Laboratory 07: Asynchronous Serially Controlled Digital Potentiometer with SPI Interface

Prof. Ken Short

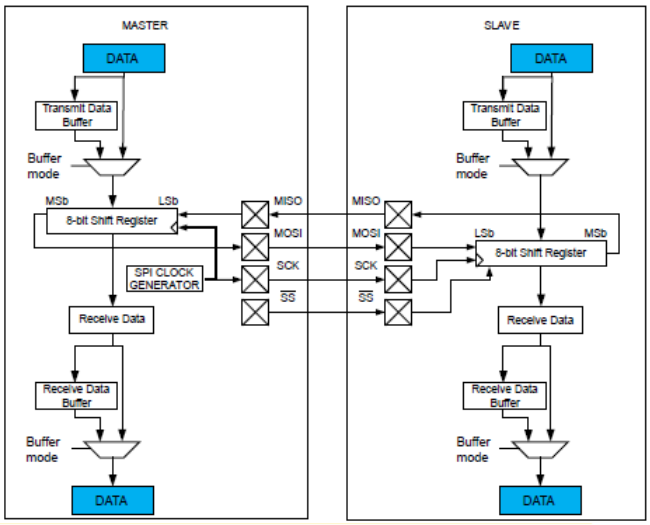
Overview

- 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. This laboratory builds to that capability.
- In the last design task, you must control the MAX5402's output serially from the Termite terminal emulator using a character string typed into the terminal emulator that represents the desired MAX5402 output voltage in decimal.

Design Tasks

- Design Task 1: Designing the Interface Connection and Verifying SPI Writes from the AVR128DB48 SPI Module to the MAX5402.
- Design Task 2: Asynchronous Serial Control of the MAX5402 Digital Potentiometer Using a Hexadecimal Value.
- Design Task 3: Asynchronous Serial Control of the MAX5402 Digital Potentiometer Via an ASCII Decimal Character String.

SPI Module Block Diagram



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

- 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.

MAX5402_SPI0_init

- Function MAX5402_SPI0_init initializes the AVR128DB48's SPI0 module to communicate with the MAX5402. SPI0 must be configured in the normal mode (no buffering). The function must check that the previous transfer is complete after writing each byte of data and then deselect the MAX5402 before returning.

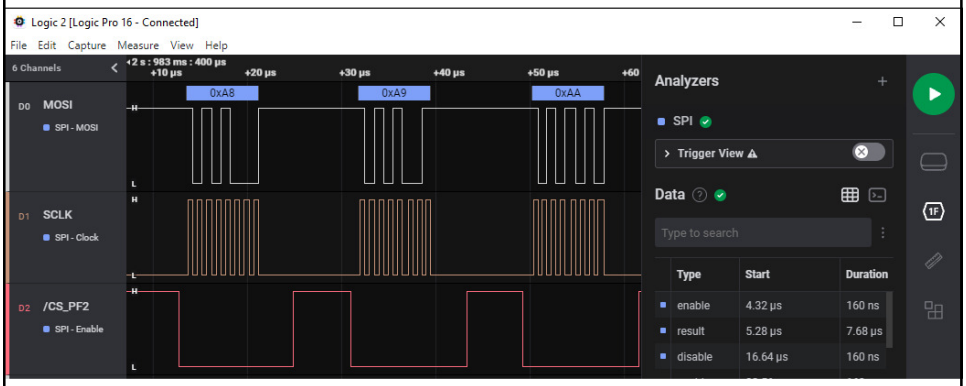
```
void MAX5402_SPI0_init(void);
```

MAX5402_SPI0_write

- Function MAX5402_SPI0_write is passed an uint8_t. This is the value used to set the position of the MAX5402's 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 future SPI devices added to the system

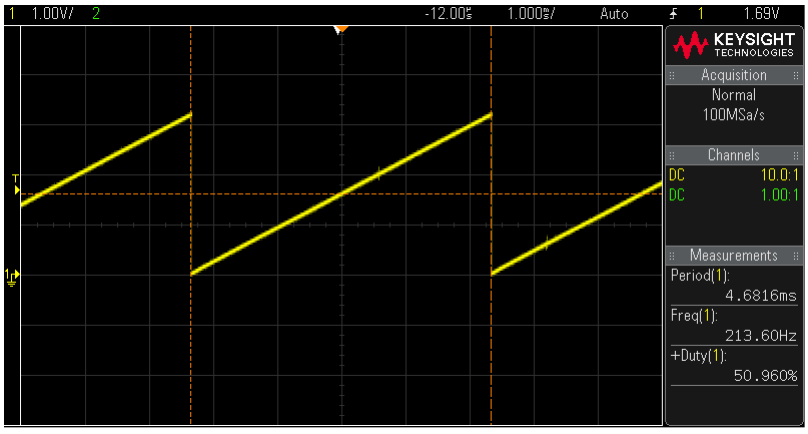
```
void MAX5402_SPI0_write(uint8_t data);
```


Saleae View of Task 1 SPI Write

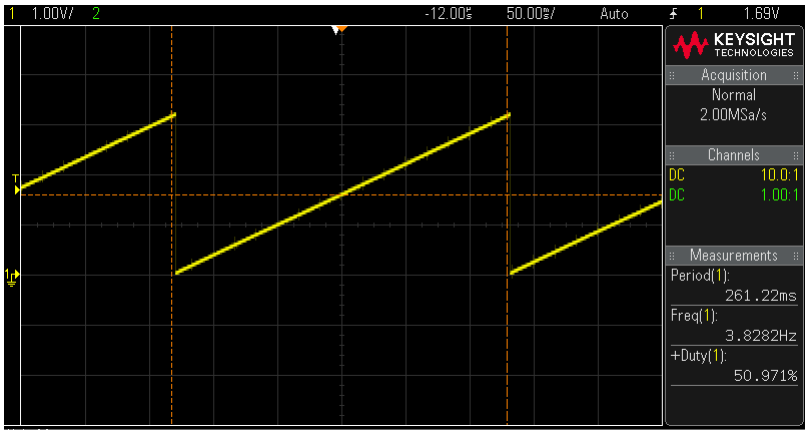


Saleae Expanded View of A Single SPI Write

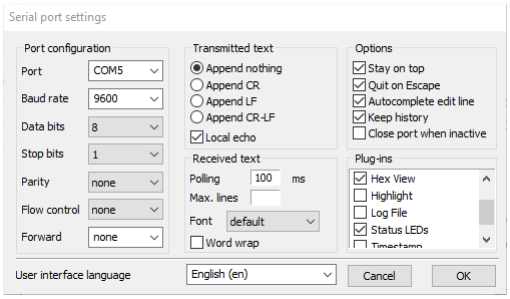
MAX5402 Output with Data Counting From 0 to 255 No Added Delay Between Outputs



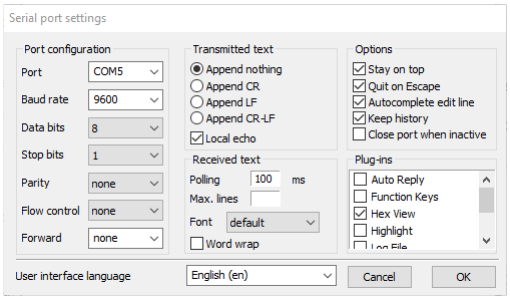
MAX5402 Output with Data Counting From 0 to 255 1ms Delay Between Outputs



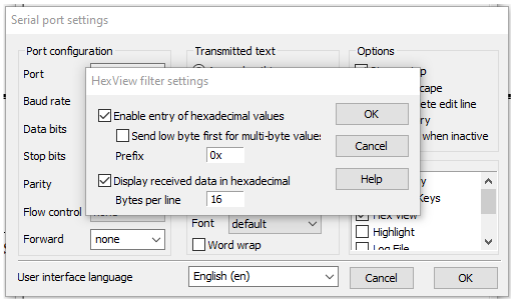
Task 2: Asynchronous Serial Control of the MAX5402 Digital Potentiometer Using a Hexadecimal Value.



Configured To Use Hex View and Append Nothing



Options When Selecting Hex View



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

- For this task you must return to sending ASCII characters, not hexadecimal values.
- In this design task you must control the MAX5402's output serially from the Termite terminal emulator using a character string you type into the terminal emulator that represents the desired MAX5402 output voltage in decimal in units of hundreds of a volt. The string has the sequence:
- Vnnn
- This string must be followed by a carriage return and line feed <CR><LF>.

Switch Implementation of FSM

```
//The switch statement labeled FSM creates a FSM to parse the command
//string received in Task 3. You will have to analyze its operation
//to answer some of the questions for this laboratory.

//You will need to include the following declarations in your code
//as global variables. Accordingly, place the outside of all functions.
//uint8_t sdr;  //serial data received
//uint8_t MAX5402_data; //data to be written to MAX5402
//uint8_t pstate = 0; //present state
//uint8_t d2, d1, d0;   //digits of the decimal value received
//uint32_t decimal;     //binary value equal to decimal value received
```

Switch Implementation of FSM (cont. 1)

```
FSM: switch (pstate)
{
    case 0:
    if (sdr == 'V')
    pstate = 1;
    else
    pstate = 0;
    break;

    case 1:
    if ((sdr >= '0') && (sdr <= '9'))
    {
        d2 = sdr & 0x0F;
        pstate = 2;
    }
    else
    pstate = 0;
    break;
```


Switch Implementation of FSM (cont. 2)

```
case 2:
if ((sdr >= '0') && (sdr <= '9'))
{
    d1 = sdr & 0x0F;
    pstate = 3;
}
else
pstate = '0';
break;

case 3:
if ((sdr >= '0') && (sdr <= '9'))
{
    d0 = sdr & 0x0F;
    pstate = 4;
}
else
pstate = 0;
break;
```

Switch Implementation of FSM (cont. 3)

```
            case 4:
            if (sdr == 0x0d)
            pstate = 5;
            else
            pstate = 0;
            break;

            case 5:
            if (sdr == 0x0a)
            {
                pstate = 0;
                decimal = (((d2 * 10) + d1) * 10) + d0;
                MAX5402_data = (uint8_t)(((decimal) * 255)/333);
            }
            else
            pstate = 0;
            break;

            default:
            pstate = 0;
        }
```


After Sending V166

Watch 1

Name	Value	Type
sdr	0x0a	uint8_t{data}@0x4002
pstate	0x00	uint8_t{data}@0x4000
d2	0x01	uint8_t{data}@0x4003
d1	0x06	uint8_t{data}@0x4001
d0	0x06	uint8_t{data}@0x4004
MAX5402_data	0x7f	uint8_t{data}@0x401a
USART_RxBuf	0x4005	char[16]{data}@0x4005
[0]	0x00	char{data}@0x4005
[1]	0x56	char{data}@0x4006
[2]	0x31	char{data}@0x4007
[3]	0x36	char{data}@0x4008
[4]	0x36	char{data}@0x4009
[5]	0x0d	char{data}@0x400a
[6]	0x0a	char{data}@0x400b
[7]	0x00	char{data}@0x400c
[8]	0x00	char{data}@0x400d
[9]	0x00	char{data}@0x400e
[10]	0x00	char{data}@0x400f
[11]	0x00	char{data}@0x4010
[12]	0x00	char{data}@0x4011
[13]	0x00	char{data}@0x4012
[14]	0x00	char{data}@0x4013
[15]	0x00	char{data}@0x4014
USART_RxHead	0x06	unsigned char{data}@0x401b
USART_RxTail	0x06	unsigned char{data}@0x4019
decimal	0x000000a6	uint32_t{data}@0x4015

After Sending V333

Watch 1

Name	Value	Type
sdr	0x0a	uint8_t{data}@0x4002
pstate	0x00	uint8_t{data}@0x4000
d2	0x03	uint8_t{data}@0x4003
d1	0x03	uint8_t{data}@0x4001
d0	0x03	uint8_t{data}@0x4004
MAX5402_data	0xff	uint8_t{data}@0x401a
USART_RxBuf	0x4005	char[16]{data}@0x4005
[0]	0x00	char{data}@0x4005
[1]	0x56	char{data}@0x4006
[2]	0x31	char{data}@0x4007
[3]	0x36	char{data}@0x4008
[4]	0x36	char{data}@0x4009
[5]	0x0d	char{data}@0x400a
[6]	0x0a	char{data}@0x400b
[7]	0x56	char{data}@0x400c
[8]	0x33	char{data}@0x400d
[9]	0x33	char{data}@0x400e
[10]	0x33	char{data}@0x400f
[11]	0x0d	char{data}@0x4010
[12]	0x0a	char{data}@0x4011
[13]	0x00	char{data}@0x4012
[14]	0x00	char{data}@0x4013
[15]	0x00	char{data}@0x4014
USART_RxHead	0x0c	unsigned char{data}@0x401b
USART_RxTail	0x0c	unsigned char{data}@0x4019
decimal	0x0000014d	uint32_t{data}@0x4015

After Sending V250, Notice Buffer Wrap Around

Watch 1

Name	Value	Type
sdr	0x0a	uint8_t{data}@0x4002
pstate	0x00	uint8_t{data}@0x4000
d2	0x02	uint8_t{data}@0x4003
d1	0x05	uint8_t{data}@0x4001
d0	0x00	uint8_t{data}@0x4004
MAX5402_data	0xbf	uint8_t{data}@0x401a
USART_RxBuf	0x4005	char[16]{data}@0x4005
[0]	0x30	char{data}@0x4005
[1]	0x0d	char{data}@0x4006
[2]	0x0a	char{data}@0x4007
[3]	0x36	char{data}@0x4008
[4]	0x36	char{data}@0x4009
[5]	0x0d	char{data}@0x400a
[6]	0x0a	char{data}@0x400b
[7]	0x56	char{data}@0x400c
[8]	0x33	char{data}@0x400d
[9]	0x33	char{data}@0x400e
[10]	0x33	char{data}@0x400f
[11]	0x0d	char{data}@0x4010
[12]	0x0a	char{data}@0x4011
[13]	0x56	char{data}@0x4012
[14]	0x32	char{data}@0x4013
[15]	0x35	char{data}@0x4014
USART_RxHead	0x02	unsigned char{data}@0x401b
USART_RxTail	0x02	unsigned char{data}@0x4019
decimal	0x000000fa	uint32_t{data}@0x4015