Customized Touch Key Protocol via I²C on ML56

Example Code Introduction for 8-bit NuMicro® Family

Information

Application	This sample code uses a customized touch key protocol to communicate with the master device through I²C interface on the ML56 series microcontroller (MCU).
BSP Version	ML51_ML54_ML56 Series BSP V2.00.001
Hardware	ML56 Touch Key Daughter Board Ver1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Overview

This sample code uses a customized touch key protocol to communicate with the master device through I^{2}C interface. The ML56 touch key daughter board is a sensing board with two touch keys, a slider and a wheel as shown in Figure 1-1. All of the sensors on this board are capacitive touch key sensing. When a finger touches any of the pads on the PCB, the current position or key value can be reported by detecting the measurement of the capacitance changes.
The slider and wheel support the resolution of 10, 20, 50, and 100 and the calibration parameters are shown in Figure 1-2. In this case, the slider and wheel with the resolution of 100 should report a position in the range of 0 -100. Please refer to NuMaker ML56SD Touch Key Design Guide for more details of touch key calibration steps.

Figure 1-1 ML56 Daughter Board

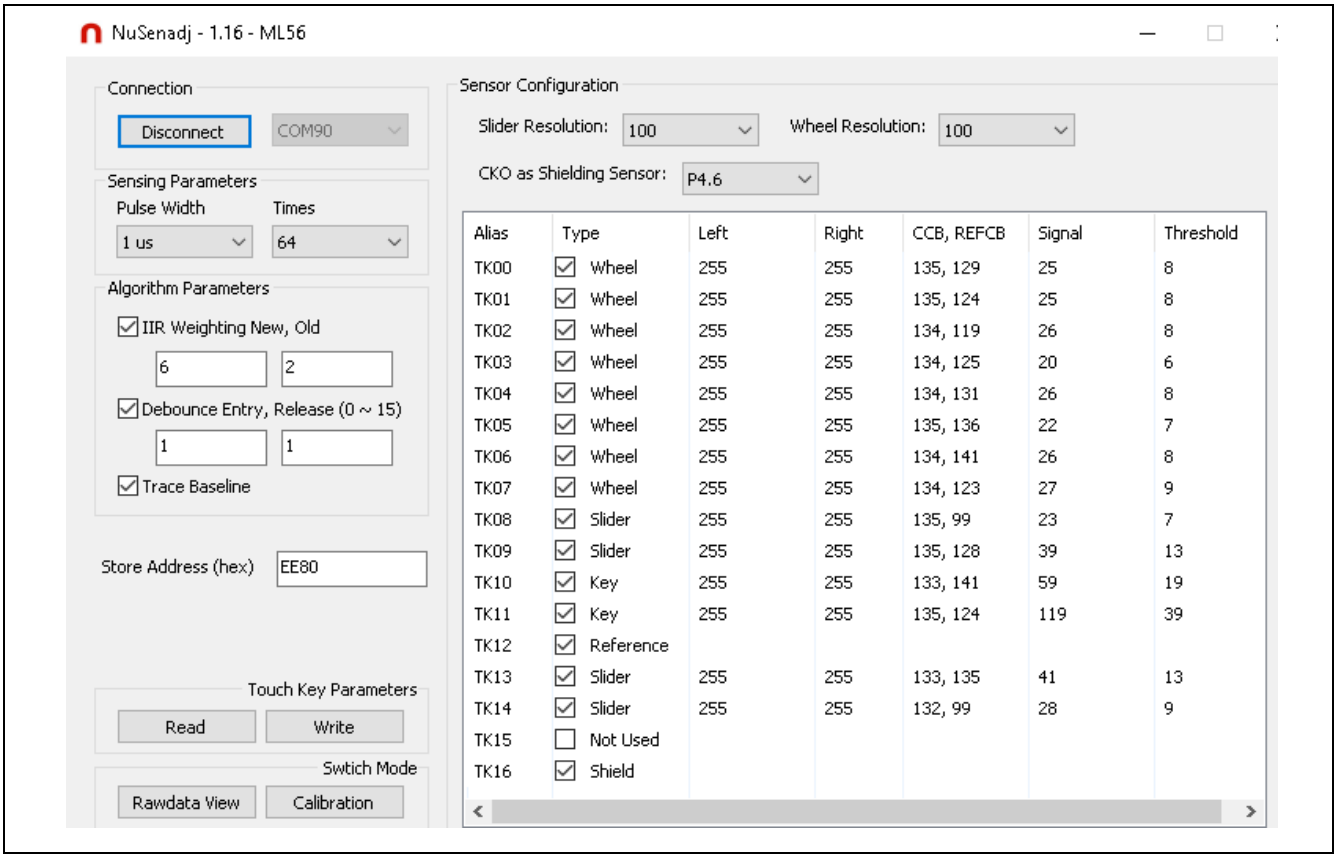

Figure 1-2 Calibration Parameters of ML56 Daughter Board

1.1 ML56 Daughter Board Overview

Figure 1-3 shows the ML56 daughter board profile and connector overview.

SWD Connector

NC	NC	NC	ICE_RX	ICE_RX
J1.2	J1.4	J1.6	J1.8	J14.10
J1.1	J1.3	J1.5	J1.7	J1.9
VCC	ICE_DAT	ICE_CLK	nRESET	GND

Extension GPIO Connector

GND	VCC	I2C_SDA	nRESET	NC	NC	NC	NC	NC	NC	NC	GND
CON3.2	CON1.22	CON1.20	CON1.18	CON1.16	CON1.14	CON1.12	CON1.10	CON1.8	CON1.6	CON1.4	CON1.2
CON3.1	CON1.21	CON1.19	CON1.17	CON1.15	CON1.13	CON1.11	CON1.9	CON1.7	CON1.5	CON1.3	CON1.1
GND	VCC	I2C_SCL	INT	NC	NC	NC	NC	NC	NC	NC	GND

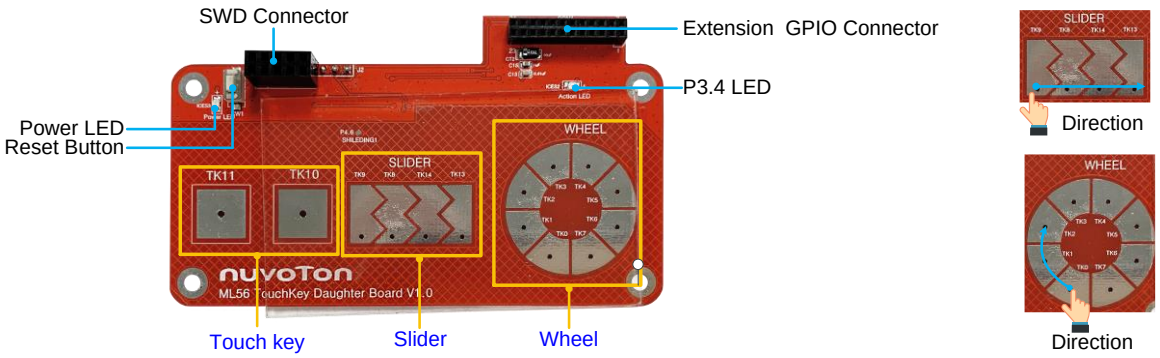

Figure 1-3 ML56 Daughter Board Connector and Function Overview

The following lists the function brief description:

- Touch keys (TK10, TK11)
- Slider (TK8, TK9, TK13, TK14)
  - The direction of slider is by TK9 to TK13
- Wheel (TK0, TK1, TK2, TK3, T4, TK5, TK6, TK7)
  - The direction of wheel is by TK0 to TK7
- SWD Connector (J1)
  - SWD host interface
- Extension GPIO Connector (CON1)
  - I^2C transmission interface
  - INT (P3.5)
  - nReset
- Status LED (ICES2)
  - Indicate the status of any touch events
- Power LED (ICES3)
  - Indicate the status of power source
- Reset Button (SW1)

1.2 Principle

1.2.1 Protocol Format

Figure 1-4 shows that the I^2C master reads data protocol and a communication consists of five parts:

- START (STA) or Repeated START (ST) signal generation
  - The master can initiate a transfer by sending a START signal; the START signal denotes the beginning of a new data transmission. Instead of sending the stop

condition, it is also allowed to send repeated start condition followed by slave address and an R/W bit.

- Slave address and R/W bit transfer
  - After START or Repeated START condition, the master sends a 7-bit slave address to identify the target device of the communication. After an address byte, a slave sensitive to the transmitted address has to acknowledge the reception.
- Command address transfer
  - After having sent the slave address byte, the master may send 1 byte command address. Please refer to section 1.2.2. The master keeps transmitting data after the slave returns acknowledge to the master
- Data read
  - A master addresses a slave with a 7-bit address and 1-bit read index to denote that the master wants to read data from the slave. The slave will start transmitting data after the slave returns acknowledge to the master.
- STOP (STO) signal generation
  - The master can terminate the communication by generating a STOP signal.

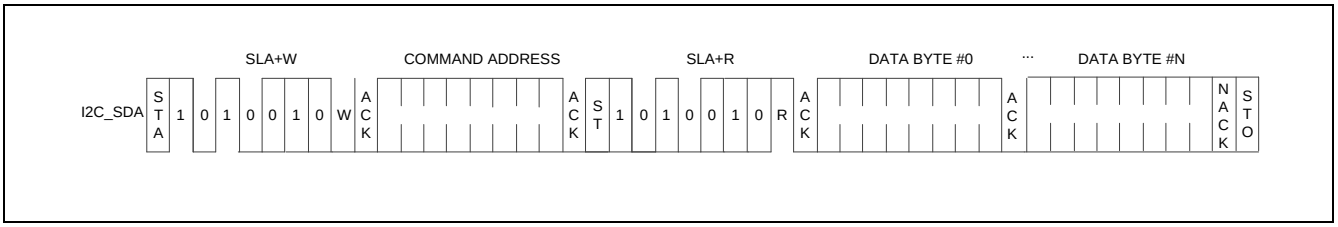

Figure 1-4 Master Reads Data Protocol Format

1.2.2 Customized Touch Key Protocol

The touch key commands are listed in Table 1-1. The commands are used for the master device to request the touch key information on the ML56 daughter board.

Address	Bits	Description
0x00	[7:0]	Reports the number of touch keys. In this sample code, the field is fixed to 2.
0x01	[7:0]	Reports the number of sliders.

		In this sample code, the field is fixed to 1.
0x02	[7:0]	Reports the slider resolution. It supports the resolution of 10, 20, 50, and 100. In this sample code, the field is fixed to 100.
0x03	[7:0]	Reports the number of wheels. In this sample code, the field is fixed to 1.
0x04	[7:0]	Reports wheel resolution. It supports the resolution of 10, 20, 50, and 100. In this sample code, the field is fixed to 100.
0x10	[7:0]	Report the touch key value. Each bit in the bitmap corresponds to each touch key. If the bit is set, it means the touched corresponding touch key. If the bit is cleared, it means the untouched corresponding touch key. For example, when TK10 is touched, the value of bit 3 for address 0x11 is set. In other words, TK10 is untouched, the value of bit 3 for address 0x11 is cleared.
0x11	[7:0]	
0x12	[7:0]	
0x13	[7:0]	
0x14	[7:0]	
0x15	[7:0]	
0x16	[7:0]	
0x17	[7:0]	
0x18	[7:0]	
0x19	[7:0]	
0x1A	[7:0]	
0x1B	[7:0]	
0x1C	[7]	0 = Slider0 event didn’t occur. 1 = Slider0 event occurred.
	[6:0]	Reports the position of slider0 in the range of 0 to ***slider resolution.*** **Note:** The address of slider resolution is 0x02.
0x1D	[7]	0 = Slider1 event didn’t occur. 1 = Slider1 event occurred.

	[6:0]	Reports the position of slider1 in the range of 0 to ***slider resolution.*** **Note:** The address of slider resolution is 0x02.
0x1E	[7]	0 = Wheel0 event didn't occur. 1 = Wheel0 event occurred.
	[6:0]	Reports the position of wheel0 in the range of 0 to ***wheel resolution***. **Note:** The address of wheel resolution is 0x04.
0x1F	[7]	0 = Wheel1 event didn't occur. 1 = Wheel1 event occurred.
	[6:0]	Reports the position of wheel1 in the range of 0 to ***wheel resolution***. **Note:** The address of wheel resolution is 0x04.

Table 1-1 Customized Touch Key Commands

1.3 Demo Result

The demo result is shown in Figure 1-5.

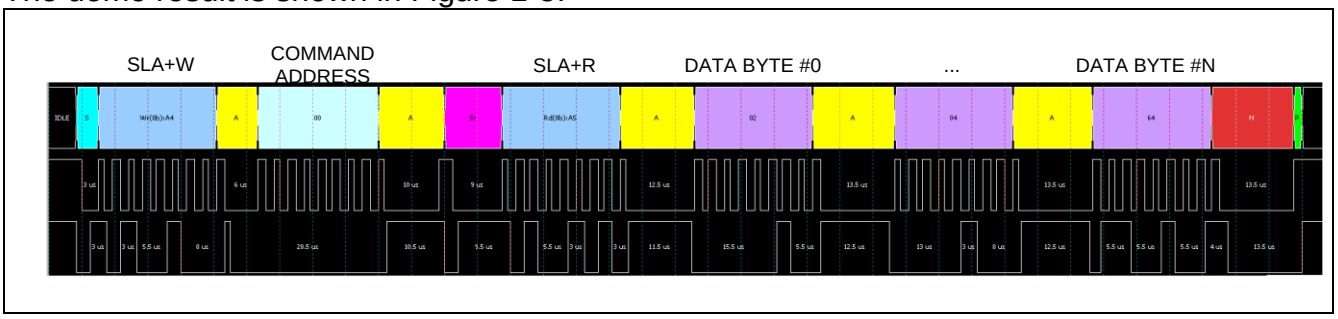

Figure 1-5 Demo Result

2 Code Description

The instructions are listed below:

1. In while-loop, call TK_RawDataView() function to wait until a touch event occurs.

2. Keep INT pin as low until master device returns NACK. The INT pin is used to trigger the master device to send the I²C clock to start transmission.

```
void main(void)
{

        do
        {
            /* Get touch key event */
            g_u8MainState = TK_RawDataView();

            switch (g_u8MainState)
            {
                case eMAIN_APP_IDLE_STATE:
                default:
                    break;

                case eMAIN_APP_TK_PRESS_STATE:
                    i8TKBak = i8TK;
                    /* If the bit is set, it means the pressed corresponding touch key */
                    SetTkChannel(i8TK);
                    /* Toggle status LED(ICES2) */
                    TK_lightLEDToggle();
                    /* INT pin is active low.
                    It is used to trigger master device to send the clock to start the
                    transmission. */
                    P35 = 0;
                    break;

                case eMAIN_APP_TK_RELEASE_STATE:
                    /* If the bit is clear, it means the released corresponding touch key
                    */
                    ClrTkChannel(i8TKBak);
                    i8TKBak = -1;
                    /* Toggle status LED(ICES2) */
                    TK_lightLEDToggle();
                    /* INT pin is active low.
                    It is used to trigger master device to send the clock to start the
                     transmission. */
                    P35 = 0;
                    break;

                case eMAIN_APP_SLIDER_STATE:
                    /* bit[7] is set, it means the slider event occurred */
                    /* bit[6~0] is filled the position of slider */
                    SetSlider0Persent(i8SliderPercentage);
                    /* Toggle status LED(ICES2) */
                    TK_lightLEDToggle();
                    /* INT pin is active low.
                    It is used to trigger master device to send the clock to start the
                    transmission. */
                    P35 = 0;
                    break;

                case eMAIN_APP_SLIDER_RELEASE_STATE:
                    /* bit[7] is clear, it means the slider event didn't occurred */
```

```
                    /* bit[6~0] is filled the position of slider */
                    ClrSlider0Persent();
                    /* Toggle status LED(ICES2) */
                    TK_lightLEDToggle();
                    /* INT pin is active low.
                    It is used to trigger master device to send the clock to start the
                    transmission. */
                    P35 = 0;
                    break;

                case eMAIN_APP_WHEEL_STATE:
                    /* bit[7] is set, it means the wheel event occurred */
                    /* bit[6~0] is filled the position of wheel */
                    SetWheel0Persent(i8WheelPercentage);
                    /* Toggle status LED(ICES2) */
                    TK_lightLEDToggle();
                    /* INT pin is active low.
                    It is used to trigger master device to send the clock to start the
                    transmission. */
                    P35 = 0;
                    break;

                case    eMAIN_APP_WHEEL_RELEASE_STATE:
                    /* bit[7] is clear, it means the wheel event didn't occurred */
                    /* bit[6~0] is filled the position of wheel */
                    ClrWheel0Persent();
                    /* Toggle status LED(ICES2) */
                    TK_lightLEDToggle();
                    /* INT pin is active low.
                    It is used to trigger master device to send the clock to start the
                    transmission. */
                    P35 = 0;
                    break;

            }

            g_u8MainState = eMAIN_APP_IDLE_STATE;
        } while (1);
}
```

```
void I2C0_ISR(void) interrupt 6
{
    …
    switch (I2C0STAT)
    {
         …

        /* Slave Transmit Data NACK */
        case 0xC0:
            set_I2C0CON_AA;
            /* INT pin is idle high */
            P35 = 1;
            break;
         …

    }
    …
}
```

3. When the TK_RawDataView() function is called, it will get the slider/wheel position and touch key value. The scan rate is 20 ms by default.

```
unsigned char TK_RawDataView(void)
```

```
{
    int8_t i8Ret = 0;
    xdata int8_t ai8Signal[TKLIB_TOL_NUM_KEY];
    int8_t ai8TmpSignal[TKLIB_TOL_NUM_KEY];

    if (u8EventKeyScan == 1)
    {
        u8EventKeyScan = 0;
        /**
          * TK_ScanKey() scan all enable key, slider and wheel channels.
          * i8Ret : Key/slider/wheel channel with max amplitude. -1: means no any key's
                    amplitude over the key's threshold.
          * ai8Signal[]: The buffer size is equal to the ML56 TK channels. It reports the
                        signal amplitude on this round
          */
        i8Ret = TK_ScanKey(&ai8Signal[0]);

        if (i8Ret == 10 || i8Ret == 11)
        {
            /* TK# is touched */
            if (i8TK != i8Ret)
            {
                u8TKPressFlag = 1;
                i8TK = i8Ret;
                return eMAIN_APP_TK_PRESS_STATE;
            }
        }
        else
        {

            /* TK# is untouched */
            i8TK = i8Ret;
            if(u8TKPressFlag == 1)
            {
               u8TKPressFlag = 0;
               return eMAIN_APP_TK_RELEASE_STATE;
            }

        }

#if defined(OPT_SLIDER)
        {
            /** To save buffer size, re-used the ai8Signal[] buffer */
            uint16_t u16ChnMsk;
            uint8_t u8Count = 0;

            u16ChnMsk = TK_GetEnabledChannelMask(TK_SLIDER);

            if (TK_CheckSliderWheelPressed(TK_SLIDER) == 1)
            {
                /* Sort sequence based on the direction of slider is according to the
                   hardware layout. */
                ai8TmpSignal[0] = ai8Signal[9];
                ai8TmpSignal[1] = ai8Signal[8];
                ai8TmpSignal[2] = ai8Signal[14];
                ai8TmpSignal[3] = ai8Signal[13];
                u8Count = 4;
                i8SliderPercentage = TK_SliderPercentage(ai8TmpSignal, u8Count);
                u8SliderPressFlag = 1;
                return eMAIN_APP_SLIDER_STATE;
            }
            else
            {
                 if(u8SliderPressFlag == 1)
                 {
                     u8SliderPressFlag = 0;
```

```
                    return eMAIN_APP_SLIDER_RELEASE_STATE;
                }
            }
        }
#endif
#if defined(OPT_WHEEL)
        {
            /** To save buffer size, re-used the ai8Signal[] buffer */
            uint32_t u32ChnMsk;
            uint8_t u8Count = 0, i;

            u32ChnMsk = TK_GetEnabledChannelMask(TK_WHEEL);

            if (TK_CheckSliderWheelPressed(TK_WHEEL)  == 1)
            {
                /* Sort sequence based on the direction of wheel is according to the
                   hardware layout. */
                for (i = 0; i < TKLIB_TOL_NUM_KEY ; i++)
                {
                    if (u32ChnMsk & (1ul << i))
                    {
                        ai8TmpSignal[u8Count] = ai8Signal[i];
                        u8Count = u8Count + 1;
                    }
                }

                i8WheelPercentage = TK_WheelPercentage(ai8TmpSignal, u8Count);
                u8WheelPressFlag = 1;
                return eMAIN_APP_WHEEL_STATE;
            }
            else
            {
                if(u8WheelPressFlag == 1)
                {
                    u8WheelPressFlag = 0;
                    return eMAIN_APP_WHEEL_RELEASE_STATE;
                }
            }
        }
#endif
    }

    return 0;
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - ML51_ML54_ML56 Series BSP v2.00.001
- IDE version
  - Keil uVersion 5.26

3.2 Hardware Requirements

- Circuit components
  - ML56 Daughter Board Ver1.0

- Pin Connect

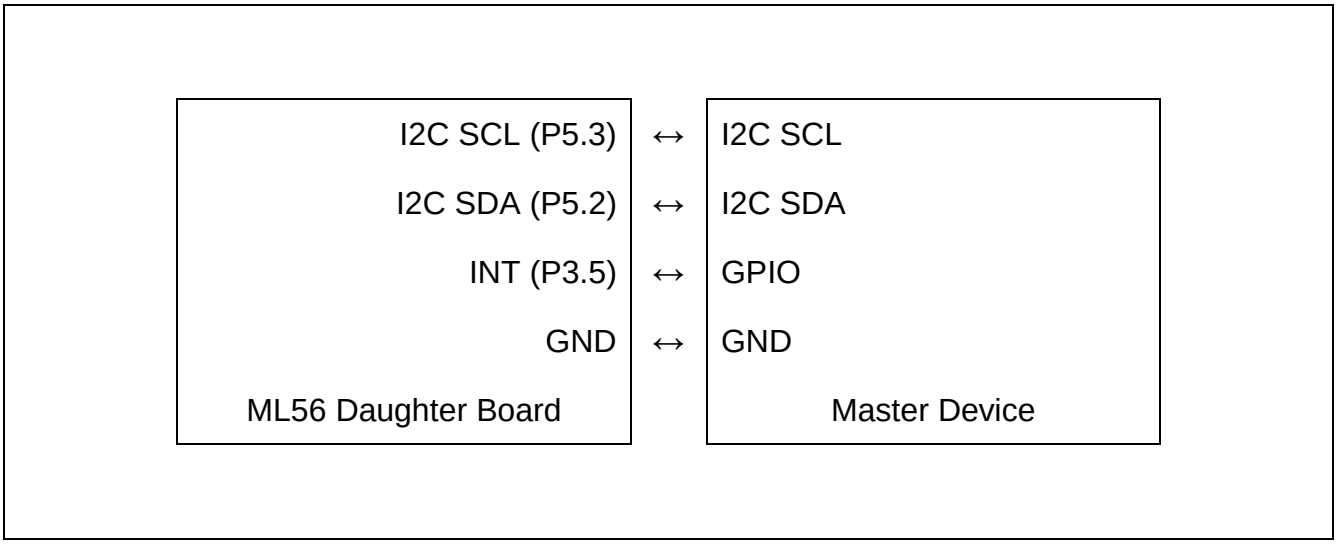

Figure 3-1 Pin Connect

4 Directory Information

The directory structure is shown in Figure 4-1.

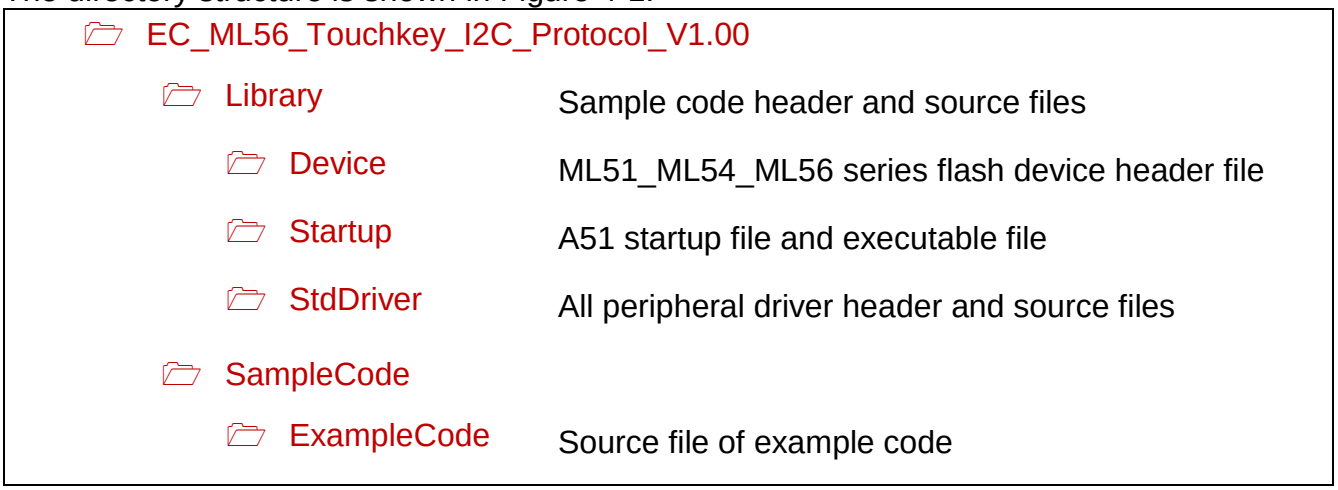

Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *ML56_Touchkey_I2C_Protocol.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2022.02.28	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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.