ESE381 Embedded Microprocessor Systems Design II

Spring 2022, K. Short revised March 14, 2022 4:20 pm

PRELIMINARY PRELIMINARY PRELIMINARY PRELIMINARY PRELIMINARY

Laboratory 07: Asynchronous Serially Controlled Digital Potentiometer with SPI Interface

To be performed the week starting March 20th.

Prerequisite Reading
(All items are posted on Blackboard.)

1 AVR128DB48 Data Sheet Section 28 SPI - Serial Peripheral Interface
2. Maxim MAX5402 Digital Potentiometer Data Sheet (on Blackboard).
3. Termite: a simple RS232 terminal.

Overview

Serial Peripheral Interface (SPI) is a defacto standard for transmitting serial data between ICs placed on the same printed circuit board. SPI allows high-speed synchronous serial data transfer between two ICs. It was originally defined by Motorola. Microcontrollers from many different manufacturers now have one or more on-chip modules that implement the SPI interface protocol in hardware. Of course you can also implement the SPI protocol in software using GPIO pins, but the transfer rate will be limited and the microcontroller will not be free to do other tasks during SPI transfers.

Using an SPI on-chip module we can eliminate the need to execute a large number of instructions to read or write data bits from or to a compatible SPI device. For maximum flexibility and wide applicability, most microcontroller on-chip SPI modules are configurable. Thus, they contain one or more control registers that are written to specify the SPI module's configuration.

While in operation, an SPI on-chip module has one or more status registers that are used to provide information about the current state of the module. On-chip SPI modules are primarily used to transfer or generate data. Accordingly, they have one or more data registers. Writing the data register initiates a data transmission. Reading this same register returns the received data.

A single SPI module can have multiple slaves connected to it. When the SPI is used to communicate with a specific external peripheral IC, the microcontroller's SPI module must be configured to be compatible with that peripheral IC's configuration and limitations. If the microcontroller is used as the master with multiple slave external peripheral ICs, its configuration must be changed by the application software to be compatible with each slave before it communicates with that slave. Accordingly, appropriate code must be written to accomplish this.

The AVR128DB48 has 2 separate SPI modules.

One major objective of this laboratory is reinforcing your understanding of the SPI protocol and familiarizing you with the use of an AVR128DB48 SPI module to implement SPI serial communications.

Design Tasks

Design Task 1: Designing the Interface Connection and Verifying SPI Writes from the AVR128DB48 SPI0 Module to the MAX5402.

Your first task is to understand the operation of the MAX5402 and its serial peripheral interface (SPI). You are to use the MAX5402 as a simple potentiometer divider with terminal H connected to +3.3V and terminal L connected to ground. The output of this simple divider is at terminal W.

Note that in your design, the only supply voltage to be used is 3.3V from the Curiosity board.

Draw a schematic of the interface of the MAX5402 to an AVR128DB48 using the AVR128DB48's SPI0. Use PF2 to drive the MAX5402's chip select ($\overline{\text{CS}}$) input.

Write a function named `MAX5402_SPI0_init` that initializes the AVR128DB48's SPI0 module to communicate with the MAX5402. SPI0 must be configured in the Normal mode (no buffering). Thus, this function must check that the previous transfer is complete after writing each byte of data and then deselect the MAX5402 before returning. Note that there are two variations of the INTFLAGS register. One for use in Normal mode and the other for use in Buffered mode.

The serial communication between the AVR128DB48 and the MAX5402 must be as fast as possible. The function prototype follows.

```
void MAX5402_SPI0_init(void);
```

Write a second function named `MAX5402_SPI0_write` that is passed an `uint8_t`. This is the value to be send to the MAX5402 to set the position of the its wiper. It is important that before this function returns, it must deselect the MAX5402. If this is not done, there could be a subsequent SPI bus conflict between the MAX5402 and other (future) SPI devices added to the system. The function prototype for this function is:

```
void MAX5402_SPI0_write(uint8_t data);
```

Write a verification program named `MAX5402_verify` that allows you to use the previous two functions to verity the operation of the MAX5402. A good verification strategy would be to have the `uint8_t` argument to `MAX5402_SPI0_write` function increment in a loop from 0 to 255. A small delay could, say 1 ms, be inserted between each call of `MAX5402_SPI0_write`. The resulting output should look like a periodic staircase or ramp and can be viewed on your oscilloscope or Saleae logic analyzer.

Submit your schematic, program C source, and program verification strategy as part of your prelab. Make sure that your program includes a program header, a header for each function, and clear comments throughout.

Design Task 2: Asynchronous Serial Control of the MAX5402 Digital Potentiometer Using a Hexadecimal Value.

In this design task, you must write a program that allows you to send data from the Termite terminal emulator and have that data control the analog output of the MAX5402. Since you will need to be use all 8-bits of the data you send to produce output voltages over the entire MAX5402's output range, you need to be able to specify the data values as hexadecimal, not ASCII. The basic ASCII code is a 7-bit code and when sent as a byte the most significant bit is always 0.

Write a program named `Terminal_to_MAX5402` that allows you to type single 8-bit hexadecimal values into the terminal emulator and have those values sent to the AVR128DB48's USART3. Use a USART3 RXC interrupt service routine (ISR) to transfer the data to a 16-byte software receive buffer. The code for this receive buffer is the same as for Design Task 2 of Laboratory 06, except for the buffer size. Implement only the receive buffer, the transmit buffer is not needed.

In the main loop of your program use the USART3_Receive function from Laboratory 06 to check whether there is data in the software receive buffer, If there is data in the receive buffer the main loop should read the data and then write it to SPI0 and, thus, to the MAX5402. Note that you must not check for a pushbutton press before reading from the receive buffer. Just check to make sure there is byte to read in the buffer. Make maximal reuse of functions that you have written for previous laboratories (particularly Laboratory 06) and for Task 1 of this laboratory.

As you type data in hexadecimal into Termite it will be sent one byte at a time to the receive buffer and then read from the buffer and sent to the MAX5402 by code in your main loop. Note the slides in the Laboratory 06 Lecture that show you how to configure Termite to send hexadecimal data and the information in Laboratory Task 2 of this laboratory.

Submit your C source file for this program as part of your prelab. Make sure that your program includes a program header, a header for each function, and clear comments throughout.

Design Task 3: Asynchronous Serial Control of the MAX5402 Digital Potentiometer Via an ASCII Decimal Character Command String.

In some embedded systems, commands are sent to the system serially in the form of ASCII character strings. The embedded system must interpret (parse) these command strings and if a string is valid carry out the command represented. Command strings are usually parsed by a finite state machine (FSM) executed by the embedded system's microcontroller. Later in this course we will look at a method for implementing an FSM that can handle any situation. For this task you will be provided with an FSM that is adequate for this design and you will have to study its implementation.

In this design task you must control the MAX5402's output serially from the Termite terminal emulator as in Task 2. However, in this case, you must type a character string into the terminal emulator that represents the desired MAX5402 output voltage in decimal. The string has the following sequence:

`Vnnn`

followed by a carriage return and line feed <CR><LF>.

The `nnn` in the string is the desired output voltage in units of hundreds of a volt. So, a value of 333 means 3.33 V and a value of 162 means 1.62 V.

A receive software buffer is to be used, but no transmit buffer. A USART0 RXC ISR puts the data into the receive buffer. The code in the super loop must read the data from the receive buffer and parse the string. If the string is valid, your code must compute the binary value to be sent to the MAX5402 to output this voltage value. The computed value is then sent to the MAX5402 via the AVR128DB48's SPI0 module. Do your computations using integer arithmetic only, do not use floating point arithmetic.

Write a program named `ASCII_str_to_MAX5402` that allows you to type a string having the specified format into the terminal emulator and have the value represented sent to the AVR128DB48's USART0. Make maximal reuse of functions that you have written for previous laboratories and for Task 1 and Task 2 of this laboratory.

A case statement labelled FSM is provided to parse the command string. It is available in a file in this laboratories folder.

Submit your C source file for this program as part of your prelab. Make sure that your program includes a program header, a header for each function, and clear comments throughout. Also submit a state diagram for FSM case statement.

Laboratory Activity

Laboratory Task 1: Designing the Interface Connection and Verifying SPI Writes from the AVR128DB48 SPI0 Module to the MAX5402.

Wire the MAX5402's SPI interface to the Curiosity board's SPI0 SPI pins and PF2. Connect and configure the Saleae logic analyzer so you can monitor the data transfer between the

AVR128DB48 and MAX5402. **Caution: in your design, the only supply voltage to be used is 3.3V from the Curiosity board.**

Create a project named `MAX5402_verify` using the program `MAX5402_verify` that you wrote for Task 1. Set the compiler optimization to -Og. Build the program. Place any variables in a watch window to observe their values. Single step through each instruction in the program. Prior to single-stepping each instruction, determine what changes you expect in the values of SPI0's registers and any variables you created.

Run your program and verify that it continually increments the binary value sent to the MAX5402. Use the Saleae logic analyzer to capture the first transaction sent from the microcontroller to the MAX5402. Submit this screen capture with your laboratory.

When your program is working correctly, have a TA verify that it performs as required. Get the TA's signature.

Laboratory Task 2: Asynchronous Serial Control of the MAX5402 Digital Potentiometer Using a Hexadecimal Value.

Create a project named `Terminal_to_MAX5402` using the program `Terminal_to_MAX5402` that you wrote for Design Task 2. Set the compiler optimization to -Og. Build the program. Place any variables in a watch window to observe their values. Single step through each instruction in the program. Prior to single-stepping each instruction, determine what changes you expect in the values of the port registers and any variables you created.

Termite focuses on text data, and specifically text that is sent and received as strings terminated with "new-line" characters. It has no provisions for transferring files or binary data. However, there is a "hexadecimal view" plug-in that you can use to send and receive non-ASCII bytes as hexadecimal values. See the slides in the Laboratory 06 Lecture and the prerequisite reading "Termite: a simple RS232 terminal."

Use the Saleae logic analyzer to capture the each transaction sent from the microcontroller to the MAX5402.

When your program is working correctly, have a TA verify that it performs as required. Get the TA's signature.

Laboratory Task 3: Asynchronous Serial Control of the MAX5402 Digital Potentiometer Via an ASCII Decimal Character Command String.

Create a project named `ASCII_str_to_MAX5402` using the program `ASCII_str_to_MAX5402` that you wrote for Task 3. Set the compiler optimization to -Og. Build the program. Place any variables in a watch window to observe their values. Single step through each instruction in the program. Prior to single-stepping each instruction, determine what changes you expect in the values of the port registers and any variables you created.

When your program is working correctly, have a TA verify that your program performs as required. Get the TA's signature.

Leave the circuit you have constructed on your breadboard, it may be used in later laboratories.

Questions

1. Fill out the logic level compatibility check list (in the laboratory folder) for the AVR128DB48 and the MAX5402. Make the AVR128DB48 device A. Both devices are operated with a supply voltage of 3.3V.

2. What is the maximum SCLK frequency you can have for an SPI transfer from the AVR128DB48 to the MAX5402 and what limits this frequency?

3. What are the required values for CPOL and CPHA for the MAX5402 and from where did you determine them?

4. Review C's switch statement. Draw the state diagram for the finite state machine (FSM) implemented by the FSM switch statement you were provided.

5. From your state diagram from Question 4, what happens in Task 3 if you send an invalid command string?

6. Why is the variable `decimal` declared as a `uint32_t`?