利用 MSP430 和 CCS5.0 学习 I2C 和 SPI 总线

制作小组：简易电子书组

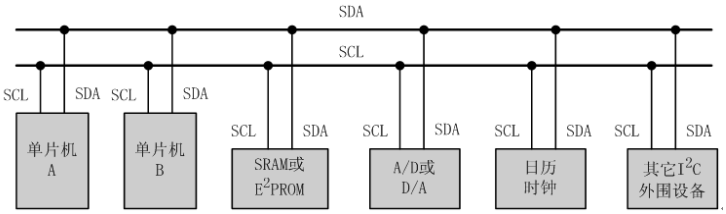
小组成员：段经璞，赵冰洁，张琢

1. I2C 总线的结构

1.1 I2C 串行总线概述

I2C 总线是 PHLIPS 公司推出的一种串行总线，是具备多主机系统所需的包括总线裁决和高低速器件同步功能的高性能串行总线。

I2C 总线只有两根双向信号线。一根是数据线 SDA，另一根是时钟线 SCL。



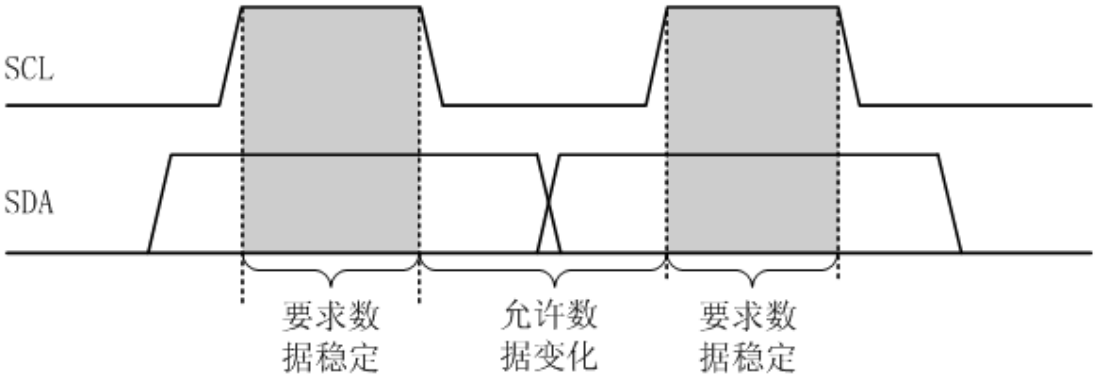
每个接到 I2C 总线上的器件都有唯一的地址。主机与其它器件间的数据传送可以是由主机发送数据到其它器件，这时主机即为发送器。由总线上接收数据的器件则为接收器。

在多主机系统中，可能同时有几个主机企图启动总线传送数据。为了避免混乱， I2C 总线要通过总线仲裁，以决定由哪一台主机控制总线。

1.2 I2C 总线的数据传送

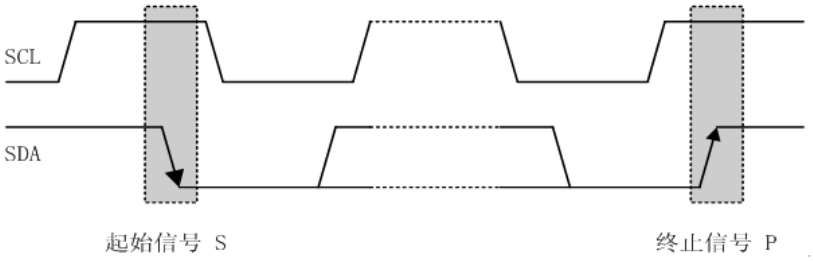
一、数据位的有效性规定

I2C 总线进行数据传送时，时钟信号为高电平期间，数据线上的数据必须保持稳定，只有在时钟线上的信号为低电平期间，数据线上的高电平或低电平状态才允许变化。



二、起始和终止信号

SCL 线为高电平期间，SDA 线由高电平向低电平的变化表示起始信号；SCL 线为高电平期间，SDA 线由低电平向高电平的变化表示终止信号。



起始和终止信号都是由主机发出的，在起始信号产生后，总线就处于被占用的状态；在终止信号产生后，总线就处于空闲状态。

连接到 I2C 总线上的器件，若具有 I2C 总线的硬件接口，则很容易检测到起始和终止信号。对于不具备 I2C 总线硬件接口的有些单片机来说，为了检测起始和终止信号，必须保证在每个时钟周期内对数据线 SDA 采样两次。

接收器件收到一个完整的数据字节后，有可能需要完成一些其它工作，如处理内部中断服务等，可能无法立刻接收下一个字节，这时接收器件可以将 SCL 线拉成低电平，从而使主机处于等待状态。直到接收器件准备好接收下一个字节时，再释放 SCL 线使之为高电平，从而使数据传送可以继续进行。

由于某种原因从机不对主机寻址信号应答时（如从机正在进行实时性的处理工作而无法接收总线上的数据），它必须将数据线置于高电平，而由主机产生一个终止信号以结束总线的数据传送。

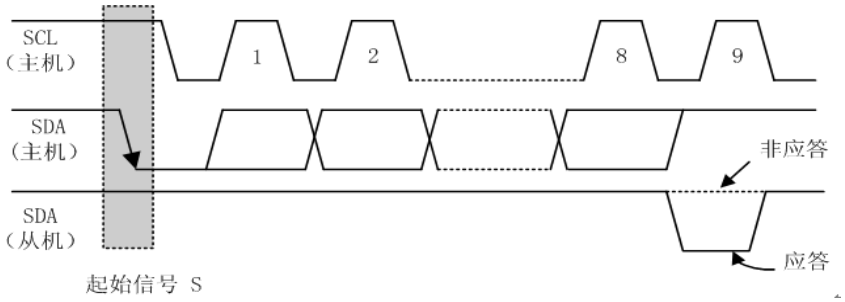
如果从机对主机进行了应答，但在数据传送一段时间后无法继续接收更多的数据时，从机可以通过对无法接收的第一个数据字节的“非应答”通知主机，主机则应发出终止信号以结束数据的继续传送。

当主机接收数据时，它收到最后一个数据字节后，必须向从机发出一个结束传送的信号。这个信号是由对从机的“非应答”来实现的。然后，从机释放 SDA 线，以允许主机产生终止信号。

三、数据传送格式

（1）字节传送与应答

每一个字节必须保证是 8 位长度。数据传送时，先传送最高位（MSB），每一个被传送的字节后面都必须跟随一位应答位（即一帧共有 9 位）。



（2）数据帧格式

I2C 总线上传送的数据信号是广义的，既包括地址信号，又包括真正的数据信号。

在起始信号后必须传送一个从机的地址（7 位），第 8 位是数据的传送方向位（R/），用“0”表示主机发送数据（T），“1”表示主机接收数据（R）。每次数据传送总是由主机产生的终止信号结束。但是，若主机希望继续占用总线进行新的数据传送，则可以不产生终止信号，

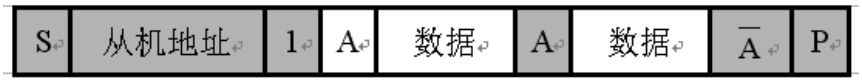
马上再次发出起始信号对另一从机进行寻址。

在总线的一次数据传送过程中，可以有以下几种组合方式：

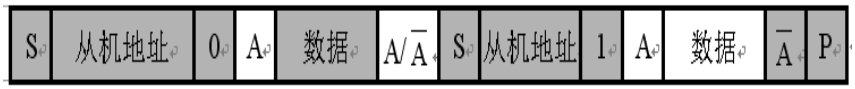
a、主机向从机发送数据，数据传送方向在整个传送过程中不变：

注：有阴影部分表示数据由主机向从机传送，无阴影部分则表示数据由从机向主机传送。A 表示应答， 表示非应答（高电平）。S 表示起始信号，P 表示终止信号。

b、主机在第一个字节后，立即由从机读数据



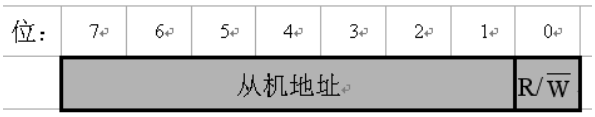
c、在传送过程中，当需要改变传送方向时，起始信号和从机地址都被重复产生一次，但两次读/写方向位正好反相。



四、总线的寻址

I2C 总线协议有明确的规定：采用 7 位的寻址字节（寻址字节是起始信号后的第一个字节）。

（1）寻址字节的位定义



D7～D1 位组成从机的地址。D0 位是数据传送方向位，为“0”时表示主机向从机写数据，为“1”时表示主机由从机读数据

主机发送地址时，总线上的每个从机都将这 7 位地址码与自己的地址进行比较，如果相同，则认为自己正被主机寻址，根据 R/位将自己确定为发送器或接收器。

从机的地址由固定部分和可编程部分组成。在一个系统中可能希望接入多个相同的从机，从机地址中可编程部分决定了可接入总线该类器件的最大数目。如一个从机的 7 位寻址位有 4 位是固定位，3 位是可编程位，这时仅能寻址 8 个同样的器件，即可以有 8 个同样的器件接入到该 I2C 总线系统中。

（2）寻址字节中的特殊地址

固定地址编号 0000 和 1111 已被保留作为特殊用途。

表 9-1 I²C 总线特殊地址表

地 址 位							R/$\overline{W}$	意 义
0	0	0	0	0	0	0	0	通用呼叫地址
0	0	0	0	0	0	0	1	起始字节
0	0	0	0	0	0	1	X	CBUS 地址
0	0	0	0	0	1	0	X	为不同总线的保留地址
0	0	0	0	0	1	1	X	保留
0	0	0	0	1	X	X	X	
1	1	1	1	1	X	X	X	
1	1	1	1	0	X	X	X	十位从机地址

起始信号后的第一字节的 8 位为“0000 0000”时，称为通用呼叫地址。通用呼叫地址的用意在第二字节中加以说明。格式为：

第一字节（通用呼叫地址）									第二字节							LSB	
0	0	0	0	0	0	0	0	A	×	×	×	×	×	×	×	B	A

第二字节为 06H 时，所有能响应通用呼叫地址的从机器件复位，并由硬件装入从机地址的可编程部分。能响应命令的从机器件复位时不拉低 SDA 和 SCL 线，以免堵塞总线。

第二字节为 04H 时，所有能响应通用呼叫地址并通过硬件来定义其可编程地址的从机器件将锁定地址中的可编程位，但不进行复位。

如果第二字节的方向位 B 为“1”，则这两个字节命令称为硬件通用呼叫命令。

在这第二字节的高 7 位说明自己的地址。接在总线上的智能器件，如单片机或其他微处理器能识别这个地址，并与之传送数据。硬件主器件作为从机使用时，也用这个地址作为从机地址。格式为：

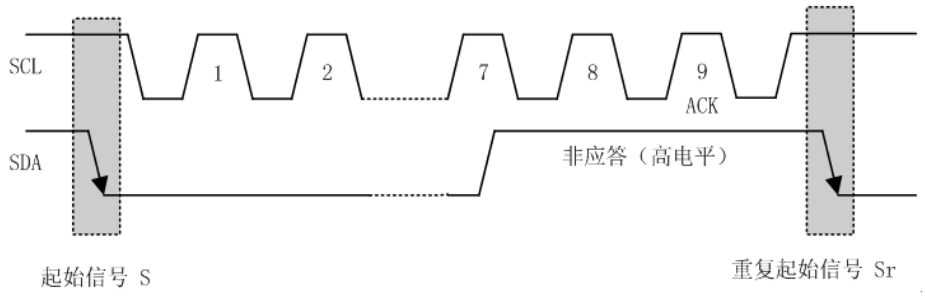
S	0000 0000	A	主机地址	1	A	数据	A	数据	A	P

在系统中另一种选择可能是系统复位时硬件主机器件工作在从机接收器方式，这时由系统中的主机先告诉硬件主机器件数据应送往的从机器件地址，当硬件主机器件要发送数据时就可以直接向指定从机器件发送数据了。

（3）起始字节

起始字节是提供给没有 I2C 总线接口的单片机查询 I2C 总线时使用的特殊字节。

不具备 I2C 总线接口的单片机，则必须通过软件不断地检测总线，以便及时地响应总线的请求。单片机的速度与硬件接口器件的速度就出现了较大的差别，为此，I2C 总线上的数据传送要由一个较长的起始过程加以引导。



引导过程由起始信号、起始字节、应答位、重复起始信号（Sr）组成。

请求访问总线的主机发出起始信号后，发送起始字节（0000 0001），另一个单片机可以用一个比较低的速率采样 SDA 线，直到检测到起始字节中的 7 个“0”中的一个为止。在检测到 SDA 线上的高电平后，单片机就可以用较高的采样速率，以便寻找作为同步信号使用的第二个起始信号 Sr。

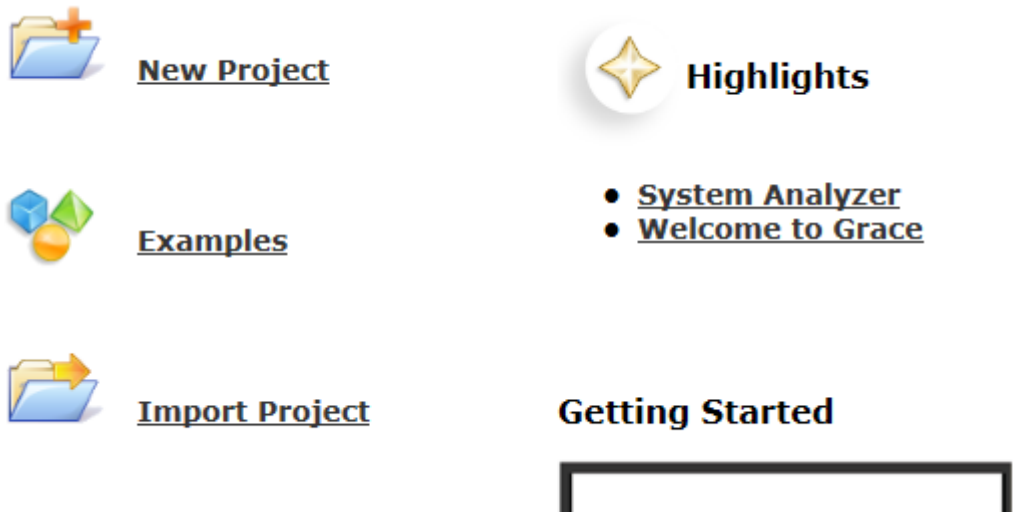
在起始信号后的应答时钟脉冲仅仅是为了和总线所使用的格式一致，并不要求器件在这个脉冲期间作应答。

2. 利用 MSP430LaunchPad 学习 I2C 总线

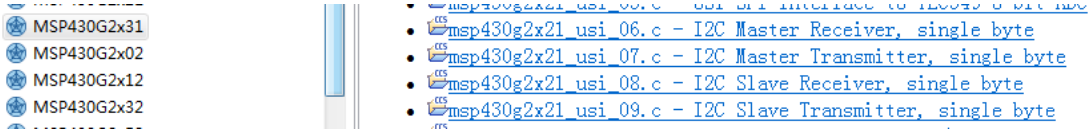
在 CCS5.0 中，提供了丰富的范例程序，利用 LauchPad 所使用的 G2231 芯片具有 I2C 总线控制器，因此就可以快速的进行 I2C 总线的实验，现在让我们看一下利用 LauchPad 学习使用 I2C 总线的全过程。

1、打开 CCS，建立新的工作空间，并将 lauchpad 连接到电脑上。

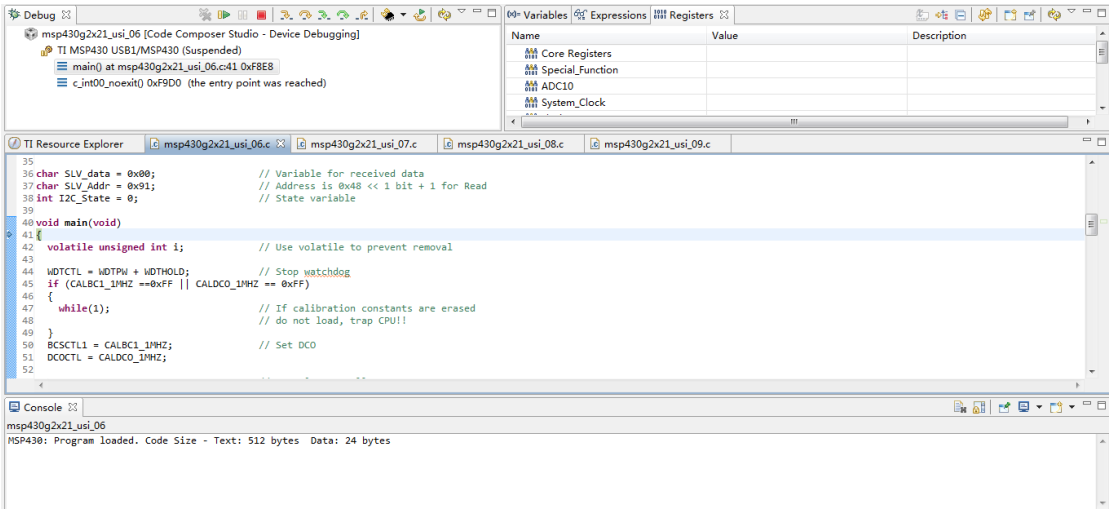
2、在欢迎界面中点击 Example，进入 CCS5.0 自带的范例程序，其界面如下：



3、找到 MSP430G2x31 系列的范例程序，并打开 I2C Master Receive, Master Transmit, Slave Receive, Slave Transmit 四个范例程序：



4、选择 I2C Master Receiver 程序，并在 Run 菜单里点击 Debug，程序将自动加载到 LauchPad 中运行，程序加载后的调试界面如下：



5、现在程序处于单步运行状态，按 F5，F6 可以让程序向下运行，并且可以设置断点，让程序直接运行到断点处。

6、程序运行到截图位置处将进入 I2C 中断，并进行 I2C Master Receive 操作：

```
  while(1)
  {
    USICTL1 |= USIIFG;                  // Set flag and start communication
    LPM0;                               // CPU off, await USI interrupt
    _NOP();                             // Used for IAR
    for (i = 0; i < 5000; i++);         // Dummy delay between communication cycles
  }
}
```

7、I2C Master Receive 中断处理程序如下：

```
#pragma vector = USI_VECTOR
__interrupt void USI_TXRX (void)
{
  switch(I2C_State)
    {
      case 0: // Generate Start Condition & send address to slave
              P1OUT |= 0x01;            // LED on: sequence start
              USISRL = 0x00;            // Generate Start Condition...
              USICTL0 |= USIGE+USIOE;
              USICTL0 &= ~USIGE;
              USISRL = SLV_Addr;        // ... and transmit address, R/W = 1
              USICNT = (USICNT & 0xE0) + 0x08; // Bit counter = 8, TX Address
              I2C_State = 2;            // Go to next state: receive address (N)Ack
              break;

      case 2: // Receive Address Ack/Nack bit
              USICTL0 &= ~USIOE;        // SDA = input
              USICNT |= 0x01;           // Bit counter = 1, receive (N)Ack bit
```

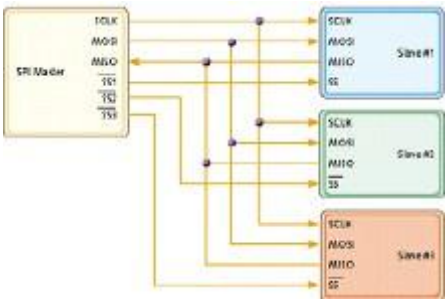
8、依次进行 I2C Master Transmit, I2C Slave Receive, I2C Slave Transmit 程序的运行。如果能取得两个 launchpad，它们就可以进行 I2C 通信了。

3. SPI 总线结构简介

SPI(Serial Peripheral Interface--串行外设接口)总线系统是一种同步串行外设接口，它可以使 MCU 与各种外围设备以串行方式进行通信以交换信息。

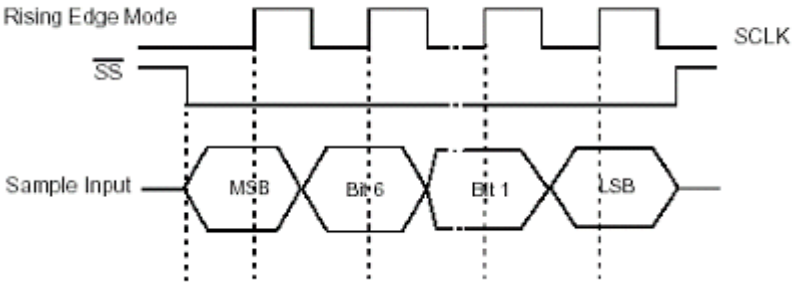
该接口一般使用 4 条线：串行时钟线（SCLK）、主机输入/从机输出数据线 MISO、主机输出/从机输入数据线 MOSI 和低电平有效的从机选择线 CS。

在点对点的通信中，SPI 接口不需要进行寻址操作，且为全双工通信，显得简单高效。



在多个从器件的系统中，每个从器件需要独立的使能信号，硬件上比 I2C 系统要稍微复杂一些。

SPI 接口在内部硬件实际上是两个简单的移位寄存器，传输的数据为 8 位，在主器件产生的从器件使能信号和移位脉冲下，按位传输，高位在前，低位在后。如下图所示，在 SCLK 的下降沿上数据改变，同时一位数据被存入移位寄存器。其时序图如下图所示：



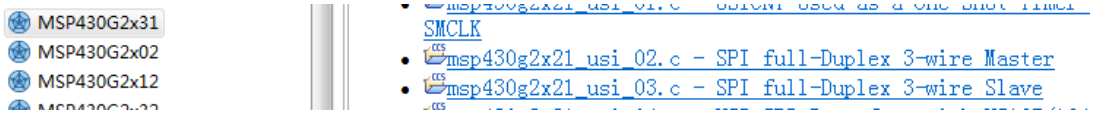
4. 利用 MSP430LaunchPad 学习 SPI 总线

在 Lauchpad 所提供的 MSP430G2231 芯片具有 SPI 控制器，利用 CCS5 中的范例程序就可以迅速的进行 SPI 总线的实验并学习 SPI 总线的使用方法。下面展示利用 CCS5 和 LauchPad 学习 SPI 总线的过程：

1、将 Lauchpad 连接到电脑上，并打开 CCS5。

2、在欢迎界面中点击 Example，进入 CCS 自带的范例程序。

3、找到 G2231 的 SPI 范例程序，如下图所示：



4、打开 SPI 全双工三线主机程序，如下图所示：

```c
#include <msp430g2221.h>


void main(void)
{
  volatile unsigned int i;

  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
  P1OUT =  0x10;                        // P1.4 set, else reset
  P1REN |= 0x10;                        // P1.4 pullup
  P1DIR = 0x01;                         // P1.0 output, else input
  USICTL0 |= USIPE7 +  USIPE6 + USIPE5 + USIMST + USIOE; // Port, SPI master
  USICTL1 |= USIIE;                     // Counter interrupt, flag remains set
  USICKCTL = USIDIV_4 + USISSEL_2;      // /16 SMCLK
  USICTL0 &= ~USISWRST;                 // USI released for operation
  USISRL = P1IN;                        // init-load data

  P1DIR |= 0x04;                        // Reset Slave
  P1DIR &= ~0x04;
  for (i = 0xFFF; i > 0; i--);          // Time for slave to ready
  USICNT = 8;                           // init-load counter
  _BIS_SR(LPM0_bits + GIE);             // Enter LPM0 w/ interrupt
}

// USI interrupt service routine
#pragma vector=USI_VECTOR
__interrupt void universal_serial_interface(void)
{
  if (0x10 & USISRL)
    P1OUT |= 0x01;
  else
    P1OUT &= ~0x01;
  USISRL = P1IN;
  USICNT = 8;                           // re-load counter
}

```

5、点击 Run，并点击 Debug，就可以将程序加载到 LauchPad 上运行，运行结果如下。

6、然后可以将 SPI 从机程序加载到 LauchPad 上运行，如果有两块 Lauchpad，就可以用这两块 LauchPad 进行 SPI 通信。