XT32H05x

XT32 microcontroller I2C

Application notes

Rev 0.0.0

Original Release Date: 28-SEP-2023

Revised :

Revision History

Release	Date	Author	Summary of Change
V0.0.0	28/09/2023	Shirling Liu	Initial

Contents

List of Figures

List of Tables

1 Introduce

This application note serves as a comprehensive guide for software developers, offering essential information on Inter-integrated circuit (I2C). It covers fundamental concepts and provides guidelines to ensure proper utilization of I2C in software development projects. Whether you're a beginner or an experienced developer, this document will equip you with the necessary knowledge and best practices to effectively configure and utilize I2Cs in your applications.

1.1 Required peripherals

This application involves modules as table 1.

Table 1. Modules in example

Sub-module	Peripheral use	Note
PADI	2 ports as I2C clock port and data port	Call HAL_PADI_Init() in code
I2C1	Pin17 as I2CSCLK, Pin18 as I2CSDA	Set as master
DMA	DMA Handshaking Interface:	**Only for DMA mode

1.2 Compatible devices

This example is compatible with the devices in Table 2.

Table 2. Device list

Product	EVB
XT32H050	XB002823

2 Design description

2.1 Feature overview

XT32H0xxx provides 2 I2C peripherals: I2C1, I2C2. The I2C software provides the following features:

- Slave and master modes
- Standard mode (up to 100Kbps)
- Fast mode (up to 400Kbps)
- Fast mode plus (up to 1Mbps)
- High speed mode (up to 3.4Mbps)
- 7-bit and 10-bit addressing mode
- Programmable setup and hold times
- Programmable digital noise filter
- DMA capability
- 8-bytes buffer

2.2 Design steps

1. Enable I2C1 source clock and set clock divider.
2. Configure pin alternate function as I2C from Peripheral PADI through PADI_InitTypeDef structure. This example uses I2C1 as host to drive the RGB sensor TCS34725.

   - PADI_IDX_IO13_I2C1_SCK, means select and enable the IO13(pin 17).
   - PADI_CFG_IO13_I2C1_SCK, means select I2C0SCLK function for IO13.
   - PADI_IDX_IO14_I2C1_SDA, means select and enable the IO14(pin 18).
   - PADI_CFG_IO14_I2C1_SDA, means select I2C0SCLK function for IO14.

   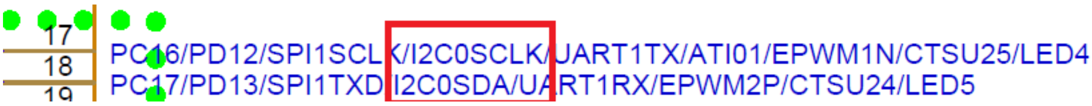
   17
18
19
PC16/PD12/SPI1SCLK/I2C0SCLK/UART1TX/ATI01/EPWM1N/CTSU25/LED4
PC17/PD13/SPI1TXD/I2C0SDA/UART1RX/EPWM2P/CTSU24/LED5

   Figure 1. IO function selection as I2C1

   Note: please refer to XT32H0xxB—reference manual document to find the assignment relationship between pin with IO.

3. Configure parameters for I2C1 module.
4. Assign system DMA handshaking interface to I2C1_RX & I2C1_TX, link DMA handle with I2C1 handle, and enable DMA1_IRQn interrupt configuration if the example under DMA mode.
5. Process to read/write data with external devices.

2.3 Design considerations

2.4 Software flowchart

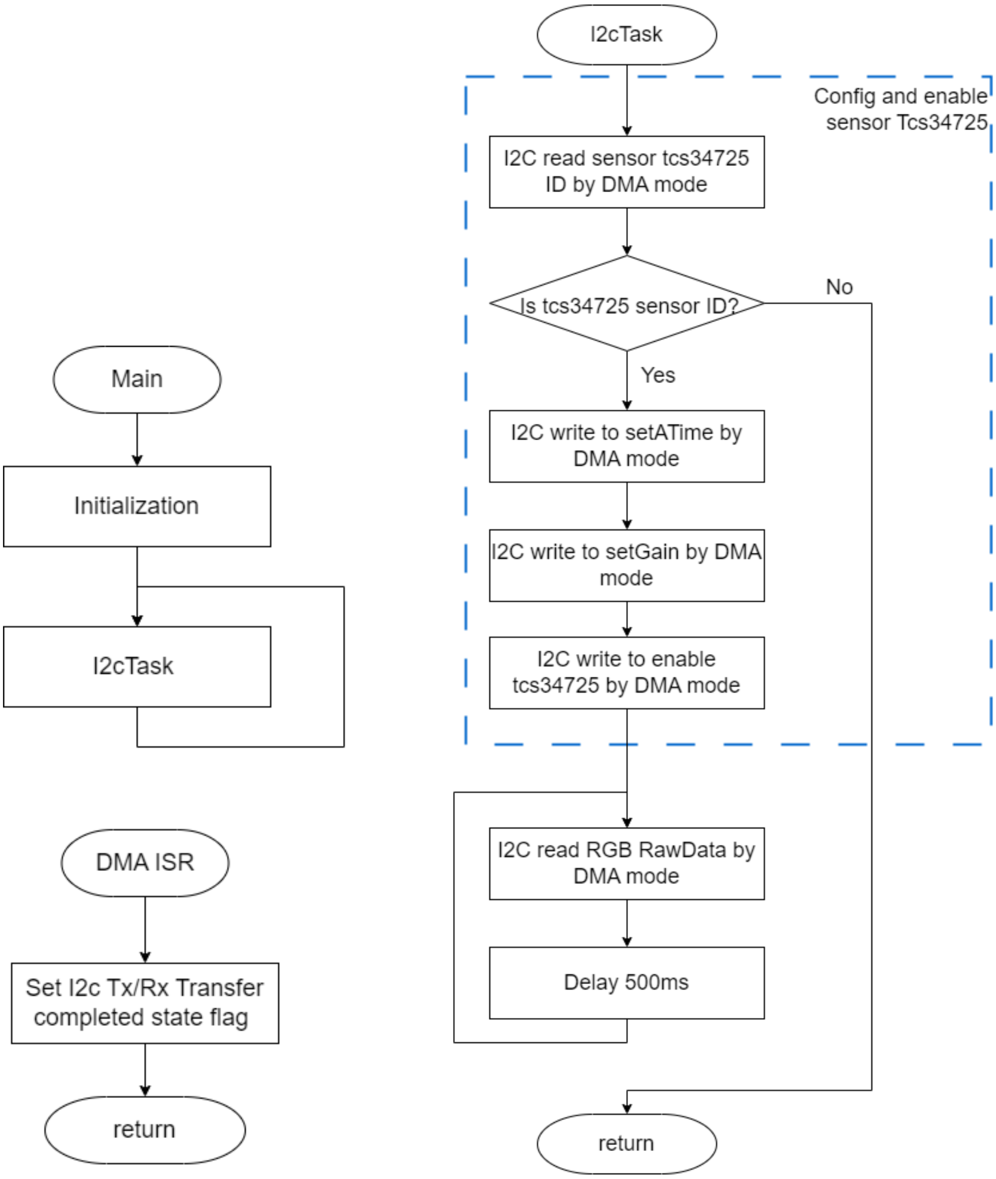
I2cTask
Config and enable sensor Tcs34725
I2C read sensor tcs34725 ID by DMA mode
Is tcs34725 sensor ID?
No
Yes
Main
Initialization
I2cTask
I2C write to setATime by DMA mode
I2C write to setGain by DMA mode
I2C write to enable tcs34725 by DMA mode
DMA ISR
I2C read RGB RawData by DMA mode
Set I2c Tx/Rx Transfer completed state flag
Delay 500ms
return
return

Figure 2. Application flow—DMA mode

2.5 Reference code

Configure Peripheral PADI through PADI_InitTypeDef structure to select alternate function as I2C1 interface as bellowing code:

```
if(hi2c->Instance==I2C1)
{
  /**mapping pad  I2C1 GPIO Configuration ***
        IO13/28/38/49/54/     ------> I2C1 SCK
        IO14/29/39/50/55     ------> I2C1 SDA
        */
    XT_IO_Option_Assigned(EVB_I2C1_SCK_IO_IDX,  EVB_I2C1_SCK_IO_CFG, PADI_PULLUP);
    XT_IO_Option_Assigned(EVB_I2C1_SDA_IO_IDX,  EVB_I2C1_SDA_IO_CFG, PADI_PULLUP);
}
```

If using DMA mode to transfer serial data, should assign system DMA handshaking interface to I2C1 as below code:

```
  //DMA ModeDMA Handshaking Interface:INDEX 8 and INDEX 9 of CFG1 I2C1 TX RX cfg0 
for I2C1
  HAL_I2C_DMAHSIFConfig(hi2c, &hI2c1Dmarx, &hI2c1Dmatx,
              I2C1_RXD_DMAHSIF_IDX,I2C1_TXD_DMAHSIF_IDX, I2C1_RXD_DMAHSIF_CFG,
               I2C1_TXD_DMAHSIF_CFG);
  HAL_I2C_LinkDMA(hi2c, &hI2c1Dmarx, &hI2c1Dmatx);
```

Configure Peripheral I2C1 :

```
{
     hI2c1.Instance = I2C1;
    hI2c1.Init.SlaveAddress    = DEVICE_TCS34725_ADDRESS
    hI2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    hI2c1.Init.OwnAddress      = XT32HX_I2C_OWNER_ADDR;
    hI2c1.Init.Speed           = I2C_SPEED_STANDARD;
    hI2c1.Init.Baudrate       = 100000;
    hI2c1.Mode                = HAL_I2C_MODE_MASTER;
    if (HAL_I2C_Init(&hI2c1) != HAL_OK)
    {
      /* Error_Handle */
    }
}
```

XT_I2c_Task handles the basic transfer process.

```
void XT_I2c_Task(void)
{
  /* USER CODE */
    TCS34725_RGBdataDef sRGBda;
    uint8_t deviceenable = FALSE;
    deviceenable = XT_I2cTcs34725_Enable();

    while(deviceenable)
    {
        XT_I2cTcs34725_getRawData(&sRGBda);
        HAL_Delay(500); //ms
    }
}
```

I2C1 write reg to drive sensor Tcs34725 function:

```
static void XT_I2cTcs34725_Write8(uint8_t reg, uint32_t value)
{
  uint8_t txbuff[8] = {0};
  uint8_t length = 0;

  txbuff[length++] = TCS34725_COMMAND_BIT |reg;
  txbuff[length++] = value & 0xFF;


if(HAL_I2C_Master_Transmit_DMA(&hI2c1,(uint16_t)(DEVICE_TCS34725_ADDRESS),txbuff,le
ngth) != HAL_OK)
  {
    if (HAL_I2C_GetError(&hI2c1) != HAL_I2C_ERROR_NONE)
     {
        Error_Handle();
     }
  }
  XT_I2c_Checksta(CB_I2C1_TXFNSH);
  HAL_Delay(3); //ms
  return;
}
```

I2C1 read reg from sensor Tcs34725:

```
static uint16_t XT_I2cTcs34725_Read16(uint8_t reg)
{
  uint8_t rxbuff[8] = {0};
  uint8_t txbuff = TCS34725_COMMAND_BIT |reg;

  HAL_I2C_Master_Transmit_DMA(&hI2c1,(uint16_t)(DEVICE_TCS34725_ADDRESS),&txbuff,1);
   XT_I2c_Checksta(CB_I2C1_TXFNSH);
  if(HAL_I2C_Master_Receive_DMA(&hI2c1,(uint16_t)(DEVICE_TCS34725_ADDRESS),(uint8_t *)rxbuff,2) != HAL_OK)
  {
     if (HAL_I2C_GetError(&hI2c1) != HAL_I2C_ERROR_NONE)
     {
        Error_Handle();
     }
  }
  XT_I2c_Checksta(CB_I2C1_RXFNSH);
  HAL_Delay(3); //ms
  return ((rxbuff[0]<<8) |rxbuff[0]);
}
```

2.6 Additional resources

- XT32H0xxB--reference manual
- XT32H0xxB--dma-AN230800