BAT32G137（库函数版本）

Rev 1.0

修订历史

版本	日期	修订人	修订内容
Rev1.1	22.6.22	缪勤文 张刚	

目录

1. 前言

串行 IICA 接口不同于 SCI 模块中的简易 IIC 功能，IICA 可以用作从机。

2. 中微 BAT32G137 系列串行 IICA 接口应用库简介

中微 BAT32G137 系类串行 IICA 应用库是一个便于移植的标准库代码风格，用户只需要对软件接口相关参数进行简单配置、以及封装接口函数调用即可实现所需功能，节约时间，提高开发效率。应用库提供了主机/从机，基于 IICA 中断读写方式，轮询方式读写。

使用方式：

使用 IICA 需要将应用层 iica_demo.c iica_demo.h 驱动层 i2ca.c i2ca.h、gpio.c gpio.h、isr.c isr.h 加入到工程中去；

2.1.应用例程使用

包括主机/从机的 IICA 初始化，读写函数以及 IICA 相关接口；演示例程包含了主机：主机驱动 AT24C02 读写数据、主机（开发板）和从机（开发板）进行数据读写

2.1.1. IICA 初始化

```
1.  void Iica0_Init(void)
2.  {
3.   GPIO_InitTypeDef GPIO_InitStruct={0};
4.   I2CA_InitTypeDef I2CA_InitStructure;
5.
6.   GPIO_PinAFConfig(GPIO_PORT6,GPIO_Pin_0,GPIO_P60,GROUP_AF_SCLAA0);//
    SCLA0 can be allocated to any desired pins in group0
7.   GPIO_PinAFConfig(GPIO_PORT6,GPIO_Pin_1,GPIO_P61,GROUP_AF_SDAA0);//S
    DAA0 can be allocated to any desired pins in group0
8.
9.   /*SCLA0 GPIO CONFIG*/
10.  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
11.  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
12.  GPIO_InitStruct.GPIO_Level  = GPIO_Level_LOW;
```

```
13.  GPIO_InitStruct.GPIO_PuPd   = GPIO_PuPd_UP;
14.  GPIO_InitStruct.GPIO_OType  = GPIO_OType_OD;
15.  GPIO_InitStruct.GPIO_Ctrl = GPIO_Control_DIG;
16.  GPIO_Init(GPIO_PORT6,&GPIO_InitStruct);
17.
18.  /*SDAA0 GPIO CONFIG*/
19.  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_1;
20.  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
21.  GPIO_InitStruct.GPIO_Level  = GPIO_Level_LOW;
22.  GPIO_InitStruct.GPIO_PuPd   = GPIO_PuPd_UP;
23.  GPIO_InitStruct.GPIO_OType  = GPIO_OType_OD;
24.  GPIO_InitStruct.GPIO_Ctrl = GPIO_Control_DIG;
25.  GPIO_Init(GPIO_PORT6,&GPIO_InitStruct);
26.
27.  I2CA_InitStructure.I2C_ClockSpeed = 100000;
28.  I2CA_InitStructure.I2C_Ack = I2CA_Ack_Enable;
29.  I2CA_InitStructure.I2C_DutyCycle = 50;
30.  I2CA_InitStructure.I2C_OwnAddress = 0xA0;
31.  I2CA_InitStructure.I2C_Mode = I2CA_Mode_SMBusMaster;
32.  I2CA_Init(&I2CA_InitStructure);
33. #ifdef I2CA_USING_INTERRUPT
34.  ISR_Register(IICA_IRQn,iica0_interrupt);
35.  INTC_EnableIRQ(IICA_IRQn);        //enable INTIICA interrupt flag
36. #endif
37. }
```

- GPIO 复用，调用 GPIO_PinAFConfig 函数，IICA 的 SCL 和 SDA 引脚可以任意映射到芯片的 GPIO，在配置 GPIO 时需要配置上拉输出、开漏输出；
- 配置 IICA 运行速率
- 设置 IICA SCL 高电平的占空比，通常设置百分之 50
- 设置自身地址：只有 IICA 用作从机才生效
- 配置 IICA 工作模式：主机模式/从机模式
- 注册中断服务函数

2.1.2. IICA 主机 轮询收/发函数

对于串行 IICA 接口收发函数，对于轮询方式来说，其无法进行 ACK 检测，默认其能收到从机响应：

发送函数：I2CA_Master_WriteData

```
/**
  * @brief  Send a data buffer through the I2CA peripheral to write.
  * @param  Address: Device address in the I2C bus.
  * @param  Reg: Register address in the Device.
  * @param  Data: Data buffer address.
  * @param  Len: Data buffer length need to transmit.
  * @retval None
  */
I2CA_Status I2CA_Master_WriteData(uint8_t Address, uint8_t Reg, uint8_t *Data, uint16_t Len)
{
 I2CA_Status status = OK;

 if (I2CA_GetFlagStaus(I2CA_STATUS_BUSBSY) != RESET)
 {
  status = BUSY;
 }
 else
 {
  I2CA_GenerateSTART();

  IICA->IICA0 = Address & 0xFE;

        while(!INTC_GetPendingIRQ(IICA_IRQn));
        INTC_ClearPendingIRQ(IICA_IRQn);

  IICA->IICA0 = Reg;

        while(!INTC_GetPendingIRQ(IICA_IRQn));
        INTC_ClearPendingIRQ(IICA_IRQn);

  do
  {
   IICA->IICA0 = *Data++;

   while(!INTC_GetPendingIRQ(IICA_IRQn));
   INTC_ClearPendingIRQ(IICA_IRQn);

```

```
38.   } while(--Len);
39.
40.         /* stop condition is generated */
41.         I2CA_GenerateSTOP();
42.  }
43.
44.  return status;
45. }
```

轮询发送函数传入的形参有：从机设备地址、寄存器地址、 要发送的数据、数据长度；

IICA 阻塞接收函数 I2CA_Master_ReadData

```
1.  I2CA_Status I2CA_Master_ReadData(uint8_t Address, uint8_t Reg, uint8
    _t *Data, uint16_t Len)
2.  {
3.   I2CA_Status status = OK;
4.
5.   if (I2CA_GetFlagStaus(I2CA_STATUS_BUSBSY) != RESET)
6.   {
7.    status = BUSY;
8.   }
9.   else
10.  {
11.         /* Generate the START condition */
12.   I2CA_GenerateSTART();
13.
14.   IICA->IICA0 = Address & 0xFE;
15.
16.         /* Wait device address send succes */
17.         while(!INTC_GetPendingIRQ(IICA_IRQn));
18.         INTC_ClearPendingIRQ(IICA_IRQn);
19.
20.   IICA->IICA0 = Reg;
21.
22.         /* Wait device register send succes */
23.         while(!INTC_GetPendingIRQ(IICA_IRQn));
24.         INTC_ClearPendingIRQ(IICA_IRQn);
25.
26.         /* Generate the RESTART condition */
27.         I2CA_GenerateSTART();
28.
29.   IICA->IICA0 = Address | 0x01;
```

```
30.
31.         /* Wait address send succes */
32.         while(!INTC_GetPendingIRQ(IICA_IRQn));
33.         INTC_ClearPendingIRQ(IICA_IRQn);
34.
35.   do {
36.             /* When the last byte to receive, we should not output A
    CK */
37.             if(Len == 1U)
38.             {
39.                 /* acknowledgment disable in the last byte to receiv
    e */
40.                 IICA->IICCTL00 &= ~(1 << I2CA_ACK_BIT);
41.             }
42.
43.             /* Release the bus status for receive data */
44.    IICA->IICCTL00 |= (1 << I2CA_WREL_BIT);
45.
46.             /* Wait data receive succes */
47.    while(!INTC_GetPendingIRQ(IICA_IRQn));
48.    INTC_ClearPendingIRQ(IICA_IRQn);
49.
50.             /* copy the receive data to target memory */
51.    *Data++ = IICA->IICA0;
52.
53.   } while(--Len);
54.
55.         /* STOP condition is generated */
56.         I2CA_GenerateSTOP();
57.  }
58.
59.  return status;
60. }
```

对于 IICA 从机读取数据，传入形参为：设备地址、寄存器地址，读取数据，以及读取数据的长度；遵循标准的 IICA 读协议；起始信号->写设备地址->寄存器地址 ->restart ->读设备地址->读取数据->停止信号；

2.1.3. IICA 主机中断收/发函数

对于 BAT32G137 系列的 IICA，中断收发函数，在中断服务函数中进行数据处理；

中断发送函数 IICA_MasterWrite

```
1.  int IICA_MasterWrite(uint8_t adr, uint8_t *tx_buf, uint16_t tx_num,
    uint8_t wait)
2.  {
3.   I2CA_Status status = OK;
4.
5.   if (I2CA_GetFlagStaus(I2CA_STATUS_BUSBSY) != RESET)
6.   {
7.    status = BUSY;
8.   }
9.   else
10.  {
11.   I2CA_GenerateSTART();
12.   /* Wait */
13.   while (wait--)
14.   {
15.    ;
16.   }
17.   if (RESET == I2CA_GetFlagStaus(I2CA_STATUS_STD))   //if start cond
    ition does not occur
18.   {
19.    status = NSTART;
20.   }
21.   /* Set parameter */
22.   pData.data = tx_buf;
23.   pData.len = tx_num;
24.   pData.flag = INT_IDLE;
25.   g_iica_tx_end = 0;
26.
27.   adr &= (uint8_t)0xFE;    // set write mode
28.   IICA->IICA0 = adr;      // write address
29.  }
30.  return (status);
31. }
```

对于中断发送函数，其功能在产生起始信号后，将数据给到中断服务函数，通过发送中断服务发送出去，在中断中处理检查应答信号。

中断接收函数 IICA_MasterReceive

```
1.  int IICA_MasterReceive(uint8_t adr, uint8_t * const rx_buf, uint16_t
     rx_num, uint8_t wait)
2.  {
3.   I2CA_Status status = OK;
```

```
4.
5.
6.   INTC_DisableIRQ(IICA_IRQn);   // disable INTIICA0 interrupt
7.
8.   if (I2CA_GetFlagStaus(I2CA_STATUS_BUSBSY) != RESET)
9.   {
10.   /* Check bus busy */
11.   INTC_EnableIRQ(IICA_IRQn);   // enable INTIICA0 interrupt
12.   status = BUSY;
13.  }
14.  else
15.  {
16.   I2CA_GenerateSTART();     //generate a start condition
17.   INTC_EnableIRQ(IICA_IRQn);   // enable INTIICA0 interrupt
18.   /* Wait */
19.   while (wait--)
20.   {
21.    ;
22.   }
23.   if (RESET == I2CA_GetFlagStaus(I2CA_STATUS_STD))   //check no star
    t condition occurs
24.   {
25.    status = NSTART;
26.   }
27.
28.   /* Set parameter */
29.   pData.data = rx_buf;
30.   pData.len = rx_num;
31.   pData.flag = INT_IDLE;
32.
33.   g_iica_rx_end = 0;
34.   g_iica_rx_cnt = 0;
35.   adr |= (uint8_t)0x01;       // set receive mode
36.   IICA->IICA0 = adr;       // write address
37.  }
38.     return (status);
39. }
```

对于 IICA 中断接收函数，通过中断方式，在中断服务函数将数据读取到指定 buffer 中；不同于 IICA 轮询方式读取；其不是标准的 iic 读协议；其只是读取部分：起始信号->读设备地址->读取数据->结束信号；

2.1.5. IICA 从机轮询收/发函数

对于串行 IICA 从机接收函数，接收地址和本站地址相同时候，就通过硬件给主控发送 ACK ，注意在从机解除等待时，必须将 IICA 置 0XFF；

接收函数：I2CA_Slave_ReceiveData

```
1.  I2CA_Status I2CA_Slave_ReceiveData(uint8_t *Data, uint16_t Size, uin
    t16_t *Len)
2.  {
3.   I2CA_Status status = OK;
4.      uint8_t Address_Match = 0;
5.      uint16_t Recv_Length = 0;
6.
7.      while(1)
8.      {
9.          while(!INTC_GetPendingIRQ(IICA_IRQn));
10.         INTC_ClearPendingIRQ(IICA_IRQn);
11.
12.         /* Check stop condition */
13.         if (IICA->IICS0 & IICA_IICS0_SPD_Msk)
14.         {
15.             /* Disable I2CA */
16.             IICA->IICCTL00 &= ~IICA_IICCTL00_SPIE_Msk;
17.             break;
18.         }
19.         else
20.         {
21.    /* When the address match is success */
22.             if (!Address_Match && (IICA->IICS0 & IICA_IICS0_COI_Msk)
    )
23.             {
24.                 Address_Match = 1;
25.
26.     /* SPIE0 = 1: enable I2CA */
27.     IICA->IICCTL00 |= IICA_IICCTL00_SPIE_Msk;
28.
29.     /* interrupt request is generated at the eighth clock's falling
    edge */
30.                 IICA->IICCTL00 &= ~IICA_IICCTL00_WTIM_Msk;
31.
32.     /* cancel wait */
33.                 IICA->IICCTL00 |= IICA_IICCTL00_WREL_Msk;
34.             }
35.             else
```

```
    {
    if (Recv_Length < Size)
    {
     *Data = IICA->IICA0;
     Data++;
     Recv_Length++;

     /* WREL0 = 1U: cancel wait */
     IICA->IICCTL00 |= IICA_IICCTL00_WREL_Msk;
     if(Size == Recv_Length)
     {
      /* WTIM0 = 1:  interrupt request is generated at the ninth clock's falling edge */
      IICA->IICCTL00 |= IICA_IICCTL00_WTIM_Msk;
                          break;
     }
    }
    else
    {
     /* When the receive data length exceed the hope length,we shoule not write data to the receive buffer */
     /* WTIM0 = 1:  interrupt request is generated at the ninth clock's falling edge */
     IICA->IICCTL00 |= IICA_IICCTL00_WTIM_Msk;

     /* WREL0 = 1U: cancel wait */
     IICA->IICCTL00 |=  IICA_IICCTL00_WREL_Msk;
    }
            }
        }
    }

 /* Set the receive data length */
 *Len = Recv_Length;

 return status;
NewState)
```

从机接收函数，形参为需要接收的 buf，buf 大小，以及接收到数据的实际长度；

IICA 作为从机轮询发送函数 I2CA_Slave_TransmitData

```
1.  I2CA_Status I2CA_Slave_TransmitData(uint8_t *Reg, uint8_t *Data, uin
    t16_t Size)
2.  {
3.   I2CA_Status status = OK;
4.   uint8_t Stop_count = 0;
5.      uint8_t Address_Match = 0;
6.      uint16_t Send_Length = 0;
7.
8.      while(1)
9.      {
10.         while(!INTC_GetPendingIRQ(IICA_IRQn));
11.         INTC_ClearPendingIRQ(IICA_IRQn);
12.
13.         /* Check stop condition */
14.         if (IICA->IICS0 & IICA_IICS0_SPD_Msk)
15.         {
16.    Stop_count++;
17.
18.    /* To avoid the repeated stop condition */
19.    if (Stop_count == 2)
20.    {
21.     /* Disable I2CA */
22.     IICA->IICCTL00 &= ~IICA_IICCTL00_SPIE_Msk;
23.     break;
24.    }
25.         }
26.         else
27.         {
28.    /* When the address match is success */
29.             if ((Address_Match < 2) && (IICA->IICS0 & IICA_IICS0_COI
    _Msk))  //接收地址和自身从机地址一致
30.             {
31.                 Address_Match ++;
32.
33.     if (Address_Match == 1)
34.     {
35.      IICA->SVA0 |= 0x01;    //第一次先读
36.     }
37.     else if (Address_Match == 2)
38.     {
39.      *Reg = IICA->IICA0;
40.      IICA->SVA0 &= ~0x01;
41.     }
42.
```

```
43.     if (!(IICA->IICCTL00 & IICA_IICCTL00_SPIE_Msk))
44.     {
45.      /* SPIE0 = 1: enable I2CA */
46.      IICA->IICCTL00 |= IICA_IICCTL00_SPIE_Msk;
47.     }
48.
49.     /* Sending or receiving */
50.     if (IICA->IICS0 & IICA_IICS0_TRC_Msk)    //send status
51.     {
52.      IICA->IICCTL00 |= IICA_IICCTL00_WTIM_Msk; //在输入 9 个时钟，将时
    钟置为低电平
53.      IICA->IICA0 = *Data;
54.      Data++;
55.      Send_Length++;
56.     }
57.     else
58.     {
59.      IICA->IICCTL00 |=  IICA_IICCTL00_ACKE_Msk;   /* ACKE0 = 1U: ena
    ble acknowledgment */
60.      IICA->IICCTL00 &= ~IICA_IICCTL00_WTIM_Msk;    // interrupt requ
    est is generated at the eighth clock's falling edge
61.      IICA->IICCTL00 |= IICA_IICCTL00_WREL_Msk;     // cancel wait
62.     }
63.             }
64.             else
65.             {
66.     /* Sending status */
67.     if (IICA->IICS0 & IICA_IICS0_TRC_Msk)
68.     {
69.      if (Send_Length == Size)
70.      {
71.       IICA->IICA0 = 0xFF;
72.
73.      }
74.      else
75.      {
76.       /* Write data to IICA0 register to send it */
77.       IICA->IICA0 = *Data;
78.       Data++;
79.       Send_Length++;
80.      }
81.     }
82.             }
83.         }
```

```
84.  }
85.
86.  return status;
87. }
```

当 IICA 作为从机时候，发送函数对应的是主机的读取函数；一般来说，主机读取函数，包括：一次写设备、写寄存器、一次读设备、读取数据；相对应的从机发送数据时候需要：读设备地址、读寄存器、发送设备地址、发送数据；

2.1.6. IICA 从机中断收/发函数

```
1.  /*************************************************
2.  * Function Name: IICA0_SlaveReceive
3.  * @brief  This function receives data as slave mode.
4.  * @param  adr - send address
5.  * @param  tx_buf - receive buffer pointer
6.  * @param  rx_num - buffer size
7.  * @return None
8.  *************************************************/
9.  void IICA_SlaveReceive(uint8_t adr, uint8_t * const rx_buf, uint16_t
     rx_num)
10. {
11.     g_iica_rx_end = 0;
12.     g_iica_rx_len = 0;
13.     g_iica_rx_cnt = rx_num;
14.     pData.len = 0;
15.     pData.data = rx_buf;
16.     pData.flag = INT_RX;
17.     IICA->SVA0 = adr; /* slave address */
18. }
```

对于从机接收函数，接收函数在中断函数中处理；IICA_SlaveReceive 函数只是将接收到数据传到 rx_buf 中 .同样，中断发送函数接口：

```
1.  /*************************************************
2.  * Function Name: IICA0_SlaveSend
3.  * @brief  This function sends data as slave mode.
4.  * @param  adr - send address
5.  * @param  tx_buf - transfer buffer pointer
6.  * @param  tx_num - buffer size
7.  * @return None
8.  *************************************************/
9.  void IICA_SlaveSend(uint8_t adr, uint8_t * const tx_buf, uint16_t tx
    _num)
```

```
{
    g_iica_tx_end = 0;
    pData.len = tx_num;
    pData.data = tx_buf;
    pData.flag = INT_TX;
    IICA->SVA0 = adr; /* slave address */
}
```

IICA 从机中断发送函数，也是将数据送到中断服务函数中，通过中断发送出去；

2.1.7. IICA 获取运行状态

```
/**
  * @brief  Checks whether the specified I2CA flag is set or not.
  * @param   I2CA_FLAG: specifies the flag to check.
  *          This parameter can be one of the following values:
  *            @arg I2CA_STATUS_SPD : the STOP condition check status.
  *            @arg I2CA_STATUS_STD : the START condition check status.
  *            @arg I2CA_STATUS_ACK : the ACK condition check status.
  *            @arg I2CA_STATUS_TRC : bus status in transmit(is 1) or receive(is 0) mode.
  *            @arg I2CA_STATUS_COI : check the address is matched.
  *            @arg I2CA_STATUS_EXC : the extend code received or not status.
  *            @arg I2CA_STATUS_ALD : the arbitrate success(is 0) or fail(is 1).
  *            @arg I2CA_STATUS_MSTS: to display the communication role is master(is 1) or slave(is 0).
  *            @arg I2CA_STATUS_BUSBSY: I2C bus is busy or idle.
  *            @arg I2CA_STATUS_STCF: Generate START condition or not.
  * @retval The new state of I2CA_FLAG (SET or RESET).
  */
FlagStatus I2CA_GetFlagStaus(uint16_t I2CA_FLAG)
```

根据 I2CA_FLAG 获取 IICA 指定的接收状态

2.1.8. 关闭 IICA 外设

```
void I2CA_DeInit(void)
```

将 I2CA 外设关闭，包括关闭其运行时钟，释放 IIC 所占用通道

3.3. IICA 主机驱动 EEPROM AT24C02 例程示例

在 main 函数中演示驱动 AT24C02 驱动程序，向 AT24C02 设备写数据，同时将所写的数据读取出来；使用了轮询方式和中断读取 2 种方式；

```
43      uint32_t msCnt;     // count value of 1ms
44      static unsigned char ledon = 1;
45      uint32_t err;
46      uint32_t i=0,j=100;;
47      uint8_t tx_buf[3]={0xAb,0x55,0x5A};
48      uint8_t tx_buf1[4]={0x01,0xA5,0x55,0x5A};
49
50  //-----------------------------------------------------------------------
51  // Systick setting
52  //-----------------------------------------------------------------------
53      SystemCoreClockUpdate();
54      msCnt = SystemCoreClock / 1000;
55      SysTick_Config(msCnt);
56      delay_init(SystemCoreClock); //延时初始化
57
58      Uart0_Init(19200);
59      Iica0_Init();
60      printf("IICA0 Start\n");
61  #ifdef USING_24C02
62      #ifdef I2CA_USING_POLLING/*offer two ways to driver 24C02*/
63
64      I2CA_Master_WriteData(SLVADDR, 0x01, tx_buf,3);
65      delayMS(20);
66      I2CA_Master_ReadData(SLVADDR,0x01,rx_buf,3);
67      delayMS(2);
68      #else
69      IICA_MasterWrite(SLVADDR,tx_buf1,4,20);
70      while(g_iica_tx_end == 0);
71
72      delayMS(20);
73
74      IICA_MasterWrite(SLVADDR,tx_buf1,1,20);
75      while(g_iica_tx_end == 0);
76      IICA_MasterReceive(SLVADDR,rx_buf, 3, 20);
77      while(g_iica_rx_end == 0);
78      delayMS(20);
79
80      #endif
81  #else
```

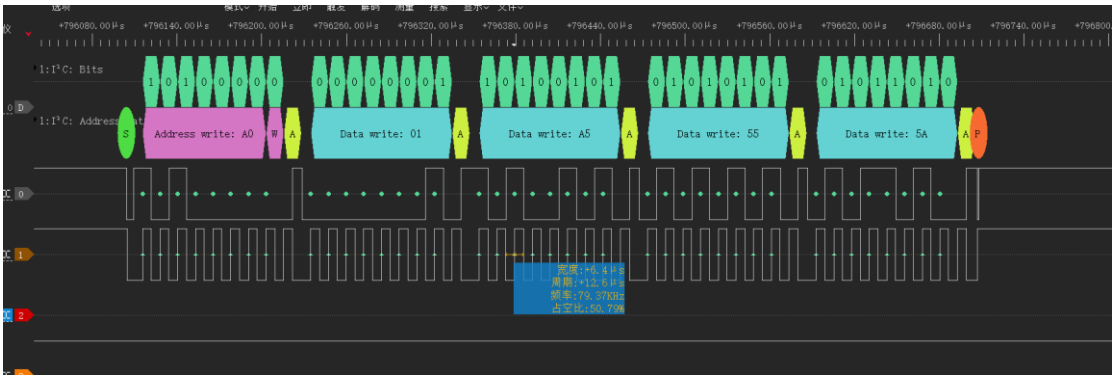
红色框为轮询方式读写 AT24C02，绿色框图为中断方式读写；通过定义宏选择使用方式。

3.4. IICA 主机和 IICA 从机进行数据回环测试

```
75      while(g_iica_tx_end == 0);
76      IICA_MasterReceive(SLVADDR,rx_buf, 3, 20);
77      while(g_iica_rx_end == 0);
78      delayMS(20);
79
80      #endif
81  #else
82      while(1)
83      {
84          IICA_MasterWrite(SLVADDR,tx_buf1,4,20);
85          while(g_iica_tx_end == 0);
86
87          delayMS(20);
88
89          IICA_MasterWrite(SLVADDR,tx_buf1,1,20);
90          while(g_iica_tx_end == 0);
91          IICA_MasterReceive(SLVADDR,rx_buf, 3, 20);
92          while(g_iica_rx_end == 0);
93          delayMS(20);
94      }
95  #endif
96
```

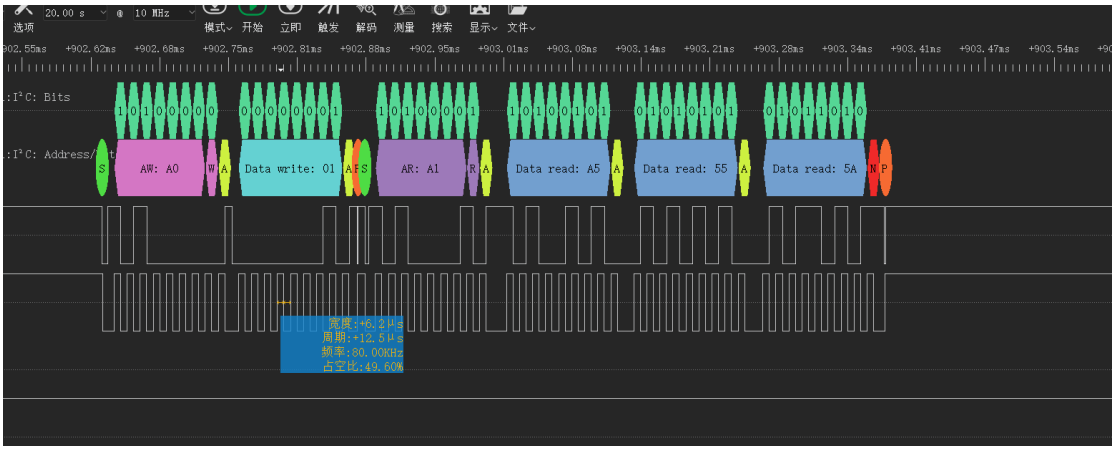
4. 示例演示

IIC 作为主机，读写 24C02



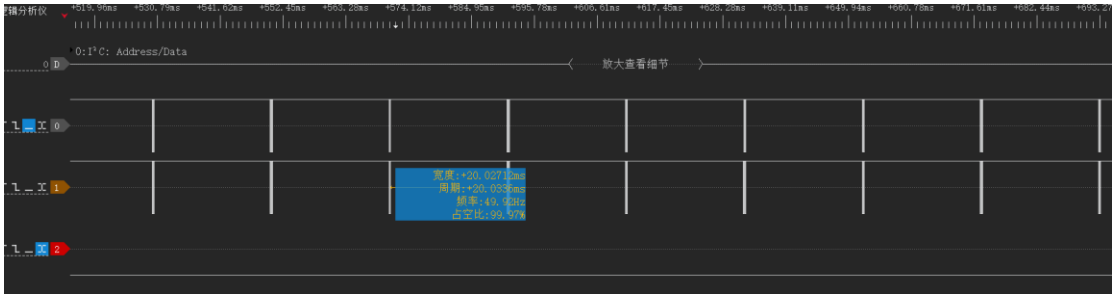
图一 写时序

注释：24C02 设备地址为 0XA0，向寄存器 0X01 地址写 3 位数据 0XA5,0X55,0X5A；

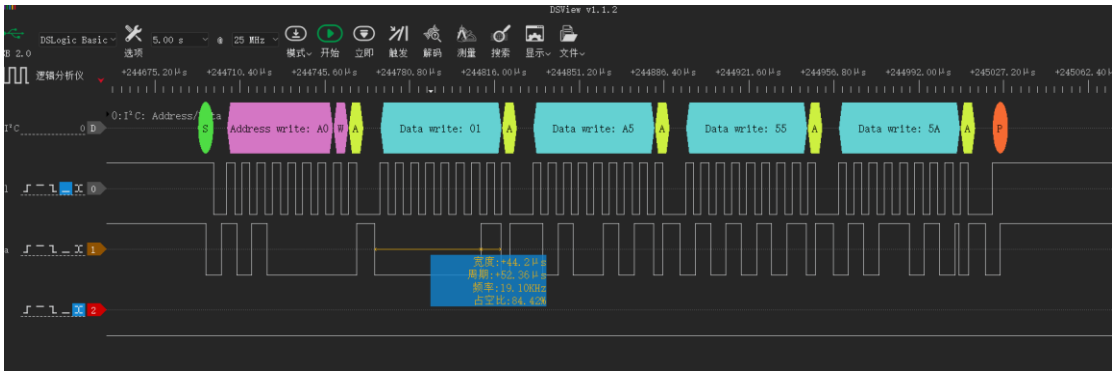


图二 读时序

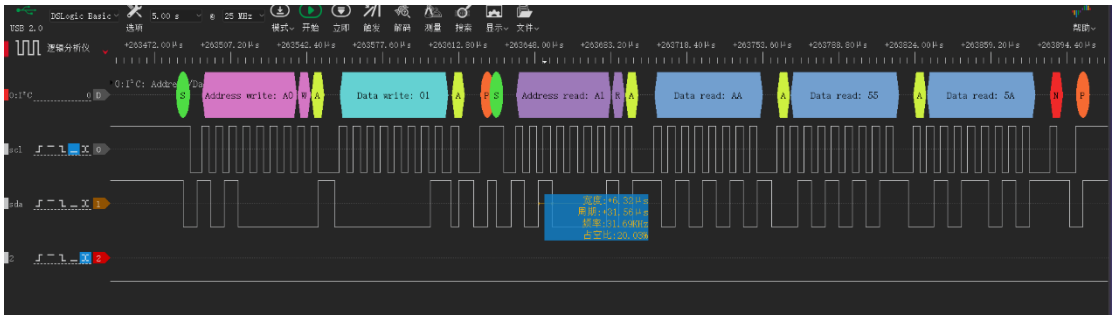
注释：在读时候，先要写设备地址，然后从设备地址读取 0X01 寄存器地址读取 3 位数据



图三 IICA 主机和 IICA 从机回环测试



图四 IICA 回环中写数据



图五 IICA 回环中读数据