Use M263 I²C to Drive TDK CH101

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code uses M263 I²C to drive TDK CH101 to achieve distance measurement function.
BSP Version	M263_Series_BSP_CMSIS_V3.00.003
Hardware	NuMaker-M263KI V1.1

1 Overview

This example code uses the M263 I²C to drive CH101 module to achieve distance measurement function. The CH101 provides accurate range measurements to target at distances up to 1.2m. Unlike other types of ToF rangefinders, such as infrared (IR) sensors, the CH101 is not affected by the color or transparency of objects and works in all lighting conditions.

The EV_MOD_CH101-03-01 module of TDK is used in this example. The module board incorporates a CH101 Ultrasonic Sensor device with an omnidirectional acoustic housing assembly, a capacitor and an FPC/FFC connector, and communicates with main control microcontroller (MCU) through I²C interface. This module can perform pitch-catch and pulse-echo range-finding at distances from 4 cm to 1.2m.

In this document, the description and information about CH101 on TDK website are quoted.

1.1 Principle

The CH101 is an ultrasonic transceiver rangefinder that uses a piezoelectric micromachined ultrasonic transducer (PMUT) to send out short pulses of soundwaves into the air. Upon hitting an object, these waves reflect back towards the PMUT, causing it to vibrate and generate an electrical signal. The time needed for the soundwaves to travel from and back to the PMUT, known as the Time-of-Flight (ToF), is measured by the built-in application-specific integrated circuit (ASIC). Using the speed of sound (343 m/s at room temperature), the system can determine the distance to the object.

Measuring distance using time of flight:

$$Range\ (R) = C\ x \frac{ToF}{2}$$

C(Speed of Sound of air) = 343 m/s.

The CH101 can be controlled and read through the I²C interface. The block diagram of reading and writing is shown below:

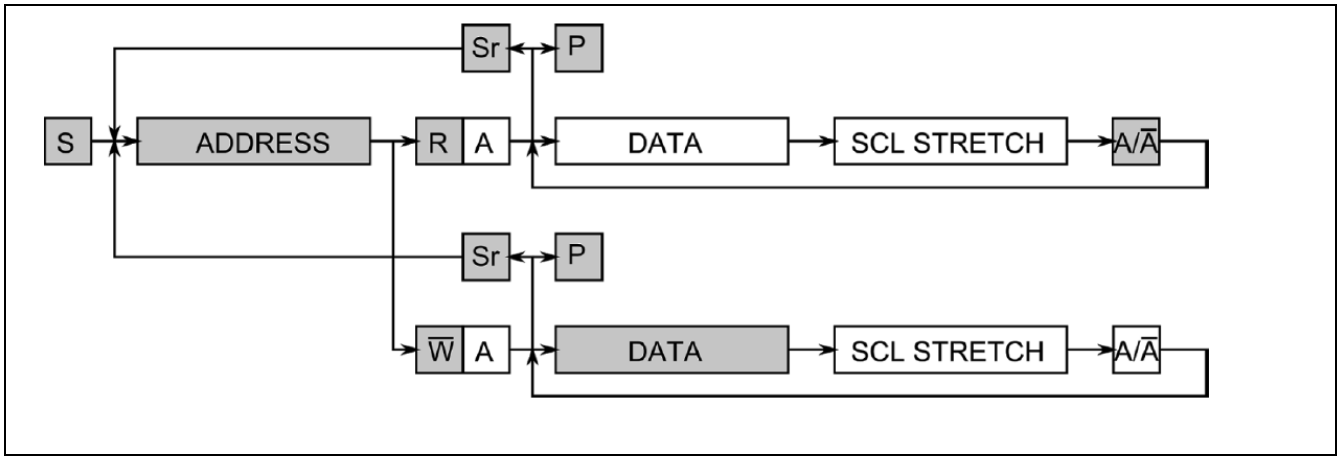

Figure 1-1 Reading and Writing Diagram

For more information about the CH101 module, please refer to the datasheet and application manual at the following website:

https://invensense.tdk.com/products/ch101/#documentation

1.2 Demo Result

The demo result is shown below.

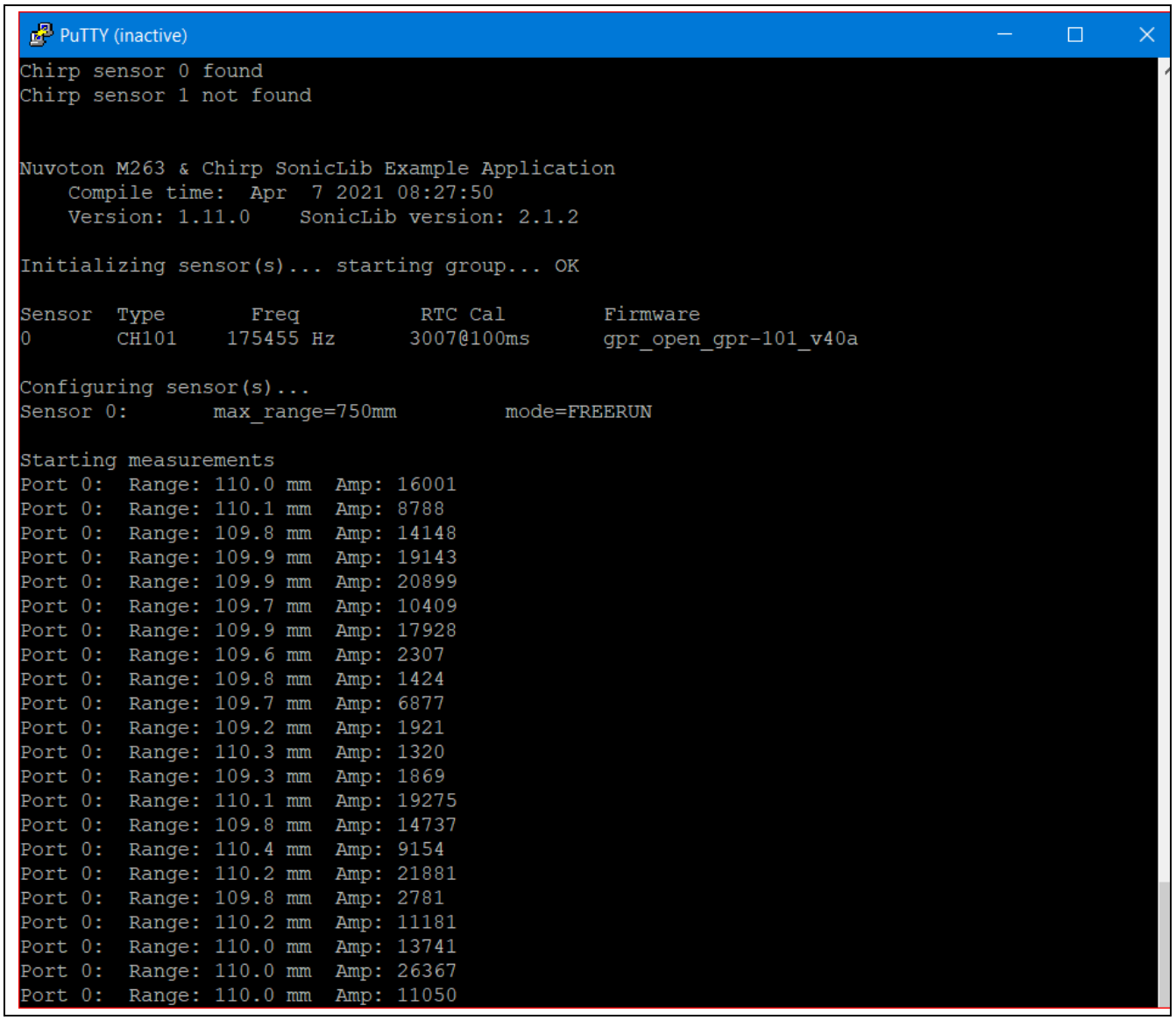

Figure 1-2 Demo Result

2 Code Description

Code initializes and finds whether the CH101 module is connected to the evaluation board.

```
void chbsp_board_init(ch_group_t *grp_ptr) {

    /* Make local copy of group pointer */
    sensor_group_ptr = grp_ptr;

    /* Initialize group descriptor */
    grp_ptr->num_ports = CHBSP_MAX_DEVICES;
    grp_ptr->num_i2c_buses = CHBSP_NUM_I2C_BUSES;
    grp_ptr->rtc_cal_pulse_ms = CHBSP_RTC_CAL_PULSE_MS;

   /* Unlock protected registers */
   SYS_UnlockReg();
   /* Init System, IP clock and multi-function I/O. */
   SYS_Init();
   /* Lock protected registers */
   SYS_LockReg();

   /* Init I2C0 */
   I2C0_Init();

   /* Configure UART0: 921600, 8-bit word, no parity bit, 1 stop bit. */
   UART_Open(UART0, 921600);

    ext_int_init();

    /* Probe I2C bus to find connected sensor(s) */
    find_sensors();
}
```

Software sets the module to CH_MODE_FREERUN mode. The RTC inside the module generates a time interval of 100 ms, and informs the M263 through INT pin to read the value of the relevant register through I^2C.

```
/*
   * handle_data_ready() - get and display data from all sensors
   *
   * This routine is called from the main() loop after all sensors have
   * interrupted. It shows how to read the sensor data once a measurement is
   * complete.  This routine always reads out the range and amplitude, and
```

```
 * optionally will read out the amplitude data or raw I/Q for all samples
 * in the measurement.
 *
 * See the comments in app_config.h for information about the amplitude data
 * and I/Q readout build options.
 *
 */
static uint8_t handle_data_ready(ch_group_t *grp_ptr) {
  uint8_t  dev_num;
  int      num_samples = 0;
  uint8_t  ret_val = 0;

  /* Read and display data from each connected sensor
   *   This loop will write the sensor data to this application's "chirp_data"
   *   array.  Each sensor has a separate chirp_data_t structure in that
   *   array, so the device number is used as an index.
   */

  for (dev_num = 0; dev_num < ch_get_num_ports(grp_ptr); dev_num++) {
      ch_dev_t *dev_ptr = ch_get_dev_ptr(grp_ptr, dev_num);

      if (ch_sensor_is_connected(dev_ptr)) {

          /* Get measurement results from each connected sensor
           *   For sensor in transmit/receive mode, report one-way echo
           *   distance,  For sensor(s) in receive-only mode, report direct
           *   one-way distance from transmitting sensor
           */

          if (ch_get_mode(dev_ptr) == CH_MODE_TRIGGERED_RX_ONLY) {
              chirp_data[dev_num].range = ch_get_range(dev_ptr,
                                                       CH_RANGE_DIRECT);
          } else {
              chirp_data[dev_num].range = ch_get_range(dev_ptr,
                                                       CH_RANGE_ECHO_ONE_WAY);
          }

          if (chirp_data[dev_num].range == CH_NO_TARGET) {
              /* No target object was detected - no range value */

              chirp_data[dev_num].amplitude = 0;  /* no updated amplitude */
```

```
                printf("Port %d:          no target found        ", dev_num);

            } else {
                /* Target object was successfully detected (range available) */

                 /* Get the new amplitude value - it's only updated if range
                  * was successfully measured.  */
                chirp_data[dev_num].amplitude = ch_get_amplitude(dev_ptr);

                printf("Port %d:  Range: %0.1f mm  Amp: %u  ", dev_num,
                        (float) chirp_data[dev_num].range/32.0f,
                        chirp_data[dev_num].amplitude);
            }

            /* Store number of active samples in this measurement */
            num_samples = ch_get_num_samples(dev_ptr);
            chirp_data[dev_num].num_samples = num_samples;

            /* Optionally read amplitude values for all samples */
#ifdef READ_AMPLITUDE_DATA
            uint16_t     start_sample = 0;
            ch_get_amplitude_data(dev_ptr, chirp_data[dev_num].amp_data,
                                  start_sample, num_samples, CH_IO_MODE_BLOCK);

#ifdef OUTPUT_AMPLITUDE_DATA
            printf("\n");
            for (uint8_t count = 0; count < num_samples; count++) {

                printf("%d\n",  chirp_data[dev_num].amp_data[count]);
            }
#endif
#endif

            /* Optionally read raw I/Q values for all samples */
#ifdef READ_IQ_DATA
            display_iq_data(dev_ptr);
#endif
            /* If more than 2 sensors, put each on its own line */
            if (num_connected_sensors > 2) {
                printf("\n");
```

```
                }
            }
        }
        printf("\n");

        return ret_val;
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M261_Series_BSP_CMSIS_V3.00.003
- IDE version
  - Keil uVersion 5.33

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M263KI
  - EV_MOD_CH101-03-01
- Pin Connect

The hardware uses M263KI I²C interface to connect MOD_CH101 sensor module.

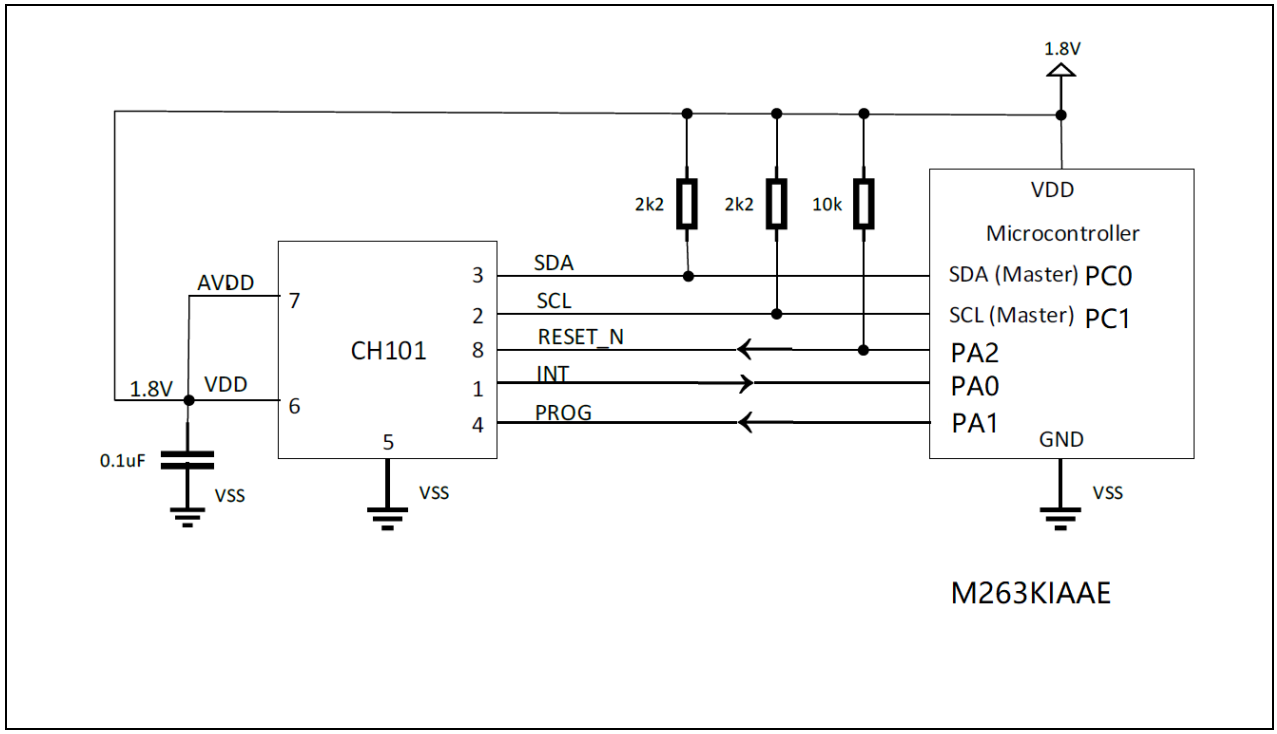

Figure 3-1 Pin Connect

The operating voltage of CH101 module is 1.8V, so the power supply voltage of NuMaker-M263KI evaluation board for the M263 must be changed to 1.8V. The resistance of the circuit board can be changed to make Nulink2-Me output 1.8V voltage.

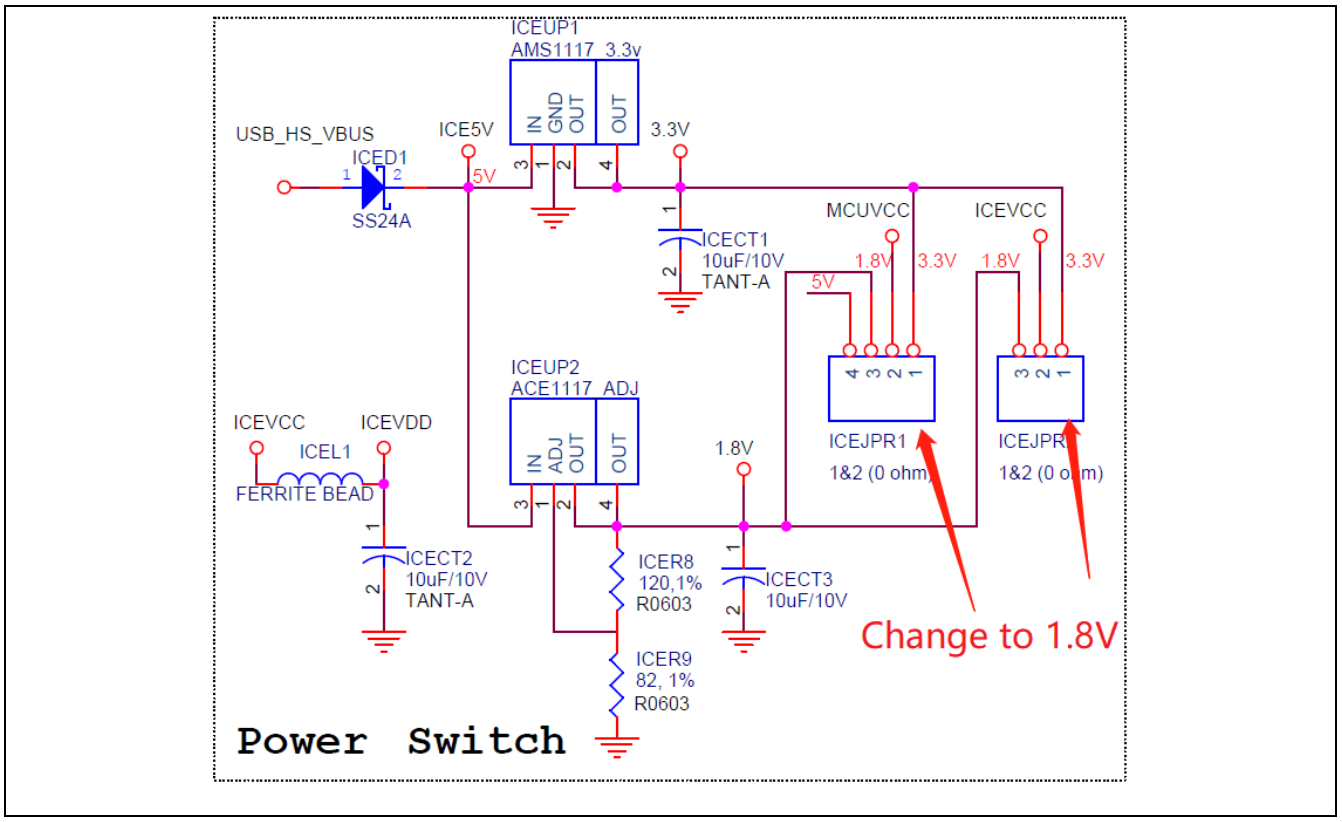

Figure 3-2 Power switch

4 Directory Information

The directory structure is shown below.

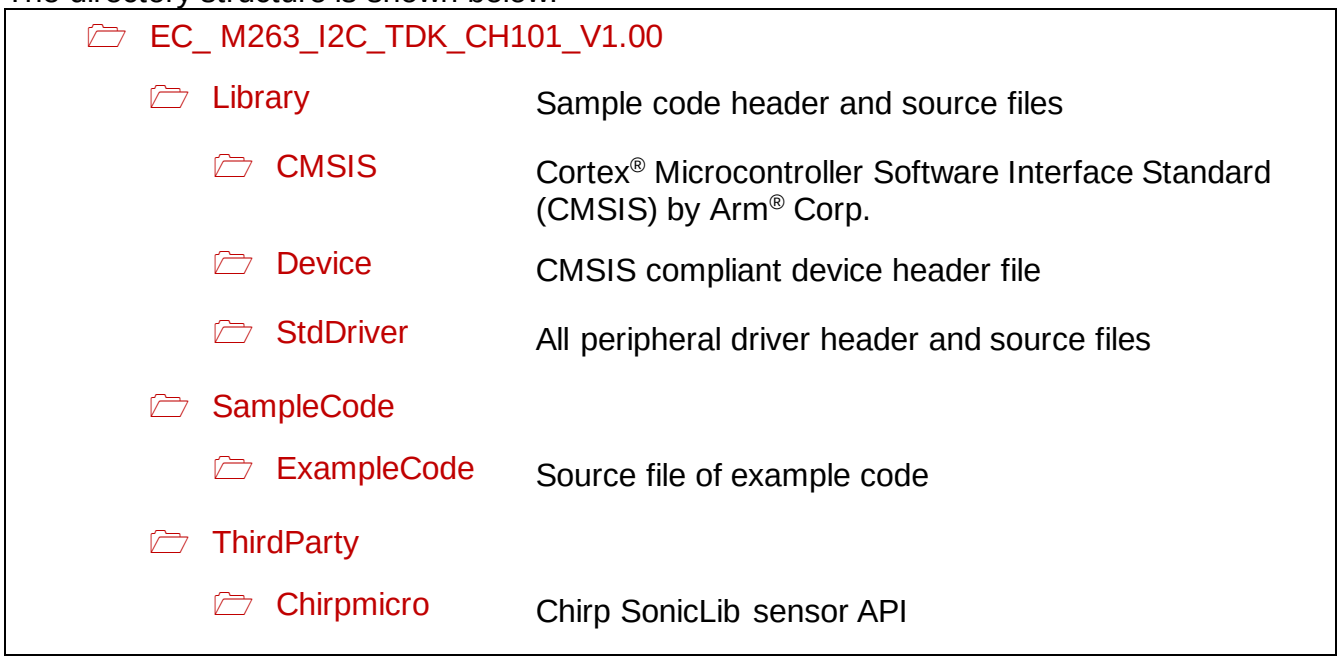

Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M263_I2C_TDK_CH101.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2021.05.06	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.