SPI Implementation Based on M2e Microprocessor

Haifan Zhang[1, a] and Fei Ye[2]

[1]School of Information Science and Technology, Fudan University,200433 Shanghai, China
[2]Camel Microelectronics, Inc., San Jose, 95129 California, United States

Abstract. Normally, the computer reads, writes and other control operations on the M2 board through the UART protocol. Our goal is to achieve control of the M2 board through the SPI protocol. The specific implementation is to use an M2 board (Master M2) as an intermediary to achieve control of the target M2 board (Slave M2). The communication between the user computer and the Master M2 still uses the UART protocol. The SPI protocol is used between the Master M2 and the Slave M2. The read/write and other operations of the Slave M2 board no longer need to occupy the UART resources of the user's computer, which eliminates the inconvenience of connecting two computers at the same time. In the case where data transmission is required for multiple M2 boards at the same time, a solution for improving efficiency and reducing cost is provided.

1 Background

1.1 M2e Microprocessor

M2e microprocessor is a 32-bit mixed-signal processor produced by Camel Microelectronics, Inc. The M2e microprocessor and its storage system (SRAM, ROM, Flash) form the M2e microprocessor system. The microprocessor system and its peripheral subsystems form an on-chip microsystem [1]. M2e microsystem Block Diagram is shown in Figure 1.

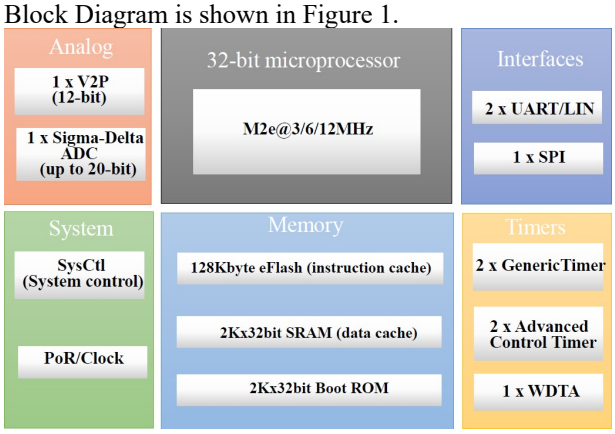

Figure 1. M2e microsystem Block Diagram

M2e microsystem Main features include:

• Processor Unit:

-3/6/12MHz 5-stage pipeline X3- 32bit microprocessor

- MIP-I, II instruction set

• Memory:

- 128 KB on-chip flash programming memory
- 2k x 32-bit data cache (SRAM)
- 2k x32-bit boot ROM

• Serial interfaces:

- Two UART/LIN
- SPI interface up to 3 Mbit/s

• Development support:

- CamelStudio Kit Please submit sources files directly to the conference organizer.

1.2 Serial Peripheral Interface

The SPI module in M2e provide serial I/O compliant to the SPI protocol. It can be configured either as a SPI master device or a SPI slave device. Figure 2 shows the SPI module in the M2e system [2].

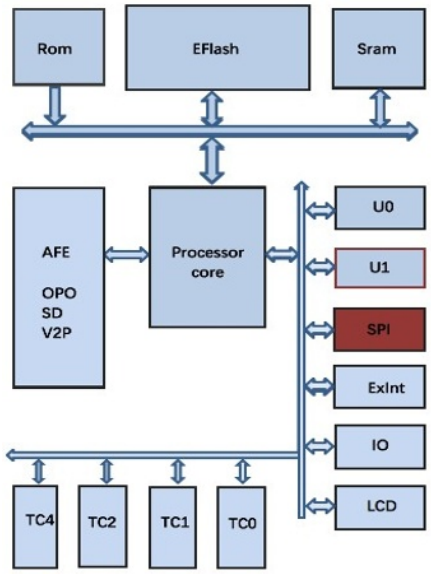

Figure 2. Schematic diagram of SPI module in M2e system

SPI (Serial Peripheral Interface) is a high-speed, full-duplex, synchronous communication bus that occupies only four wires on the pins of the chip, saving the pins of

[a]Corresponding author: 15307130032@fudan.edu.cn

the chip and saving space on the layout of the PCB. For convenience, the SPI works in master-slave mode. This mode usually has one master device and one or more slave devices. It requires at least 4 wires and is common to all SPI-based devices. They are SDI (data input), SDO (data output), SCLK (clock), CS (chip select).

The CS is a control signal selected from whether the chip is selected by the main chip, that is, when only the chip select signal is a predetermined enable signal (high potential or low potential), the master chip is effective for the operation of the slave chip.

1.3 SPI protocol

The SPI implements the MCU to communicate with various peripheral devices in a serial manner. Since the SPI has a clock, its corresponding data transmission is synchronous, which is different from the UART transmission. The result is that the transmission speed of the SPI can be relatively high, almost reaching a rate comparable to the internal clock of the chip [3].

M2's SPI module can be used as a maser device or as a slave device. When the SPI is used as the master device, the SPI clock is provided by M2 and its speed is equivalent to the internal system clock. One byte (8 bits) is transmitted each time as a byte unit.

2 Tasks and system structure

In the Room space of the M2 board, there is an interface program, as shown in Figure 3, which can be used to implement eight functions such as Memory Write, Read, and Jump Address.

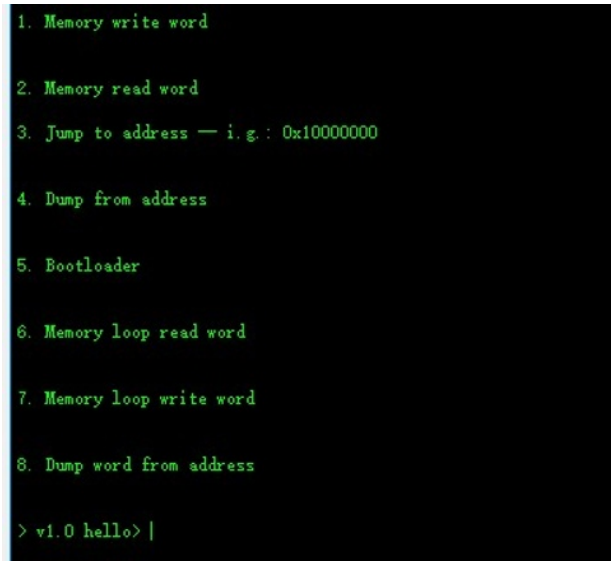

Figure 3. Stored interface program

2.1 Goals and tasks

When the M2 board function is traditionally enabled, the communication protocol for connecting the computer to the UART is used to write to the memory or read the memory. We tried to change the communication method, using the SPI communication protocol between the board and the board, so that the user can use a M2 to do the UART to SPI mediation, realize the memory read and write operation of another M2 board, and no longer pass the UART serial port. In this way, when transmitting data and controlling reading and writing, the trouble of having to connect multiple computers can be saved, and the efficiency of downloading the program on the M2 board can be improved.

2.2 system structure

This project connects the first M2 board through the computer UART serial port, and the first M2 board and the second M2 board are connected by SPI protocol wiring.

When the two boards are running in the user space at the same time, the first board appears as shown in FIG. 3, and the second board can be read and written by the computer, and the interface is displayed through the first board

The function of the first board is to obtain the data hub from the computer UART port and display the read and write status of the second board; the second board is the actually operated M2 board.

In order to facilitate the later debugging, we make the code also output the data acquisition status of the second board, so that the actual transmission and reception of the second board can also be seen after connecting the computer. The block diagram is shown in Figure 4.

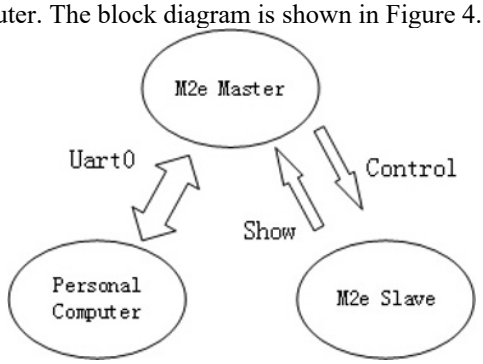

Figure 4. system structure

3 Hardware solution

The hardware equipment of this project is simple, no need for extra peripherals, only five wires, two serial cable, two M2 boards. The two boards should be shared by one wire, one wire is common clock (SCK), one is connected to the enable terminal (CEN), and two are connected to input (SI) and output (SO). The connection is shown in Figure 5.

4 Software solution

This project aims to achieve eight functions in the Rom space through SPI communication. First, through the puts function, 8 function menus are printed on the master board for selection. With the switch function, users can choose which function to enter. This only requires a simple Master to transmit the slave information in one

direction. In the slave board, the switch function should also be set to jump to the corresponding 8 functions according to the contents of the received master [4-6].

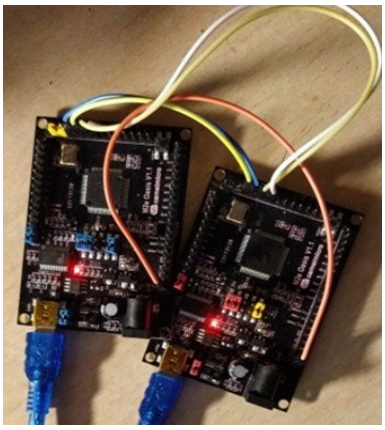

Figure 5. hardware solution

4.1 Memory Write

One board is set to Master and the other board to Slave. Among them, the function in the Master board mainly performs the sending function, and the Slave board mainly performs the receiving information.

The Master board enters the MemoryWrite1 function by entering 1. After receiving the 1st, the Slave board enters the MemoryWrite2 function and sends 1 to the Master at the same time, and then enters the read mode. Before the Master receives the 1 returned by the Slave, due to the loopback of the SPI communication protocol, it needs to send another message to receive the 1 successfully. If you read using the RT_SPI_Read_ () function before sending the message, you can only read the blank value. Here, we need to send another message in advance, in preparation for sending the address in advance, and send an address in the past.

When the Master receives 1, it enters the address continuous transmission mode, with the carriage return symbol '\r' or the input of 8 as the end marker. After receiving the consecutive addresses, Slave will return the signal 2 and enter the Value reading mode. After the Master receives 2, it continues to send the value. After the Slave is received, use the MemoryWrite32 function to directly write the value to the address and jump back to the beginning of the program with the (*(void (*) ())0x10000000) () statement. This completes the writing.

4.2 Memory Read

This mode is similar to Memory Write. The only difference is that after the Slave board read values through the MemoryRead32 function, we need to transfer the reading value to sending value, and continuously send the value to the Master board, and then output it by the Master board. Since the value read is the integer data, the d2h function is used to convert the shape into a hexadecimal character, which is stored in the array and output to the Master one by one. When the Master reads, it still needs to send a meaningless value each time to push the clock cycle so that the correct value can be read.

4.3 Jump to address

It is identical to the input address portion of the Memory Write section. When the Slave receives the address sent by the Master, it can jump to the address by directly using the (*(void (*) ()) address) () function.

4.4 Dump from address

The function of this function is to continuously read the address value. You need to enter the starting address first, and then enter the value that needs to be read several times. When the number of inputs is 1-4 times, the value of the starting address is read; when the number of input is 5-8, the starting address and the address value after adding four are read, so the analogy is 4. When the number of inputs is 0, jump directly back to the beginning of the program.

This function comprehensively uses the read and write methods in the Write and Read functions. After several conversions, it is the most cumbersome function of code and logic. It is necessary to input the address to the Slave by the Master first, and then input the number of times to the Slave by the Master. After Slave is calculated, the number of values that need to be displayed is returned to the Master. The Master then continuously reads the value sent from the Slave, and the output is continuously displayed on the screen.

The time of sending and receiving in the function should be payed attention. The items such as puts (xtoa ()) after receiving in the function cannot be omitted, otherwise the sending time will be insufficient and the receiving will be incomplete.

4.5 Bootloader

This function is to read data from the Uart0 port and write it to the flash space. Due to the special nature of the flash space read and write, it is necessary to perform the operation of erasing the space in advance and adjusting the baud rate. The Bootloader function in the original Rom space itself does not have the ability to modify the baud rate and clear the function. Since the master and slave programs are written in the flash space, it is impossible to completely clear the flash to download new data. Only the flash space is manually cleared, and the master and slave control programs are reserved in the previous segment of the flash. This function reserves a space of 10000000 to 1000f000 for the original function. After that, the 30k space starting from 1000f000 is erased as a space for new data reading.

Both the main board and the slave board must change the baud rate to 19200. After that, the main board reads the value from the computer through the UART and transmits it to the slave board in two digits (hexadecimal). The last four digits (hexadecimal) received from the board are written into the flash memory that has been emptied since 1000f000. The end

symbol of the setting is automatically ended when reading to 0xFFFFFFFF, and jumps back to the main interface of the program.

4.6 Memory loop read word

This function is similar to the Memory Read function. It can be read once and changed continuously for multiple times.

4.7 Memory loop write word

As above, change Memory Wrtie to write multiple times in succession.

4.8 Dump word from address

Similar to the Dump function, you also need to enter the Count value, but this count value directly corresponds to the value of how many addresses to read. Others are no different from the Dump function.

5 Compile on CamelStudio

Use the compilation tool CamelStudio to complete the software solution. In the directory where CamelStudio is installed, double-click the CamelStudio icon to get the main interface of CMStudio as shown in Figure 6.

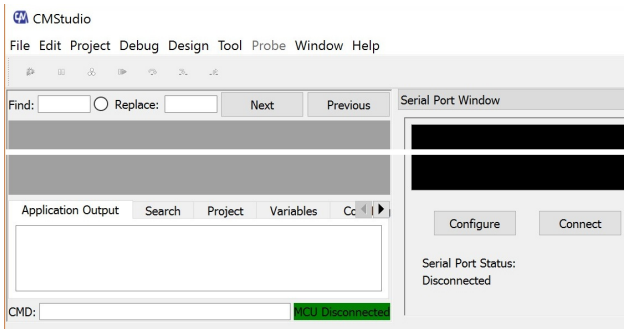

Figure 6. CMStudio main interface

6 Implementation status

Two boards can be connected to the same computer through the serial port. The computer controls the Master board to operate the Slave board. It can watch the data reception and transmission of the Master and Slave at the same time, which is more intuitive. The experiment found that 8 kinds of function functions can be successfully reflected, but there are also some problems.

Before running, you need to ensure that the clock is fully aligned, otherwise the value of the slave board will continue to run and cannot be fixed at the normal initial interface. You can use the method of continuously inputting '1'. When it is observed that the motherboard can successfully enter the address without jumping out of the way, it means that the clock pairing is successful. At this time, once the program starts running correctly, it enters a steady state.

Clock polarity configuration needs to be noted:

The 81xx module will always input data bits at the rising edge of the clock, and the host will always output data bits on the falling edge of the clock.

The master transmits data on the falling edge of the clock, and the slave receives data on the rising edge of the clock. Therefore, the SPI clock polarity of the master device should be configured as a falling edge.

SDI is shifted into 8-bit shift register on every rising edge of SCK in the order of data bit 7, data bit 6 ... data bit 0.

Slave SSD1289 receives data on the rising edge of the clock and receives data in order from high to low. Therefore, the SPI clock polarity of the master should also be configured as a falling edge.

Once the clock polarity and phase are configured correctly, the data can be accurately transmitted and received. Therefore, the master's clock should be properly configured against the slave's SPI interface timing or Spec documentation.

7 Conclusion

When the SPI interface clock on the master device side is configured, one should make sure understand the clock requirements of the slave device, because the clock polarity and phase of the master device are based on the slave device. Therefore, in the configuration of the clock polarity, it must be clear whether the slave device receives data on the rising or falling edge of the clock, and outputs data on the falling or rising edge of the clock. It should be noted that since the SDO of the master device is connected to the SDI of the slave device, the SDO of the slave device is connected to the SDI of the master device, and the data received by the slave device SDI is sent by the SDO of the master device, and the data received by the master device SDI is the slave device. The SDO is sent, so the configuration of the SPI clock polarity of the master device (i.e., the configuration of the SDO) is opposite to the polarity of the SDI received data of the slave device, and the polarity of the data sent by the slave device SDO is the same.

References

1. M2e User guide _20170703, Camel Microelectronics, Inc. July 3, 2017
2. Bo Hu, Xiaofeng Wu, Fei Ye and Weize Xie, SoC microsystem principle and application. (to be published)
3. Better SPI Bus Design in 3 Steps, https://www.dorkbotpdx.org/, September 3, 2018
4. Serial Peripheral Interface, https://en.wikipedia.org, September 10, 2018
5. SSCOM3.3 Serial port assistant, http://www.daxia.com/, September 10, 2018
6. hex workshop , http://www.hexworkshop.com/, Breakpoint Software, September 10, 2018