Realising the SPI communication in a multiprocessor system

Ákos Székács[*], Tibor Szakáll[**] and Zoltán Hegyközi[***]

[*] Polytechnical Engineering College/Department Of Electronics And Telecommunication, Subotica, Serbia
[**] Polytechnical Engineering College/Department Of Electronics And Telecommunication, Subotica, Serbia
[***] Polytechnical Engineering College/Department Of Electronics And Telecommunication, Subotica, Serbia

akos.szekacs@gmail.com
tibor.szakall@gmail.com
hzoli@gmail.com

***Abstract*—The SPI communication was originally realised for the Fully Configurable Freely Scalable Digital Audio System. First we used Philipses I^2C (Inter IC Communication), but it was too slow, and the number of the connected devices was limited.**

With SPI we can connect as many devices as many pins we have on the main microcontroller. The speed of the communication between ICs is much faster thanks for the Full Duplex proper of the SPI communication.

I. Introduction

Before I could introduce the whole communication, I have to introduce the SPI communication and the situation what led us to realize an SPI communication in a multiprocessor system:

A. *The Situation*

We had a board named MainBox with one microcontroller and with 6 more ICs, which was used in a Fully Configurable Freely Scalable Digital Audio System. The MainBox's function was to receive control data from a PC, mix it to an AES3 digital audio signal and send it forward to the digital amplifiers, I^2C communication was used, it was satisfying until we were forced to use a 5 Channel Digital Audio Mixer instead of a MainBox. The mixer includes the MainBox 5 times. The output of the first MainBox was connected serially to the input of the second MainBox, the second MainBox's output was connected to the third MainBox's input, the third MainBox's output to the fourth MainBox's input, the fourth MainBox's output to the fifth MainBox's input and the output of the fifth MainBox was the output what we needed, the mixed audio signal with control data mixed in too.

We had to connect the microcontrollers, but we could not because of the lack of the I^2C communication. I^2C communication uses a bus and every IC connects to it, every IC has its own address (I^2C address). On our board, from one type of IC we had 5 piece. The I^2C address is a 7 bit address, from this seven bits we can change only 2 bits hardwarely (connecting the IC's pins to Vcc or GND). So we had only 4 possible addresses, but 5 ICs plus I^2Cs maximum speed was around 100 Kbps -> I^2C was not enough. We had to use another communication.

The only other communication what was supported by all of the used ICs was SPI. Then came the idea, we have connected the 5 microcontrollers with each other, using SPI, placed one more microcontroller to the board and used it as a master for the SPI communication. So, the micorocontrollers were communicating with each other through the master microcontroller using SPI and every microcontroller communicates with his "own" ICs over I^2C.

B. *The SPI Communication*

The Serial Peripheral Interface Bus (SPI) bus is a synchronous serial data link standard named by Motorola that operates in full duplex mode. Sometimes SPI is called a "four wire" serial bus. There is one master and one or more slave devices in the communication.

Four logic signals are necessary to connect 2 ore more devices with SPI:

- SCLK – Serial Clock (output from master)
- MOSI / SIMO – Master Out Slave In (output from master)

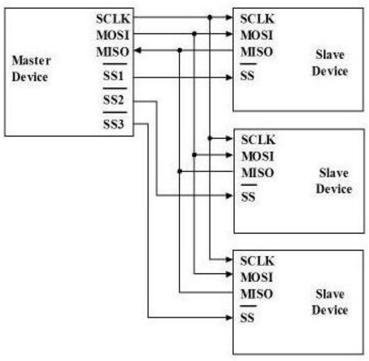

Figure 1. The SPI Bus system with 1 master device and with 3 slave devices

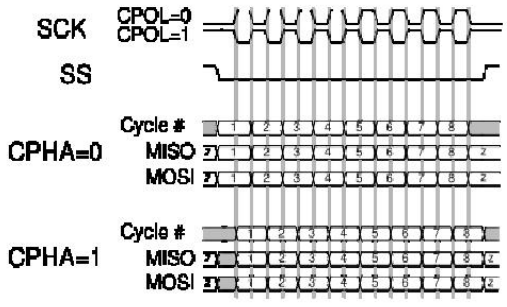

Figure 2. Clock dependencies from CPHA and CPOL

CPOL	CPHA	Active edge
0	0	Rising
0	1	Falling
1	0	Falling
1	1	Rising

Figure 3. CPOL and CPHA setup table

- MISO / SOMI – Master In Slave Out (output from slave)
- SS – Slave Select (active low, output from master)

If there is only one slave device then the SS pin on the slave device can be fixed to logic low state.

If there is 2 or more slave devices in the system, then an independent SS signal is required from the master device for each slave device.

When the master device wants to start a communication it has to set the clocks, that is less then or equal to the slave device's maximum frequency (most commonly from 1 to a few MHz).

SPI communication is a full duplex communication, the master device sends a byte to the desired slave device in the meantime it receives a byte from the slave device. Transmissions may involve any number of clock cycles. When there are no more data to be transmitted, the master device stops toggling its clock. Normally, it then deselects the slave device.

Every slave device on the bus that hasn't been activated using its Slave Select line must disregard the input clock and MOSI signals, and may not drive MISO. The master device selects only one slave at a time.

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. With CPOL and CPHA options we can adjust the clock polarity.

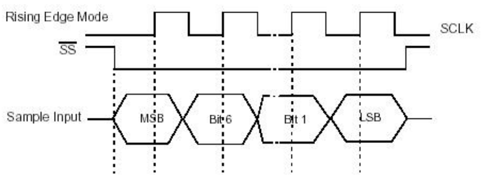

Figure 4. Rising Edge Mode. CPOL = 0; CPHA = 0

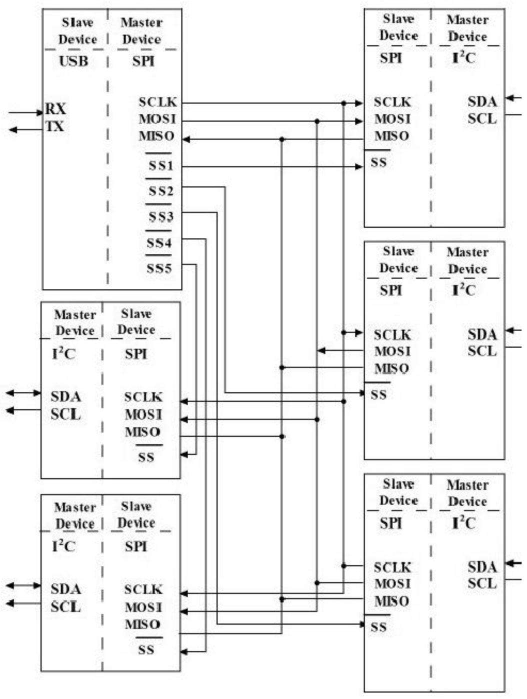

Figure 5. Block Diagram of the Digital Audio Mixer Communication

II. Realising The Communication

A. *The Idea*

Because of I^{2}C's lack, we had two choices to make our board work, we start all over using only the SPI protocol, or we mix the communication protocols on our board.

If we use only SPI, then the communication would be much faster on the whole board, but then we would have to redesign firstly the MainBox board, then the Digital Audio Mixer board, it would have serious consequents such as testing the new board, testing SPI with 30 or more devices etc.

B. *The Plan*

We decided to use SPI with I^{2}C. On Figure 5. we can see the block diagram of the communication, the microcontroller what is master in the SPI communication, communicates with the PC, sends information, about the board state and receives instructions. Interprets the header of the received data, and forwards the information to the desired slave microcontroller. The slave microcontroller interprets the information, using the Audio System Controlling Protocol (ASCP) [3], and sends out information (commands, coefficients, biquad filters etc) over I^{2}C to it's his group of ICs.

C. *The Implementation*

As we can see on Figure 5, the whole system consists of 6 micorocontrollers (Texas Instruments MSP430F1611). Every microcontroller has 2 USART (Universal Synchronous / Asynchronous Receive / Transmit) peripheral interface, named USART0 and USART1.

On the main microcontroller (which is communicating with the PC), USART0 is used to communicate with the

PC via serial communication protocol, when the microcontroller is communicating with the PC then it is in slave mode (the PC with a Virtual Serial Communication via USB starts and ends the communication). The microcontroller's other USART port, the USART1 is used to communicate with the other microcontrollers on the board using SPI communication protocol, in this case (USART1, SPI) the microcontroller is a master device (it generates the clock, starts and ends a communication).

The other microcontrollers on the board using the USART1 interface too, to communicate with the master device via SPI, but they are slaves in the communication, the master device selects the desired microcontroller with the his SS pin, sends the clock, starts and ends the communication. When one of these microcontrollers receives data from the master device over SPI, then switches to I^{2}C and sends the data out using Audio System Controlling Protocol (ASCP) [3] to other IC. I^{2}C module is on the USART0 peripheral interface on the microcontroller. When they switched to I^{2}C communication, then they became master devices.

None of these microcontrollers can communicate I^{2}C and SPI in the same time, because of that, their default communication setup is SPI on USART1 interface. When they received something from the master device via SPI, they switch to I^{2}C mode transmits and receives all the data necessary then switches back to SPI waiting for the master device to start another communication.

III. Code Details

The Master microcontroller's SPI initialization routine:

```
1.  void initMasterSPI ( void )
2.  {
3.  P5SEL |= 0x0F;
4.  P5DIR = 0xFA;
5.  U1CTL = SWRST;
6.  U1TCTL = CKPH + SSEL1 + SSEL0;
7.  UBR01 = 0x03;
8.  UBR11 = 0x00;
9.  UMCTL1 = 0x00;
10. U1CTL = CHAR + SYNC + MM;
11. U1CTL &= ~SWRST;
12. ME2 |= USPIE1;
13. IE2 |= UTXIE1 + URXIE1;
14. }
```

The Slave microcontroller's SPI initialization routine:

```
1.  void initSlaveSPI ( void )
2.  {
3.  P5SEL |=  0x0F;
4.  P5DIR =   0xF4;
5.  U1CTL = SWRST;
6.  U1TCTL = CKPH;
7.  UBR01 = 0x03;
8.  UBR11 = 0x00;
9.  UMCTL1 = 0x00;
10. UCTL1 = CHAR + SYNC + SWRST;
11. UCTL1 &= ~SWRST;
12. ME2 |= USPIE1;
13. IE2 |= UTXIE1 + URXIE1;
14. }
```

IV. Commenting The Code

A. *Master MSP*

The microcontrollers SCLK, MOSI, MISO and SS pins are on the fifth port (P5.0 – SCLK; P5.1 – MOSI; P5.1 – MISO; P5.3 - SS).

On the third row we initialize the MSP's desired pins to act as an SPI interface.

The fourth row is the direction setup for the fifth port (1 is for output, 0 is for input).

On the fifth row we take the MSP to a software reset state by taking 0x01 (SWRST) in to the USART1 Control Register (U1CTL), the other seven bits of this register has to be 0, because the USART1 interface is in reset state, now we can set up all the other control registers of the interface.

In the sixth row we set up the USART1 Transmit Control Register (U1TCTL). SSEL1 with SSEL0 sets up the clock source. In our case it will be SSEL1 = 1 and SSEL0 = 1, it means that the master clock source is the master clock of the MSP, with CKPH we delay UCLK by one half cycle.

Seventh and eighth row: The baud-rate generator uses the content of (UBR01 + UBR11) to set the baud rate. Unpredictable SPI operation occurs if UBR < 2.

In the ninth row we set the Modulation Control Register (UMCTL1) to 0x00, because it is not used in SPI mode.

In the tenth row we write in to the U1CTL again: CHAR – 8 bit character length, SYNC – SPI mode (if 0, then the USART1 is in I^{2}C mode), MM – Master Mode.

In the eleventh row we put 0, only to the LSB of the USART1 register. It means that it is not anymore in software reset mode.

In the next row we enable the SPI mode for USART1 interface.

In the thirteenth row we enable interrupts for receive and transmit.

B. *Slave MSP*

The microcontrollers SCLK, MOSI, MISO and SS pins are on the fifth port (P5.0 – SCLK; P5.1 – MOSI; P5.1 – MISO; P5.3 - SS).

On the third row we initialize the MSP's desired pins to act as an SPI interface.

The fourth row is the direction setup for the fifth port (1 is for output, 0 is for input).

On the fifth row we take the MSP to a software reset state by taking 0x01 (SWRST) in to the USART1 Control Register (U1CTL), the other seven bits of this register has to be 0, because the USART1 interface is in reset state, now we can set up all the other control registers of the interface.

In the sixth row we set up the USART1 Transmit Control Register (U1TCTL). We wont use SSEL1 snd

SSEL0 because it is a slave MSP and the master clock is provided by the master microcontroller.

Seventh and eighth row: The baud-rate generator uses the content of (UBR01 + UBR11) to set the baud rate. Unpredictable SPI operation occurs if UBR < 2.

In the ninth row we set the Modulation Control Register (UMCTL1) to 0x00, because it is not used in SPI mode.

In the tenth row we write in to the U1CTL again: CHAR – 8 bit character length, SYNC – SPI mode (if 0, then the USART1 is in I^2C mode).

In the eleventh row we put 0, only to the LSB of the USART1 register. It means that it is not anymore in software reset mode.

In the next row we enable the SPI mode for USART1 interface.

In the thirteenth row we enable interrupts for receive and transmit.

C. All Together

The slaves default communication is SPI, they are waiting for instructions and data from the master, when they have received it, they switch to I^2C mode by taking 0 to the U1CTL register's SYNC bit, then it can communicate with "his" IC's.

If a slave wants to "tell" something to the master device, then sends an interrupt to the master and the master starts a communication with the desired slave device

V. Final Words

Finally we have set up all of the devices to communicate with the ICs what are in his scope. The communication is working properly to every direction. We have to mention that Texas Instruments MSP430F1611 is a matured microcontroller, the datasheet of it still has some mistakes, a few of them is with SPI communication.

References

[1] Motorola, "MC68HC11 manual,".
[2] Texas Instruments, "MSP430x1xx family users guide,".
[3] Péter Kaszás, Ákos Székács, Tibor Szakáll, "Audio system controlling protocol (ASCP) with AES3," unpublished.
[4] Texas Instruments website, www.ti.com.
[5] Philips's website, www.philips.com.