstand a peak surge current of about 150 mA. Protection diode D2 is required if the output is shorted or crowbarred to ground and C_{adj} is greater than 1.0 μF.

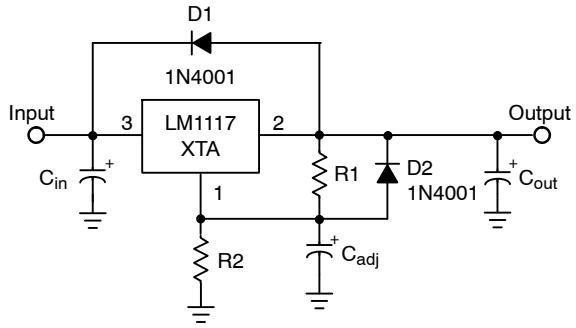


Figure 25. Protection Diode Placement

A combination of protection diodes D1 and D2 may be required in the event that V_{in} is shorted to ground and C_{adj} is greater than 50 μF. The peak current capability stated for the internal diodes are for a time of 100 μs with a junction temperature of 25°C. These values may vary and are to be used as a general guide.

Load Regulation

The LM1117 series is capable of providing excellent load regulation; but since these are three terminal devices, only partial remote load sensing is possible. There are two conditions that must be met to achieve the maximum available load regulation performance. The first is that the top side of programming resistor R1 should be connected as close to the regulator case as practicable. This will minimize the voltage drop caused by wiring resistance RW + from appearing in series with reference voltage that is across R1. The second condition is that the ground end of R2 should be connected directly to the load. This allows true Kelvin sensing where the regulator compensates for the voltage drop caused by wiring resistance RW −.

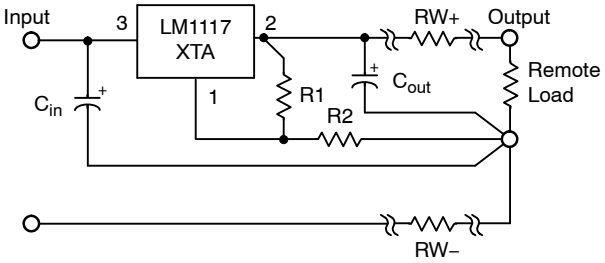


Figure 26. Load Sensing

Thermal Considerations

This series contains an internal thermal limiting circuit that is designed to protect the regulator in the event that the maximum junction temperature is exceeded. When activated, typically at 175°C, the regulator output switches off and then back on as the die cools. As a result, if the device is continuously operated in an overheated condition, the output will appear to be oscillating. This feature provides protection from a catastrophic device failure due to accidental overheating. It is not intended to be used as a substitute for proper heatsinking. The maximum device power dissipation can be calculated by:

$$P_D = \frac{T_{J(max)} - T_A}{R_{\theta JA}}$$

The devices are available in surface mount SOT−223 and DPAK packages. Each package has an exposed metal tab that is specifically designed to reduce the junction to air thermal resistance, $R_{\theta JA}$, by utilizing the printed circuit board copper as a heat dissipater. Figures 21 and 22 show typical $R_{\theta JA}$ values that can be obtained from a square pattern using economical single sided 2.0 ounce copper board material. The final product thermal limits should be tested and quantified in order to insure acceptable performance and reliability. The actual $R_{\theta JA}$ can vary considerably from the graphs shown. This will be due to any changes made in the copper aspect ratio of the final layout, adjacent heat sources, and air flow.

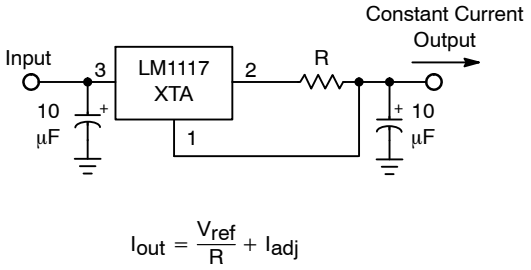


Figure 27. Constant Current Regulator

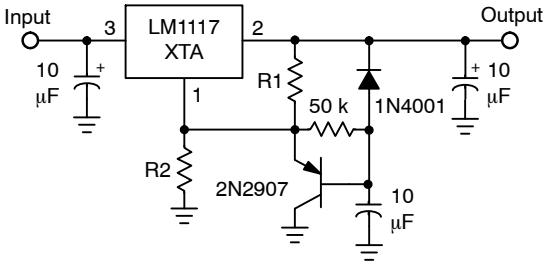
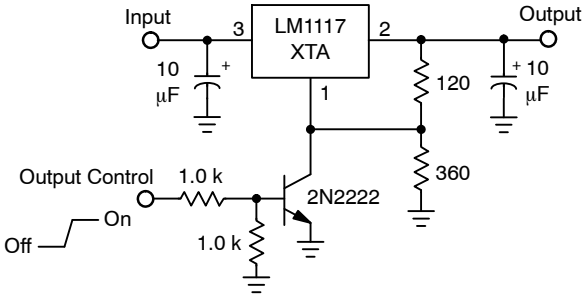
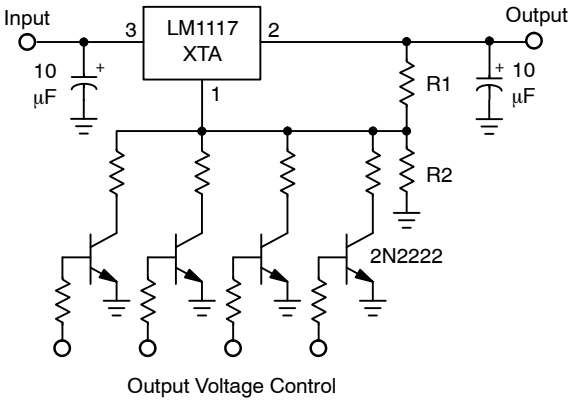


Figure 28. Slow Turn−On Regulator



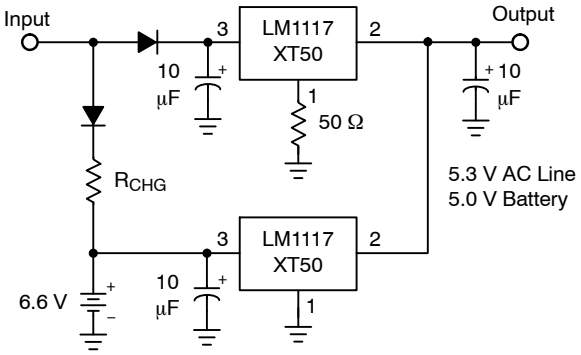
$V_{out(Off)} = V_{ref}$

Figure 29. Regulator with Shutdown



Resistor R2 sets the maximum output voltage. Each transistor reduces the output voltage when turned on.

Figure 30. Digitally Controlled Regulator



The 50 Ω resistor that is in series with the ground pin of the upper regulator level shifts its output 300 mV higher than the lower regulator. This keeps the lower regulator off until the input source is removed.

Figure 31. Battery Backed−Up Power Supply

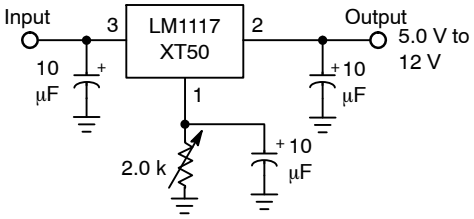


Figure 32. Adjusting Output of Fixed Voltage Regulators

ORDERING INFORMATION – (LM1117)

Device	Nominal Output Voltage	Package	Shipping†
LM1117MPX−ADJNOPB	Adjustable	SOT−223 (Pb−Free)	4000 / Tape & Reel
LM1117MPX−18NOPB	1.8		
LM1117MPX−25NOPB	2.5		
LM1117MPX−33NOPB	3.3		
LM1117MPX−50NOPB	5.0		

†For information on tape and reel specifications, including part orientation and tape sizes, please refer to our Tape and Reel Packaging Specifications Brochure, BRD8011/D.

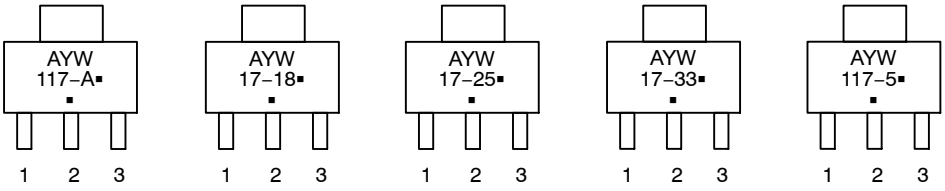
ORDERING INFORMATION – (LM1117I)

Device	Nominal Output Voltage	Package	Shipping†
LM1117IMPX−ADJNOPB	Adjustable	SOT−223 (Pb−Free)	4000 / Tape & Reel

†For information on tape and reel specifications, including part orientation and tape sizes, please refer to our Tape and Reel Packaging Specifications Brochure, BRD8011/D.

MARKING DIAGRAMS – LM1117

SOT−223
CASE 318H

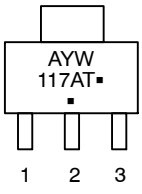


A = Assembly Location
Y = Year
W = Work Week
▪ = Pb−Free Package

(Note: Microdot may be in either location)

MARKING DIAGRAMS – LM1117I

SOT−223
CASE 318H



Adjustable

A = Assembly Location
Y = Year
W = Work Week
▪ = Pb−Free Package

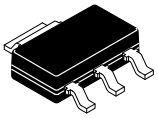
(Note: Microdot may be in either location)

MECHANICAL CASE OUTLINE
PACKAGE DIMENSIONS

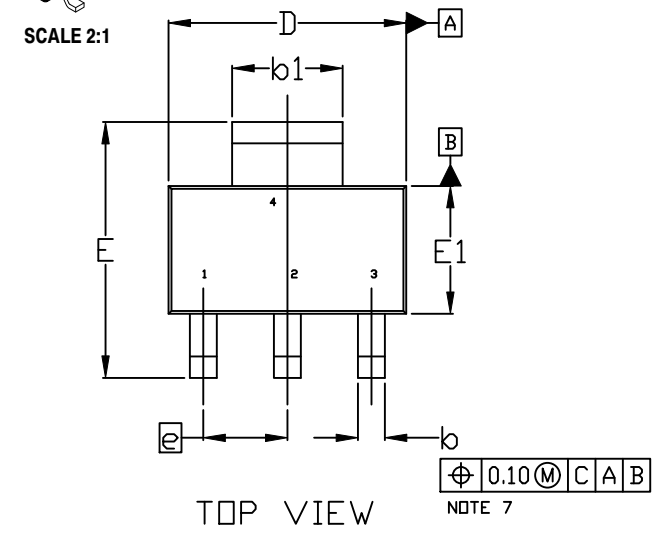
ON Semiconductor®

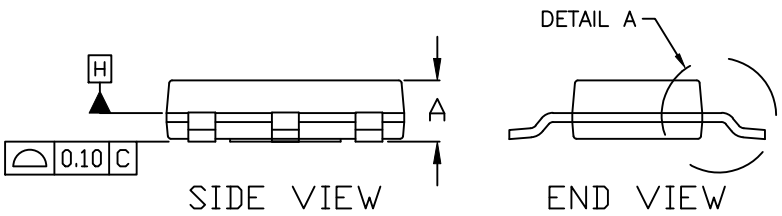
SOT−223
CASE 318H
ISSUE B

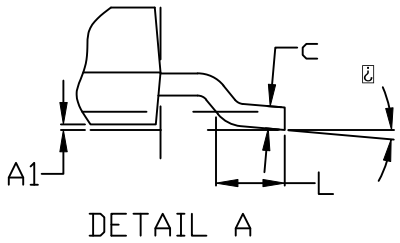
DATE 13 MAY 2020



SCALE 2:1

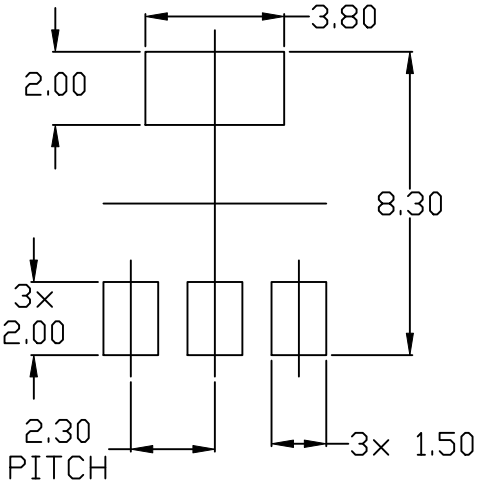






NOTES:
1. DIMENSIONING AND TOLERANCING PER ASME Y14.5M, 2009.
2. CONTROLLING DIMENSION: MILLIMETERS
3. DIMENSIONS D & E1 ARE DETERMINED AT DATUM H. DIMENSIONS DO NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS. SHALL NOT EXCEED 0.23mm PER SIDE.
4. LEAD DIMENSIONS b AND b1 DO NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBBAR PROTRUSION IS 0.08mm PER SIDE.
5. DATUMS A AND B ARE DETERMINED AT DATUM H.
6. A1 IS DEFINED AS THE VERTICAL DISTANCE FROM THE SEATING PLANE TO THE LOWEST POINT OF THE PACKAGE BODY.
7. POSITIONAL TOLERANCE APPLIES TO DIMENSIONS b AND b1.

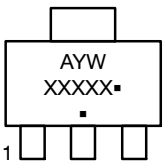
DIM	MILLIMETERS		
	MIN.	NOM.	MAX.
A	---	---	1.80
A1	0.02	0.06	0.11
b	0.60	0.74	0.88
b1	2.90	3.00	3.10
c	0.24	---	0.35
D	6.30	6.50	6.70
E	6.70	7.00	7.30
E1	3.30	3.50	3.70
e	2.30 BSC		
L	0.25	---	---
∠	0°	---	10°



RECOMMENDED MOUNTING FOOTPRINT

* For additional information on our Pb-Free strategy and soldering details, please download the ON Semiconductor Soldering and Mounting Techniques Reference Manual, SOLDERRM/D.

GENERIC MARKING DIAGRAM*



A = Assembly Location
Y = Year
W = Work Week
XXXXX = Specific Device Code
▪ = Pb−Free Package

(Note: Microdot may be in either location)

*This information is generic. Please refer to device data sheet for actual part marking. Pb−Free indicator, "G" or microdot "▪", may or may not be present. Some products may not follow the Generic Marking.

DOCUMENT NUMBER:	**98ASH70634A**	Electronic versions are uncontrolled except when accessed directly from the Document Repository. Printed versions are uncontrolled except when stamped "CONTROLLED COPY" in red.
DESCRIPTION:	**SOT−223**	**PAGE 1 OF 1**

ADDITIONAL INFORMATION

TECHNICAL PUBLICATIONS:
Technical Library: www.onsemi.com/design/resources/technical−documentation
onsemi Website: www.onsemi.com

ONLINE SUPPORT: www.onsemi.com/support
For additional information, please contact your local Sales Representative at
www.onsemi.com/support/sales

Single chip 2.4 GHz Transceiver nRF24L01

FEATURES

- True single chip GFSK transceiver
- Complete OSI Link Layer in hardware
- Enhanced ShockBurst™
- Auto ACK & retransmit
- Address and CRC computation
- On the air data rate 1 or 2Mbps
- Digital interface (SPI) speed 0-8 Mbps
- 125 RF channel operation
- Short switching time enable frequency hopping
- Fully RF compatible with nRF24XX
- 5V tolerant signal input pads
- 20-pin package (QFN20 4x4mm)
- Uses ultra low cost +/- 60 ppm crystal
- Uses low cost chip inductors and 2-layer PCB
- Power supply range: 1.9 to 3.6 V

APPLICATIONS

- Wireless mouse, keyboard, joystick
- Keyless entry
- Wireless data communication
- Alarm and security systems
- Home automation
- Surveillance
- Automotive
- Telemetry
- Intelligent sports equipment
- Industrial sensors
- Toys

GENERAL DESCRIPTION

nRF24L01 is a single chip radio transceiver for the world wide 2.4 - 2.5 GHz ISM band. The transceiver consists of a fully integrated frequency synthesizer, a power amplifier, a crystal oscillator, a demodulator, modulator and Enhanced ShockBurst™ protocol engine. Output power, frequency channels, and protocol setup are easily programmable through a SPI interface. Current consumption is very low, only 9.0mA at an output power of -6dBm and 12.3mA in RX mode. Built-in Power Down and Standby modes makes power saving easily realizable.

QUICK REFERENCE DATA

Parameter	Value	Unit
Minimum supply voltage	1.9	V
Maximum output power	0	dBm
Maximum data rate	2000	kbps
Supply current in TX mode @ 0dBm output power	11.3	mA
Supply current in RX mode @ 2000 kbps	12.3	mA
Temperature range	-40 to +85	°C
Sensitivity @ 1000 kbps	-85	dBm
Supply current in Power Down mode	900	nA

Table 1 nRF24L01 quick reference data

Type Number	Description	Version
nRF24L01	20 pin QFN 4x4, RoHS & SS-00259 compliant	D
nRF24L01 IC	Bare Dice	D
nRF24L01-EVKIT	Evaluation kit (2 test PCB, 2 configuration PCB, SW)	1.0

Table 2 nRF24L01 ordering information

BLOCK DIAGRAM

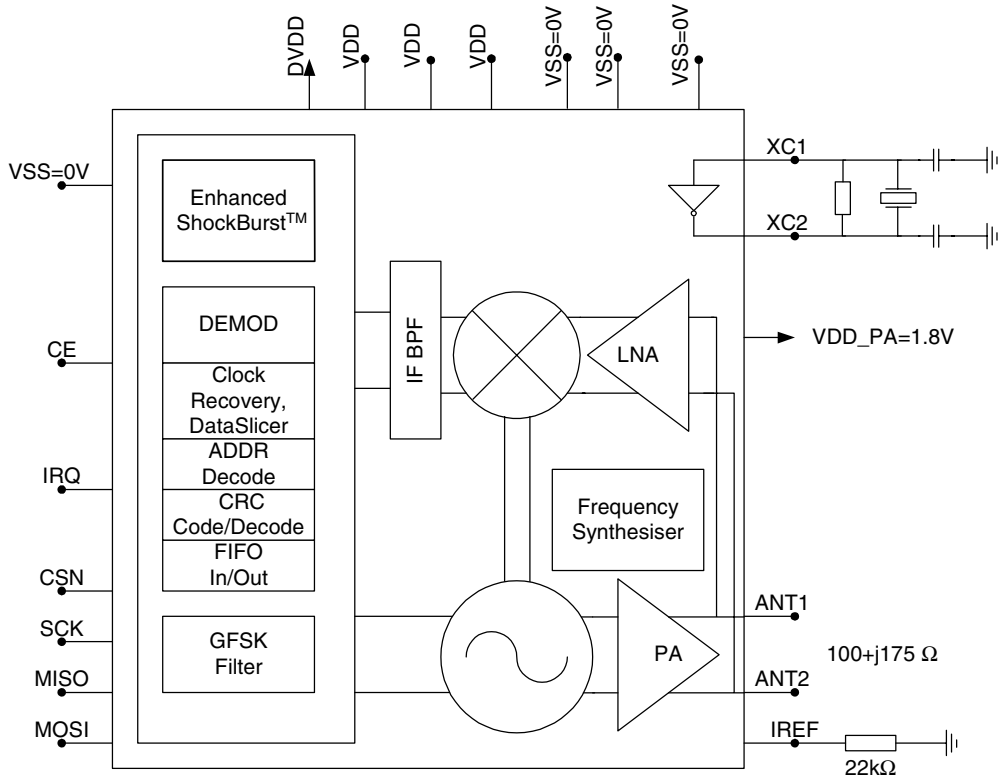


Figure 1 nRF24L01 with external components.

PIN FUNCTIONS

Pin	Name	Pin function	Description
1	CE	Digital Input	Chip Enable Activates RX or TX mode
2	CSN	Digital Input	SPI Chip Select
3	SCK	Digital Input	SPI Clock
4	MOSI	Digital Input	SPI Slave Data Input
5	MISO	Digital Output	SPI Slave Data Output, with tri-state option
6	IRQ	Digital Output	Maskable interrupt pin
7	VDD	Power	Power Supply (+3V DC)
8	VSS	Power	Ground (0V)
9	XC2	Analog Output	Crystal Pin 2
10	XC1	Analog Input	Crystal Pin 1
11	VDD_PA	Power Output	Power Supply (+1.8V) to Power Amplifier
12	ANT1	RF	Antenna interface 1
13	ANT2	RF	Antenna interface 2
14	VSS	Power	Ground (0V)
15	VDD	Power	Power Supply (+3V DC)
16	IREF	Analog Input	Reference current
17	VSS	Power	Ground (0V)
18	VDD	Power	Power Supply (+3V DC)
19	DVDD	Power Output	Positive Digital Supply output for de-coupling purposes
20	VSS	Power	Ground (0V)

Table 3 nRF24L01 pin function

PIN ASSIGNMENT

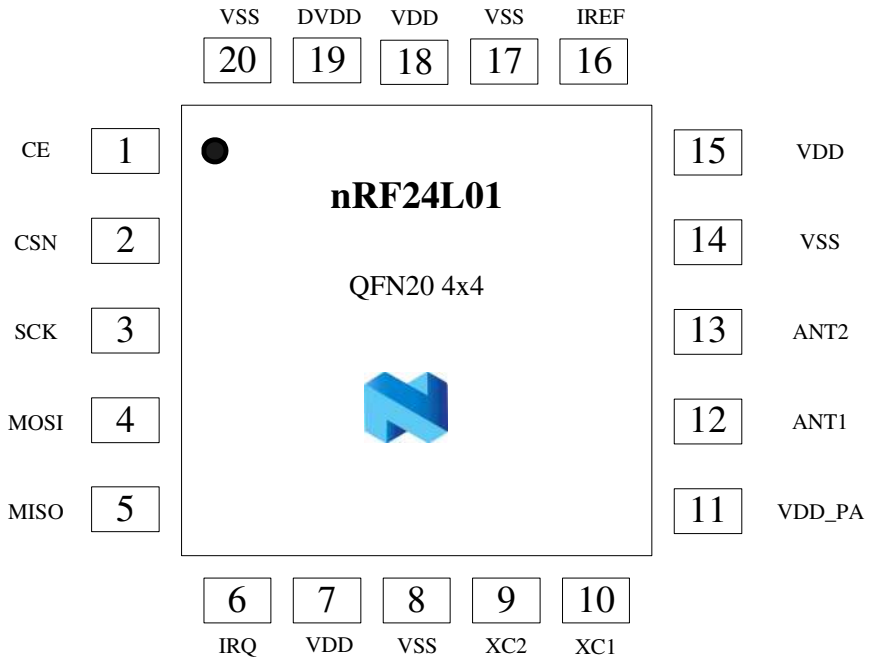


Figure 2 nRF24L01 pin assignment (top view) for a QFN20 4x4 package.

ELECTRICAL SPECIFICATIONS

Conditions: VDD = +3V, VSS = 0V, T_A = - 40ºC to + 85ºC

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
	Operating conditions					
VDD	Supply voltage		1.9	3.0	3.6	V
TEMP	Operating Temperature		-40	+27	+85	ºC
	Digital input pin					
V_{IH}	HIGH level input voltage	[1]	0.7VDD		5.25	V
V_{IL}	LOW level input voltage		VSS		0.3VDD	V
	Digital output pin					
V_{OH}	HIGH level output voltage (I_{OH}=-0.25mA)		VDD- 0.3		VDD	V
V_{OL}	LOW level output voltage (I_{OL}=0.25mA)		VSS		0.3	V
	General RF conditions					
f_{OP}	Operating frequency	[2]	2400		2525	MHz
f_{XTAL}	Crystal frequency			16		MHz
Δf_{1M}	Frequency deviation @ 1000kbps			±160		kHz
Δf_{2M}	Frequency deviation @ 2000kbps			±320		kHz
R_{GFSK}	Data rate ShockBurst™		>0		2000	kbps
$F_{CHANNEL}$	Channel spacing @ 1000kbps			1		MHz
$F_{CHANNEL}$	Channel spacing @ 2000kbps			2		MHz
	Transmitter operation					
P_{RF}	Maximum Output Power	[3]		0	+4	dBm
P_{RFC}	RF Power Control Range		16	18	20	dB
P_{RFCR}	RF Power Accuracy				±4	dB
P_{BW}	20dB Bandwidth for Modulated Carrier (2000kbps)			1800	2000	kHz
P_{RF1}	1st Adjacent Channel Transmit Power 2MHz				-20	dBm
P_{RF2}	2nd Adjacent Channel Transmit Power 4MHz				-50	dBm
I_{VDD}	Supply current @ 0dBm output power	[4]		11.3		mA
I_{VDD}	Supply current @ -18dBm output power			7.0		mA
I_{VDD}	Average Supply current @ -6dBm output power, Enhanced ShockBurst™	[5]		0.05		mA
I_{VDD}	Supply current in Standby-I mode	[6]		32		µA
I_{VDD}	Supply current in power down			900		nA

[1] All digital inputs handle up to 5.25V signal inputs. Keep in mind that the VDD of the nRF24L01 must match the V_{IH} of the driving device for output pins.
[2] Usable band is determined by local regulations
[3] Antenna load impedance = 15Ω+j88Ω
[4] Antenna load impedance = 15Ω+j88Ω. Effective data rate 1000kbps or 2000 kbps
[5] Antenna load impedance = 15Ω+j88Ω. Effective data rate 10kbps and full packets
[6] Given for a 12pF crystal. Current when using external clock is dependent on signal swing.

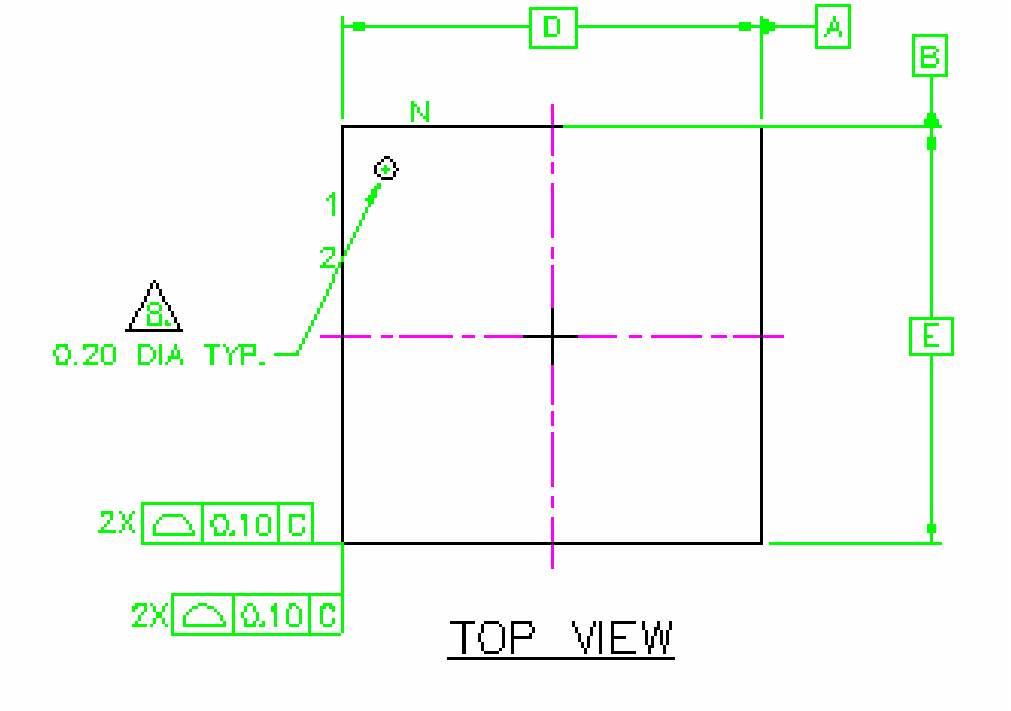
	Receiver operation					
I_{VDD}	Supply current one channel 2000kbps			12.3		mA
I_{VDD}	Supply current one channel 1000kbps			11.8		mA
RX_{SENS}	Sensitivity at 0.1%BER (@2000kbps)			-82		dBm
RX_{SENS}	Sensitivity at 0.1%BER (@1000kbps)			-85		dBm
C/I_{CO}	C/I Co-channel (@2000kbps)	[7]		[8]/[9]		dB
C/I_{1ST}	1st Adjacent Channel Selectivity C/I 2MHz			1/4		dB
C/I_{2ND}	2nd Adjacent Channel Selectivity C/I 4MHz			-21/-20		dB
C/I_{3RD}	3rd Adjacent Channel Selectivity C/I 6MHz			-27/-27		dB
C/I_{CO}	C/I Co-channel (@1000kbps)	[10]		[11]/[12]		dB
C/I_{1ST}	1st Adjacent Channel Selectivity C/I 1MHz			8/8		dB
C/I_{2ND}	2nd Adjacent Channel Selectivity C/I 2MHz			-22/-21		dB
C/I_{3RD}	3rd Adjacent Channel Selectivity C/I 3MHz			-30/-30		dB

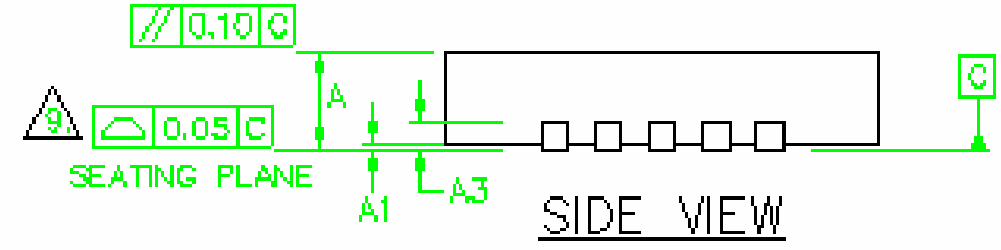
Table 4 nRF24L01 RF specifications

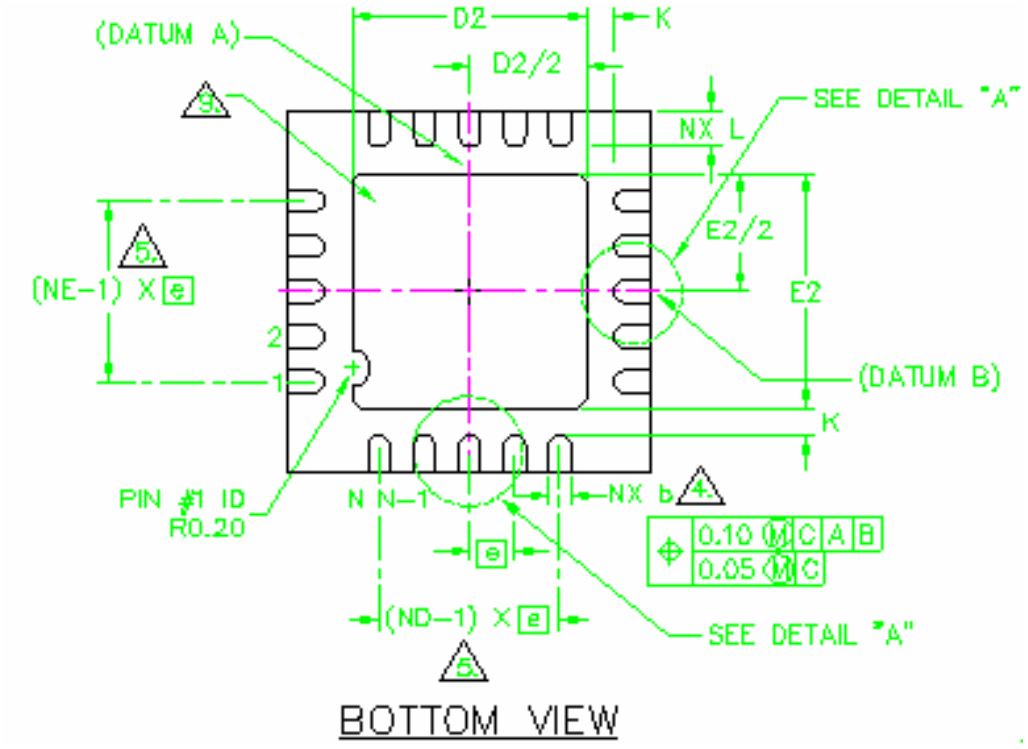
[7] Data rate is 2000kbps for the following C/I measurements
[8] According to ETSI EN 300 440-1 V1.3.1 (2001-09) page 27
[9] nRF24L01 equal modulation on interfering signal
[10] Data rate is 1000kbps for the following C/I measurements
[11] According to ETSI EN 300 440-1 V1.3.1 (2001-09) page 27
[12] nRF24L01 equal modulation on interfering signal

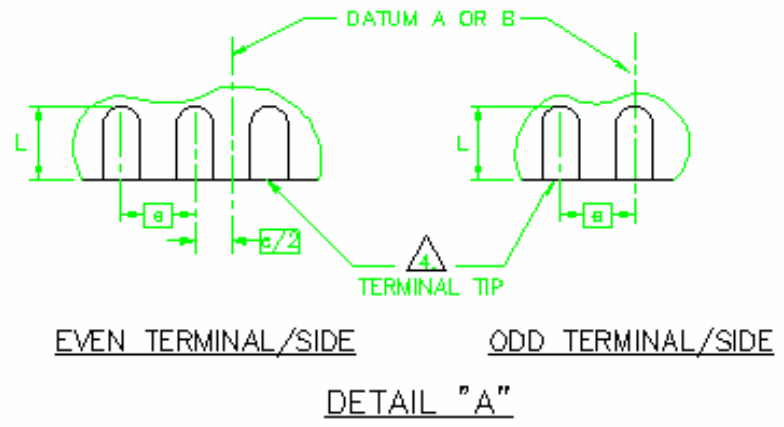
PACKAGE OUTLINE

nRF24L01 uses the QFN20 4x4 package, with matt tin plating.









Package Type		A	A1	A3	K	D/E	e	D2/E2	L	L1	b
Saw QFN20 (4x4 mm)	**Min**	0.80	0.00					2.50	0.35		0.18
	Typ.	0.85	0.02	0.20	0.20	4.0 BSC[13]	0.5 BSC	2.60	0.40	0.15	0.25
	Max	0.95	0.05	REF.	min			2.70	0.45	max	0.30

Figure 3 nRF24L01 Package Outline.

[13] BSC: Basic Spacing between Centers, ref. JEDEC standard 95, page 4 17-11/A

Package marking:

n	R	F		*B*	*X*
2	4	L	0	1	
Y	*Y*	*W*	*W*	*L*	*L*

Abbreviations:

B – Build Code, i.e. unique code for production sites, package type and test platform
X – "X" grade, i.e. Engineering Samples (optional)
YY – 2 digit Year number
WW – 2 digit Week number
LL – 2 letter wafer lot number code

Absolute Maximum Ratings

Supply voltages
VDD ..... - 0.3V to + 3.6V
VSS 0V
Input voltage
V_I ..... - 0.3V to 5.25V
Output voltage
V_O ..... VSS to VDD

Total Power Dissipation
P_D (T_A=85°C) 60mW

Temperatures
Operating Temperature…. - 40°C to + 85°C
Storage Temperature……. - 40°C to + 125°C

Note: Stress exceeding one or more of the limiting values may cause permanent damage to the device.

ATTENTION!

Electrostatic Sensitive Device
Observe Precaution for handling.

Glossary of Terms

Term	Description
ACK	Acknowledgement
ART	Auto Re-Transmit
CE	Chip Enable
CLK	Clock
CRC	Cyclic Redundancy Check
CSN	Chip Select NOT
ESB	Enhanced ShockBurst™
GFSK	Gaussian Frequency Shift Keying
IRQ	Interrupt Request
ISM	Industrial-Scientific-Medical
LNA	Low Noise Amplifier
LSB	Least Significant Bit
LSByte	Least Significant Byte
Mbps	Megabit per second
MCU	Micro Controller Unit
MISO	Master In Slave Out
MOSI	Master Out Slave In
MSB	Most Significant Bit
MSByte	Most Significant Byte
PCB	Printed Circuit Board
PER	Packet Error Rate
PID	Packet Identity Bits
PLD	Payload
PRX	Primary RX
PTX	Primary TX
PWR_DWN	Power Down
PWR_UP	Power Up
RoHS	Restriction of use of Certain Hazardous Substances
RX	Receive
RX_DR	Receive Data Ready
SPI	Serial Peripheral Interface
TX	Transmit
TX_DS	Transmit Data Sent

Table 5 Glossary

FUNCTIONAL DESCRIPTION

Modes of operation

The nRF24L01 can be set in the following main modes depending on the level of the following primary I/Os and configuration registers:

Mode	PWR_UP register	PRIM_RX register	CE	FIFO state
RX mode	1	1	1	-
TX mode	1	0	1	Data in TX FIFO
TX mode	1	0	1➔0	Stays in TX mode until packet transmission is finished
Standby-II	1	0	1	TX FIFO empty
Standby-I	1	-	0	No ongoing packet transmission
Power Down	0	-	-	-

Table 6 nRF24L01 main modes

An overview of the nRF24L01 I/O pins in different modes is given in Table 7.

Pin functions in the different modes of nRF24L01

Pin Name	Direction	TX Mode	RX Mode	Standby Modes	Power Down
CE	Input	High Pulse >10µs	High	Low	-
CSN	Input	SPI Chip Select, active low			
SCK	Input	SPI Clock			
MOSI	Input	SPI Serial Input			
MISO	Tri-state Output	SPI Serial Output			
IRQ	Output	Interrupt, active low			

Table 7 Pin functions of the nRF24L01

Standby Modes

Standby-I mode is used to minimize average current consumption while maintaining short start up times. In this mode, part of the crystal oscillator is active. In Standby-II mode some extra clock buffers are active compared to Standby-I mode. Standby-II occurs when CE is held high on a PTX device with empty TX FIFO. The configuration word content is maintained during Standby modes. SPI interface may be activated. For start up time see Table 13.

Power Down Mode

In power down nRF24L01 is disabled with minimal current consumption. When entering this mode the device is not active, but all registers values available from the SPI interface are maintained during power down and the SPI interface may be activated (CSN=0). For start up time see Table 13. The power down is controlled by the PWR_UP bit in the CONFIG register.

Packet Handling Methods

nRF24L01 has the following Packet Handling Methods:

- ShockBurst™ (compatible with nRF2401, nRF24E1, nRF2402 and nRF24E2 with 1Mbps data rate, see page 26)
- Enhanced ShockBurst™

ShockBurst™

ShockBurst™ makes it possible to use the high data rate offered by nRF24L01 without the need of a costly, high-speed microcontroller (MCU) for data processing/clock recovery. By placing all high speed signal processing related to RF protocol on-chip, nRF24L01 offers the application microcontroller a simple SPI compatible interface, the data rate is decided by the interface-speed the micro controller itself sets up. By allowing the digital part of the application to run at low speed, while maximizing the data rate on the RF link, ShockBurst™ reduces the average current consumption in applications.
In ShockBurst™ RX, IRQ notifies the MCU when a valid address and payload is received respectively. The MCU can then clock out the received payload from an nRF24L01 RX FIFO.
In ShockBurst™ TX, nRF24L01 automatically generates preamble and CRC, see Table 12. IRQ notifies the MCU that the transmission is completed. All together, this means reduced memory demand in the MCU resulting in a low cost MCU, as well as reduced software development time. nRF24L01 has a three level deep RX FIFO (shared between 6 pipes) and a three level deep TX FIFO. The MCU can access the FIFOs at any time, in power down mode, in standby modes, and during RF packet transmission. This allows the slowest possible SPI interface compared to the average data-rate, and may enable usage of an MCU without hardware SPI.

Enhanced ShockBurst™

Enhanced ShockBurst™ is a packet handling method with functionality that makes bi-directional link protocol implementation easier and more efficient. In a typical bi-directional link, one will let the terminating part acknowledge received packets from the originating part in order to make it possible to detect data loss. Data loss can then be recovered by retransmission. The idea with Enhanced ShockBurst™ is to let nRF24L01 handle both acknowledgement of received packets and retransmissions of lost packets, without involvement from the microcontroller.

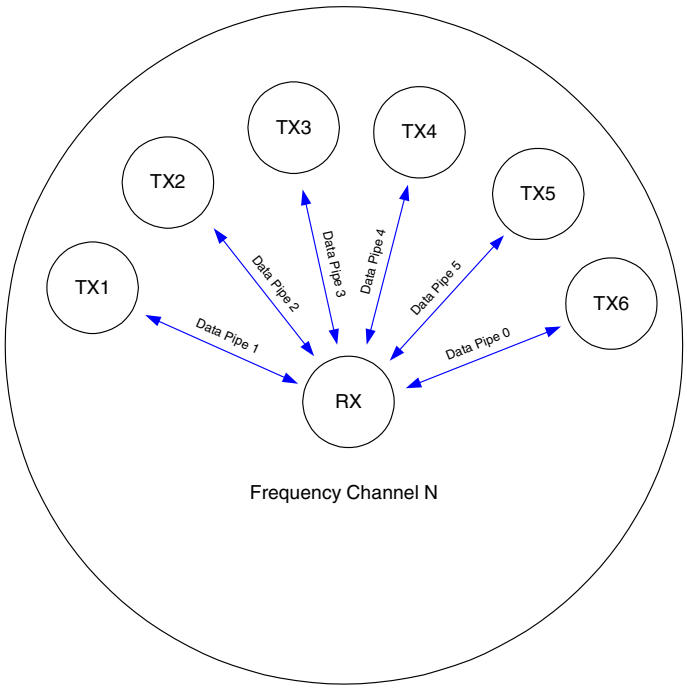


Figure 4: nRF24L01 in a star network configuration

An nRF24L01 configured as primary RX (PRX) will be able to receive data trough 6 different data pipes, see Figure 4. A data pipe will have a unique address but share the same frequency channel. This means that up to 6 different nRF24L01 configured as primary TX (PTX) can communicate with one nRF24L01 configured as PRX, and the nRF24L01 configured as PRX will be able to distinguish between them. Data pipe 0 has a unique 40 bit configurable address. Each of data pipe 1-5 has an 8 bit unique address and shares the 32 most significant address bits. All data pipes can perform full Enhanced ShockBurst™ functionality.
nRF24L01 will use the data pipe address when acknowledging a received packet. This means that nRF24L01 will transmit ACK with the same address as it receives payload at. In the PTX device data pipe 0 is used to received the acknowledgement, and therefore the receive address for data pipe 0 has to be equal to the transmit address to be able to receive the acknowledgement. See Figure 5 for addressing example.

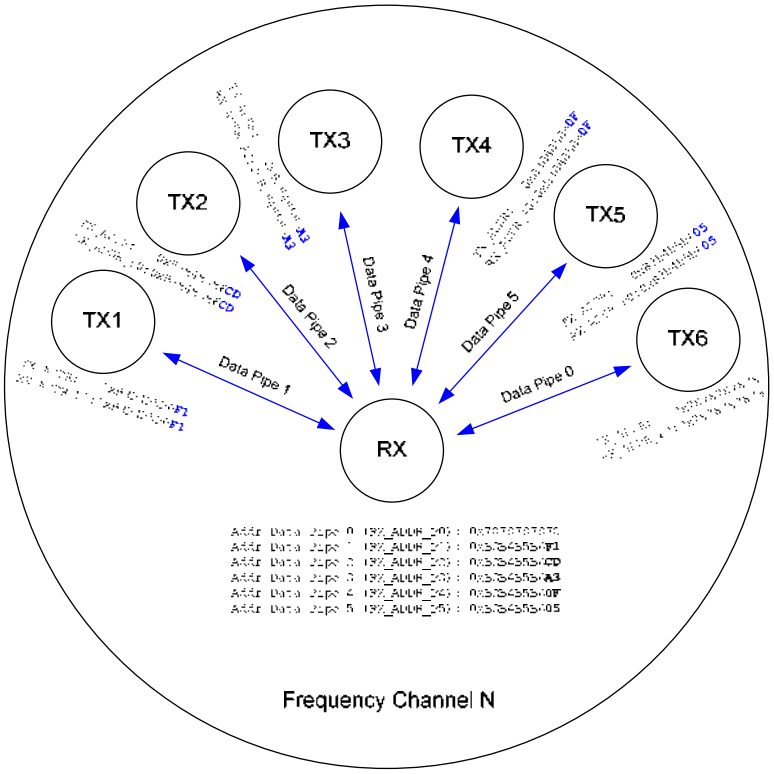


Figure 5: Example on how the acknowledgement addressing is done

An nRF24L01 configured as PTX with Enhanced ShockBurst™ enabled, will use the ShockBurst™ feature to send a packet whenever the microcontroller wants to. After the packet has been transmitted, nRF24L01 will switch on its receiver and expect an acknowledgement to arrive from the terminating part. If this acknowledgement fails to arrive, nRF24L01 will retransmit the same packet until it receives an acknowledgement or the number of retries exceeds the number of allowed retries given in the SETUP_RETR_ARC register. If the number of retries exceeds the number of allowed retries, this will be showed by the STATUS register bit MAX_RT which gives an interrupt.
Whenever an acknowledgement is received by an nRF24L01 it will consider the last transmitted packet as delivered. It will then be cleared from the TX FIFO, and the TX_DS IRQ source will be set high.

With Enhanced ShockBurst™ nRF24L01 offers the following benefits:

- Highly reduced current consumption due to short time on air and sharp timing when operating with acknowledgement traffic
- Lower system cost. Since the nRF24L01 handles all the high-speed link layer operations, like re-transmission of lost packet and generating acknowledgement to received packets, it is no need for hardware SPI on the system microcontroller to interface the nRF24L01. The interface can be done by using general purpose IO pins on a low cost microcontroller where the SPI is emulated in firmware. With the nRF24L01 this will be sufficient speed even when running a bi-directional link.
- Greatly reduced risk of “on-air” collisions due to short time on air
- Easier firmware development since the link layer is integrated on chip

Enhanced ShockBurst™ Transmitting Payload:

1. The configuration bit PRIM_RX has to be low.
2. When the application MCU has data to send, the address for receiving node (TX_ADDR) and payload data (TX_PLD) has to be clocked into nRF24L01 via the SPI interface. The width of TX-payload is counted from number of bytes written into the TX FIFO from the MCU. TX_PLD must be written continuously while holding CSN low. TX_ADDR does not have to be rewritten if it is unchanged from last transmit. If the PTX device shall receive acknowledge, data pipe 0 has to be configured to receive the acknowledgement. The receive address for data pipe 0 (RX_ADDR_P0) has to be equal to the transmit address (TX_ADDR) in the PTX device. For the example in Figure 5 the following address settings have to be performed for the TX5 device and the RX device:
   TX5 device: TX_ADDR = 0xB3B4B5B605
   TX5 device: RX_ADDR_P0 = 0xB3B4B5B605
   RX device: RX_ADDR_P5 = 0xB3B4B5B605
3. A high pulse on CE starts the transmission. The minimum pulse width on CE is 10 µs.
4. nRF24L01 ShockBurst™:
   - Radio is powered up
   - 16 MHz internal clock is started.
   - RF packet is completed (see the packet description)
   - Data is transmitted at high speed (1 Mbps or 2 Mbps configured by MCU).
5. If auto acknowledgement is activated (ENAA_P0=1) the radio goes into RX mode immediately. If a valid packet has been received in the valid acknowledgement time window, the transmission is considered a success. The TX_DS bit in the status register is set high and the payload is removed from TX FIFO. If a valid acknowledgement is not received in the specified time window, the payload is resent (if auto retransmit is enabled). If the auto retransmit counter (ARC_CNT) exceeds the programmed maximum limit (ARC), the MAX_RT bit in the status register is set high. The payload in TX FIFO is NOT removed. The IRQ pin will be active when MAX_RT or TX_DS is high. To turn off the IRQ pin, the interrupt source must be reset by writing to the status register (see Interrupt chapter). If no acknowledgement is received for a packet after the maximum number of retries, no further packets can be sent before the MAX_RX interrupt is cleared. The packet loss counter (PLOS_CNT) is incremented at each MAX_RT interrupt. I.e. ARC_CNT counts the number of retries that was required to get a single packet through. PLOS_CNT counts the number of packets that did not get through after maximum number of retries.
6. The device goes into Standby-I mode if CE is low. Otherwise next payload in TX FIFO will be sent. If TX FIFO is empty and CE is still high, the device will enter Standby-II mode.
7. If the device is in Standby-II mode, it will go to Standby-I mode immediately if CE is set low.

Enhanced ShockBurst™ Receive Payload:

1. RX is selected by setting the PRIM_RX bit in the configuration register to high. All data pipes that shall receive data must be enabled (EN_RXADDR register),

auto acknowledgement for all pipes running Enhanced ShockBurst™ has to be enabled (EN_AA register), and the correct payload widths must be set (RX_PW_Px registers). Addresses have to be set up as described in item 2 in the Enhanced ShockBurst™ transmit payload chapter above.

2. Active RX mode is started by setting CE high.
3. After 130µs nRF24L01 is monitoring the air for incoming communication.
4. When a valid packet has been received (matching address and correct CRC), the payload is stored in the RX-FIFO, and the RX_DR bit in status register is set high. The IRQ pin will be active when RX_DR is high. RX_P_NO in status register will indicate what data pipe the payload has been received in.
5. If auto acknowledgement is enabled, an acknowledgement is sent back.
6. MCU sets the CE pin low to enter Standby-I mode (low current mode).
7. MCU can clock out the payload data at a suitable rate via the SPI interface.
8. The device is now ready for entering TX or RX mode or power down mode.

Two way communication with payload in both directions

If payload shall be sent in both directions, the PRIM_RX register must be toggled by redefining the device from PRX to PTX or vice versa. The controlling processors must handle the synchronicity between a PTX and a PRX. Data buffering in both RX FIFO and TX FIFO simultaneously is possible, but restricted to data pipes 1 to 5. The third level in TX FIFO shall only be written in RX, TX or Standby-II mode if data is stored in RX FIFO

Auto Acknowledgement (RX)

The auto acknowledgement function reduces the load of the external microcontroller, and may remove the need for dedicated SPI hardware in a mouse/keyboard or comparable systems, and hence reduce cost and average current consumption. Auto acknowledgement can be configured individually for each data pipe via the SPI interface.

If auto acknowledgement is enabled and a valid packet (correct data pipe address and CRC) is received, the device will enter TX mode and send an acknowledgement packet. After the device has sent the acknowledgement packet, normal operation resumes, and the mode is determined by the PRIM_RX register and CE pin.

Auto Re-Transmission (ART) (TX)

An auto retransmission function is available. It will be used at the TX side in an auto acknowledgement system. In the SETUP_RETR register it will be possible to state how many times the data in the data register will be resent if data is not acknowledged. After each sending, the device will enter RX mode and wait a specified time period for acknowledgement. When the acknowledgement packet is received, the device will return to normal transmit function. If there is no more unsent data in the TX FIFO and the CE pin is low, the device will go into Standby-I mode. If the acknowledgement is not received, the device will go back to TX mode and resend the data. This will continue until acknowledgment is received, or a time out occurs

(i.e. the maximum number of sending is reached). The only way to reset this is to set the PWR_UP bit low or let the auto retransmission finish. A packet loss counter will be incremented each time a packet does not succeed to reach the destination before time out. (Time out is indicated by the MAX_RT interrupt.) The packet loss counter is reset when writing to the RF channel register.

Packet Identity (PID) and CRC used by Enhanced ShockBurst™

Each packet contains a two bit wide PID field to detect if the received packet is new or resent. The PID will prevent that the PRX device presents the same payload more than once to the microcontroller. This PID field is incremented at the TX side for each new packet received via the SPI interface. The PID and CRC field is used by the PRX device to determine whether a packet is resent or new. When several data is lost on the link, the PID fields may in some cases become equal to last received PID. If a packet has the same PID as the previous packet, nRF24L01 will compare the CRC sums from both packets. If they also are equal, the last received packet is considered as a copy of the previous and is discarded.

1: PRX device:
The PRX device compares the received PID with the last PID. If the PID fields are different, the packet is considered to be new. If the PID is equal to last received PID, the received packet might be the same as last time. The receiver must check if the CRC is equal to the previous CRC. If the CRC is equal to the previous one, the packet is probably the same, and will be discarded.

2: PTX device:
The transmitter increments the PID field each time it sends a new packet.

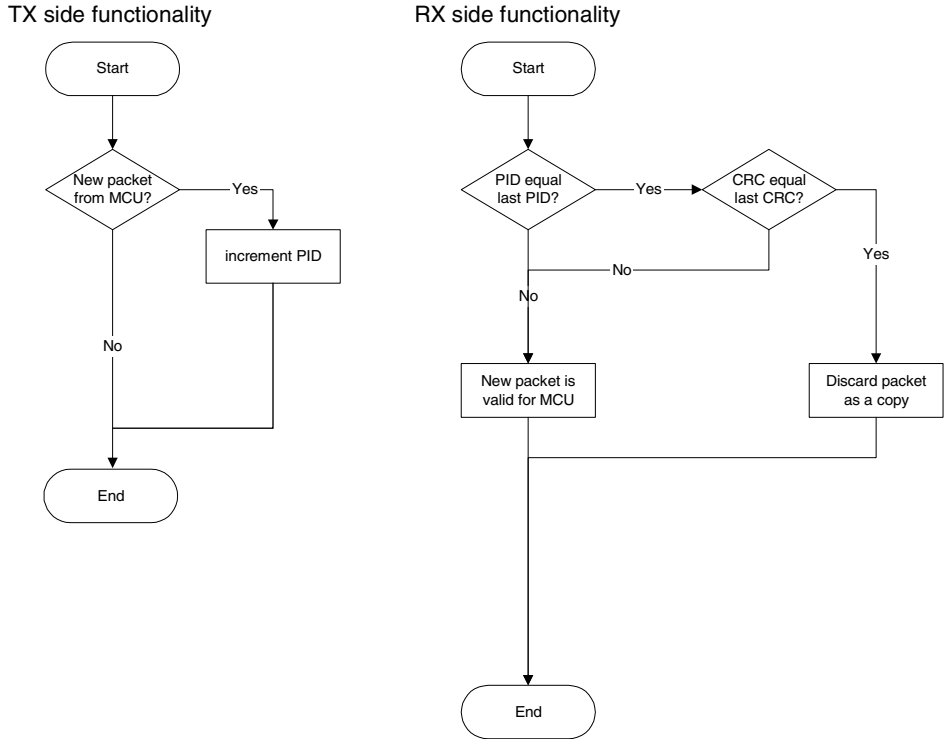


Figure 6 PID generation/detection

The length of the CRC is configurable through the SPI interface. It is important to notice that the CRC is calculated over the whole packet including address, PID and payload. No packet is accepted as correct if the CRC fails. This is an extra requirement for packet acceptance that is not illustrated in the figure above.

Stationary Disturbance Detection – CD

Carrier Detect (CD) is set high when an in-band RF signal is detected in RX mode, otherwise CD is low. The internal CD signal is filtered before presented to CD register. The internal CD signal must be high for at least 128µs.
In Enhanced ShockBurst™ it is recommended to use the Carrier Detect functionality only when the PTX device does not succeed to get packets through, as indicated by the MAX_RT interrupt for single packets and by the packet loss counter (PLOS_CNT) if several packets are lost. If the PLOS_CNT in the PTX device indicates to high rate of packet losses, the device can be configured to a PRX device for a short time (T_{stbt2a} + CD-filter delay = 130µs+128µs = 258µs) to check CD. If CD was high (jam situation), the frequency channel should be changed. If CD was low (out of range), it may continue on the same frequency channel, but perform other adjustments. (A dummy write to the RF_CH will clear the PLOS_CNT.)

Data Pipes

nRF24L01 configured as PRX can receive data addressed to 6 different data pipes in one physical frequency channel. Each data pipe has its own unique address and can be configured to have individual behavior.

The data pipes are enabled with the bits in the EN_RXADDR register. By default only data pipe 0 and 1 are enabled.
The address for each data pipe is configured in the RX_ADDR_Px registers. Always ensure that none of the data pipes have the exact same address.

Data pipe 0 has a unique 40 bit configurable address. Data pipes 1-5 share the 32 most significant address bits and have only the LSByte unique for each data pipe. Figure 7 shows an example of how data pipes 0-5 are addressed. All pipes can have up to 40 bit address, but for pipe 1-5 only the LSByte is different, and the LSByte must be unique for all pipes.

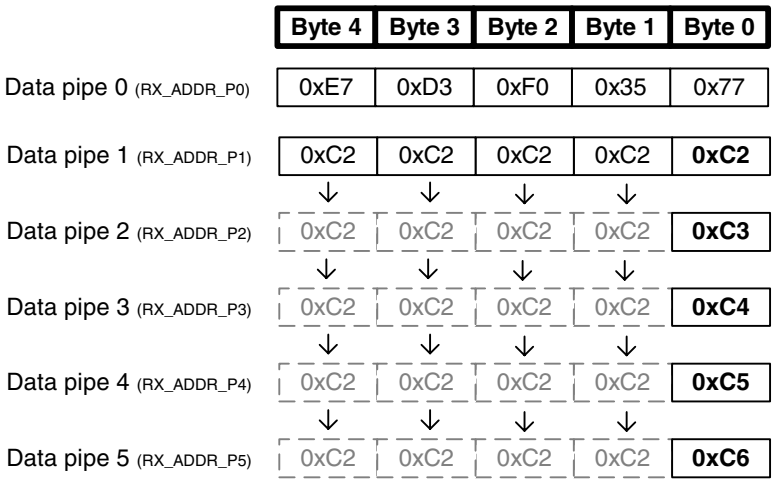


Figure 7: Addressing data pipes 0-5

When a packet has been received at one of the data pipes and the data pipe is setup to generate acknowledgement, nRF24L01 will generate an acknowledgement with an address that equals the data pipe address where the packet was received.

Some configuration settings are common to all data pipes and some are individual. The following settings are common to all data pipes:

- CRC enabled/disabled (CRC always enabled when ESB is enabled)
- CRC encoding scheme
- RX address width
- Frequency channel
- RF data rate
- LNA gain
- RF output power

DEVICE CONFIGURATION

All configuration of nRF24L01 is defined by values in some configuration registers. All these registers are writable via the SPI interface.

SPI Interface

The SPI interface is a standard SPI interface with a maximum data rate of 10Mbps. Most registers are readable.

SPI Instruction Set

The available commands to be used on the SPI interface are shown below. Whenever CSN is set low the interface expects an instruction. Every new instruction must be started by a high to low transition on CSN.

In parallel to the SPI instruction word applied on the MOSI pin, the STATUS register is shifted serially out on the MISO pin.

The serial shifting SPI commands is on the format:

<**Instruction word**: MSBit to LSBit (one byte)>

<**Data bytes**: LSByte to MSByte, MSBit in each byte first>

See Figure 8 and Figure 9.

Instruction Name	Instruction Format [binary]	# Data Bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read registers. AAAAA = 5 bit Memory Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write registers. AAAAA = 5 bit Memory Map Address *Executable in power down or standby modes only.*
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation will always start at byte 0. Payload will be deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Used in TX mode. Write TX-payload: 1 – 32 bytes. A write operation will always start at byte 0.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, i.e. acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last sent payload. Packets will be repeatedly resent as long as CE is high. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

Table 8 Instruction set for the nRF24L01 SPI interface.

The W_REGISTER and R_REGISTER may operate on single or multi-byte registers. When accessing multi-byte registers one will read or write MSBit of LSByte first. The

writing can be terminated before all bytes in a multi-byte register has been written. In this case the unwritten MSByte(s) will remain unchanged. E.g. the LSByte of RX_ADDR_P0 can be modified by writing only one byte to the RX_ADDR_P0 register. The content of the status register will always be read to MISO after a high to low transition on CSN.

Interrupt

The nRF24L01 has an active low interrupt pin (IRQ). The interrupt pin is activated when TX_DS, RX_DR or MAX_RT is set high in status register. When MCU writes '1' to the interrupt source, the IRQ pin will go inactive. The interrupt mask part of the CONFIG register is used to mask out the interrupt sources that are allowed to set the IRQ pin low. By setting one of the MASK bits high, the corresponding interrupt source will be disabled. By default all interrupt sources are enabled.

SPI Timing

The interface supports SPI. SPI operation and timing is given in Figure 8 to Figure 10 and in Table 9 and Table 10. The device must be in one of the standby modes or power down mode before writing to the configuration registers. In Figure 8 to Figure 10 the following notations are used:
Cn – SPI Instruction Bit
Sn – Status Register Bit
Dn – Data Bit (note: LSByte to MSByte, MSBit in each byte first)

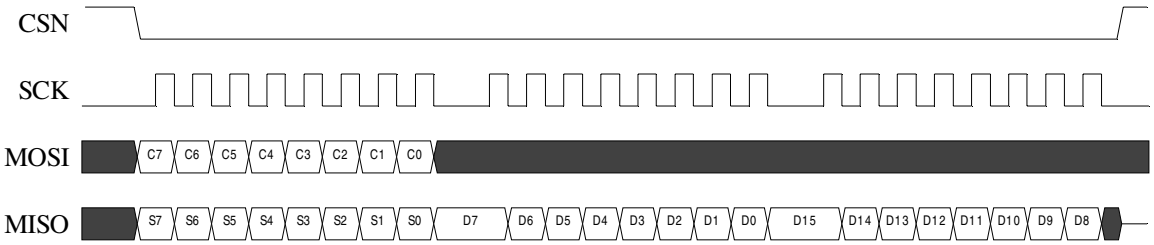


Figure 8 SPI read operation.

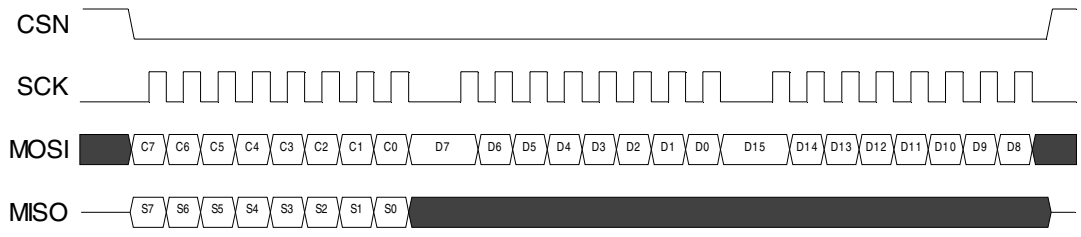


Figure 9 SPI write operation.

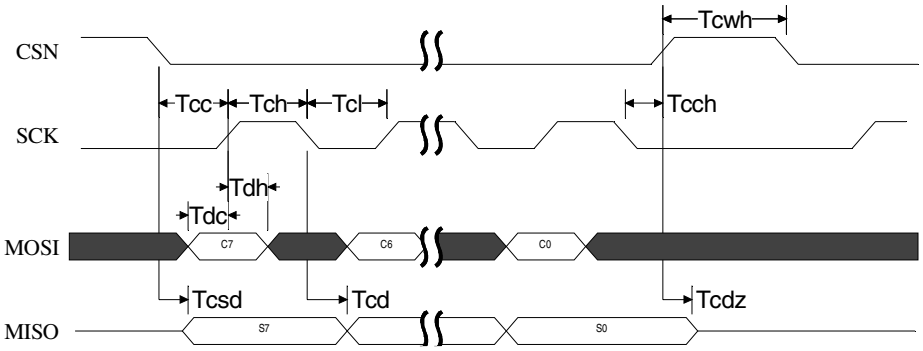


Figure 10 SPI NOP timing diagram.

PARAMETER	SYMBOL	MIN	MAX	UNITS
Data to SCK Setup	Tdc	2		ns
SCK to Data Hold	Tdh	2		ns
CSN to Data Valid	Tcsd		38	ns
SCK to Data Valid	Tcd		55	ns
SCK Low Time	Tcl	40		ns
SCK High Time	Tch	40		ns
SCK Frequency	Fsck	0	8	MHz
SCK Rise and Fall	Tr,Tf		100	ns
CSN to SCK Setup	Tcc	2		ns
SCK to CSN Hold	Tcch	2		ns
CSN Inactive time	Tcwh	50		ns
CSN to Output High Z	Tcdz		38	ns

Table 9 SPI timing parameters (C_{Load} = 5pF).

PARAMETER	SYMBOL	MIN	MAX	UNITS
Data to SCK Setup	Tdc	2		ns
SCK to Data Hold	Tdh	2		ns
CSN to Data Valid	Tcsd		42	ns
SCK to Data Valid	Tcd		58	ns
SCK Low Time	Tcl	40		ns
SCK High Time	Tch	40		ns
SCK Frequency	Fsck	0	8	MHz
SCK Rise and Fall	Tr,Tf		100	ns
CSN to SCK Setup	Tcc	2		ns
SCK to CSN Hold	Tcch	2		ns
CSN Inactive time	Tcwh	50		ns
CSN to Output High Z	Tcdz		42	ns

Table 10 SPI timing parameters (C_{Load} = 10pF).

MEMORY MAP

All undefined bits in the table below are redundant. They will be read out as '0'.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
00	CONFIG				Configuration Register
	Reserved	7	0	R/W	Only '0' allowed
	MASK_RX_DR	6	0	R/W	Mask interrupt caused by RX_DR 1: Interrupt not reflected on the IRQ pin 0: Reflect RX_DR as active low interrupt on the IRQ pin
	MASK_TX_DS	5	0	R/W	Mask interrupt caused by TX_DS 1: Interrupt not reflected on the IRQ pin 0: Reflect TX_DS as active low interrupt on the IRQ pin
	MASK_MAX_RT	4	0	R/W	Mask interrupt caused by MAX_RT 1: Interrupt not reflected on the IRQ pin 0: Reflect MAX_RT as active low interrupt on the IRQ pin
	EN_CRC	3	1	R/W	Enable CRC. Forced high if one of the bits in the EN_AA is high
	CRCO	2	0	R/W	CRC encoding scheme '0' - 1 byte '1' – 2 bytes
	PWR_UP	1	0	R/W	1: POWER UP, 0:POWER DOWN
	PRIM_RX	0	0	R/W	1: PRX, 0: PTX
01	EN_AA Enhanced ShockBurst™				Enable 'Auto Acknowledgment' Function Disable this functionality to be compatible with nRF2401, see page 26
	Reserved	7:6	00	R/W	Only '00' allowed
	ENAA_P5	5	1	R/W	Enable auto ack. data pipe 5
	ENAA_P4	4	1	R/W	Enable auto ack. data pipe 4
	ENAA_P3	3	1	R/W	Enable auto ack. data pipe 3
	ENAA_P2	2	1	R/W	Enable auto ack. data pipe 2
	ENAA_P1	1	1	R/W	Enable auto ack. data pipe 1
	ENAA_P0	0	1	R/W	Enable auto ack. data pipe 0
02	EN_RXADDR				Enabled RX Addresses
	Reserved	7:6	00	R/W	Only '00' allowed
	ERX _P5	5	0	R/W	Enable data pipe 5.
	ERX _P4	4	0	R/W	Enable data pipe 4.
	ERX _P3	3	0	R/W	Enable data pipe 3.
	ERX _P2	2	0	R/W	Enable data pipe 2.
	ERX _P1	1	1	R/W	Enable data pipe 1.
	ERX _P0	0	1	R/W	Enable data pipe 0.
03	SETUP_AW				Setup of Address Widths (common for all data pipes)
	Reserved	7:2	000000	R/W	Only '000000' allowed
	AW	1:0	11	R/W	RX/TX Address field width '00' - Illegal '01' - 3 bytes '10' - 4 bytes '11' – 5 bytes LSByte will be used if address width below 5 bytes
04	SETUP_RETR				Setup of Automatic Retransmission
	ARD	7:4	0000	R/W	Auto Re-transmit Delay '0000' – Wait 250+86uS

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
					‘0001’ – Wait 500+86uS ‘0010’ – Wait 750+86uS …….. ‘1111’ – Wait 4000+86uS (Delay defined from end of transmission to start of next transmission)[14]
	ARC	3:0	0011	R/W	Auto Retransmit Count ‘0000’ –Re-Transmit disabled ‘0001’ – Up to 1 Re-Transmit on fail of AA …… ‘1111’ – Up to 15 Re-Transmit on fail of AA
05	RF_CH				RF Channel
	Reserved	7	0	R/W	Only '0' allowed
	RF_CH	6:0	0000010	R/W	Sets the frequency channel nRF24L01 operates on
06	RF_SETUP				RF Setup Register
	Reserved	7:5	000	R/W	Only '000' allowed
	PLL_LOCK	4	0	R/W	Force PLL lock signal. Only used in test
	RF_DR	3	1	R/W	Data Rate ‘0’ – 1 Mbps ‘1’ – 2 Mbps
	RF_PWR	2:1	11	R/W	Set RF output power in TX mode '00' – -18 dBm '01' – -12 dBm '10' – -6 dBm '11' – 0 dBm
	LNA_HCURR	0	1	R/W	Setup LNA gain
07	STATUS				Status Register (In parallel to the SPI instruction word applied on the MOSI pin, the STATUS register is shifted serially out on the MISO pin)
	Reserved	7	0	R/W	Only '0' allowed
	RX_DR	6	0	R/W	Data Ready RX FIFO interrupt. Set high when new data arrives RX FIFO[15]. Write 1 to clear bit.
	TX_DS	5	0	R/W	Data Sent TX FIFO interrupt. Set high when packet sent on TX. If AUTO_ACK is activated, this bit will be set high only when ACK is received. Write 1 to clear bit.
	MAX_RT	4	0	R/W	Maximum number of TX retries interrupt Write 1 to clear bit. If MAX_RT is set it must be cleared to enable further communication.
	RX_P_NO	3:1	111	R	Data pipe number for the payload

[14] Accurate formula for delay from start of transmission, to start of re-transmission:
TRD (us) = 250us * (ARD+1) + 4us *(AW + PW + CRCW) +138,5us.
TRD= total retransmit delay, AW=Address Width (#bytes), PW=Payload Width(#bytes)
, CRCW= CRC Width (#bytes)

[15] The Data Ready interrupt is set by a new packet arrival event. The procedure for handling this interrupt should be: 1) read payload via SPI, 2) clear RX_DR interrupt, 3) read FIFO_STATUS to check if there are more payloads available in RX FIFO, 4) if there are more data in RX FIFO, repeat from 1).

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
					available for reading from RX_FIFO 000-101: Data Pipe Number 110: Not Used 111: RX FIFO Empty
	TX_FULL	0	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
08	OBSERVE_TX				Transmit observe register
	PLOS_CNT	7:4	0	R	Count lost packets. The counter is overflow protected to 15, and discontinue at max until reset. The counter is reset by writing to RF_CH. See page 14 and 17.
	ARC_CNT	3:0	0	R	Count resent packets. The counter is reset when transmission of a new packet starts. See page 14.
09	CD				
	Reserved	7:1	000000	R	
	CD	0	0	R	Carrier Detect. See page 17.
0A	RX_ADDR_P0	39:0	0xE7E7E7E7E7	R/W	Receive address data pipe 0. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0B	RX_ADDR_P1	39:0	0xC2C2C2C2C2	R/W	Receive address data pipe 1. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0C	RX_ADDR_P2	7:0	0xC3	R/W	Receive address data pipe 2. Only LSB. MSBytes will be equal to RX_ADDR_P1[39:8]
0D	RX_ADDR_P3	7:0	0xC4	R/W	Receive address data pipe 3. Only LSB. MSBytes will be equal to RX_ADDR_P1[39:8]
0E	RX_ADDR_P4	7:0	0xC5	R/W	Receive address data pipe 4. Only LSB. MSBytes will be equal to RX_ADDR_P1[39:8]
0F	RX_ADDR_P5	7:0	0xC6	R/W	Receive address data pipe 5. Only LSB. MSBytes will be equal to RX_ADDR_P1[39:8]
10	TX_ADDR	39:0	0xE7E7E7E7E7	R/W	Transmit address. Used for a PTX device only. (LSByte is written first) Set RX_ADDR_P0 equal to this address to handle automatic acknowledge if this is a PTX device with Enhanced ShockBurst™ enabled. See page 14.
11	RX_PW_P0				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P0	5:0	0	R/W	Number of bytes in RX payload in data pipe 0 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
12	RX_PW_P1				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P1	5:0	0	R/W	Number of bytes in RX payload in data pipe 1 (1 to 32 bytes). 0 Pipe not used

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
					1 = 1 byte … 32 = 32 bytes
13	RX_PW_P2				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P2	5:0	0	R/W	Number of bytes in RX payload in data pipe 2 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
14	RX_PW_P3				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P3	5:0	0	R/W	Number of bytes in RX payload in data pipe 3 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
15	RX_PW_P4				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P4	5:0	0	R/W	Number of bytes in RX payload in data pipe 4 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
16	RX_PW_P5				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P5	5:0	0	R/W	Number of bytes in RX payload in data pipe 5 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
17	FIFO_STATUS				FIFO Status Register
	Reserved	7	0	R/W	Only '0' allowed
	TX_REUSE	6	0	R	Reuse last sent data packet if set high. The packet will be repeatedly resent as long as CE is high. TX_REUSE is set by the SPI instruction REUSE_TX_PL, and is reset by the SPI instructions W_TX_PAYLOAD or FLUSH TX
	TX_FULL	5	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
	TX_EMPTY	4	1	R	TX FIFO empty flag. 1: TX FIFO empty. 0: Data in TX FIFO.
	Reserved	3:2	00	R/W	Only '00' allowed
	RX_FULL	1	0	R	RX FIFO full flag. 1: RX FIFO full. 0: Available locations in RX FIFO.
	RX_EMPTY	0	1	R	RX FIFO empty flag. 1: RX FIFO empty. 0: Data in RX FIFO.
N/A	TX_PLD	255:0	X	W	Written by separate SPI command TX data payload register 1 - 32 bytes. This register is implemented as a FIFO with 3 levels. Used in TX mode only
N/A	RX_PLD	255:0	X	R	Written by separate SPI command RX data payload register. 1 - 32 bytes. This register is implemented as a FIFO

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
					with 3 levels. All receive channels share the same FIFO

Table 11 Memory map of nRF24L01

Configuration for compatibility with nRF24XX

How to setup nRF24L01 to receive from an nRF2401/nRF2402/nRF24E1/nRF24E2:

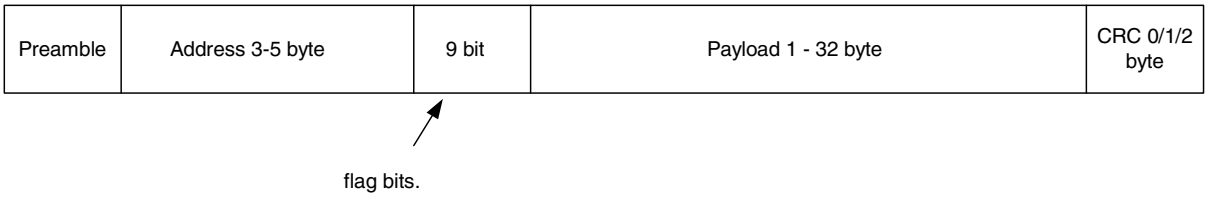
- Use same CRC configuration as the nRF2401/nRF2402/nRF24E1/nRF24E2 uses
- Set the PRIM_RX bit to 1
- Disable auto acknowledgement on the data pipe that will be addressed
- Use the same address width as the PTX device
- Use the same frequency channel as the PTX device
- Select data rate 1Mbit/s on both nRF24L01 and nRF2401/nRF2402/nRF24E1/nRF24E2
- Set correct payload width on the data pipe that will be addressed
- Set PWR_UP and CE high

How to setup nRF24L01 to transmit to an nRF2401/nRF24E1:

- Use same CRC configuration as the nRF2401/nRF2402/nRF24E1/nRF24E2 uses
- Set the PRIM_RX bit to 0
- Set the Auto Retransmit Count to 0 to disable the auto retransmit functionality
- Use the same address width as the nRF2401/nRF2402/nRF24E1/nRF24E2 uses
- Use the same frequency channel as the nRF2401/nRF2402/nRF24E1/nRF24E2 uses
- Select data rate 1Mbit/s on both nRF24L01 and nRF2401/nRF2402/nRF24E1/nRF24E2
- Set PWR_UP high
- Clock in a payload that has the same length as the nRF2401/nRF2402/nRF24E1/nRF24E2 is configured to receive
- Pulse CE to send the packet

PACKET DESCRIPTION

An Enhanced ShockBurst™ packet with payload (1-32 bytes).



A ShockBurst™ packet compatible to nRF2401/nRF2402/nRF24E1/nRF24E2 devices.

Preamble	• Preamble is used to detect 0 and 1 levels. It is stripped off (RX) and added (TX) by nRF24L01.
Address	• The address field contains the receiver address. • The address can be 3, 4 or 5 bytes wide • The address fields can be individually configured for all RX channels and the TX channel • Address is automatically removed from received packets.[16]
Flags	• PID: Packet Identification. 2 bits that is incremented for each new payload • 7 bits reserved for packet compatibility with future products • Not used when compatible to nRF2401/nRF24E1
Payload	• 1 - 32 bytes wide.
CRC	• The CRC is optional. • 0-2 bytes wide CRC • The polynomial for 8 bits CRC check is $X^8 + X^2 + X + 1$ • The polynomial for 16 bits CRC check is $X^{16} + X^{12} + X^5 + 1$.

Table 12 Data packet description

[16] Suggested use of addresses. In general more bits in the address gives less false detection, which in the end may give lower data Packet-Error-Rate (PER).

A. The address made by (5, 4, or 3) equal bytes are not recommended because it in general will make the packet-error-rate increase.
B. Addresses where the level shift only one time (i.e. 000FFFFFFF) could often be detected in noise that may give a false detection, which again may give raised packet-error-rate.
C. Addresses as a continuation of the preamble (hi-low toggling) will raise the Packet-Error-Rate (PER).

IMPORTANT TIMING DATA

The following timing applies for operation of nRF24L01.

nRF24L01 Timing Information

nRF24L01 timing	Max.	Min.	Name
Power Down ➔ Standby mode	1.5ms		Tpd2stby
Standby modes ➔ TX/RX mode	130µs		Tstby2a
Minimum CE high		10µs	Thce
Delay from CE pos. edge to CSN low		4µs	Tpece2csn

Table 13 Operational timing of nRF24L01

When the nRF24L01 is in power down it must always settle in Standby for 1.5ms before it can enter one of the TX or RX modes. Note that the configuration word will be lost if VDD is turned off and that the device then must be configured before going to one of the TX or RX mode.

Enhanced ShockBurst™ timing

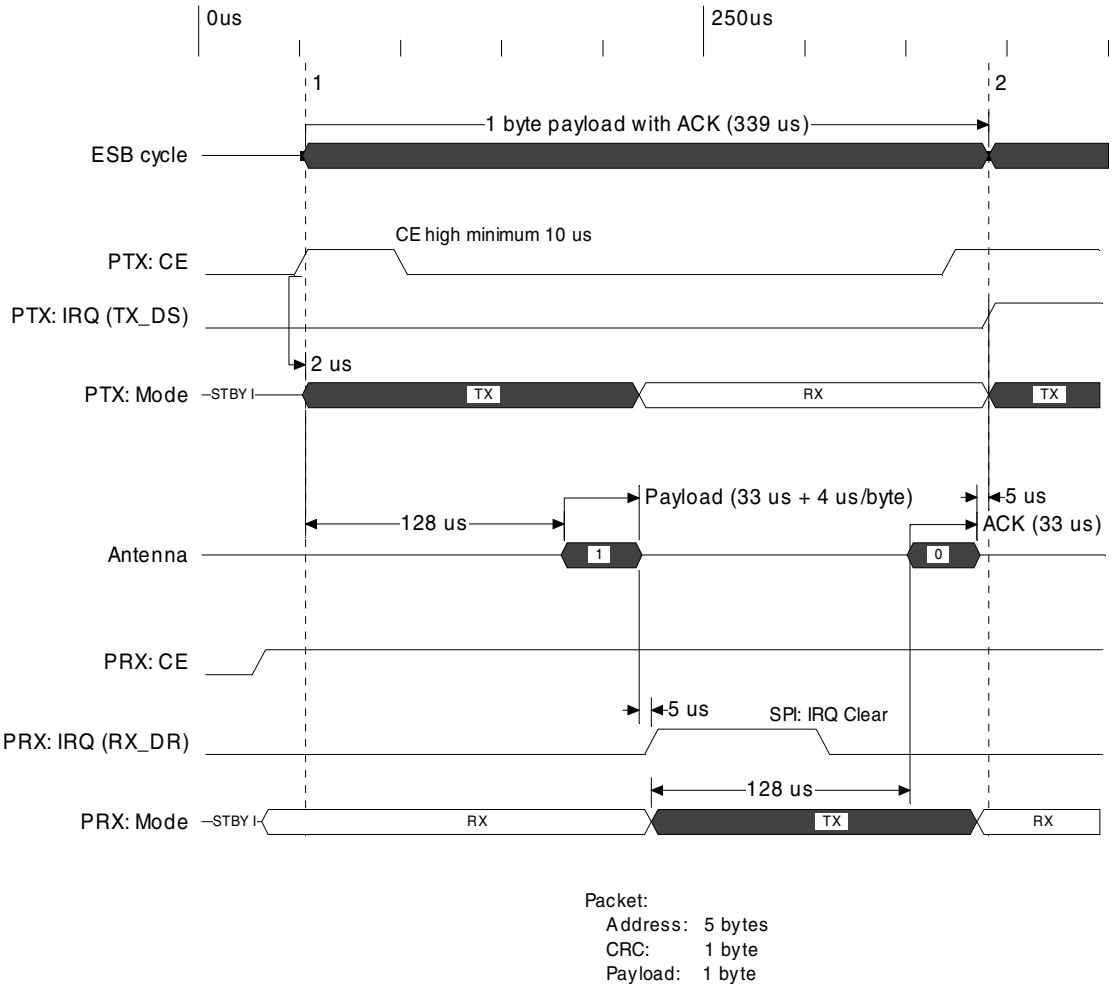


Figure 11 Timing of Enhanced ShockBurst™ for one packet upload (2Mbps).

In Figure 11 the sending of one packet and the acknowledgement of this packet is shown. The loading of payload to the PTX device is not shown in the figure. The PRX device is turned into RX mode (CE=1), and the PTX device is set into TX mode (CE=1 for minimum 10 µs). After 130 µs the transmission starts and is finished after another 37 µs (1 byte payload). The transmission ends, and the PTX device is automatically turned around to RX mode to wait for the acknowledgement from the PRX device. After the PTX device has received the acknowledgement it gives an interrupt to the MCU (IRQ (TX_DS) =>TX-data sent). After the PRX device has received the packet it gives an interrupt to the MCU (IRQ (RX_DR) =>RX-data ready).

PERIPHERAL RF INFORMATION

Antenna output

The ANT1 & ANT2 output pins provide a balanced RF output to the antenna. The pins must have a DC path to VDD, either via a RF choke or via the center point in a dipole antenna. A load of 15Ω+j88Ω (simulated values) is recommended for maximum output power (0dBm). Lower load impedance (for instance 50 Ω) can be obtained by fitting a simple matching network between the load and ANT1 and ANT2.

Output Power adjustment

SPI RF-SETUP (RF_PWR)	RF output power	DC current consumption
11	0 dBm	11.3 mA
10	-6 dBm	9.0 mA
01	-12 dBm	7.5 mA
00	-18 dBm	7.0 mA

Conditions: VDD = 3.0V, VSS = 0V, T_A = 27ºC, Load impedance = 15Ω+j88Ω.

Table 14 RF output power setting for the nRF24L01.

Crystal Specification

Frequency accuracy includes initial accuracy (tolerance) and stability over temperature and aging.

Frequency	C_L	ESR max	C_{0max}	Frequency accuracy
16MHz	8 – 16 pF	100 Ω	7.0pF	±60ppm

Table 15 Crystal specification of the nRF24L01

To achieve a crystal oscillator solution with low power consumption and fast start-up time, it is recommended to specify the crystal with a low value of crystal load capacitance. Specifying a lower value of crystal parallel equivalent capacitance, C_0 will also work, but this can increase the price of the crystal itself. Typically C_0=1.5pF at a crystal specified for C_{0max}=7.0pF.

The crystal load capacitance, C_L, is given by:

$C_L = \frac{C_1' \cdot C_2'}{C_1' + C_2'}$, where $C_1' = C_1 + C_{PCB1} + C_{I1}$ and $C_2' = C_2 + C_{PCB2} + C_{I2}$

C_1 and C_2 are SMD capacitors as shown in the application schematics. C_{PCB1} and C_{PCB2} are the layout parasitic on the circuit board. C_{I1} and C_{I2} are the capacitance seen into the XC1 and XC2 pin respectively; the value is typical 1pF.

nRF24L01 sharing crystal with a micro controller.

When using a micro controller to drive the crystal reference input XC1 of the nRF24L01 transceiver some rules must be followed.

Crystal parameters:

When the micro controller drives the nRF24L01 clock input, the requirement of load capacitance C_L is set by the micro controller only. The frequency accuracy of ±60 ppm is still required to get a functional radio link. The nRF24L01 will load the crystal by 0.5pF at XC1 in addition to the PBC routing.

Input crystal amplitude & Current consumption

The input signal should not have amplitudes exceeding any rail voltage, but any DC-voltage within this is OK. Exceeding rail voltage will excite the ESD structure and the radio performance is degraded below specification. If testing the nRF24L01 with a RF source with no DC offset as the reference source, the input signal will go below the ground level, which is not acceptable.

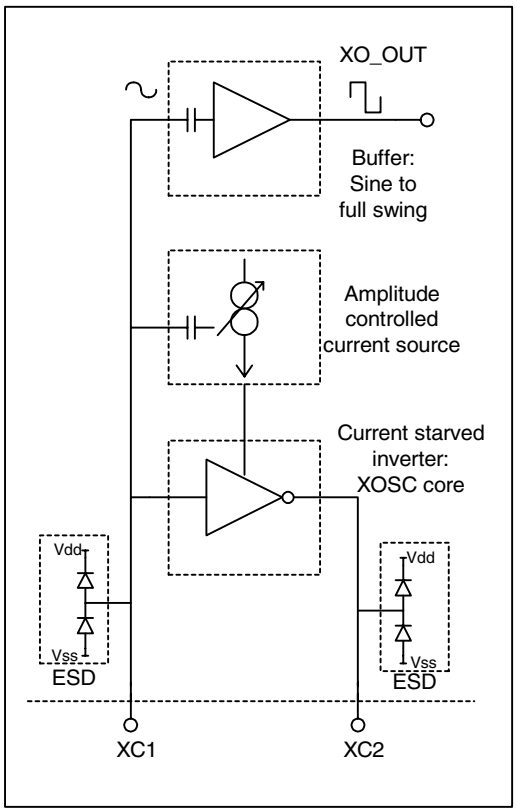


Figure 12 Principle of crystal oscillator

The nRF24L01 crystal oscillator is amplitude regulated. To achieve low current consumption and also good signal-to-noise ratio when using an external clock, it is recommended to use an input signal larger than 0.4 V-peak. When clocked externally, XC2 is not used and can be left as an open pin.

PCB layout and de-coupling guidelines

A well-designed PCB is necessary to achieve good RF performance. Keep in mind that a poor layout may lead to loss of performance, or even functionality, if due care is not taken. A fully qualified RF-layout for the nRF24L01 and its surrounding components, including matching networks, can be downloaded from **www.nordicsemi.no**.

A PCB with a minimum of two layers including a ground plane is recommended for optimum performance. The nRF24L01 DC supply voltage should be de-coupled as close as possible to the VDD pins with high performance RF capacitors, see Table 16. It is preferable to mount a large surface mount capacitor (e.g. 4.7µF tantalum) in parallel with the smaller value capacitors. The nRF24L01 supply voltage should be filtered and routed separately from the supply voltages of any digital circuitry.

Long power supply lines on the PCB should be avoided. All device grounds, VDD connections and VDD bypass capacitors must be connected as close as possible to the nRF24L01 IC. For a PCB with a topside RF ground plane, the VSS pins should be connected directly to the ground plane. For a PCB with a bottom ground plane, the best technique is to have via holes as close as possible to the VSS pads. At least one via hole should be used for each VSS pin.

Full swing digital data or control signals should not be routed close to the crystal or the power supply lines.

APPLICATION EXAMPLE

nRF24L01 with single ended matching network crystal, bias resistor, and decoupling capacitors.

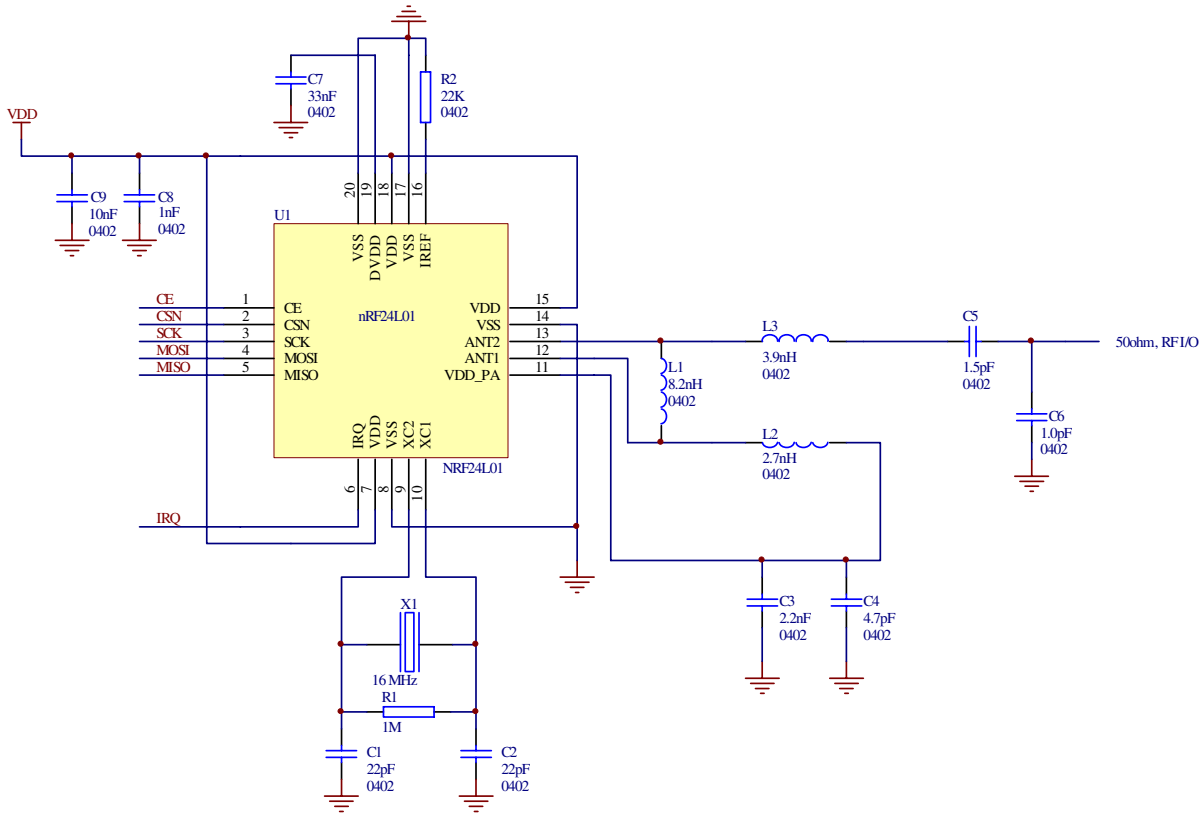


Figure 13 nRF24L01 schematic for RF layouts with single ended 50Ω RF output.

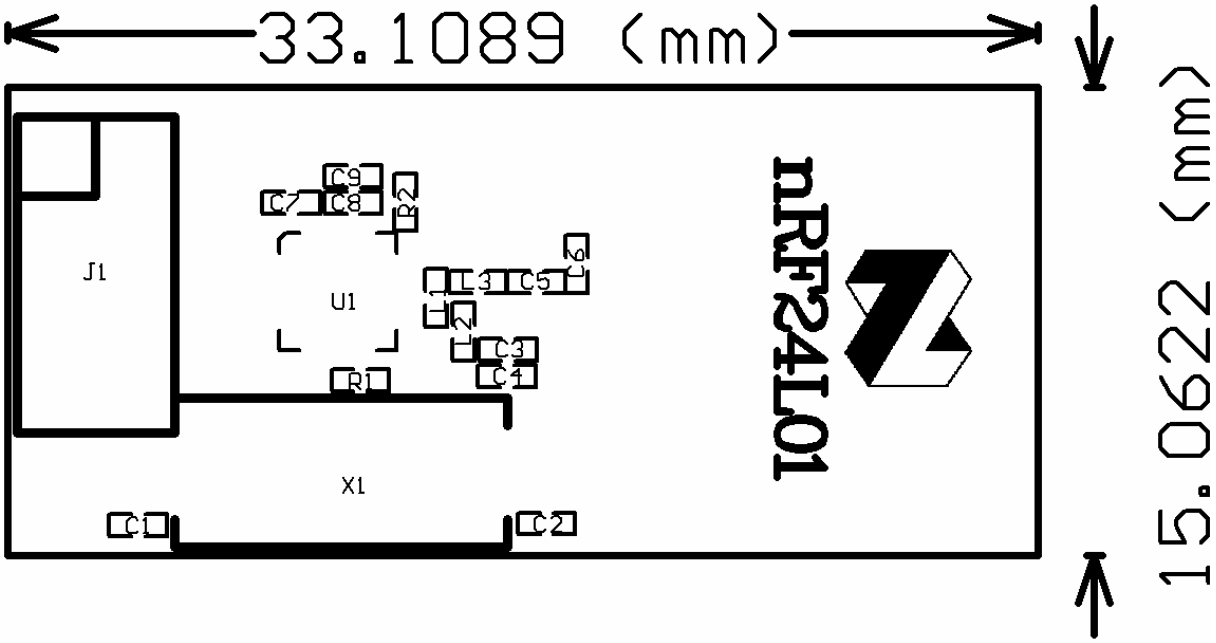
Part	Designator	Footprint	Description
22pF[17]	C1	0402	NPO, +/- 2%, 50V
22pF[17]	C2	0402	NPO, +/- 2%, 50V
2.2nF	C3	0402	X7R, +/- 10%, 50V
4.7pF	C4	0402	NPO, +/- 0.25 pF, 50V
1.5pF	C5	0402	NPO, +/- 0.1 pF, 50V
1,0pF	C6	0402	NPO, +/- 0.1 pF, 50V
33nF	C7	0402	X7R, +/- 10%, 50V
1nF	C8	0402	X7R, +/- 10%, 50V
10nF	C9	0402	X7R, +/- 10%, 50V
8,2nH	L1	0402	chip inductor +/- 5%
2.7nH	L2	0402	chip inductor +/- 5%
3,9nH	L3	0402	chip inductor +/- 5%
1M	R1	0402	+/-10%
22K	R2	0402	+/- 1 %
nRF24L01	U1	QFN20 4x4	
16MHz	X1		+/-60ppm, C_L=12pF[17]

Table 16 Recommended components (BOM) in nRF24L01 with antenna matching network

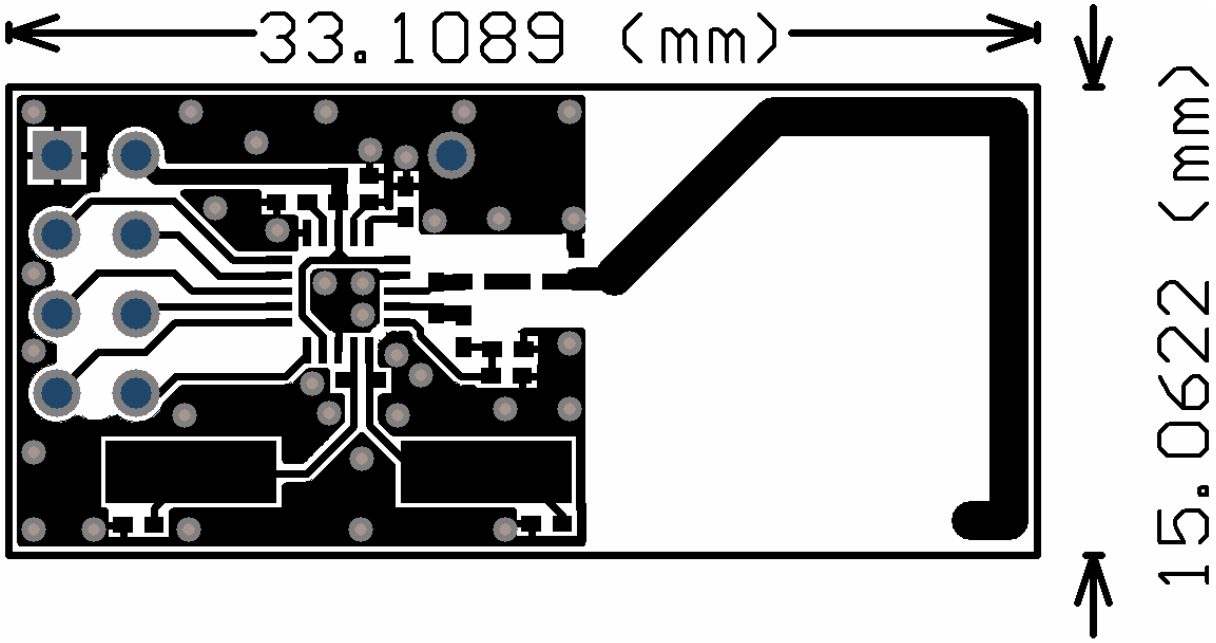
[17] C1 and C2 must have values that match the crystals load capacitance, C_L.

PCB layout examples

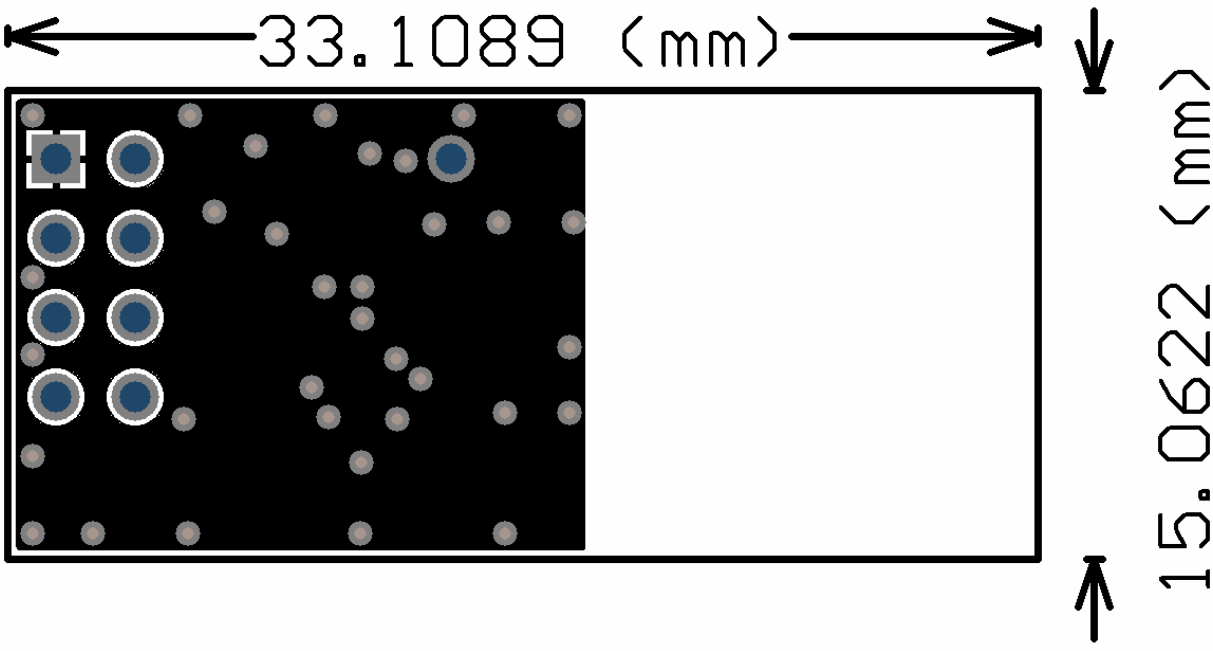
Figure 14 shows a PCB layout example for the application schematic in Figure 13. A double-sided FR-4 board of 1.6mm thickness is used. This PCB has a ground plane on the bottom layer. Additionally, there are ground areas on the component side of the board to ensure sufficient grounding of critical components. A large number of via holes connect the top layer ground areas to the bottom layer ground plane.



Top overlay



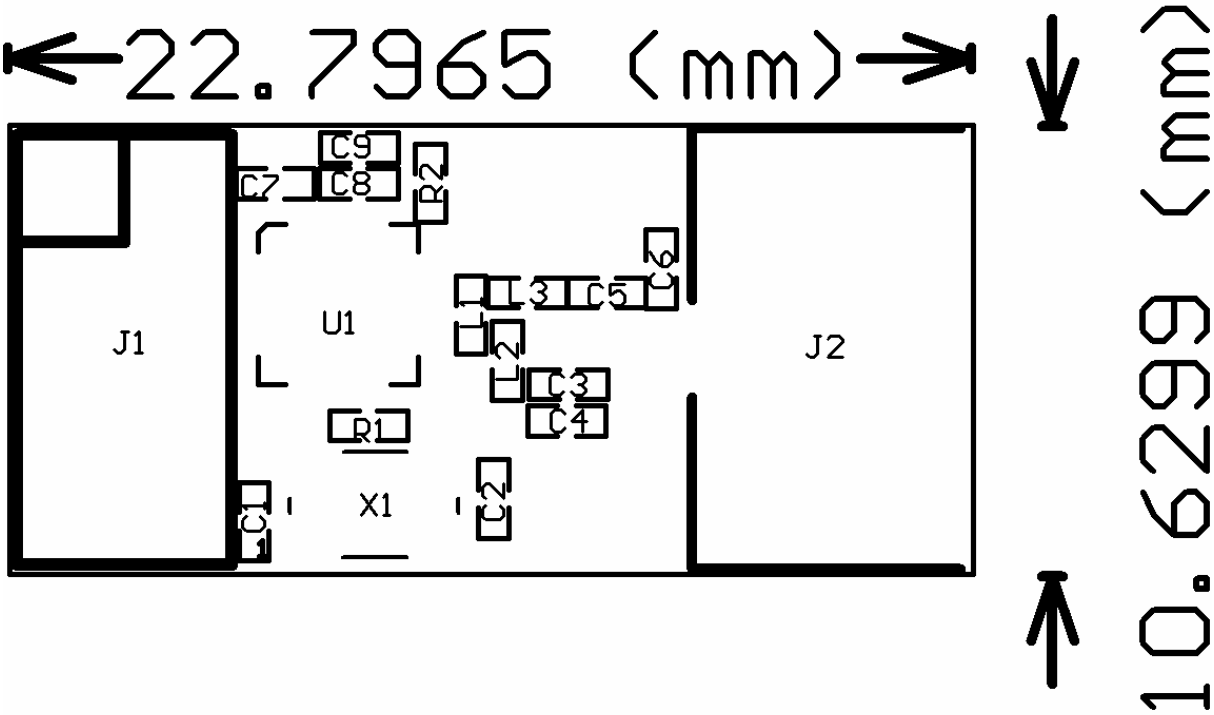
Top layer



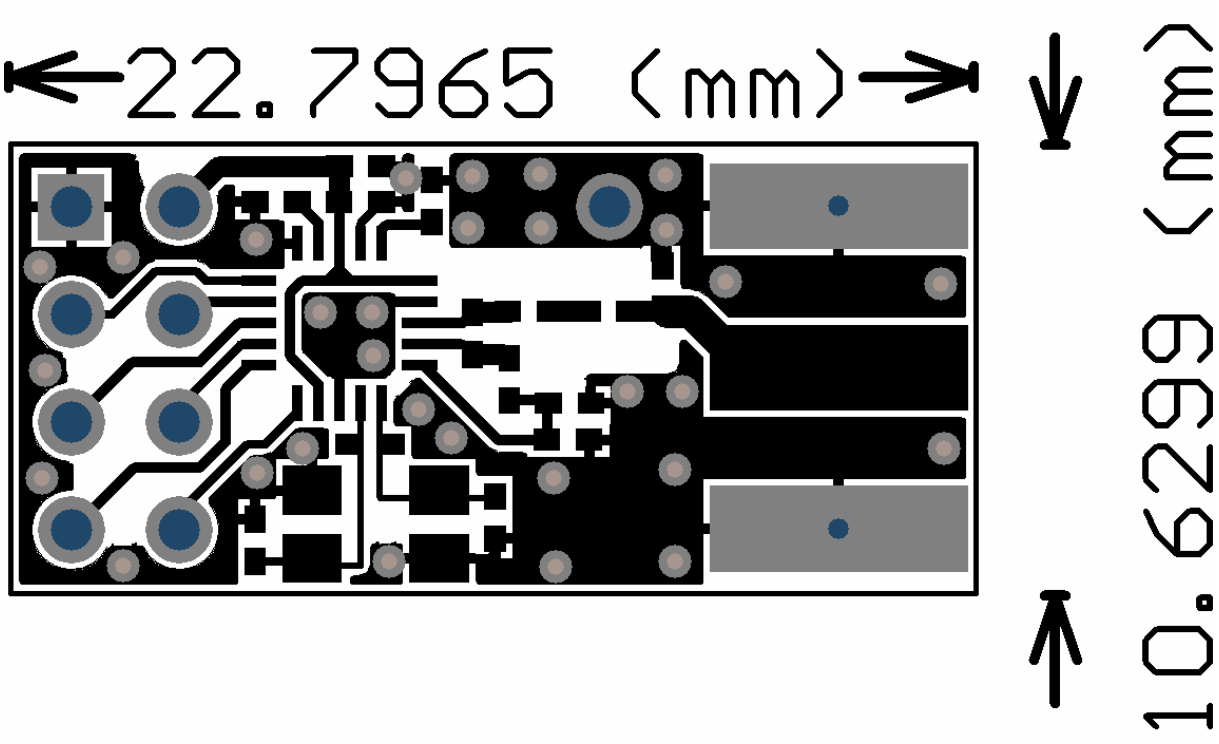
Bottom layer

Figure 14 nRF24L01 RF layout with single ended connection to PCB antenna and 0603 size passive components

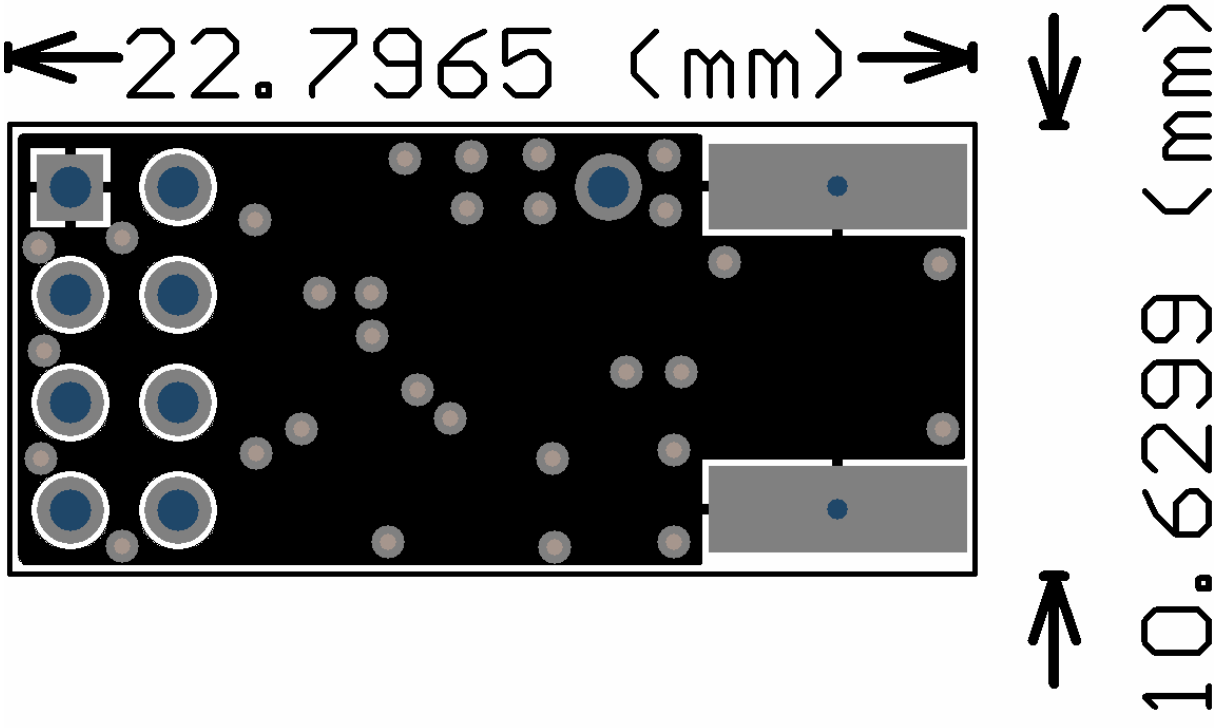
The nest figure (Figure 15) is for the SMA output to have a board for direct measurements at a 50Ω SMA connector.



Top Overlay



Top Layer



Bottom Layer

Figure 15 Module with OFM crystal and SMA connector

DEFINITIONS

Data sheet status	
Objective product specification	This data sheet contains target specifications for product development.
Preliminary product specification	This data sheet contains preliminary data; supplementary data may be published from Nordic Semiconductor ASA later.
Product specification	This data sheet contains final product specifications. Nordic Semiconductor ASA reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
Limiting values	
Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Specifications sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.	
Application information	
Where application information is given, it is advisory and does not form part of the specification.	

Table 17. Definitions

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

LIFE SUPPORT APPLICATIONS

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Preliminary Product Specification: Revision Date: 08.03.2006.

Data sheet order code: 080306-nRF24L01

YOUR NOTES

Nordic Semiconductor ASA – World Wide Distributors

For Your nearest dealer, please see http://www.nordicsemi.no

Main Office:
Vestre Rosten 81, N-7075 Tiller, Norway
Phone: +47 72 89 89 00, Fax: +47 72 89 89 89

Visit the Nordic Semiconductor ASA website at http://www.nordicsemi.no