# XT32H05x

# XT32 microcontroller SPI

# Application notes

Rev 0.0.0

**Original Release Date: 12-Sep-2023**

**Revised :**

# Revision History

| Release | Date | Author | Summary of Change |
|---|---|---|---|
| V0.0.0 | 12/09/2023 | 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 Serial Peripheral Interface (SPI). It covers fundamental concepts and provides guidelines to ensure proper utilization of SPI 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 SPIs 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 | 4 ports as SPI transmitter port and receiver port | Call HAL_PADI_Init() in code |
| SPI1 | Pin26 as SPI1SCB, Pin27 as SPI1TXD<br>Pin29 as SPI1CLK, Pin28 as SPI1RXD | Set as master |
| SPI2 | Pin26 as SPI1SCB, Pin27 as SPI1TXD<br>Pin29 as SPI1CLK, Pin28 as SPI1RXD | Set as slaver |

### 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 SPI peripherals: SPI1, SPI2. SPI1 and SPI2 both can be used for master or slaver with full duplex serial communication with external devices. The SPI software provides the following features:

- Support 4 to 16 bits serial data transfer
- 16 bits width and 8 entries depth of transmit/receive FIFO buffer
- Support DMA transfer
- Generate interrupts
- Multi-master contention detection
- programmable serial clock polarity and phase

- Choice of Motorola SPI and Texas Instrument Synchronous Serial Protocol

## 2.2 Design steps

1. Enable SPI1/SPI2 Baudrate source clock and set clock divider.
2. Configure pin alternate function as SPI from Peripheral PADI through PADI_InitTypeDef structure. This example uses SPI1 as master, SPI2 as slaver.
   - PADI_IDX_IO23_SPI1_TXD, means select and enable the IO23(pin 27).
   - PADI_CFG_IO23_SPI1_TXD, means select SPI1 TX function for IO27.

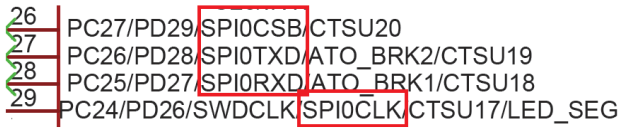



Figure 1. IO function selection as SPI1

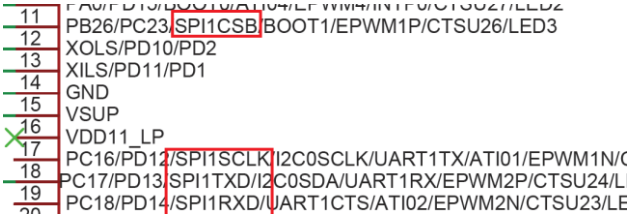


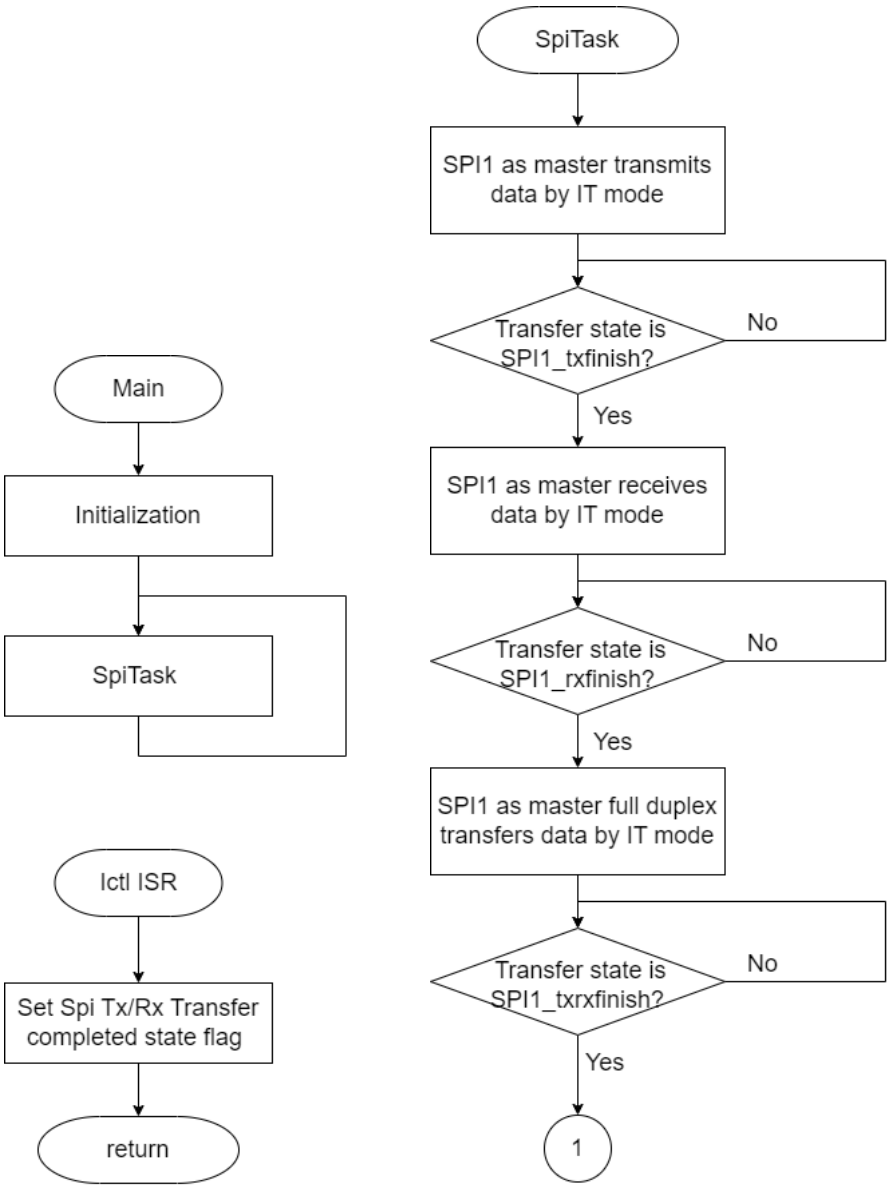

Figure 2. IO function selection as SPI2

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

3. Configure parameters for SPI1 module and SPI2 module.
4. Enable ICTL_IRQn interrupt configuration if the example under IT mode, else ignore this step.
5. Process to transfer serial data with external devices.

## 2.3 Design considerations

## 2.4 Software flowchart



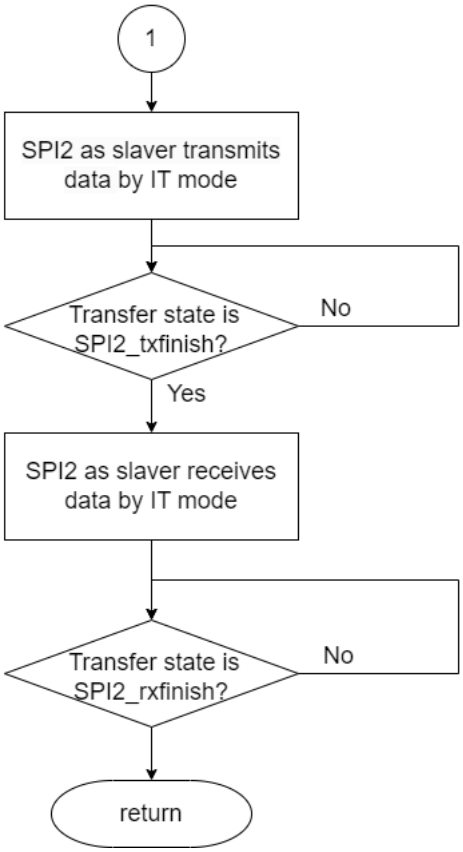



Figure 3. Application flow—IT mode

## 2.5 Reference code

Configure Peripheral PADI through PADI_InitTypeDef structure to select alternate function as SPI1 interface as bellowing code:

```
if((hspi->Instance== SPI1M)|| (hspi->Instance== SPI1S))
{
```

```
  /**mapping IO  SPI1 GPIO Configuration ***
       IO22 ------> SPI1_CS
       IO25 ------> SPI1_SCK
       IO24 ------> SPI1_MISO (RXD)
       IO23 ------> SPI1_MOSI (TXD)
       */    /* mapping IO */
  XT_IO_Option_Assigned(EVB_SPI1_CS_IO_IDX,EVB_SPI1_CS_IO_CFG,PADI_PULLDOWN);
  XT_IO_Option_Assigned(EVB_SPI1_TXD_IO_IDX,EVB_SPI1_TXD_IO_CFG,PADI_PULLUP);
  XT_IO_Option_Assigned(EVB_SPI1_RXD_IO_IDX,EVB_SPI1_RXD_IO_CFG,PADI_PULLUP);
  XT_IO_Option_Assigned(EVB_SPI1_CLK_IO_IDX,EVB_SPI1_CLK_IO_CFG,PADI_PULLDOWN);
}
```

Enable ICTL IRQ:

```
    HAL_NVIC_SetPriority(ICTL_IRQn, 2, 0);
    HAL_NVIC_EnableIRQ(ICTL_IRQn);
```

Configure Peripheral SPI1 :

```
    hSpi1master.Instance           = SPI1M;
    hSpi1master.Init.Direction     = SPI_DIRECTION_2LINES;
    hSpi1master.Init.DataSize      = SPI_DATASIZE_8BIT;
    hSpi1master.Init.ClockPolarity = SPI_POLARITY_LOW;
    hSpi1master.Init.ClockPhase    = SPI_PHASE_1EDGE;
    hSpi1master.Init.BaudRate      = 64;
    hSpi1master.Init.Standard      = SPI_FRF_MOTO;
    hSpi1master.Init.ControlSize   = SPI_CTRLSIZE_1BIT;
    hSpi1master.Init.NumberDataFrame = 1;

    if (HAL_SPI_Init(&hSpi1master) != HAL_OK)
    { /* Error_Handle */
      Error_Handle();
    }
```

XT_Spi_Task handles the basic transfer process.

```
{
    /* USER CODE */
    /*test spi1 as master transfer comunication*/
    if(HAL_SPI_Transmit_IT(&hSpi1master, (uint8_t *)aSpi1TxBuffer,
sizeof(aSpi1TxBuffer)-1)!=HAL_OK)
    {
        Error_Handle();
    }
    while (HAL_SPI_GetState(&hSpi1master) != HAL_SPI_STATE_READY);
    while(u1Spi_cbState!=CB_SPI1_TXFNSH);
    u1Spi_cbState = CB_SPI_IDLE;

    if(HAL_SPI_Receive_IT(&hSpi1master, (uint8_t *)aRxBuffer,
sizeof(aSpi1TxBuffer)-1)!=HAL_OK)
    {
        Error_Handle();
    }
    while (HAL_SPI_GetState(&hSpi1master) != HAL_SPI_STATE_READY);
    while(u1Spi_cbState!=CB_SPI1_RXFNSH);
    u1Spi_cbState = CB_SPI_IDLE;

    if(HAL_SPI_TransmitReceive_IT(&hSpi1master, (uint8_t *)aSpi1TxBuffer, (uint8_t
*)aRxBuffer, sizeof(aSpi1TxBuffer)-1)!=HAL_OK)
    {
        Error_Handle();
    }
    while(u1Spi_cbState!=CB_SPI1_TXRXFNSH);
    u1Spi_cbState = CB_SPI_IDLE;

    /*test spi2 as slaver transfer comunication*/
    if(HAL_SPI_Transmit_IT(&hSpi2slaver, (uint8_t *)aSpi2TxBuffer,
sizeof(aSpi2TxBuffer))!=HAL_OK)
    {
        Error_Handle();
```

```
    }
    while (HAL_SPI_GetState(&hSpi2slaver) != HAL_SPI_STATE_READY);
    while(u1Spi_cbState!=CB_SPI2_TXFNSH);
    u1Spi_cbState = CB_SPI_IDLE;

    if(HAL_SPI_Receive_IT(&hSpi2slaver, (uint8_t *)aRxBuffer,
sizeof(aSpi2TxBuffer)-1)!=HAL_OK)
    {
        Error_Handle();
    }
    while (HAL_SPI_GetState(&hSpi2slaver) != HAL_SPI_STATE_READY);
    while(u1Spi_cbState!=CB_SPI2_RXFNSH);
    u1Spi_cbState = CB_SPI_IDLE;

}
```

## 2.6 Additional resources

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