App Note 3524: Efficient Bit-Banged SPI Port for 8051-Compatible Microcontrollers

A fast SPI port can be bit-banged through GPIO pins and serve as a lower cost alternative to 8051-compatible microcontrollers with SPI ports. The code shown in this application note takes advantage of 8051-specific features to create a fast SPI port using minimal extra code.

Although 8051-compatible microcontrollers with SPI ports are available, a lower cost device with an SPI port bit-banged through GPIO pins often suffices for many applications. The code shown here takes advantage of features specific to the 8051 core to create a fast SPI port with minimal effort. The CPHA, CPOL, and CS_TOGGLE_BETWEEN_BYTES constants in the #define statements initialize macros that tailor the code to the type of SPI port being implemented. The preprocessor performs this code tailoring at compile time rather than at runtime, saving precious clock cycles that would be wasted if decision structures (i.e., regular if-else statements) were used.

The code below includes 8051-specific C commands required to take advantage of the 8051 core's features. Although these commands are compiler specific (Keil µVision v2 Development Tools for 8051 in this case), all "good" C compilers for 8051-compatible devices incorporate similar commands.

Examining the code, PORT_0 is defined as type sfr, which alerts the compiler that this label is an 8051 Special Function Register (SFR). Because this SFR is bit-addressable, the sbit type defines identifiers that reference specific SFR bits to act as SPI port pins. The bdata type used in the spiTmp declaration allows this variable to be placed in special bit-addressable memory within the 8051 core's directly addressable RAM. Again, the sbit type defines identifiers that will reference specific bits in the spiTmp variable.

The bytes to be sent through the SPI port are loaded into the global byte array, spiData. Declaring this variable as global allows the SPI transmit/receive function to access spiData without needing to pass it as a parameter. Declaring it with the data identifier forces the compiler to store the array in the fastest accessible memory (directly addressable memory) inside the 8051 core.

The spiReadWriteBlock function contains the code for the bit-banged SPI port. It efficiently transmits every byte in the spiData array using this SPI port, starting from the last element in

the array to the first. Accessing the array using this reverse order allows a comparison with zero (see code), which translates into faster assembly due to the 8051 instruction set. When the `spiReadWriteBlock` function is complete, bytes read with the SPI port will have replaced the original data in the `spiData` array, again starting from the last element in the array to the first.

Note that the code is optimized to send and receive blocks of data larger than one byte. For single-byte transfers, the looping structure and local variable inside `spiReadWriteBlock` should be removed. (This can be done using the preprocessor.)

When compiled for the Maxim DS89C420/430/440/450 family of 8051-compatible microcontrollers running at 32MHz, this bit-banged SPI port runs at just over 2Mbps, as shown in **Figure 1**. Also, the code requires only two bytes of directly addressable RAM and 139 bytes of flash memory for code space (including SPI port initialization and the main program loop).
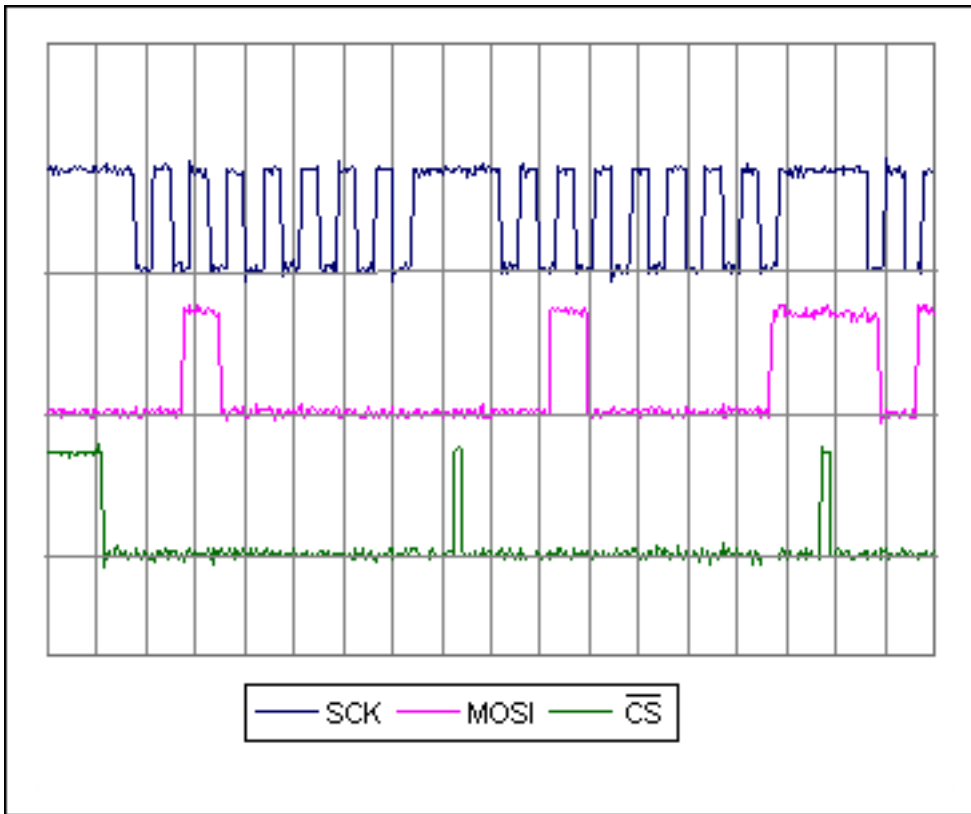

Figure 1. These waveforms represent the output from the bit-banged SPI port when the `CPHA`, `CPOL`, and `CS_TOGGLE_BETWEEN_BYTES` constants are set to 1. This firmware uses bit-addressable memory in the 8051 core to increase the speed of the SPI port.

```
/*
 ************************************************************
 * 8051 Bit-Banged SPI
 *
 * MAXIM INTEGRATED PRODUCTS
 * Author: Paul Holden (Paul_Holden@maximhq.com)
```

```
 *
 *********************************************************
 */

// CONSTANTS -------------------------------------------
#define CPOL                      1  // Set CPOL to 1 or 0
#define CPHA                      1  // Set CPHA to 1 or 0
#define CS_TOGGLE_BETWEEN_BYTES  1  // Set CS toggle
                                     //   0=false  1=true
#define N_OF_SPI_BYTES           3

// MACROS ----------------------------------------------
#if CPHA
   #define SCK_POST
   #if CPOL
      #define SCK_INIT 1
      #define SCK_PRE  SCK=0
      #define SCK_MID  SCK=1
   #else
      #define SCK_INIT 0
      #define SCK_PRE  SCK=1
      #define SCK_MID  SCK=0
   #endif
#else
   #define SCK_PRE
   #if CPOL
      #define SCK_INIT 1
      #define SCK_MID  SCK=0
      #define SCK_POST SCK=1
   #else
      #define SCK_INIT 0
      #define SCK_MID  SCK=1
      #define SCK_POST SCK=0
   #endif
#endif

#if CS_TOGGLE_BETWEEN_BYTES
   #define CS_TOGGLE CS=1;CS=0
#else
   #define CS_TOGGLE
#endif

// PIN DEFINITIONS -------------------------------------
sfr  PORT_0 = 0x80;
sbit CS     = PORT_0 ^ 1;
```

```
sbit SCK     = PORT_0 ^ 2;
sbit MOSI    = PORT_0 ^ 3;
sbit MISO    = PORT_0 ^ 4;

// GLOBAL VARIABLES ----------------------------------------
unsigned char data spiData[N_OF_SPI_BYTES];

// BIT-ADDRESSABLE GLOBAL VARIABLES ----------------------
unsigned char bdata spiTmp;
sbit                spiTmp7 = spiTmp ^ 7;
sbit                spiTmp6 = spiTmp ^ 6;
sbit                spiTmp5 = spiTmp ^ 5;
sbit                spiTmp4 = spiTmp ^ 4;
sbit                spiTmp3 = spiTmp ^ 3;
sbit                spiTmp2 = spiTmp ^ 2;
sbit                spiTmp1 = spiTmp ^ 1;
sbit                spiTmp0 = spiTmp ^ 0;

// FUNCTION PROTOTYPES ------------------------------------
void spiReadWriteBlock(void);

// FUNCTION spiReadWriteByte ------------------------------
//
// Data is transfered starting from spiData[N_OF_SPI_BYTES-1]
// to spiData[0] MSb first. The received data replaces the
// existing data from spiData[N_OF_SPI_BYTES-1] to spiData[0].
//
// NOTE: this function assumes that
//       SCK=SCK_INIT and CS=1
void spiReadWriteBlock(void)
{
   unsigned char data i = N_OF_SPI_BYTES-1;

   CS = 0;
   while(1)
   {
      spiTmp = spiData[i];
      SCK_PRE; MOSI=spiTmp7; SCK_MID; spiTmp7=MISO; SCK_POST; // bit 7
      SCK_PRE; MOSI=spiTmp6; SCK_MID; spiTmp6=MISO; SCK_POST; // bit 6
      SCK_PRE; MOSI=spiTmp5; SCK_MID; spiTmp5=MISO; SCK_POST; // bit 5
      SCK_PRE; MOSI=spiTmp4; SCK_MID; spiTmp4=MISO; SCK_POST; // bit 4
      SCK_PRE; MOSI=spiTmp3; SCK_MID; spiTmp3=MISO; SCK_POST; // bit 3
      SCK_PRE; MOSI=spiTmp2; SCK_MID; spiTmp2=MISO; SCK_POST; // bit 2
      SCK_PRE; MOSI=spiTmp1; SCK_MID; spiTmp1=MISO; SCK_POST; // bit 1
      SCK_PRE; MOSI=spiTmp0; SCK_MID; spiTmp0=MISO; SCK_POST; // bit 0
      spiData[i] = spiTmp;
```

```
        if (i == 0)
            break;
        i--;

        CS_TOGGLE;
    }
    CS = 1;
}

// MAIN ---------------------------------------------------
void main(void)
{
    // 0. Init SPI Pins
    CS  = 1;
    SCK = SCK_INIT;

    // 1. Program Loop...
    while (1)
    {
        spiData[2] = 0x40;
        spiData[1] = 0x41;
        spiData[0] = 0x42;
        spiReadWriteBlock();
    }
}
```

A similar version of this article appeared in the May 2, 2005 issue of *EE Times* magazine.

More Information

DS89C420: QuickView -- Full (PDF) Data Sheet -- Free Samples

DS89C430: QuickView -- Full (PDF) Data Sheet

DS89C440: QuickView -- Full (PDF) Data Sheet

DS89C450: QuickView -- Full (PDF) Data Sheet