CommonSense

Sony’s Spresense Sensor Board

Datasheet

by SensiEDGE and Edge Impulse

# List of Content

| 1 Overview | 4 |
| --- | --- |
| 1.1 General information | 4 |
| 1.2 CXD5602 pinout | 5 |
| 1.3 Sensor board pin functions | 7 |
| 2 Main Hardware Components | 10 |
| 2.1 CXD5602 microcontroller | 10 |
| 2.2 Sensors | 11 |
| 2.2.1 LSM6DS3: inertial module: 3D accelerometer and 3D gyroscope | 11 |
| 2.2.2 LIS2MDL: 3-Axis Magnetometer | 11 |
| 2.2.3 HTS221TR: humidity and temperature | 11 |
| 2.2.4 LPS22HH: pressure sensor | 11 |
| 2.2.5 VL53L1X: Proximitry sensor | 12 |
| 2.2.6 SGP-40: Air quality sensor | 12 |
| 2.2.7 APDS-9250: Digital RGB, IR and Ambient Light Sensor | 12 |
| 2.2.8 PCA9538: Port expander | 12 |
| 2.2.9 SDCS/16: micro SD card | 12 |
| 2.3 User Interface | 13 |
| 2.3.1 TE044003-1: Magnetic Buzzer | 13 |
| 2.3.2 SS304BS: Button | 13 |
| 2.3.3 LTST-C195KGJRKT Dual color chip LED | 14 |
| 3 CXD5602 Microcontroller | 14 |
| 3.1 Introduction | 14 |
| 3.2 Features | 17 |
| 4 Sensors | 17 |
| 4.1 3D accelerometer and 3D gyroscope | 17 |
| 4.1.1 General Description | 17 |
| 4.1.2 Features | 18 |
| 4.1.3 Schematic connections | 19 |
| 4.2 3 Axis magnetometer | 19 |
| 4.2.1 General description | 19 |
| 4.2.2 Features | 20 |
| 4.2.3 Schematic connection | 21 |
| 4.3 Humidity and temperature sensor | 21 |
| 4.3.1 General description | 21 |
| 4.3.2 Features | 22 |
| 4.3.3 Schematic connection | 23 |
| 4.4 Pressure sensor | 23 |
| 4.4.1 General description | 23 |
| 4.4.2 Features | 24 |
| 4.4.3 Schematic connection | 25 |
| 4.5 Proximity sensor | 25 |
| 4.5.1 General description | 25 |
| 4.5.2 Features | 26 |
| 4.5.3 Schematic connection | 27 |
| 4.6 Air quality sensor | 27 |
| 4.6.1 General description | 28 |
| 4.6.2 Schematic connection | 29 |
| 4.7 Digital RGB, IR and Ambient Light Sensor | 29 |
| 4.7.1 General description | 29 |
| 4.7.2 Features | 30 |
| 4.7.3 Schematic connection | 31 |
| 4.8 Port expander | 31 |
| 4.8.1 General description | 31 |
| 4.8.2 Features | 33 |
| 4.8.3 Schematic connection | 34 |
| 4.9 Micro SD card | 34 |
| 4.9.1 General description | 34 |
| 4.9.2 Schematic connection | 35 |
| 5 Examples of use | 36 |
| 5.1 LEDs | 36 |
| 5.2 Button | 36 |
| 5.3 Speaker | 36 |
| 5.4 HTS221TR: humidity and temperature | 37 |
| 5.5 LIS2MDL: 3-Axis Magnetomet | 38 |
| 5.6 LPS22HH: pressure sensor | 40 |
| 5.7 LSM6DS3: inertial module: 3D accelerometer and 3D gyroscope | 41 |
| 5.8 VL53L1X: Proximitry sensor | 43 |
| 5.9 APDS-9250: Digital RGB, IR and Ambient Light Sensor | 45 |
| 5.10 SGP-40: Air quality sensor | 45 |
| 5.11 Low power test | 46 |
| 5.12 SD card | 46 |
| 5.13 FatFS SD card | 47 |
| Best regards | 48 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# **1 Overview**

## 1.1 General information

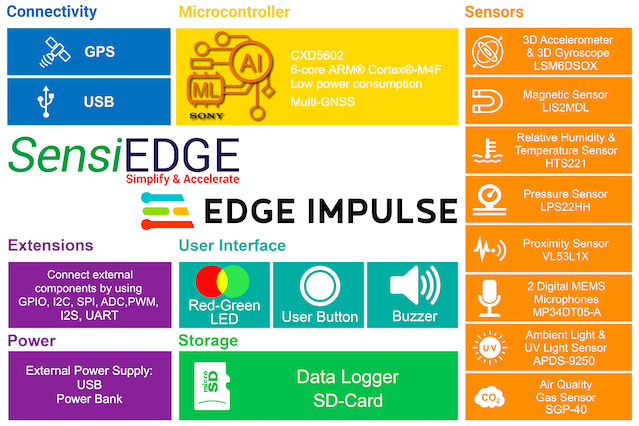


Figure 1. SensiEDGE Block Diagram

System consist (Fig. 1) from the main board and sensor board

The main board is Spresense CXD5602PWBMAIN1C

The sensor board consist from sensors and GPIO expander. The button and LEDs are connected to GPIO expander.

## **1.2 CXD5602 pinout**

CXD6502 pinout and pin functions are in (Fig. 2) and (Tabl. 1).

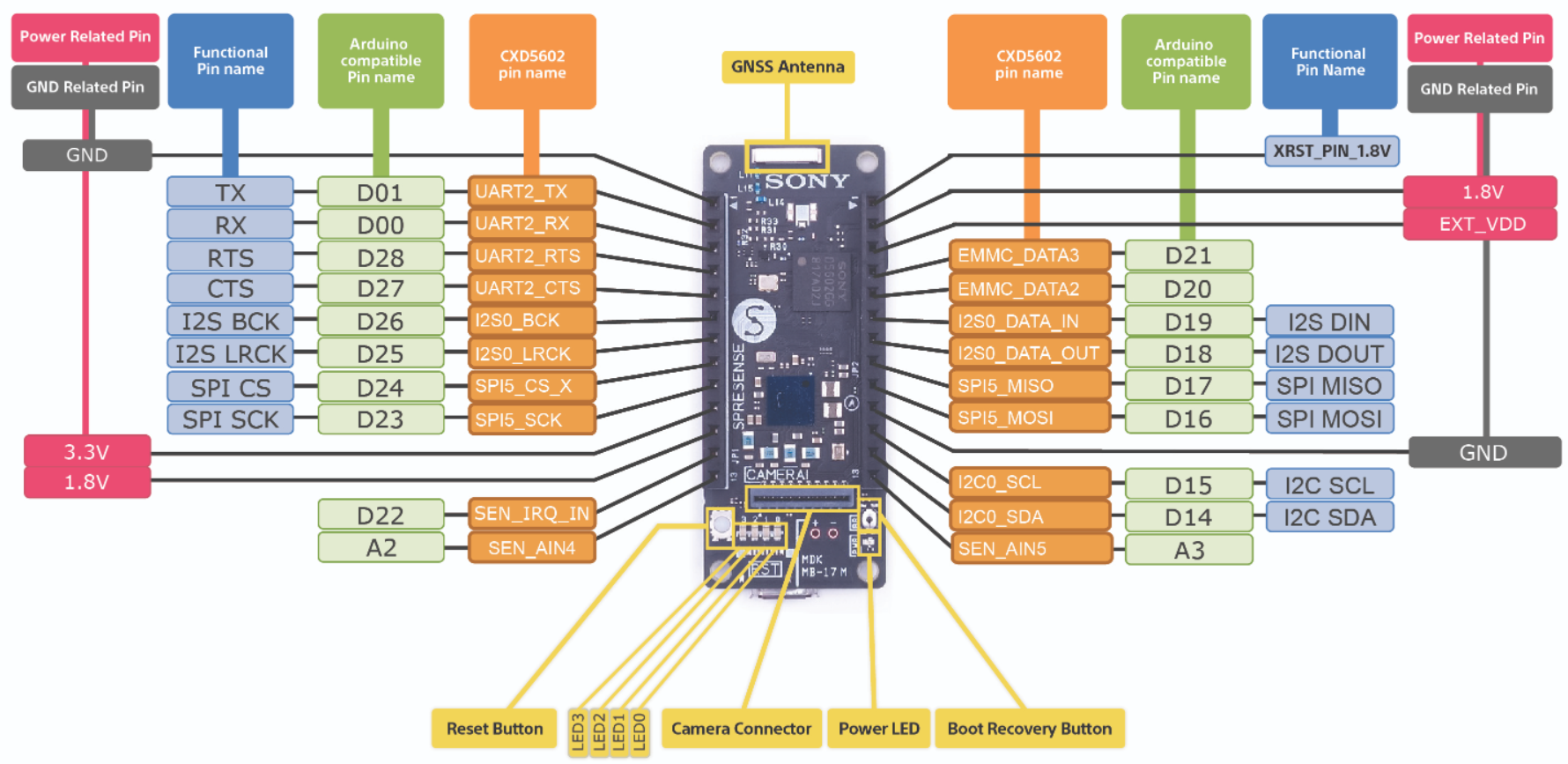


Figure 2. CXD6502 board general view and pinout

Table 1. Pin functions

J1:

| Pin\Mode | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| 1 | GND | | | |
| 2 | GPIO | UART2\_TXD | - | GPIO |
| 3 | GPIO | UART2\_RXD | - | GPIO |
| 4 | GPIO | UART2\_RTS | - | GPIO |
| 5 | GPIO | UART2\_CTS | - | GPIO |
| 6 | GPIO | I2S\_BCK | - | GPIO |
| 7 | GPIO | I2S\_LRCK | - | GPIO |
| 8 | GPIO | EMMC\_CMD | SPI5\_CS\_X | GPIO |
| 9 | GPIO | EMMC\_CLK | SPI5\_SCK | GPIO |
| 10 | 3,3V | | | |
| 11 | 1,8V | | | |
| 12 | GPIO | SEN\_IRQ\_IN | - | - |
| 13 | SEN\_AIN4 | | | |

J2:

| Pin\Mode | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| 1 | RST | | | |
| 2 | 1,8V | | | |
| 3 | EXT\_VDD | | | |
| 4 | GPIO | EMMC\_DATA3 | - | GPIO |
| 5 | GPIO | EMMC\_DATA2 | - | GPIO |
| 6 | GPIO | I2S0\_DATA\_IN | - | GPIO |
| 7 | GPIO | I2S0\_DATA\_OUT | - | GPIO |
| 8 | GPIO | EMMC\_DATA1 | SPI5\_MISO | GPIO |
| 9 | GPIO | EMMC\_DATA0 | SPI5\_MOSI | GPIO |
| 10 | GND | | | |
| 11 | GPIO | I2C0\_BCK | - | - |
| 12 | GPIO | I2C0\_BDT | - | - |
| 13 | SEN\_AIN5 | | | |

## **1.3 Sensor board pin functions**

Sensor board pinout and pin functions are in (Fig. 3) and (Tabl. 2).

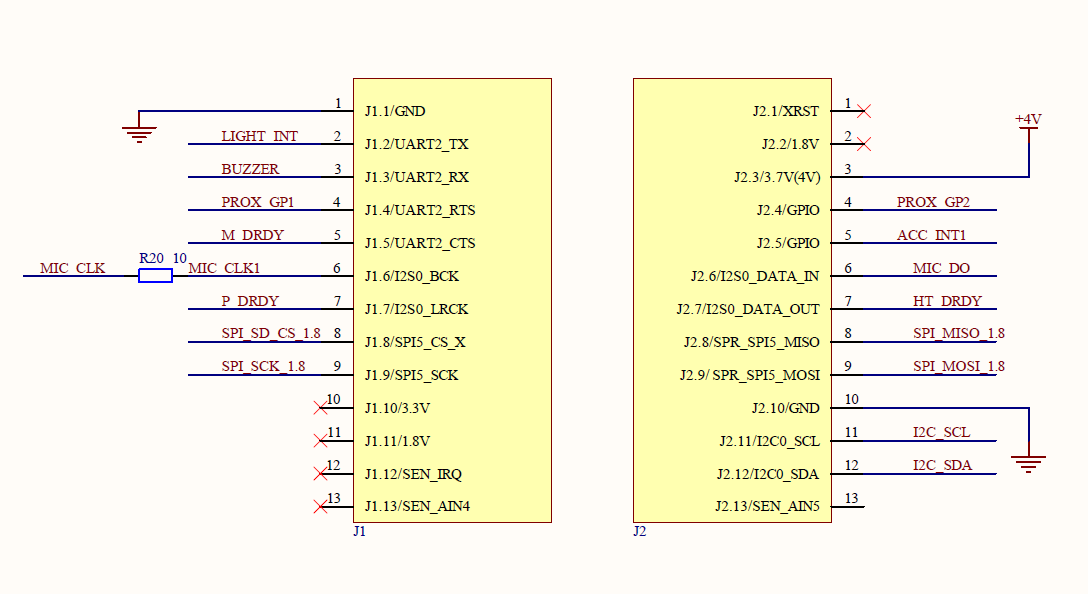


Figure 3. Sensors board pinout

Table 2. Sensors board pinout

J1:

| Pin | Mode | Note |
| --- | --- | --- |
| 1 | GND | Power |
| 2 | GPO | APDS-9250 INT |
| 3 | GPI\PWM | Buzzer control input |
| 4 | GPIO | VL53L1 GPIO1 |
| 5 | GPO | LIS2MDL INT |
| 6 | I2S\_BCK | MP34DT05 I2S CLK |
| 7 | GPO | LPS22HH INT\DRDY |
| 8 | SPI\_CS | SD card CS |
| 9 | SPI\_SCK | SD card SCK |
| 10 | x |  |
| 11 | x |  |
| 12 | x |  |
| 13 | x |  |

J2:

| Pin | Mode | Note |
| --- | --- | --- |
| 1 | x |  |
| 2 | x |  |
| 3 | +4V | Power |
| 4 | GPI | VL53L1 XSHUT |
| 5 | GPO | LSM6 INT1 |
| 6 | I2S\_DATA\_OUT | MP34DT05 data |
| 7 | GPO | HTS221 DRDY |
| 8 | SPI\_MISO | SD card MISO |
| 9 | SPI\_MOSI | SD card MOSI |
| 10 | GND | Power |
| 11 | I2C\_SCL | Sensors and extra GPIO I2C SCL |
| 12 | I2C\_SDA | Sensors and extra GPIO I2C SDA |
| 13 | x |  |

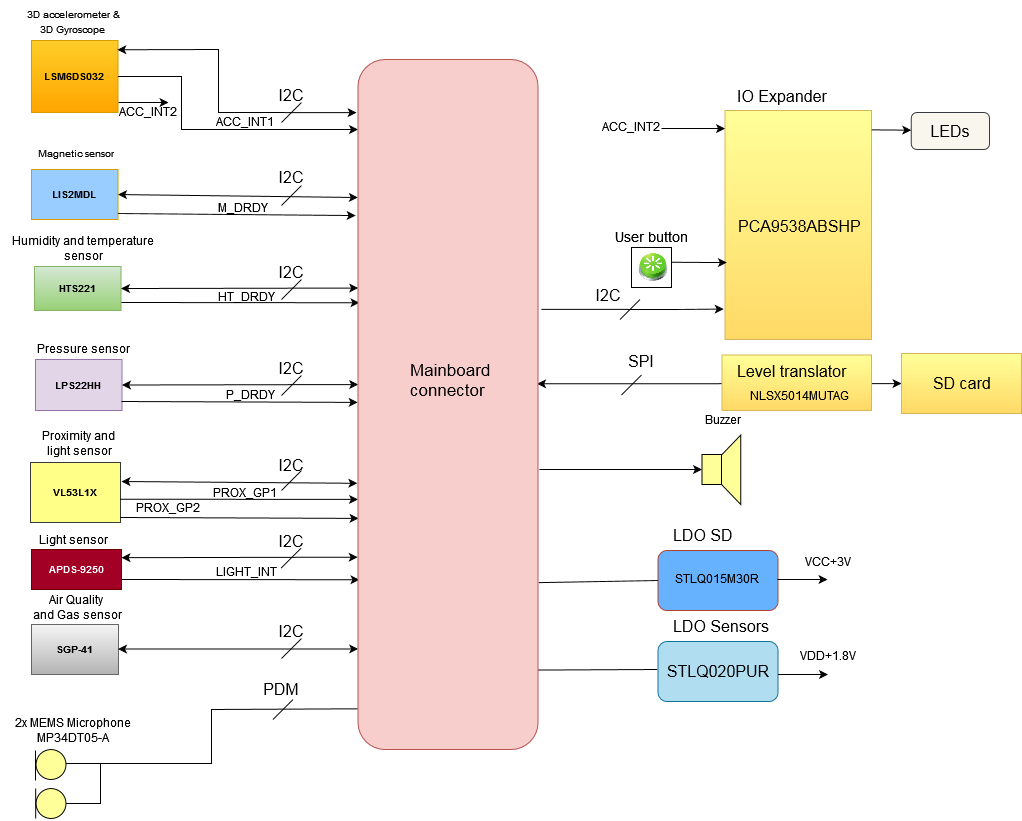
****

Figure 4. System board pinout

# 2 Main Hardware Components

The system consist from the main board based on CXD5602 microcontroller and sensor board with sensors and LEDs and button (Fig. 4). The LEDs, button works from port expander PCA9538. Pin INT2 of LSM6DSOX is also connected to port expander.

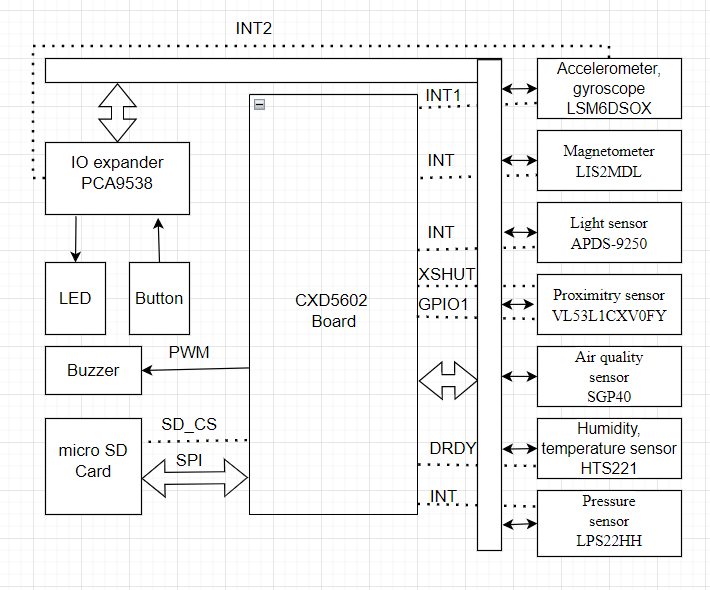


Figure 5. Systems block diagram

## 2.1 CXD5602 microcontroller

CXD5602GF/GG is a 32 bit RISC low-power microprocessor solution for wearable applications. It is based on the Arm® Cortex®-M4 processor with FPU 32 bit RISC and It integrates Arm® Cortex®-M0+ 32 bit RISC specifically for the system controller (power management, clock, reset) and I/O processor. It incorporates embedded 1.5 MByte of SRAM, 64 KByte of backup SRAM, and 256 KByte of I/O processor SRAM. The Arm® Cortex®-M4 processor with FPU and Arm® Cortex®-M0+ are power-gated by the Power Management Unit, respectively, that is, the CPUs are powered off by internal power switches. Processor SRAMs are able to retain data, and it’s possible to restart processors quickly. To provide optimized hardware performance for sensor fusion and audio processing services, the device integrates ultra-low power GNSS Domain, Audio Codec Domain and Sensor Domain. Integrated Audio Codec Domain supports digital noise cancelling and digital equalizer. Sensor Domain provides the specialized engine for sensor processing.

## 2.2 Sensors

The *Sensi*BLE module contains verity of sensors :

▪ ST’s 3D accelerometer and 3D gyroscope

▪ ST’s 3-Axis Magnetometer

▪ ST’s humidity and temperature

▪ ST’s pressure sensor

▪ ST’s Proximitry sensor

▪ Sensirion’s Air quality sensor

▪ Avago’s Digital RGB, IR and Ambient Light Sensor

▪ NXP’s Port expander

▪ Kingston’s micro SD card

### 2.2.1 LSM6DS3: inertial module: 3D accelerometer and 3D gyroscope

The LSM6DS3 is a system-in-package featuring a 3D digital accelerometer and a 3D digital gyroscope. Enabling always-on low-power features for an optimal motion experience.

### 2.2.2 LIS2MDL: 3-Axis Magnetometer

The LIS2MDL is an ultra low-power high-performance 3-Axis Magnetometer. This device offers the unique flexibility for designers to implement movement and position detection in space-constrained products such as personal navigation devices.

### 2.2.3 HTS221TR: humidity and temperature

The HTS221 is an ultra compact sensor for relative humidity and temperature. It includes a sensing element consists of a polymer dielectric planar capacitor structure and a mixed signal ASIC to provide the measurement information through digital serial interfaces.

### 2.2.4 LPS22HH: pressure sensor

The LPS22HH is an ultra compact absolute piezoresistive pressure sensor. It includes a monolithic sensing element capable to detect.

### 2.2.5 VL53L1X: Proximitry sensor

The VL53L1X is a state-of-the-art, Time-of-Flight (ToF), laser-ranging sensor, enhancing the ST FlightSense product family. It is the fastest miniature ToF sensor on the market with accurate ranging up to 4 m and fast ranging frequency up to 50 Hz

### 2.2.6 SGP-40: Air quality sensor

The SGP40 is a digital gas sensor designed for easy integration into air purifier s or demand controlled ventilation systems. Sensirion’s CMOSens ® technology offers a complete , easy to use sensor system on a single chip featuring a digital I 2 C interface and a temperature controlle d micro hotplate, providing a humidity compensated VOC based indoor air quality signal . The output signal can be directly processed by Sensirion’s powerful VOC Algorithm to translate the raw signal into a VOC Index as a robust measure for indoor air quality. The VOC Algorithm automatically adapts to the environment the sensor is exposed to.

### 2.2.7 APDS-9250: Digital RGB, IR and Ambient Light Sensor

The Avago APDS-9250 is a low-voltage digital RGB, IR and ambient light sensor device that converts light intensity to digital output signal. The color-sensing feature is useful in applications such as LED RGB backlight control, solid-state lighting, reflected LED color sampler and fluorescent light color temperature detection. With the IR sensing feature, the device can be used to read the IR content in certain lighting condition and detect the type of light source.

### 2.2.8 PCA9538: Port expander

The PCA9538 is an 8-bit I/O expander of general purpose parallel input and output (I/O) expansion for the two-line bidirectional I2C bus (or SMBus) protocol. This device can operate with a power supply range from 2.3 V to 5.5 V. This device supports both 100-kHz (Standard-mode) and 400-kHz (Fast-mode) clock frequencies. This device, along with other I/O expanders, provides a simple solution when additional I/Os are needed for switches, sensors, push-buttons, LEDs, fans, and so on.

### 2.2.9 SDCS/16: micro SD card

16 Gb micro SD card

## 2.3 User Interface

The *Sensi*EDGE module contains verity of user interfaces :

▪ Buzzer TE044003-1

▪ Button SS304BS

▪ RG-LED 5988510207F

### 2.3.1 TE044003-1: Magnetic Buzzer

Buzzers Transducer, Externally Driven Electromechanical 3 V 90mA 4kHz 70dB @ 3V, 10cm Surface Mount Solder Pads

### 2.3.2 SS304BS: Button

Surface mount type tact switch. Outline dimension 4.5mm x 2.3mm and 1.8 mm height. Horizontal operating direction Anti-fl ux penetration (1100J)

### 2.3.3 LTST-C195KGJRKT Dual color chip LED

Green, Red 565nm Green, 635nm Red LED Indication - Discrete 2V Green, 2V Red 2-SMD, No Lead

# 3 CXD5602 Microcontroller

## 3.1 Introduction

CXD5602GF/GG is a 32 bit RISC low power microprocessor solution for wearable applications. It is based on the Arm® Cortex®-M4 processor with FPU 32 bit RISC and It integrates Arm® Cortex®-M0+ 32 bit RISC specifically for the system controller (power management, clock, reset) and I/O processor. It incorporates embedded 1.5 MByte of SRAM, 64 KByte of backup SRAM, and 256 KByte of I/O processor SRAM. The Arm® Cortex®-M4 processor with FPU and Arm® Cortex®-M0+ are power-gated by the Power Management Unit, respectively, that is, the CPUs are powered off by internal power switches. Processor SRAMs are able to retain data, and it’s possible to restart processors quickly. To provide optimized hardware performance for sensor fusion and audio processing services, the device integrates ultra-low power GNSS Domain, Audio Codec Domain and Sensor Domain. Integrated Audio Codec Domain supports digital noise cancelling and digital equalizer. Sensor Domain provides the specialized engine for sensor processing. Eliminating the need for a discrete sensor hub, these features enable various sensor applications (activity recognition, voice recognition, etc.) low power audio applications such as music playback (MP3 decode, Bluetooth A2DP, etc.) and hands-free communication (Bluetooth HFP).

## 3.2 Features

The features of CXD5602GF/GG are:

* Application Processor

Arm® Cortex®-M4 processor with FPU 32 bit RISC

Operating frequency up to 156 MHz

* 1.5 MByte Application Memory
* Application Multi-layer Bus

32 bit Multi-layer bus architecture

Application Domain for Arm® Cortex®-M4 processor with FPU 32 bit RISC, Audio Codec,

Connectivity, Storage and Image Block

* Audio Codec

Digital Equalizer (DEQ)

Two I2S Interfaces supported

Unique Audio Data Format (Pulse Density Modulation) between CXD5602GF/GG and CXD5247GF

* 8 bit parallel Camera Interface supported
* 2D Graphics Acceleration

BitBLT, Rotate, Scaling, Blender

* Connectivity/Storage Interface

On-chip USB2.0 Device supported

eMMC 4.41 for eMMC Device

SD3.0 Host Controller interface

SPI and SDIO support for external Wi-Fi transceivers

UART support for external Bluetooth transceivers

Quad SPI-FLASH Interface

* Display Interface

SPI Interface up to 40.96 Mbps

8/16/24/32 bpp LCD or E-Ink recommend up to QVGA resolution

* System and I/O Processor (SYSCPU)

Arm® Cortex®-M0+ 32 bit RISC

Operating frequency up to 100 MHz at 1.0 V

256 KByte SRAM

128 KByte ROM for secure booting

* System and IOP Domain multi-layer bus

32 bit Multi-layer bus architecture

SYSIOP for Arm® Cortex®-M0+ 32 bit RISC, PMU, GNSS, Sensor engine, HostIF, Configurable IO

* Power management

I2C and GPIO interface connections to Power Management IC (assuming CXD5247GF)

power on reset

power gate control

* Clock and Reset management

X'tal, RTC, RCOSC, PLL

* 64 KByte Backup SRAM
* Timer

RTC

A general-purpose 32 bit timer each Processor Unit

* Host Interface

I2C, SPI or UART interface

1 KByte Host communication Memory

* Sensor engine

SPI and Two I2C Interfaces

40 KByte Sensory Data FIFO

Pre-processing unit for sensor fusion

Up to Four PWMs

* ADCs

Four channel 10 bit low power ADC

Two channels 10 bit high performance ADC

* Multi-GNSS Controller

Arm® Cortex®-M4 processor with FPU 32 bit RISC

Operating frequency up to 98.208 MHz

64 KByte ROM

640 KByte SRAM

CORDIC engine for GNSS support

* Multi-GNSS receiver

GPS (L1 C/A)

GLONASS (L1OF)

QZSS (L1 C/A, L1 SAIF)

SBAS (L1 C/A)

WAAS, EGNOS, MSAS

BeiDou (B1)

Galileo (E1 CBOC)

* Configurable I/O

I2C/SPI/GPIO Interfaces

* Debug

Serial wire debug (SWD), Embedded Trace Macrocell

UART supported

# 4 Sensors

## 4.1 3D accelerometer and 3D gyroscope

### 4.1.1 General Description

The LSM6DSOX is a system-in-package featuring a 3D digital accelerometer and a 3D digital gyroscope boosting performance at 0.55 mA in high-performance mode and enabling always-on low-power features for an optimal motion experience for the consumer. The LSM6DSOX supports main OS requirements, offering real, virtual and batch sensors with 9 kbytes for dynamic data batching. ST’s family of MEMS sensor modules leverages the robust and mature manufacturing processes already used for the production of micromachined accelerometers and gyroscopes. The various sensing elements are manufactured using specialized micromachining processes, while the IC interfaces are developed using CMOS technology that allows the design of a dedicated circuit which is trimmed to better match the characteristics of the sensing element. The LSM6DSOX has a full-scale acceleration range of ±2/±4/±8/±16 g and an angular rate range of ±125/±250/±500/±1000/±2000 dps. The LSM6DSOX fully supports EIS and OIS applications as the module includes a dedicated configurable signal processing path for OIS and auxiliary SPI, configurable for both the gyroscope and accelerometer. The LSM6DSOX OIS can be configured from the Auxiliary SPI and primary interface (SPI / I²C & MIPI I3CSM). High robustness to mechanical shock makes the LSM6DSOX the preferred choice of system designers for the creation and manufacturing of reliable products. The LSM6DSOX is available in a plastic land grid array (LGA) package.

### 4.1.2 Features

• Power consumption: 0.55 mA in combo high-performance mode

• “Always-on" experience with low power consumption for both accelerometer and gyroscope

• Smart FIFO up to 9 kbyte

• Android compliant

• ±2/±4/±8/±16 g full scale

• ±125/±250/±500/±1000/±2000 dps full scale

• Analog supply voltage: 1.71 V to 3.6 V

• Independent IO supply (1.62 V)

• Compact footprint: 2.5 mm x 3 mm x 0.83 mm

• SPI / I²C & MIPI I3CSM serial interface with main processor data synchronization

• Auxiliary SPI for OIS data output for gyroscope and accelerometer

• OIS configurable from Aux SPI, primary interface (SPI / I²C & MIPI I3CSM)

• Advanced pedometer, step detector and step counter

• Significant Motion Detection, Tilt detection

• Standard interrupts: free-fall, wakeup, 6D/4D orientation, click and double-click

• Programmable finite state machine: accelerometer, gyroscope and external sensors

• Machine Learning Core

• S4S data synchronization

• Embedded temperature sensor

• ECOPACK®, RoHS and “Green” compliant

### 4.1.3 Schematic connections

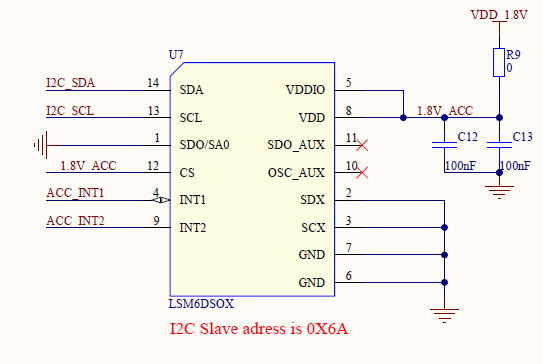


Figure 6. LSM6DSOX connection

For normal sensor work LSM6DSOX is connected to I2C0, for extra functions INT1 is connected to J2.5, and INT2 is connected to P1 of the port expander (Fig. 5).

## 4.2 3-Axis magnetometer

### 4.2.1 General description

The LIS2MDL is an ultra-low-power, high-performance 3-axis digital magnetic sensor. The LIS2MDL has a magnetic field dynamic range of ±50 gauss. The LIS2MDL includes an I2C serial bus interface that supports standard, fast mode, fast mode plus, and high-speed (100 kHz, 400 kHz, 1 MHz, and 3.4 MHz) and an SPI serial standard interface.

The device can be configured to generate an interrupt signal for magnetic field detection. The LIS2MDL is available in a plastic land grid

array package (LGA) and is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

### 4.2.2 Features

* 3 magnetic field channels
* ±50 gauss magnetic dynamic range
* 16-bit data output
* SPI/I2C serial interfaces
* Analog supply voltage 1.71 V to 3.6 V
* Selectable power mode/resolution
* Single measurement mode
* Programmable interrupt generator
* Embedded self-test
* Embedded temperature sensor
* ECOPACK®, RoHS and “Green” compliant

### **4.2.3 Schematic connection**

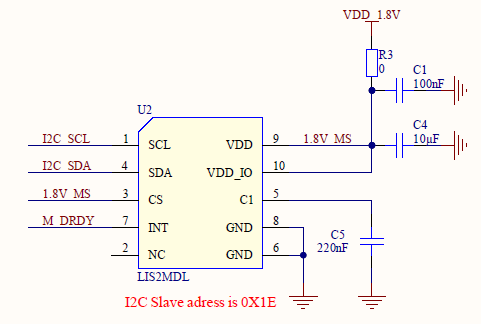


Figure 7. LIS2MDL connection

For normal sensor work LIS2MDL is connected to I2C0, for extra functions INT is connected to UART2\_CTS (Fig. 6).

## 4.3 Humidity and temperature sensor

### 4.3.1 General description

The HTS221 is an ultra-compact sensor for relative humidity and temperature. It includes a sensing element and a mixed signal ASIC to provide the measurement information through digital serial interfaces. The sensing element consists of a polymer dielectric planar capacitor structure capable of detecting relative humidity variations and is manufactured using a dedicated ST process.

The HTS221 is available in a small top-holed cap land grid array (HLGA) package guaranteed to operate over a temperature range from -40 °C to +120 °C.

### 4.3.2 Features

* 0 to 100% relative humidity range
* Supply voltage: 1.7 to 3.6 V
* Low power consumption: 2 μA @ 1 Hz ODR
* Selectable ODR from 1 Hz to 12.5 Hz
* High rH sensitivity: 0.004% rH/LSB
* Humidity accuracy: ± 3.5% rH, 20 to +80% rH
* Temperature accuracy: ± 0.5 °C,15 to +40 °C
* Embedded 16-bit ADC
* 16-bit humidity and temperature output data
* SPI and I²C interfaces
* Factory calibrated
* Tiny 2 x 2 x 0.9 mm package
* ECOPACK® compliant

### 4.3.3 Schematic connection

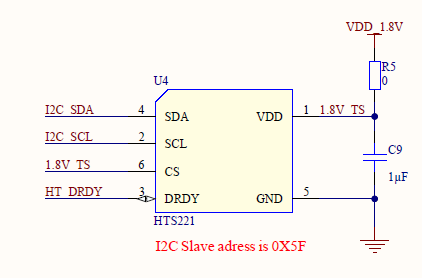


Figure 8. HTS221 connection

For normal sensor work HTS221 is connected to I2C0, for extra functions DRDY is connected to I2S0\_DATA\_OUT (Fig. 7).

## 4.4 Pressure sensor

### 4.4.1 General description

The LPS22HH is an ultra-compact piezoresistive absolute pressure sensor which functions as a digital output barometer. The device comprises a sensing element and an IC interface which communicates through I²C, MIPI I3CSM or SPI from the sensing element to the application. The sensing element, which detects absolute pressure, consists of a suspended membrane manufactured using a dedicated process developed by ST.

The LPS22HH is available in a full-mold, holed LGA package (HLGA). It is guaranteed to operate over a temperature range extending from -40 °C to +85 °C. The package is holed to allow external pressure to reach the sensing element.

### 4.4.2 Features

* 260 to 1260 hPa absolute pressure range
* Current consumption down to 4 μA
* Absolute pressure accuracy: 0.5 hPa
* Low pressure sensor noise: 0.65 Pa
* High-performance TCO: 0.65 Pa/°C
* Embedded temperature compensation
* 24-bit pressure data output
* ODR from 1 Hz to 200 Hz
* SPI, I²C or MIPI I3CSM interfaces
* Embedded FIFO
* Interrupt functions: Data-Ready, FIFO flags,
* pressure thresholds
* Supply voltage: 1.7 to 3.6 V
* High shock survivability: 22,000 g
* Small and thin package
* ECOPACK® lead-free compliant

### 4.4.3 Schematic connection

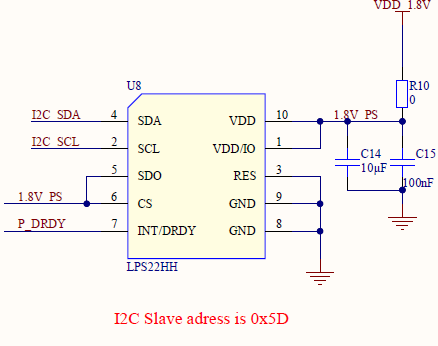


Figure 9. LPS22HH connection

For normal sensor work LPS22HH is connected to I2C0 (Fig. 8).

## 4.5 Proximity sensor

### 4.5.1 General description

The VL53L1X is a state-of-the-art, Time-of-Flight (ToF), laser-ranging sensor, enhancing the ST FlightSense product family. It is the fastest miniature ToF sensor on the market with accurate ranging up to 4 m and fast ranging frequency up to 50 Hz

Housed in a miniature and reflowable package, it integrates a SPAD receiving array, a 940 nm invisible Class1 laser emitter, physical infrared filters, and optics to achieve the best ranging performance in various ambient lighting conditions with a range of cover window options.

Unlike conventional IR sensors, the VL53L1X uses ST’s latest generation ToF technology which allows absolute distance measurement whatever the target color and reflectance.

It is also possible to program the size of the ROI on the receiving array, allowing the sensor FoV to be reduced.

### 4.5.2 Features

Fully integrated miniature module

– Size: 4.9x2.5x1.56 mm

– Emitter: 940 nm invisible laser (Class1)

– SPAD (single photon avalanche diode)

receiving array with integrated lens

– Low-power microcontroller running advanced

digital firmware

Pin-to-pin compatible with the VL53L0X

FlightSense ranging sensor

Fast and accurate long distance ranging

– Up to 400 cm distance measurement

– Up to 50 Hz ranging frequency

Typical full field-of-view (FoV): 27°

Programmable region-of-interest (ROI) size on

the receiving array, allowing the sensor FoV to be

reduced

Programmable ROI position on the receiving

array, providing multizone operation control from

the host

Easy integration

– Single reflowable component

– Can be hidden behind many cover window

materials

– Software driver and code examples for

turnkey ranging

– Single power supply (2v8)

– I²C interface (up to 400 kHz)

### **4.5.3 Schematic connection**

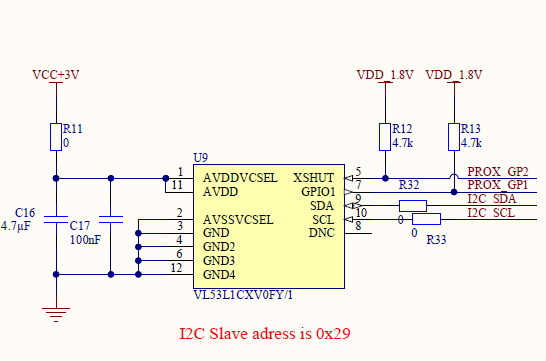


Figure 10. VL53L1X connection

For normal sensor work VL53L1X connected to I2C0, for extra functions XSHUT connected to UART2\_RTS and GPIO1 connected to J2.4 (Fig. 9).

## 4.6 Air quality sensor

### 4.6.1 General description

The SGP40 is a digital gas sensor designed for easy integration into air purifier s or demand controlled ventilation systems. Sensirion’s CMOSens ® technology offers a complete , easy to use sensor system on a single chip featuring a digital I 2 C interface and a temperature controlle d micro hotplate, providing a humidity compensated VOC based indoor air quality signal . The output signal can be directly processed by Sensirion’s powerful VOC Algorithm to translate the raw signal into a VOC Index as a robust measure for indoor air quality. The VOC Algorithm automatically adapts to the environment the sensor is exposed to. Both sensing element and

VOC Algorithm feature an unmatched robustness against contaminating gases present in real world applications enabling a unique long term stability as well as low drift and device to device variation. The very small 2.44 x 2.44 x 0.85 mm 3 DFN package enables

applications in limited spaces. Sensirion’s state of the art production process guarantees high reproducibility and reliability. Tape and reel packaging together with suitability for standard SMD a ssembly processes make the SGP40 prede stined for high volume applications.

### **4.6.2 Schematic connection**

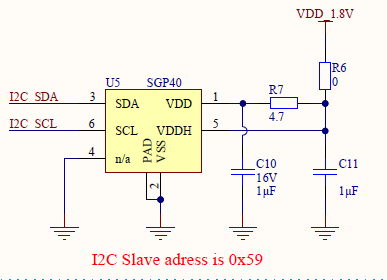


Figure 11. SGP40 connection

For normal sensor work SGP40 connected to I2C0 (Fig. 10).

## 4.7 Digital RGB, IR and Ambient Light Sensor

### 4.7.1 General description

The APDS-9250 device uses 4 individual channels of red, green, blue, and IR (RGB+IR) in a specially designed matrix arrangement. This allows the device to have optimal angular response and accurate RGB spectral response with high lux accuracy over various light sources. APDS-9250 supports the I2C interface and has a programmable interrupt controller that frees up micro-controller resources.

The device detects light intensity under a variety of lighting conditions and through a variety of attenuation materials, including dark glass. APDS-9250 could be configured as Ambient Light Sensor and RGB+IR Sensor. The color-sensing feature is useful in applications such as LED RGB backlight control, solid-state lighting, reflected LED color sampler, or fluorescent light color temperature detection. The integrated IR blocking filter makes this device an excellent ambient light sensor and color temperature monitor sensor together with the temperature compensation that allows output to have less variation over the temperature.

### 4.7.2 Features

• Colour and Ambient Light Sensing (CS-RGB and ALS)

- Accuracy of Correlated Color Temperature (CCT)

- Individual channels for Red, Green, Blue and Infared

- Approximates Human Eye Response with Green Channel

- Red, Green, Blue, Infrared and ALS Sensing

- High Sensitivity in low lux condition – Ideally suited for Operation Behind Dark Glass

- Wide Dynamic Range: 18,000,000: 1

- Up to 20-Bit Resolution

• Power Management

- Low Active Current – 130 μA typical

- Low Standby Current – 1μA typical

• I2C-bus Fast Mode Compatible Interface

- Up to 400 kHz (I2C Fast-Mode)

- Dedicated Interrupt Pin

• Small Package L 2.0 × W 2.0 × H 0.65 mm

### 4.7.3 Schematic connection

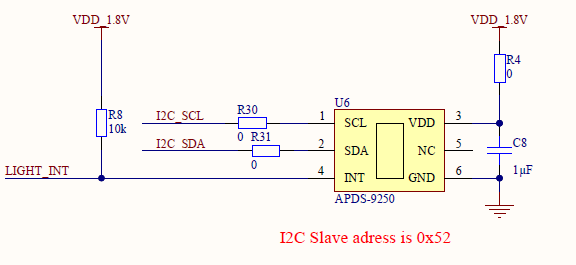


Figure 12. APDS-9250 connection

For normal sensor work APDS-9250 connected to I2C0, for extra functions INTconnected to UART2\_TX (Fig.11).

## 4.8 Port expander

### 4.8.1 General description

The PCA9538 is an 8-bit I/O expander of general purpose parallel input and output (I/O) expansion for the two-line bidirectional I2C bus (or SMBus) protocol. This device can operate with a power supply range from 2.3 V to 5.5 V. This device supports both100-kHz (Standard-mode) and 400-kHz (Fast-mode) clock frequencies. This device, along with other I/O expanders, provides a simple solution when additional I/Os are needed for switches, sensors, push-buttons, LEDs, fans, and so on.

The features of PCA9538 include an interrupt that is generated on the INT pin whenever an input port changes state. The A0 and A1 hardware selectable address pins allow up to four PCA9538 devices on the same I2C bus. This device can also be reset to its default state by using the RESET feature or by cycling the power supply to cause a power-on reset.

INT can be connected to the interrupt input of a microcontroller. By sending an interrupt signal on this line, the remote I/O can inform the microcontroller if there is incoming data on its ports without having to communicate via the I2C bus. Thus, the PCA9538 can remain a simple slave device.

The device outputs (latched) have high-current drive capability for directly driving LEDs. It has low current consumption.

Two hardware pins (A0 and A1) are used to program and vary the fixed I2C address and allow up to four devices to share the same I2C bus or SMBus.

### 4.8.2 Features

• Low standby current consumption of 1 μA max

• I2C to parallel port expander

• Open-drain active-low interrupt output

• Active-low reset input

• Operating power-supply voltage range of 2.3 V to

5.5 V

• 5-V Tolerant I/O ports

• 400-kHz Fast I2C bus

• Two hardware address pins allow up to four

devices on the I2C/SMBus

• Input and output configuration register

• Polarity inversion register

• Power-up with all channels configured as inputs

• No glitch on power up

• Noise filter on SCL/SDA inputs

• Latched outputs with high-current drive maximum

capability for directly driving LEDs

• Latch-up performance exceeds 100 mA Per JESD

78, class II

• ESD protection exceeds JESD 22

– 2000-V Human-body model (A114-A)

– 200-V Machine model (A115-A)

– 1000-V Charged-device model (C101)

### **4.8.3 Schematic connection**

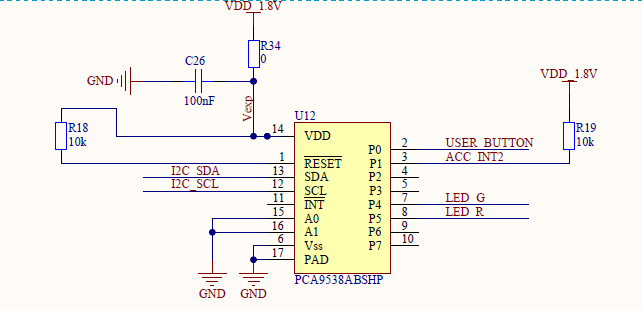


Figure 13. APDS-9250 connection

For normal sensor work PCA9538 is connected to I2C0 (Fig.12).

## **4.9 Micro SD card**

### 4.9.1 General description

Should be used micro SD card 3.3V. Software developed and tested with Kingston micro SD card with 16 Gb of memory. The system might not work with some kinds of micro SD cards.

### 4.9.2 Schematic connection

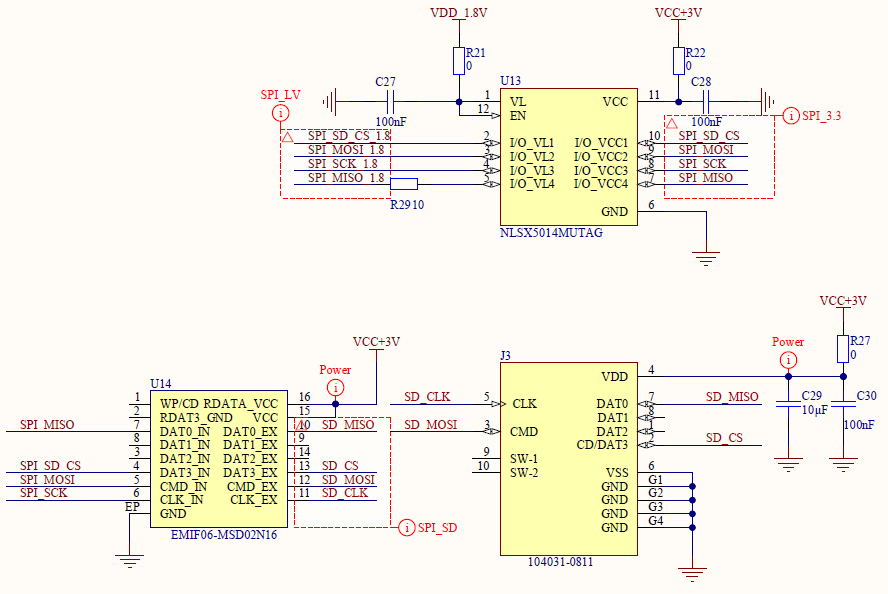


Figure 14. micro SD card connection

For normal sensor work micro SD card connected to SPI5, micro SD CS connected to SPI5\_CS\_X (Fig.14).

# 5 Examples of use

In this part added some examples of usage of equipment. All tests are enabled in default project and going one by one. For separate test you have to comment “#define ALL\_TESTS” in Tests.cpp file and uncomment just one from defines below.

If you want to use some equipment in your app see recommendations below.

## 5.1 LEDs

LEDs works with port expander

For working with LEDs you have to include Led.h header file.

After that you have to init the LEDs:

led\_init();

After that you can use for functions to control LEDs:

led\_green\_off();

led\_red\_on();

led\_red\_off();

led\_green\_on();

## 5.2 Button

Button works with port expander

For working with button you have to include Led.h header file.

After that you have to init the button:

button\_init ();

After that you can check button state with:

button\_is\_pressed();

## 5.3 Speaker

The speaker works with PWM

For work with PWM you have to include Pwm.h

To set PWM parameters you have to use:

pwm\_run(10, 20, 500);

where 10 – frequency 1 – 1000 Hz, 20 – duty cycle 1 – 100 %, 500 – time in ms.

For PWM work

pwm\_act();

should be run in loop, when PWM is finished function returns false.

## 5.4 HTS221TR: humidity and temperature

To run HTS221 you have to include Hts221.h and I2c.h

Variables example and other functions:

typedef struct {

float x0;

float y0;

float x1;

float y1;

} lin\_t;

static uint8\_t hts221\_whoamI = 0;

static int16\_t hts221\_data\_raw\_humidity;

static int16\_t hts221\_data\_raw\_temperature;

static float hts221\_humidity\_perc;

static float hts221\_temperature\_degC;

static lin\_t lin\_hum;

static lin\_t lin\_temp;

static float linear\_interpolation(lin\_t \*lin, int16\_t x) {

return ((lin->y1 - lin->y0) \* x + ((lin->x1 \* lin->y0) -

(lin->x0 \* lin->y1)))

/ (lin->x1 - lin->x0);

}

To init the device you can use:

i2c\_init();

hts221\_device\_id\_get(nullptr, &hts221\_whoamI);

printf("hts221\_whoamI = %d\r\n", hts221\_whoamI);

/\* Read humidity calibration coefficient \*/

hts221\_hum\_adc\_point\_0\_get(nullptr, &lin\_hum.x0);

hts221\_hum\_rh\_point\_0\_get(nullptr, &lin\_hum.y0);

hts221\_hum\_adc\_point\_1\_get(nullptr, &lin\_hum.x1);

hts221\_hum\_rh\_point\_1\_get(nullptr, &lin\_hum.y1);

/\* Read temperature calibration coefficient \*/

hts221\_temp\_adc\_point\_0\_get(nullptr, &lin\_temp.x0);

hts221\_temp\_deg\_point\_0\_get(nullptr, &lin\_temp.y0);

hts221\_temp\_adc\_point\_1\_get(nullptr, &lin\_temp.x1);

hts221\_temp\_deg\_point\_1\_get(nullptr, &lin\_temp.y1);

/\* Enable Block Data Update \*/

hts221\_block\_data\_update\_set(nullptr, PROPERTY\_ENABLE);

/\* Set Output Data Rate \*/

hts221\_data\_rate\_set(nullptr, HTS221\_ODR\_1Hz);

/\* Device power on \*/

hts221\_power\_on\_set(nullptr, PROPERTY\_ENABLE);

For get data from senor you can use:

/\* Read output only if new value is available \*/

hts221\_reg\_t reg;

hts221\_status\_get(nullptr, &reg.status\_reg);

if (reg.status\_reg.h\_da) {

/\* Read humidity data \*/

memset(&hts221\_data\_raw\_humidity, 0x00, sizeof(int16\_t));

hts221\_humidity\_raw\_get(nullptr, (uint8\_t\*) &hts221\_data\_raw\_humidity);

hts221\_humidity\_perc = linear\_interpolation(&lin\_hum, hts221\_data\_raw\_humidity);

printf("Hymidity\_raw = %d\r\n", hts221\_data\_raw\_humidity);

hts221\_humidity\_perc = 0.0 - hts221\_humidity\_perc;

if (hts221\_humidity\_perc < 0) {

hts221\_humidity\_perc = 0;

}

if (hts221\_humidity\_perc > 100) {

hts221\_humidity\_perc = 100;

}

printf("Humidity [%%]:%3.2f\r\n", hts221\_humidity\_perc);

reg.status\_reg.h\_da = 0;

}

if (reg.status\_reg.t\_da) {

/\* Read temperature data \*/

memset(&hts221\_data\_raw\_temperature, 0x00, sizeof(int16\_t));

hts221\_temperature\_raw\_get(nullptr, (uint8\_t\*) &hts221\_data\_raw\_temperature);

hts221\_temperature\_degC = linear\_interpolation(&lin\_temp, hts221\_data\_raw\_temperature);

printf("Temperature [degC]:%6.2f\r\n", hts221\_temperature\_degC );

reg.status\_reg.t\_da = 0;

}

## 5.5 LIS2MDL: 3-Axis Magnetometer

To work with LIS2MDL you have to include Lis2mdl.h and I2c.h

Variables example:

static int16\_t data\_raw\_magnetic[3];

static int16\_t lis2\_data\_raw\_temperature;

static float magnetic\_mG[3];

static float lis2\_temperature\_degC;

static uint8\_t lis2\_whoamI, lis2\_rst;

To intialisation you may use:

i2c\_init();

/\* Check device ID \*/

lis2mdl\_device\_id\_get(nullptr, &lis2\_whoamI);

printf("LIS2MDL\_ID = %d\r\n", lis2\_whoamI);

if (lis2\_whoamI != LIS2MDL\_ID) {

while (1) {

/\* manage here device not found \*/

}

}

/\* Restore default configuration \*/

lis2mdl\_reset\_set(nullptr, PROPERTY\_ENABLE);

do {

lis2mdl\_reset\_get(nullptr, &lis2\_rst);

} while (lis2\_rst);

/\* Enable Block Data Update \*/

lis2mdl\_block\_data\_update\_set(nullptr, PROPERTY\_ENABLE);

/\* Set Output Data Rate \*/

lis2mdl\_data\_rate\_set(nullptr, LIS2MDL\_ODR\_10Hz);

/\* Set / Reset sensor mode \*/

lis2mdl\_set\_rst\_mode\_set(nullptr, LIS2MDL\_SENS\_OFF\_CANC\_EVERY\_ODR);

/\* Enable temperature compensation \*/

lis2mdl\_offset\_temp\_comp\_set(nullptr, PROPERTY\_ENABLE);

/\* Set Low-pass bandwidth to ODR/4 \*/

//lis2mdl\_low\_pass\_bandwidth\_set(nullptr, LIS2MDL\_ODR\_DIV\_4);

/\* Set device in continuous mode \*/

lis2mdl\_operating\_mode\_set(nullptr, LIS2MDL\_CONTINUOUS\_MODE);

/\* Enable interrupt generation on new data ready \*/

lis2mdl\_drdy\_on\_pin\_set(nullptr, PROPERTY\_ENABLE);

To get data and pint:

uint8\_t reg;

/\* Read output only if new value is available \*/

lis2mdl\_mag\_data\_ready\_get(nullptr, &reg);

if (reg) {

/\* Read magnetic field data \*/

memset(data\_raw\_magnetic, 0x00, 3 \* sizeof(int16\_t));

lis2mdl\_magnetic\_raw\_get(nullptr, (uint8\_t\*) data\_raw\_magnetic);

magnetic\_mG[0] = lis2mdl\_from\_lsb\_to\_mgauss( data\_raw\_magnetic[0]);

magnetic\_mG[1] = lis2mdl\_from\_lsb\_to\_mgauss( data\_raw\_magnetic[1]);

magnetic\_mG[2] = lis2mdl\_from\_lsb\_to\_mgauss( data\_raw\_magnetic[2]);

printf("Mag field [mG]:%4.2f\t%4.2f\t%4.2f\r\n", magnetic\_mG[0], magnetic\_mG[1], magnetic\_mG[2]);

/\* Read temperature data \*/

memset(&lis2\_data\_raw\_temperature, 0x00, sizeof(int16\_t));

lis2mdl\_temperature\_raw\_get(nullptr, (uint8\_t\*) &lis2\_data\_raw\_temperature);

lis2\_temperature\_degC =

lis2mdl\_from\_lsb\_to\_celsius(lis2\_data\_raw\_temperature);

printf("Temperature [degC]:%6.2f\r\n", lis2\_temperature\_degC);

## 5.6 LPS22HH: pressure sensor

To work with LPS22HH you have to include Lps22hh.h and I2c.h

Variables example:

static uint32\_t data\_raw\_pressure;

static int16\_t lps22\_data\_raw\_temperature;

static float pressure\_hPa;

static float lps22\_temperature\_degC;

static uint8\_t lps22\_whoamI, lps22\_rst;

To initialization:

i2c\_init();

/\* Check device ID \*/

lps22\_whoamI = 0;

lps22hh\_device\_id\_get(nullptr, &lps22\_whoamI);

printf("LPS22HH\_ID = %d", lps22\_whoamI);

if (lps22\_whoamI != LPS22HH\_ID ) {

printf("LPS22HH\_ID Init Error\r\n");

while (1); /\*manage here device not found \*/

}

/\* Restore default configuration \*/

lps22hh\_reset\_set(nullptr, PROPERTY\_ENABLE);

do {

lps22hh\_reset\_get(nullptr, &lps22\_rst);

} while (lps22\_rst);

/\* Enable Block Data Update \*/

lps22hh\_block\_data\_update\_set(nullptr, PROPERTY\_ENABLE);

/\* Set Output Data Rate \*/

lps22hh\_data\_rate\_set(nullptr, LPS22HH\_10\_Hz\_LOW\_NOISE);

To get data and pint:

lps22hh\_reg\_t reg;

/\* Read output only if new value is available \*/

lps22hh\_read\_reg(nullptr, LPS22HH\_STATUS, (uint8\_t \*)&reg, 1);

if (reg.status.p\_da) {

memset(&data\_raw\_pressure, 0x00, sizeof(uint32\_t));

lps22hh\_pressure\_raw\_get(nullptr, (uint8\_t \*) &data\_raw\_pressure);

pressure\_hPa = lps22hh\_from\_lsb\_to\_hpa( data\_raw\_pressure);

printf("pressure [hPa]:%6.2f\r\n", pressure\_hPa);

}

if (reg.status.t\_da) {

memset(&lps22\_data\_raw\_temperature, 0x00, sizeof(int16\_t));

lps22hh\_temperature\_raw\_get(nullptr, (uint8\_t \*) &lps22\_data\_raw\_temperature);

lps22\_temperature\_degC = lps22hh\_from\_lsb\_to\_celsius(

lps22\_data\_raw\_temperature );

printf("temperature [degC]:%6.2f\r\n", lps22\_temperature\_degC );

}

## 5.7 LSM6DS3: inertial module: 3D accelerometer and 3D gyroscope

To work with LSM6DS3 you have to include Lsm6dso32.h and I2c.h

Variables example:

static int16\_t data\_raw\_acceleration[3];

static int16\_t data\_raw\_angular\_rate[3];

static int16\_t lsm6\_data\_raw\_temperature;

static float acceleration\_mg[3];

static float angular\_rate\_mdps[3];

static float lsm6\_temperature\_degC;

static uint8\_t lsm6\_whoamI, rst;

static uint8\_t tx\_buffer[1000];

static stmdev\_ctx\_t dev\_ctx;

To initialization:

i2c\_init();

/\* Check device ID \*/

lsm6dso32\_device\_id\_get(&dev\_ctx, &lsm6\_whoamI);

printf("LSM6DS32\_ID = %d\r\n", lsm6\_whoamI);

if (lsm6\_whoamI != LSM6DSO32\_ID)

while (1);

/\* Restore default configuration \*/

lsm6dso32\_reset\_set(&dev\_ctx, PROPERTY\_ENABLE);

do {

lsm6dso32\_reset\_get(&dev\_ctx, &rst);

} while (rst);

/\* Disable I3C interface \*/

lsm6dso32\_i3c\_disable\_set(&dev\_ctx, LSM6DSO32\_I3C\_DISABLE);

/\* Enable Block Data Update \*/

lsm6dso32\_block\_data\_update\_set(&dev\_ctx, PROPERTY\_ENABLE);

/\* Set full scale \*/

lsm6dso32\_xl\_full\_scale\_set(&dev\_ctx, LSM6DSO32\_4g);

lsm6dso32\_gy\_full\_scale\_set(&dev\_ctx, LSM6DSO32\_2000dps);

/\* Set ODR (Output Data Rate) and power mode\*/

lsm6dso32\_xl\_data\_rate\_set(&dev\_ctx, LSM6DSO32\_XL\_ODR\_12Hz5\_LOW\_PW);

lsm6dso32\_gy\_data\_rate\_set(&dev\_ctx,

LSM6DSO32\_GY\_ODR\_12Hz5\_HIGH\_PERF);

To get data and pint:

lsm6dso32\_reg\_t reg;

/\* Read output only if new data is available \*/

lsm6dso32\_status\_reg\_get(&dev\_ctx, &reg.status\_reg);

if (reg.status\_reg.xlda) {

/\* Read acceleration data \*/

memset(data\_raw\_acceleration, 0x00, 3 \* sizeof(int16\_t));

lsm6dso32\_acceleration\_raw\_get(&dev\_ctx, (uint8\_t\*) data\_raw\_acceleration);

acceleration\_mg[0] =

lsm6dso32\_from\_fs4\_to\_mg(data\_raw\_acceleration[0]);

acceleration\_mg[1] =

lsm6dso32\_from\_fs4\_to\_mg(data\_raw\_acceleration[1]);

acceleration\_mg[2] =

lsm6dso32\_from\_fs4\_to\_mg(data\_raw\_acceleration[2]);

printf("Acceleration [mg]:%4.2f\t%4.2f\t%4.2f\r\n",

acceleration\_mg[0], acceleration\_mg[1], acceleration\_mg[2]);

}

if (reg.status\_reg.gda) {

/\* Read angular rate field data \*/

memset(data\_raw\_angular\_rate, 0x00, 3 \* sizeof(int16\_t));

lsm6dso32\_angular\_rate\_raw\_get(&dev\_ctx, (uint8\_t\*) data\_raw\_angular\_rate);

angular\_rate\_mdps[0] =

lsm6dso32\_from\_fs2000\_to\_mdps(data\_raw\_angular\_rate[0]);

angular\_rate\_mdps[1] =

lsm6dso32\_from\_fs2000\_to\_mdps(data\_raw\_angular\_rate[1]);

angular\_rate\_mdps[2] =

lsm6dso32\_from\_fs2000\_to\_mdps(data\_raw\_angular\_rate[2]);

printf("Angular rate [mdps]:%4.2f\t%4.2f\t%4.2f\r\n",

angular\_rate\_mdps[0], angular\_rate\_mdps[1], angular\_rate\_mdps[2]);

}

if (reg.status\_reg.tda) {

/\* Read temperature data \*/

memset(&lsm6\_data\_raw\_temperature, 0x00, sizeof(int16\_t));

lsm6dso32\_temperature\_raw\_get(&dev\_ctx, (uint8\_t\*) &lsm6\_data\_raw\_temperature);

lsm6\_temperature\_degC = lsm6dso32\_from\_lsb\_to\_celsius(

lsm6\_data\_raw\_temperature);

printf("Temperature [degC]:%6.2f\r\n", lsm6\_temperature\_degC);

}

## 5.8 VL53L1X: Proximitry sensor

To work with VL53L1X you have to include VL53L1X\_api.h and I2c.h

Variables example:

static uint8\_t sensorState = 0;

static uint8\_t first\_range = 1;

static int status;

static uint8\_t byteData;

static uint16\_t wordData;

static uint16\_t Dev = 0;

static VL53L1X\_Result\_t Results;

To initialization:

i2c\_init();

usleep(1000 \* 1000);

status = VL53L1\_RdByte(Dev, 0x010F, &byteData);

printf("VL53L1X Model\_ID: %X\n", byteData);

status += VL53L1\_RdByte(Dev, 0x0110, &byteData);

printf("VL53L1X Module\_Type: %X\n", byteData);

status += VL53L1\_RdByte(Dev, 0x0111, &byteData);

printf("VL53L1X Revision: %X\n", byteData);

while (sensorState == 0) {

status += vl53l1x\_boot\_state(Dev, &sensorState);

VL53L1\_WaitMs(Dev, 20);

}

printf("Chip booted\n");

status = vl53l1x\_sensor\_init(Dev);

/\* status += vl53l1x\_setInterrupt\_polarity(Dev, 0); \*/

status += vl53l1x\_set\_distance\_mode(Dev, 2); /\* 1=short, 2=long \*/

status += vl53l1x\_set\_timing\_budgetIn\_ms(Dev, 100);

status += vl53l1x\_set\_inter\_measurementIn\_ms(Dev, 100);

status += vl53l1x\_start\_ranging(Dev);

To get data and pint:

static uint8\_t dataReady = 0;

while (dataReady == 0) {

status = vl53l1x\_check\_for\_data\_ready(Dev, &dataReady);

//printf("status = %u, dataReady = %u\r\n", status, dataReady);

usleep(20 \* 1000);

}

dataReady = 0;

/\* Get the data the new way \*/

status += vl53l1x\_get\_result(Dev, &Results);

printf("Status = %2d, dist = %5d, Ambient = %2d, Signal = %5d, #ofSpads = %5d\n",

Results.Status, Results.Distance, Results.Ambient,

Results.SigPerSPAD, Results.NumSPADs);

/\* trigger next ranging \*/

status += vl53l1x\_clear\_interrupt(Dev);

if (first\_range) {

/\* very first measurement shall be ignored

\* thus requires twice call

\*/

status += vl53l1x\_clear\_interrupt(Dev);

first\_range = 0;

}

status = vl53l1x\_sensor\_init(Dev);

status += vl53l1x\_set\_distance\_mode(Dev, 2); /\* 1=short, 2=long \*/

status += vl53l1x\_set\_timing\_budgetIn\_ms(Dev, 100);

status += vl53l1x\_set\_inter\_measurementIn\_ms(Dev, 100);

status += vl53l1x\_start\_ranging(Dev);

## 5.9 APDS-9250: Digital RGB, IR and Ambient Light Sensor

To work with APDS-9250 you have to include Apds9250.h and I2c.h

Variables example:

Apds9250 myApds9250;

To initialization:

i2c\_init();

if (myApds9250.begin())

{

printf("myApds9250.begin() OK\r\n");

}

myApds9250.setMode(modeColorSensor);

myApds9250.setResolution(res18bit);

myApds9250.setGain(gain1);

myApds9250.setMeasurementRate(rate100ms);

To get data and pint:

uint32\_t red = 0;

uint32\_t green = 0;

uint32\_t blue = 0;

uint32\_t ir = 0;

myApds9250.getAll(&red, &green, &blue, &ir);

printf("red = %lu\r\ngreen = %lu\r\nblue = %lu\r\nir = %lu\r\n", red, green, blue, ir);

## 5.10 SGP-40: Air quality sensor

To work with APDS-9250 you have to include sgp40\_i2c.h and I2c.h

Variables example:

static int16\_t error = 0;

static uint16\_t serial\_number[3];

static uint8\_t serial\_number\_size = 3;

// Parameters for deactivated humidity compensation:

static uint16\_t default\_rh = 0x8000;

static uint16\_t default\_t = 0x6666;

To initialization:

i2c\_init();

error = sgp40\_get\_serial\_number(serial\_number, serial\_number\_size);

if (error) {

printf("Error executing sgp40\_get\_serial\_number(): %i\r\n", error);

} else {

printf("serial: 0x%04x%04x%04x\r\n", serial\_number[0], serial\_number[1],

serial\_number[2]);

printf("\r\n");

}

To get data and pint:

uint16\_t sraw\_voc;

error = sgp40\_measure\_raw\_signal(default\_rh, default\_t, &sraw\_voc);

if (error) {

printf("Error executing sgp40\_measure\_raw\_signal(): %i\n", error);

} else {

printf("SRAW VOC: %u\n", sraw\_voc);

}

## 5.11 Low power test

For using low power modes LowPower.h should be included.

After that should be run:

LowPower.begin();

And selected one from Low power modes, ex:

LowPower.deepSleep();

## 5.12 SD card

To use SD card to write a raw data should be included Sdcard.h

To initialization:

sdcard\_init();

After that you can read and write SD card block(s) ex:

sdcard\_write\_single\_block(0, blockDataWr);

sdcard\_read\_single\_block(0, blockData);

## 5.13 FatFS SD card

To work with FS ff.h and Sdcard.h should be included

To initialization:

int result = sdcard\_init();

if( f\_mount(&USERFatFS,(TCHAR const\*)USERPath, 0) != FR\_OK) {

printf("FS mount error\r\n");

}

To work with Fat FS refer to

http://elm-chan.org/fsw/ff/00index\_e.html

**Best regards**

We wish you an interesting using of the board and discovering new possibilities with this powerful sensor set and MCU