M031G Manchester Codec Software Solution

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code uses software TX and hardware RX solution to implement Manchester codec function.
BSP Version	M030G BSP CMSIS V3.01.000
Hardware	M031_GPON_GFN33_NU_AUTO Module

1 Overview

In 5G GPON standard, the China Mobile does not send the IDLE pattern between two frames. However, the M031G Manchester codec needs to use the IDLE pattern for sending and receiving frames. To solve this problem, this sample provides an workaround solution to comply with the China Mobile 5G GPON requirements.

1.1 Principle

In the M031G Manchester hardware codec, since the Manchester TX sends the IDLE pattern in the beginning of each frame, it is replaced by another pure software mechanism in this sample. The mechanism is implemented by Timer triggering two different PDMA channels to make the TX pin output High and Low levels, respectively. Specifically, if the TX signal needs to be modulated with frequency modulation, this sample also demonstrates how to generate the corresponding sinusoidal waveform.

The M031G Manchester RX decoder needs the IDLE pattern before decoding each frame. But the China Mobile standard sends the consecutive frames without the IDLE pattern. To solve this issue, while receiving RX signal from the assigned GPIO pin, the first byte of several Preamble patterns is replaced by one pseudo IDLE pattern, and the modified frame is outputted to another GPIO pin (RXm). At the same time, the RXm pin is connected to the real M031G Manchester RX pin. Since the reformed RX signal includes the IDLE pattern in the beginning of each frame, the M031G Manchester decoder can decode it by hardware.

1.1.1 Software TX

This section describes how to generate the TX signal by software. The procedure is listed below.

1. Prepare the sinusoidal waveform table for the DAC0 if the TX frequency modulation is necessary.
   - The table can be created by the following statement.

```
/* prepare sinusoidal waveform data table for the DAC0 */
for (i=0; i<SINE_SAMPLE; i++)
{
    /* Add 1.0 to offset sine result from [-1, 1] to [0, 2],
       and divided with 2.0 to compress to [0, 1] */
    g_sineBuf[i] = (uint16_t)(((sin((double)(((i+1) * PI) /
                        (SINE_SAMPLE/2))) + 1.0) / 2.0) * 0xFFF);
}
```

2. Prepare the encoded buffer.
   - Before sending the raw data buffer, it must be encoded to the desired Manchester format, including G.E. Thomas (1→10, 0→01) or IEEE 802.3 (0→10, 1→01). In this sample code, G.E. Thomas is selected.

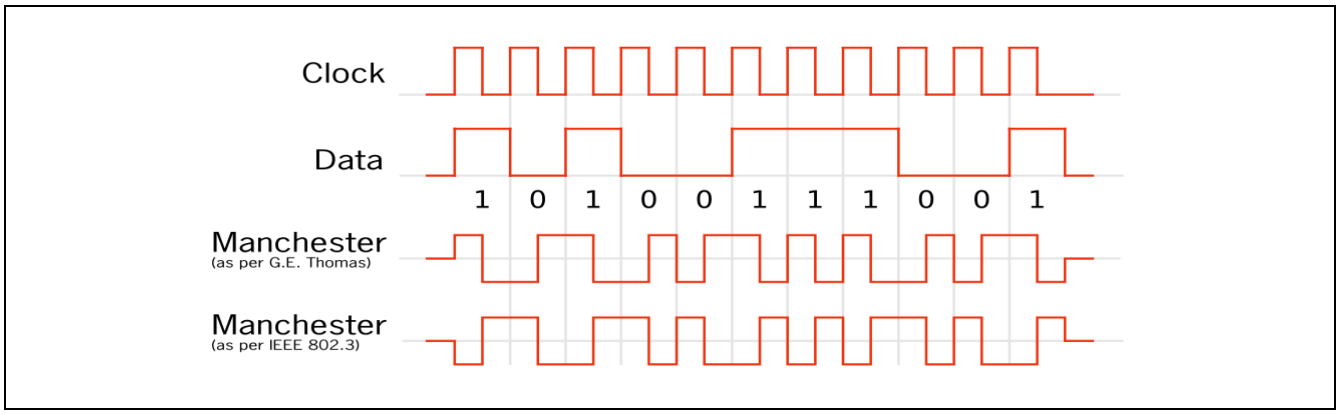

Figure 1-1 Manchester Encoded Format

To generate the TX signal in the dedicated GPIO pin, each bit of the raw data buffer is encoded to one two-byte pattern. This two-byte pattern can trigger two different PDMA channels to generate High or Low level on the TX pin, respectively. In addition, if the TX frequency modulation is required, the PDMA channel can also trigger DAC0 to generate sine waveform when the TX is in High or Low state.

```
/* encode raw data buffer */
void Encode_Buf_Fill(uint8_t *u8ManchTxBuf, uint8_t u8Id)
{
    for(i=0; i<FRAME_LENGTH; i++)
    {
        for(j=0; j<8; j++)
        {
            u8Dat = u8ManchTxBuf[i] >> (7-j);
            if(u8Dat & 0x1) /* 1 is encode to 10 */
            {
                g_u8EncodeBuf[u8Id][u16BitCount] = ENCODE_TX_HIGH;
                u16BitCount++;
                g_u8EncodeBuf[u8Id][u16BitCount] = ENCODE_TX_LOW;
                u16BitCount++;
            }
            else          /* 0 is encode to 01 */
            {
                g_u8EncodeBuf[u8Id][u16BitCount] = ENCODE_TX_LOW;
                u16BitCount++;
                g_u8EncodeBuf[u8Id][u16BitCount] = ENCODE_TX_HIGH;
                u16BitCount++;
            }
```

```
    }
  }
}
```

3. Set PDMA for the TX and DAC0 output.
   - Based on different PDMA channels, the TX pin can output High or Low level, and DAC0 can generate the sine waveform at the same time if requred. Several scatter-gather tables for different PDMA channels have been set up in the function “PDMA_Encode_Init()” as described below.

   a. PDMA_ENCODE_CH0

   Two scatter-gather tables are listed for PDMA_ENCODE_CH0. Based on the content of g_u8EncodeBuf[], the PDMA_SWREQ register will be written 0x02 or 0x04 to trigger PDMA_ENCODE_CH1 or PDMA_ENCODE_CH2, respectively.

```
void PDMA_Encode_Init(void)
{
  …

  /* Enable PDMA channels */
  PDMA_Open(PDMA, 1<<PDMA_ENCODE_CH0);
  PDMA_SetTransferMode(PDMA, PDMA_ENCODE_CH0,
  PDMA_TMR1, TRUE, (uint32_t)&PDMA_TX0_DESC[0]);

  PDMA_TX0_DESC[0].ctl = ((1024 - 1) <<
  PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_8 |
  PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_SINGLE |
  PDMA_OP_SCATTER;
  PDMA_TX0_DESC[0].src = (uint32_t)g_u8EncodeBuf[0];
  PDMA_TX0_DESC[0].dest = (uint32_t)&PDMA->SWREQ;
  PDMA_TX0_DESC[0].offset = (uint32_t)&PDMA_TX0_DESC[1] –
  (PDMA->SCATBA);

  PDMA_TX0_DESC[1].ctl = ((1024 - 1) <<
  PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_8 |
  PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_SINGLE |
  PDMA_OP_SCATTER;
  PDMA_TX0_DESC[1].src = (uint32_t)g_u8EncodeBuf[1];
  PDMA_TX0_DESC[1].dest = (uint32_t)&PDMA->SWREQ;
  PDMA_TX0_DESC[1].offset = (uint32_t)&PDMA_TX0_DESC[0] –
  (PDMA->SCATBA);   //link to first description

  …
}
```

b. PDMA_ENCODE_CH1

If the DAC0 is assigned to generate sine waveform while TX is in High state, two scatter-gather tables are set up for PDMA_ENCODE_CH1. The first table enables the DAC0 auto-sine function, and DAC0 will generate the sine waveform on DAC0 output pin. The second table can make the TX output High state. However, if the DAC0 is assigned to generate sine waveform while TX is in Low state, three scatter-gather tables are set up for PDMA_ENCODE_CH1. The first table disables the DAC0 auto-sine function and the second table sets the DAC0 to the assigned value (0x000 ~ 0xFFF). Then, the third table sets the TX output to High state.

```
void PDMA_Encode_Init(void)
{
    …

#ifdef OPT_AUTO_SINE_HIGH
    /* Enable PDMA channels */
    PDMA_Open(PDMA, 1<<PDMA_ENCODE_CH1);
    PDMA_SetTransferMode(PDMA, PDMA_ENCODE_CH1,
    PDMA_MEM, TRUE, (uint32_t)&PDMA_TX1_DESC[0]);

    PDMA_TX1_DESC[0].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_BURST |
    PDMA_OP_SCATTER;

    PDMA_TX1_DESC[0].src = (uint32_t)&g_u32DacEnable;
    PDMA_TX1_DESC[0].dest = (uint32_t)&DAC0->ADGCTL;
    PDMA_TX1_DESC[0].offset = (uint32_t)&PDMA_TX1_DESC[1] –
    (PDMA->SCATBA);

    PDMA_TX1_DESC[1].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_SINGLE |
    PDMA_OP_SCATTER;
    PDMA_TX1_DESC[1].src = (uint32_t)&g_u32PinHigh;
    PDMA_TX1_DESC[1].dest = (uint32_t)&ENCODE_TXD;
    PDMA_TX1_DESC[1].offset = (uint32_t)&PDMA_TX1_DESC[0] –
    (PDMA->SCATBA);   //link to first description
    …

#else
    /* Enable PDMA channels */
    PDMA_Open(PDMA, 1<<PDMA_ENCODE_CH1);

    PDMA_SetTransferMode(PDMA, PDMA_ENCODE_CH1,
    PDMA_MEM, TRUE, (uint32_t)&PDMA_TX1_DESC[0]);
```

```
    PDMA_TX1_DESC[0].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_BURST |
    PDMA_OP_SCATTER;
    PDMA_TX1_DESC[0].src = (uint32_t)&g_u32DacDisable;
    PDMA_TX1_DESC[0].dest = (uint32_t)&DAC0->ADGCTL;
    PDMA_TX1_DESC[0].offset = (uint32_t)&PDMA_TX1_DESC[1] –
    (PDMA->SCATBA);

    PDMA_TX1_DESC[1].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_BURST |
    PDMA_OP_SCATTER;
    PDMA_TX1_DESC[1].src = (uint32_t)&g_u32DacData;
    PDMA_TX1_DESC[1].dest = (uint32_t)&DAC0->DAT;
    PDMA_TX1_DESC[1].offset = (uint32_t)&PDMA_TX1_DESC[2] –
    (PDMA->SCATBA);   //link to first description

    PDMA_TX1_DESC[2].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_SINGLE |
    PDMA_OP_SCATTER;
    PDMA_TX1_DESC[2].src = (uint32_t)&g_u32PinHigh;
    PDMA_TX1_DESC[2].dest = (uint32_t)&ENCODE_TXD;
    PDMA_TX1_DESC[2].offset = (uint32_t)&PDMA_TX1_DESC[0] -
    (PDMA->SCATBA);   //link to first description
    …

#endif
    …
}
```

c. PDMA_ENCODE_CH2

If the DAC0 is assigned to generate sine waveform while TX is in High state, three scatter-gather tables are set up for PDMA_ENCODE_CH2. The first table disables the DAC0 auto-sine function and the second table set the DAC0 to the assigned value (0x000 ~ 0xFFF). Then, the third table sets the TX output to Low state. However, If the DAC0 is assigned to generate sine waveform while TX is in Low state, two scatter-gather tables are set up for PDMA_ENCODE_CH2. The first table enables the DAC0 auto-sine function, and the DAC0 will generate the sine waveform on DAC0 output pin. The second tables can make the TX output Low state.

```
void PDMA_Encode_Init(void)
{
    …
```

```
#ifdef OPT_AUTO_SINE_HIGH
    …

    /* Enable PDMA channels */
    PDMA_Open(PDMA, 1<<PDMA_ENCODE_CH2);

    PDMA_SetTransferMode(PDMA, PDMA_ENCODE_CH2,
    PDMA_MEM, TRUE, (uint32_t)&PDMA_TX2_DESC[0]);

    PDMA_TX2_DESC[0].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_BURST |
    PDMA_OP_SCATTER;
    PDMA_TX2_DESC[0].src = (uint32_t)&g_u32DacDisable;
    PDMA_TX2_DESC[0].dest = (uint32_t)&DAC0->ADGCTL;
    PDMA_TX2_DESC[0].offset = (uint32_t)&PDMA_TX2_DESC[1] –
    (PDMA->SCATBA);

    PDMA_TX2_DESC[1].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_BURST |
    PDMA_OP_SCATTER;
    PDMA_TX2_DESC[1].src = (uint32_t)&g_u32DacData;
    PDMA_TX2_DESC[1].dest = (uint32_t)&DAC0->DAT;
    PDMA_TX2_DESC[1].offset = (uint32_t)&PDMA_TX2_DESC[2] -
    (PDMA->SCATBA);   //link to first description

    PDMA_TX2_DESC[2].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_SINGLE |
    PDMA_OP_SCATTER;
    PDMA_TX2_DESC[2].src = (uint32_t)&g_u32PinLow;
    PDMA_TX2_DESC[2].dest = (uint32_t)&ENCODE_TXD;
    PDMA_TX2_DESC[2].offset = (uint32_t)&PDMA_TX2_DESC[0] –
    (PDMA->SCATBA);   //link to first description
#else
    …

    /* Enable PDMA channels */
    PDMA_Open(PDMA, 1<<PDMA_ENCODE_CH2);
    PDMA_SetTransferMode(PDMA, PDMA_ENCODE_CH2,
    PDMA_MEM, TRUE, (uint32_t)&PDMA_TX2_DESC[0]);

    PDMA_TX2_DESC[0].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_BURST |
    PDMA_OP_SCATTER;
    PDMA_TX2_DESC[0].src = (uint32_t)&g_u32DacEnable;
    PDMA_TX2_DESC[0].dest = (uint32_t)&DAC0->ADGCTL;
```

```
    PDMA_TX2_DESC[0].offset = (uint32_t)&PDMA_TX2_DESC[1] –
    (PDMA->SCATBA);

    PDMA_TX2_DESC[1].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_INC | PDMA_DAR_FIX | PDMA_REQ_SINGLE |
    PDMA_OP_SCATTER;
    PDMA_TX2_DESC[1].src = (uint32_t)&g_u32PinLow;
    PDMA_TX2_DESC[1].dest = (uint32_t)&ENCODE_TXD;
    PDMA_TX2_DESC[1].offset = (uint32_t)&PDMA_TX2_DESC[0] –
    (PDMA->SCATBA);   //link to first description
#endif
}
```

1.1.2 Hardware RX

This section describes how to decode RX signal by the M031G Manchester hardware decoder. As described in Section 1.1, the M031G Manchester decoder needs the IDLE pattern in the beginning of each frame. Therefore, the received RX signal needs to be reformed to another RXm with the IDLE pattern. In this sample code, the RX signal is received from PB15 pin and reformed to another RXm (PB5). In Figure 1-2, the IDLE pattern 0xFF is inserted between two frames.

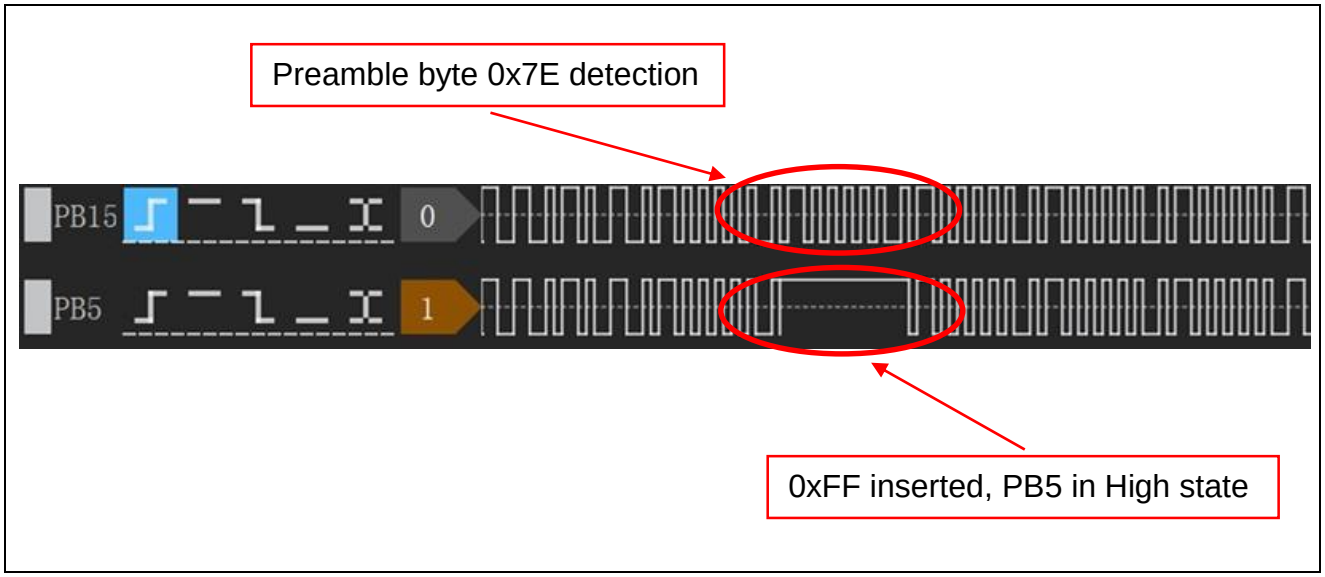

Figure 1-2 RX and Reformed RX

At the same time, the reformed RXm is connected to the real Manchester RX pin, and the Manchester hardware decoder can decode the RXm as shown in Figure 1-3.

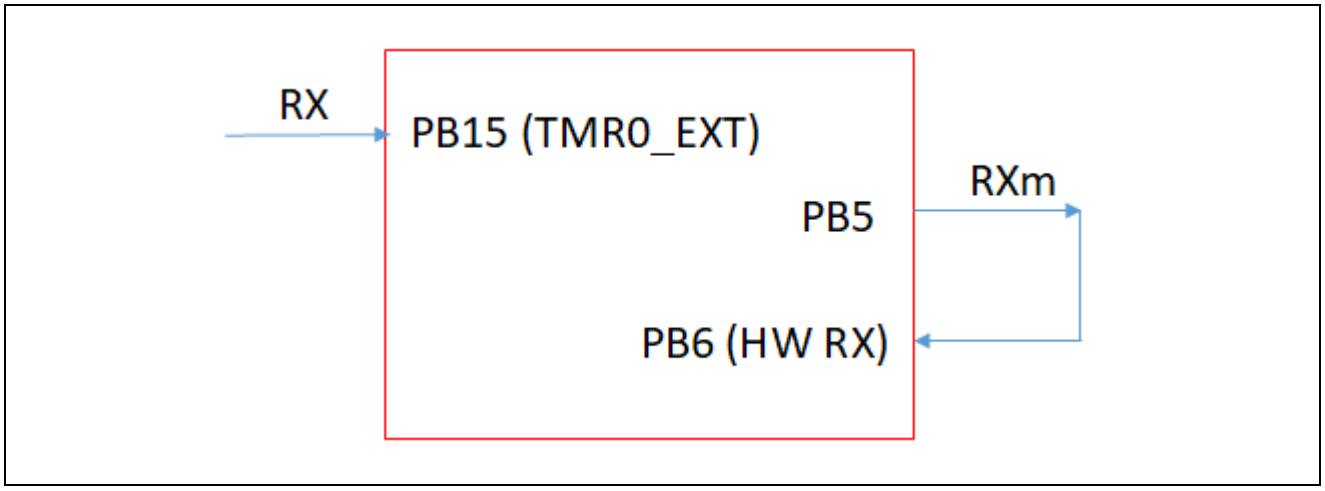

Figure 1-3 RXm Loopback to Hardware RX

The procedure to decode the RX is listed below.

1. Decode the first Preamble byte by software.
   - In this sample code, the bit stream is encoded by G.E. Thomas format in 2 kHz. The M031G Timer0 capturing function with 1 MHz time base is used to decode the first byte of Preamble byte (0x7E).

   - To begin searching the pattern 0x7E, the Timer0 is set to monitor the rising-edge of external input in the beginning. Whenever the rising-edge of RX has been detected, the Timer0 capture function is changed to detect the falling-edge. When the falling-edge is detected later, the Timer0 count value that is captured by the falling-edge will be checked if the captured value is beyond 1,000+/-100. If the value does not exceed 1,000+/-100, keep the Timer0 capture function on falling-edge detection. Otherwise, ignore the current detection and return to the first rising-edge detection again.

   - If the captured values of six continuous falling-edges do not exceed 1,000+/-100, it means that 0x7E or 0x7F pattern has been detected. To confirm if 0x7E has been detected, the Timer0 capture function is changed to detect the rising-edge. If the captured value of next rising-edge is still in 1,000+/-100, it is confirmed 0x7E detected.

   - Figure 1-4 indicates that three continuous differences are captured by one rising-edge and three falling-edge can be between 1,000+/-100 and means 0x7# to be

detected. Specifically, the RXm pin is kept at High state before 0x7E pattern is detected as shown in Figure 1-2.

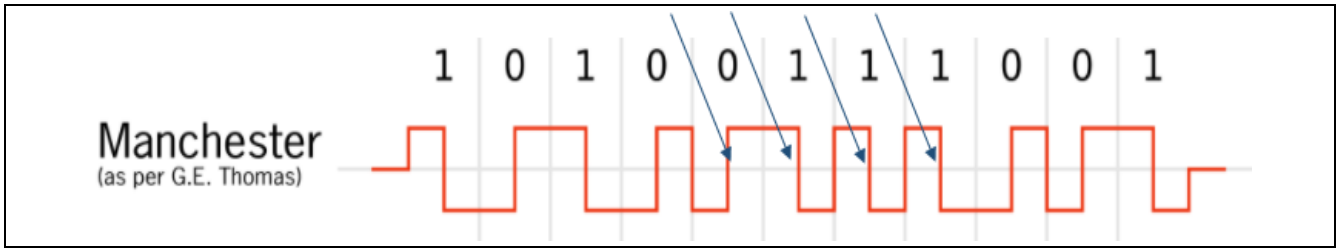

Figure 1-4 Preamble Byte Detection

```
/* For detecting Preamble pattern 0x7E by software */
void Manch_Receive_By_Software(uint32_t u32Time)
{
    /* Check if the first rising-edge detected */
    if(g_u8RxStatus == 0)
    {
        /* Change to detect falling-edge */
        Timer_Decode_Capture_Set(TIMER_CAPTURE_FALLING_EDGE);
        g_u8RxStatus = 1;
    }
    else if((g_u8RxStatus==1) || (g_u8RxStatus==2) || (g_u8RxStatus==3)
            || (g_u8RxStatus==4) || (g_u8RxStatus==5) || g_u8RxStatus==6))
    {
        /* Check the range of deviation */
        if((u32Time>(TIME_2T-TIME_OFFSET)) &&
           (u32Time<(TIME_2T+TIME_OFFSET)))
        {
            /* If the deviation of six falling-edge detections less than 100
               (TIME_OFFSET), change to rising-edge detection */
            if(g_u8RxStatus == 6)
                Timer_Decode_Capture_Set(TIMER_CAPTURE
                _RISING_EDGE);
            g_u8RxStatus++;
        }
        else
        {
            Timer_Decode_Capture_Set(TIMER_CAPTURE_
            RISING_EDGE);
            g_u8RxStatus = 0;
        }
    }
    else if(g_u8RxStatus == 7)
    {
        /* If consecutive seven deviations are less than 100,
           It means that 0x7E is detected */
        if((u32Time>(TIME_2T-TIME_OFFSET)) &&
```

```
        (u32Time<(TIME_2T+TIME_OFFSET)))
    {
        g_u8RxStatus = 0;
        NVIC_DisableIRQ(TMR0_IRQn);
        TIMER_Start(TIMER2);
        Timer_Decode_Capture_Set(TIMER_CAPTURE_FALLING
        _AND_RISING_EDGE);
        Timer_PDMA_Enable();
    }
    else
    {
        Timer_Decode_Capture_Set(TIMER_CAPTURE_
        RISING_EDGE);
        g_u8RxStatus = 0;
    }
  }
}
```

2. Duplicate RX signal to RXm.
   - After the first Preamble byte 0x7E is detected, the PDMA_DECODE_CH0 is enabled. Specifically, the PDMA_DECODE_CH0 is triggered by Timer0 capture RX rising-edge and falling-edge, and it can duplicate the RX signal to the RXm. Two scatter-gather tables are set up for the PDMA_DECODE_CH0 as listed below.

```
void PDMA_Rx_Init(void)
{
    /* Enable PDMA channels */
    PDMA_Open(PDMA, 1<<PDMA_DECODE_CH0);
    PDMA_SetTransferMode(PDMA, PDMA_DECODE_CH0,
    PDMA_TMR0, TRUE, (uint32_t)&PDMA_RX0_DESC[0]);

    PDMA_RX0_DESC[0].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_FIX | PDMA_DAR_FIX | PDMA_REQ_SINGLE |
    PDMA_OP_SCATTER;
    PDMA_RX0_DESC[0].src = (uint32_t)&g_u32DecodeTxBuf[0];
    PDMA_RX0_DESC[0].dest = (uint32_t)&DECODE_TXD;
    PDMA_RX0_DESC[0].offset = (uint32_t)&PDMA_RX0_DESC[1] –
    (PDMA->SCATBA);

    PDMA_RX0_DESC[1].ctl = ((1 - 1) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_32 |
    PDMA_SAR_FIX | PDMA_DAR_FIX | PDMA_REQ_SINGLE |
    PDMA_OP_SCATTER;
    PDMA_RX0_DESC[1].src = (uint32_t)&g_u32DecodeTxBuf[1];
    PDMA_RX0_DESC[1].dest = (uint32_t)&DECODE_TXD;
    PDMA_RX0_DESC[1].offset = (uint32_t)&PDMA_RX0_DESC[0] –
    (PDMA->SCATBA);   //link to first description
```

```
    …

}
```

3. Set PDMA for Manchester hardware decoder.
   - The PDMA_DECODE_CH1 is for Manchester hardware decoder and two scatter-gather tables are set. Specifically, the hardware decoder is always enabled and can only decode the frame with the IDLE pattern in the beginning as shown in Figure 1-2.

```
void PDMA_Rx_Init(void)
{
    …

    /* Disable RX DMA */
    MANCH_DISABLE_RX_DMA(MANCH);

    /* MANCH RX PDMA channel configuration */
    PDMA_Open(PDMA, 1<<PDMA_DECODE_CH1);
    PDMA_SetTransferMode(PDMA, PDMA_DECODE_CH1,
    PDMA_MANCH_RX, TRUE, (uint32_t)&PDMA_RX1_DESC[0]);

    PDMA_RX1_DESC[0].ctl = ((FRAME_LENGTH - 2) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_8 |
    PDMA_SAR_FIX | PDMA_DAR_INC | PDMA_REQ_SINGLE |
    PDMA_OP_SCATTER;
    PDMA_RX1_DESC[0].src = (uint32_t)&MANCH->RXDAT;
    PDMA_RX1_DESC[0].dest = (uint32_t)g_u8ManchRxBuf[0];
    PDMA_RX1_DESC[0].offset = (uint32_t)&PDMA_RX1_DESC[1] –
    (PDMA->SCATBA);

    PDMA_RX1_DESC[1].ctl = ((FRAME_LENGTH - 2) <<
    PDMA_DSCT_CTL_TXCNT_Pos) | PDMA_WIDTH_8 |
    PDMA_SAR_FIX | PDMA_DAR_INC | PDMA_REQ_SINGLE |
    PDMA_OP_SCATTER;
    PDMA_RX1_DESC[1].src = (uint32_t)&MANCH->RXDAT;
    PDMA_RX1_DESC[1].dest = (uint32_t)g_u8ManchRxBuf[1];
    PDMA_RX1_DESC[1].offset = (uint32_t)&PDMA_RX1_DESC[0] –
    (PDMA->SCATBA);   //link to first description

    /* Enable RX DMA */
    MANCH_ENABLE_RX_DMA(MANCH);
}
```

1.1.3 CRC Checking

In the M031G Manchester decoder, there is no hardware mechanism to do the CRC checking. After receiving the input frame from the dedicated RX pin, the CRC can be checked by the M031G CRC. This sample code provides two selections for the user to do the CRC checking, as described below.

1. CRC checking by PDMA

```
int check_crc(uint8_t* buf)
{
    …

    /* Open Channel PDMA_DECODE_CRC_CH */
    PDMA_Open(PDMA,1 << PDMA_DECODE_CRC_CH);

    PDMA_SetTransferCnt(PDMA, PDMA_DECODE_CRC_CH,
    PDMA_WIDTH_8, MSG_LENGTH);
    PDMA_SetTransferAddr(PDMA, PDMA_DECODE_CRC_CH, (uint32_t)buf,
    PDMA_SAR_INC, (uint32_t)&CRC->DAT, PDMA_DAR_FIX);
    PDMA_SetTransferMode(PDMA, PDMA_DECODE_CRC_CH, DMA_MEM,
    FALSE, 0);
    PDMA_SetBurstType(PDMA, PDMA_DECODE_CRC_CH,
    PDMA_REQ_BURST, PDMA_BURST_1);

    /* Generate a software request to trigger transfer with PDMA */
    PDMA_Trigger(PDMA, PDMA_DECODE_CRC_CH);

    /* Wait transfer done */
    while(!((PDMA_GET_TD_STS(PDMA)&(PDMA_TDSTS_TDIF0_Msk<<PD
    MA_DECODE_CRC_CH))));

    /* Clear transfer done flag */
    PDMA_CLR_TD_FLAG(PDMA,
    (PDMA_TDSTS_TDIF0_Msk<<PDMA_DECODE_CRC_CH));

    /* Get CRC-8 checksum value */
    checksum = CRC_GetChecksum();

    …
}
```

2. CRC checking by no PDMA

```
int check_crc(uint8_t* buf)
{
    …

    /* Start to calculate CRC checksum in buffer */
    for(i = 0; i < MSG_BEFORE_CRC; i++)
    {
        CRC_WRITE_DATA(buf[j]);
```

```
        j++;
    }

  /* Store checksum in buffer */
  buf_checksum = buf[j];
  j++;

    /* Continue to calculate CRC checksum in buffer */
    for(i = 0; i < MSG_LENGTH; i++)
    {
        CRC_WRITE_DATA(buf[j]);
        j++;
    }

    /* Get CRC-8 checksum value */
    checksum = CRC_GetChecksum();

    …
}
```

1.2 Operation Process

Before doing the test, the related hardware connection must be ready. This sample can be verified by self-loopback test or full duplex test between two M031G evaluation boards. When this sample code begins to be executed, the console can display the message as shown in Figure 1-5.

```
+------------------------------------------------------------------------------+
|                            MANCH Driver Sample Code                          |
+------------------------------------------------------------------------------+
  This sample code will send MANCH encoded data to PA2 (TX) by pure software.
  Please connect PA2 to PB15 (pseudo RX, TMR0_EXT) for loopback test. In addition,
  the modified RX (RXm) which one preamble byte is replaced by IDLE pattern
  is generated at PB5. Please also connect PB5 to the real RX (PB.6) at the same time.

  Press any key when to be ready.
```

Figure 1-5 Console Message

2 Demo Result

From the test result, the received frame size is only 63-byte. This reason is that one Preamble byte is served as the IDLE pattern, and the hardware decoder does not receive it. This sample also provides another option to copy the received buffer to another rearranged 64-byte buffer, g_u8ManchRxBuf_ReArranged[].

```
-> CRC: OK


RX: 0x7e, 0x7e, 0x7e, 0x7e, 0xbe, 0xbe, 0xbe, 0x36, 0xb9, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0x
be, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0x
be, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe, 0xbe,

-> CRC: OK


RX: 0x7e, 0x7e, 0x7e, 0x7e, 0xbf, 0xbf, 0xbf, 0x36, 0xfa, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0x
bf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0x
bf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf, 0xbf,

-> CRC: OK


RX: 0x7e, 0x7e, 0x7e, 0x7e, 0xc0, 0xc0, 0xc0, 0x36, 0xe6, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0x
c0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0x
c0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0,

-> CRC: OK


RX: 0x7e, 0x7e, 0x7e, 0x7e, 0xc1, 0xc1, 0xc1, 0x36, 0xa5, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0x
c1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0x
c1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1, 0xc1,

-> CRC: OK
```

Figure 2-1 Output Result

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M030G BSP CMSIS V3.01.000
- IDE version
  - Keil uVersion 5.27

3.2 Hardware Requirements

- M031_GPON_GFN33_NU_AUTO Module

- Pins Connection

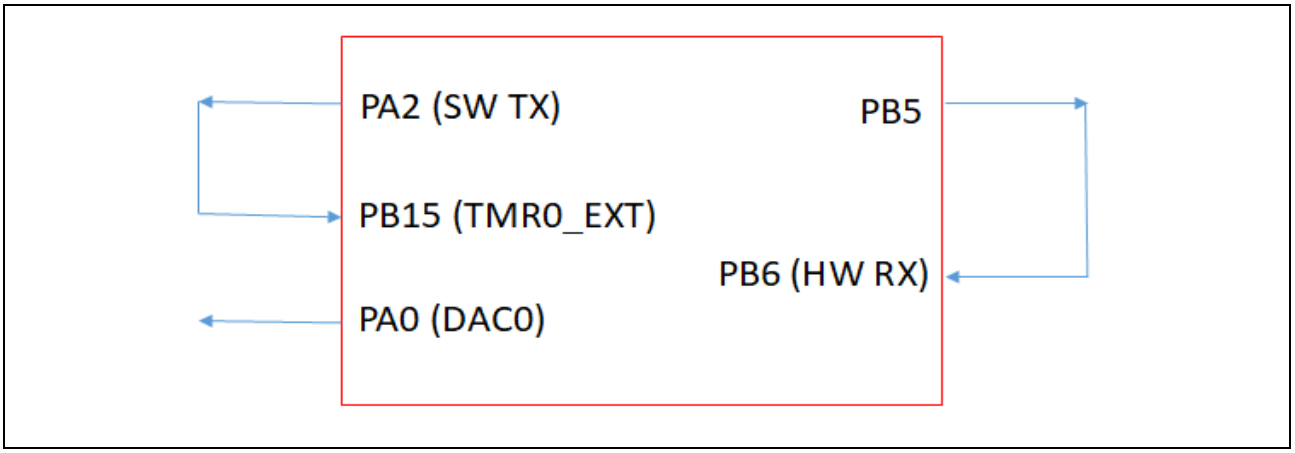

Figure 3-1 Pins Connection

4 Directory Information

The directory structure is shown below.

M030G BSP	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
MANCH_TXRXLoopback_NoIdlePatternInFrames	Source files of example code

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 MANCH_TXRXLoopback_NoIdlePatternInFrames.uvproj.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2021.09.07	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.