DAWSON COLLEGE – Electronics Engineering Technology Department

Fall 2012

Embedded System Hardware (243-513-DW)

Laboratory Project

SPI, ADC and USART

Submitted by: Matthew Masi

Student Number: 1036987

Date of Lab: November 28, 2012

Date Submitted: December 8, 2012

PURPOSE

The purpose of this experiment is to understand and explore some of the unexplored functions of the ATMEGA8 and ATMEGA8515 microprocessors. The two functions explored in this experiment are the analog to digital converter (ADC) and serial peripheral interface (SPI). The ADC will be used to control the frequency of a square wave and calculate the voltage that is being inputted. SPI will be used to transfer the data from one microprocessor to another, and then display it on the terminal using the USART.

EQUIPMENT & COMPONENTS

- ATMEGA8
- ATMEGA8515
- MAX232
- 0.1uf Capacitor x 6
- 100nf Capacitor
- 10uh inductor
- 10kohm Potentiometer
- DB9 connector
- Rollover Cable
- DMM
- Oscilloscope
- Power Supply
- Computer (AVR Studio 4 & HyperTerminal)
- AVRISP mkII programmer

CIRCUIT DIAGRAM

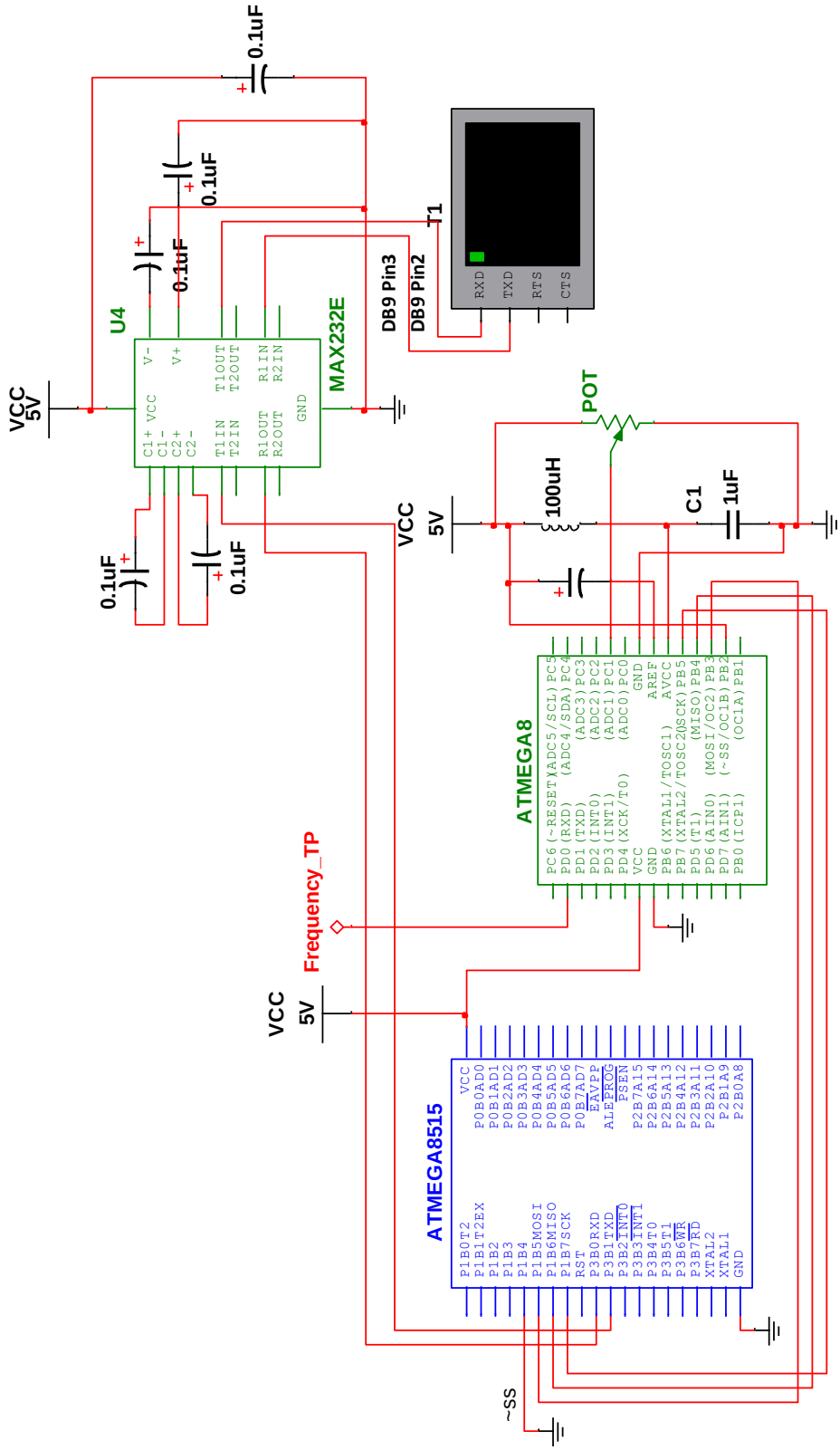

Figure 1: ATMEGA8 transmitting data to ATMEGA8515 which transmits the data to the terminal using MAX232

MEASUREMENTS & RESULTS

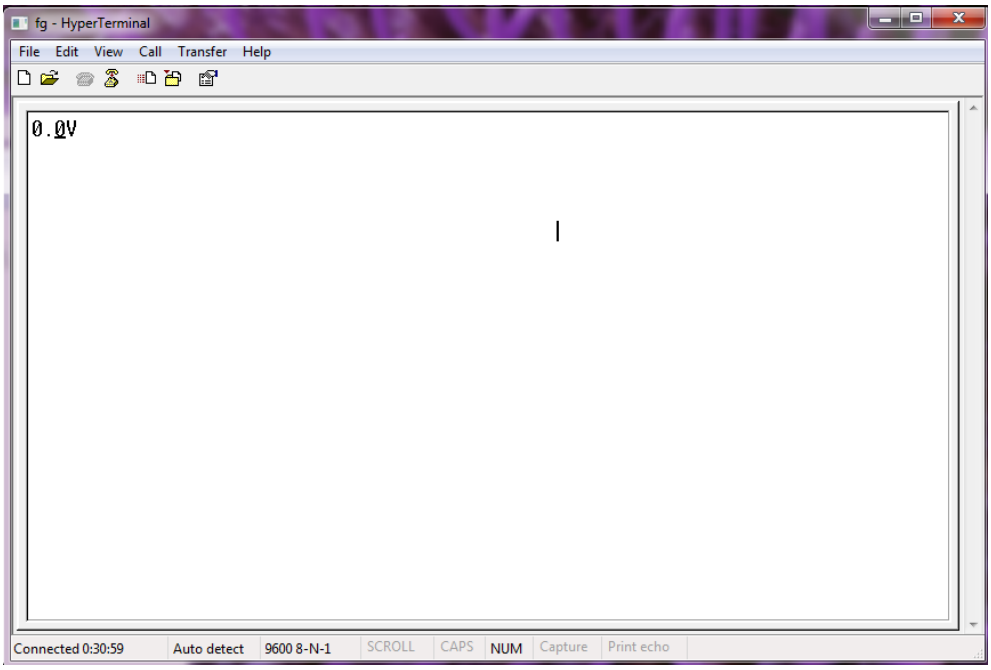

Figure 2: Potentiometer at 100% and voltage of 0V

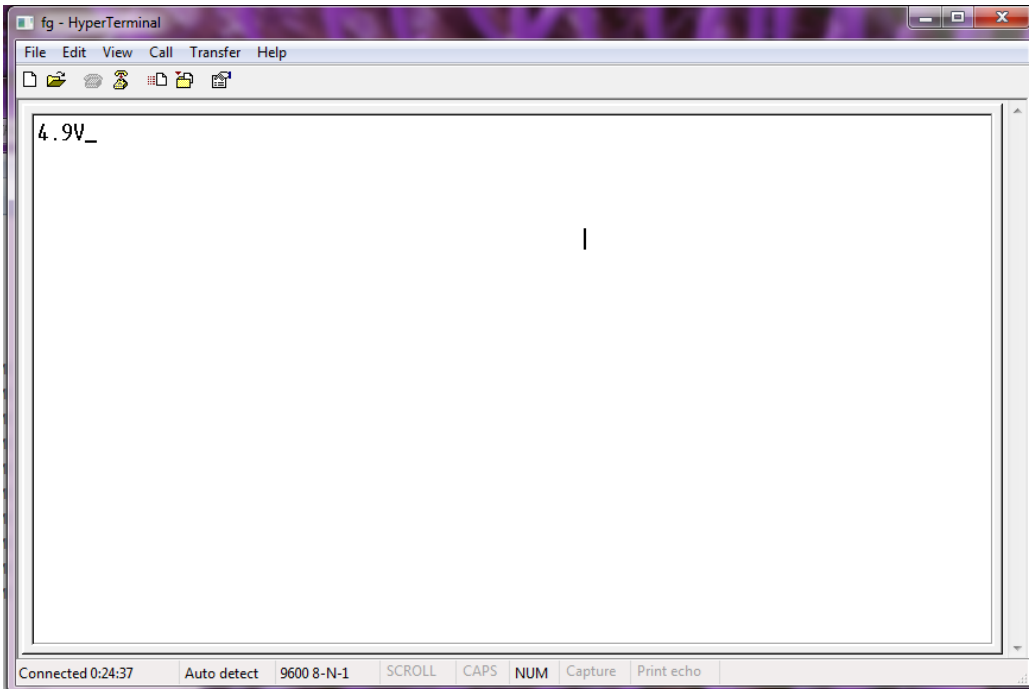

Figure 3: Potentiometer at 0% and voltage at 4.9V

The input voltage was displayed in real time on the terminal by using the ADC function to vary the input voltage. SPI was then used to transmit the data to the other microprocessor which used it's USART to send the data to the terminal.

Waveforms at 4.9V:

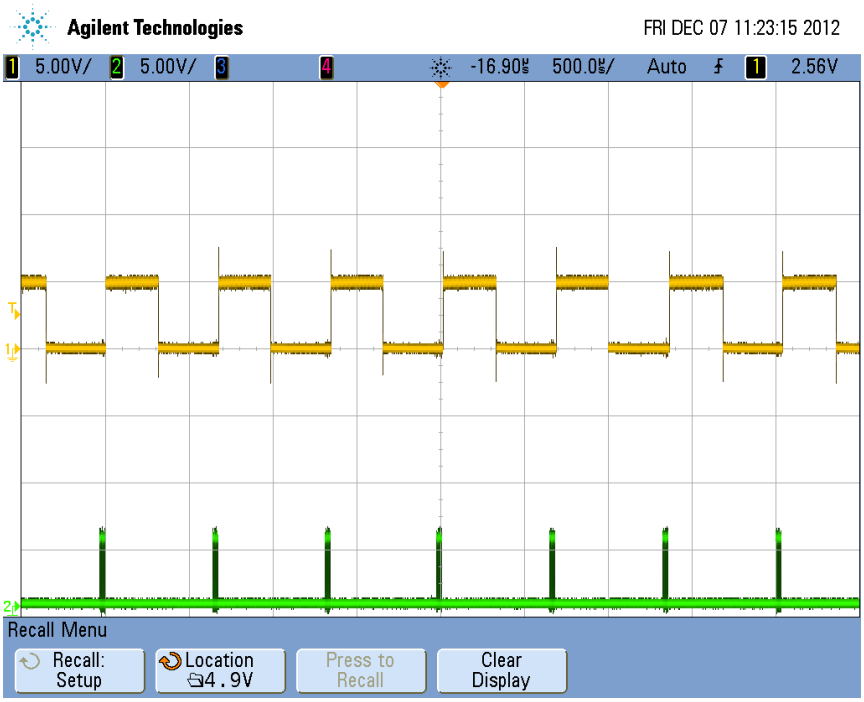

Figure 4: 1.5kHz square wave at 4.9V vs. SPI's SCK

This capture compares the square wave that is being altering by the potentiometer with the SCK pin that is connected to both the ATMEGA8 and ATMEGA8515.

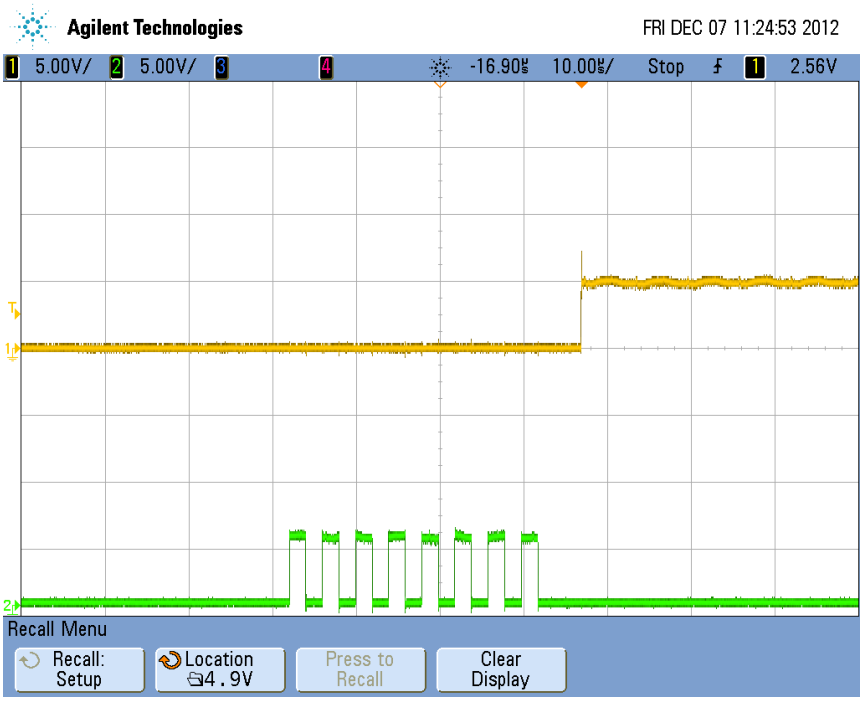

Figure 5: zooming into SCK square wave

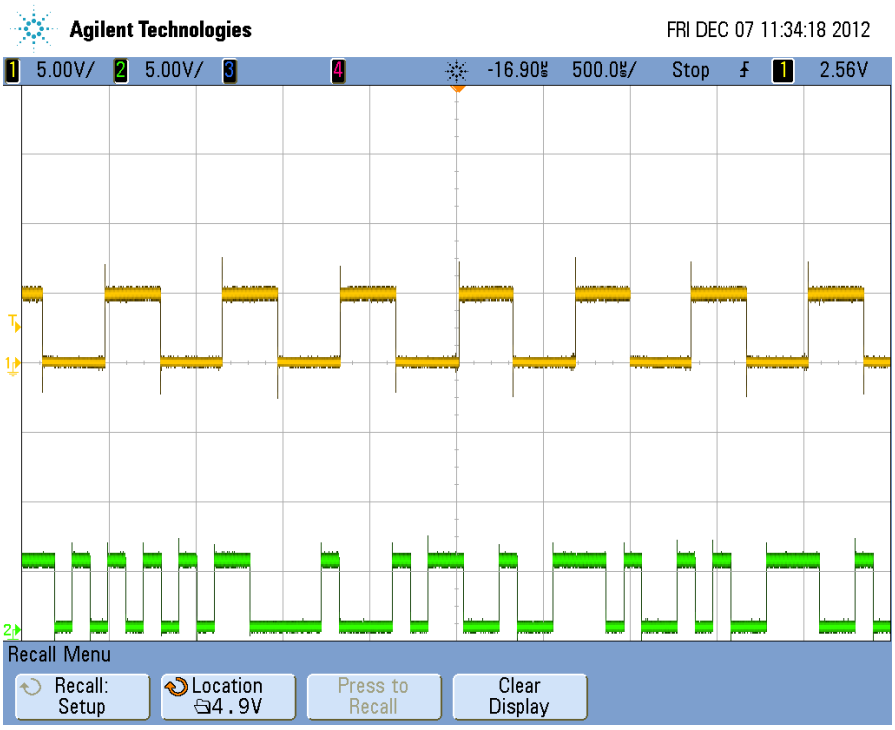

Figure 6: 1.5kHz square wave vs. data being transmitted by USART

Waveforms at 4V:

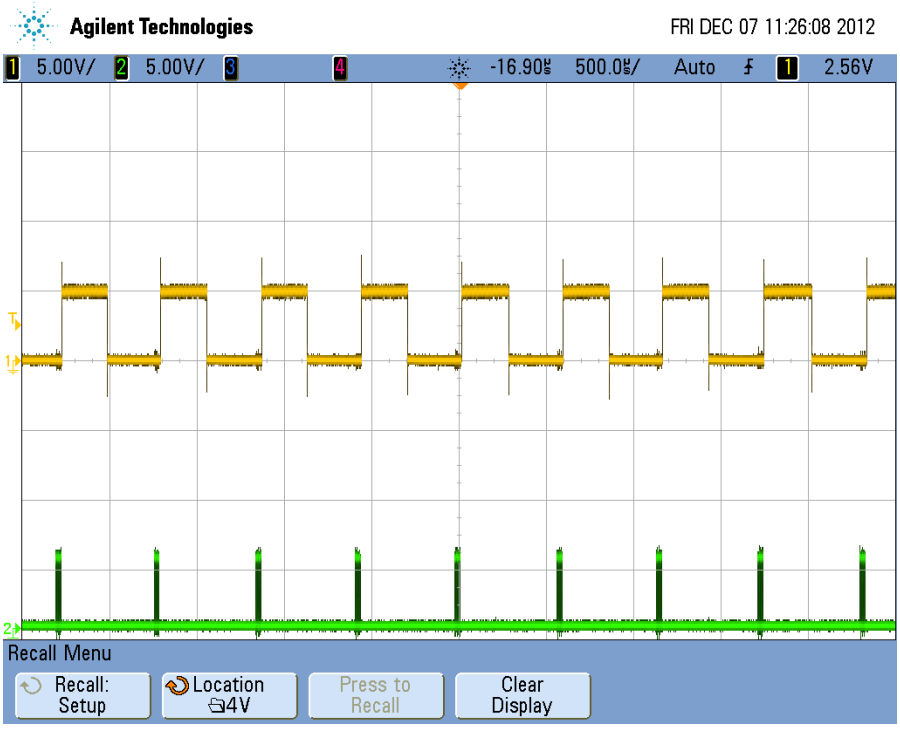

Figure 7: 1.8kHz square wave at 4.9V vs. SPI's SCK

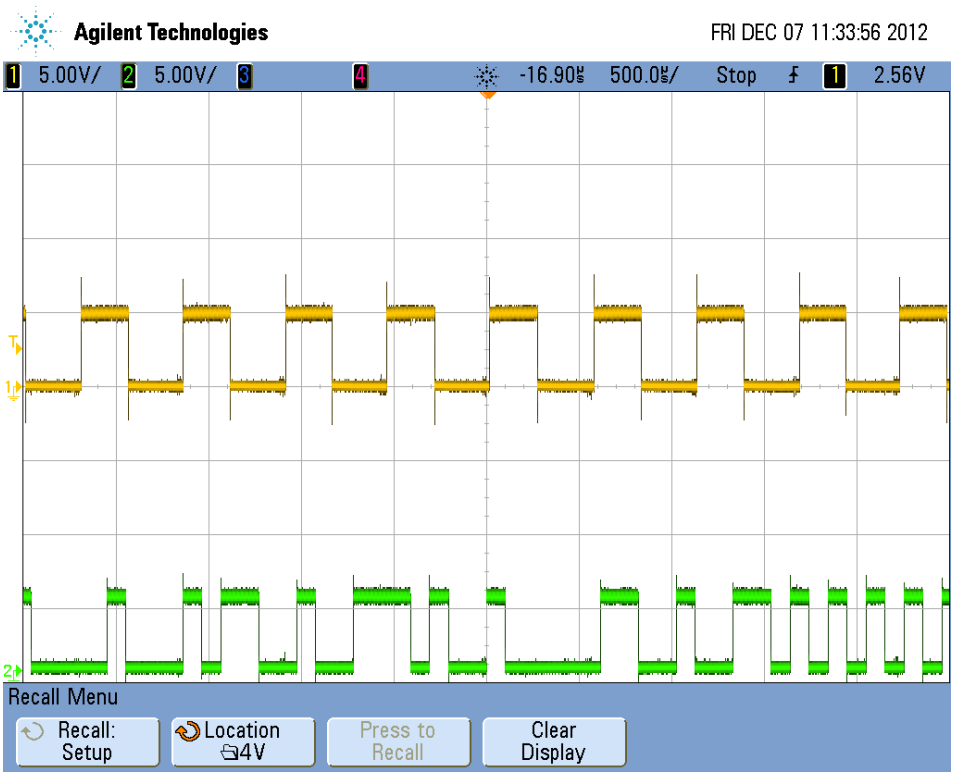

Figure 8: 1.8kHz square wave vs. data being transmitted by USART

Waveforms at 3V:

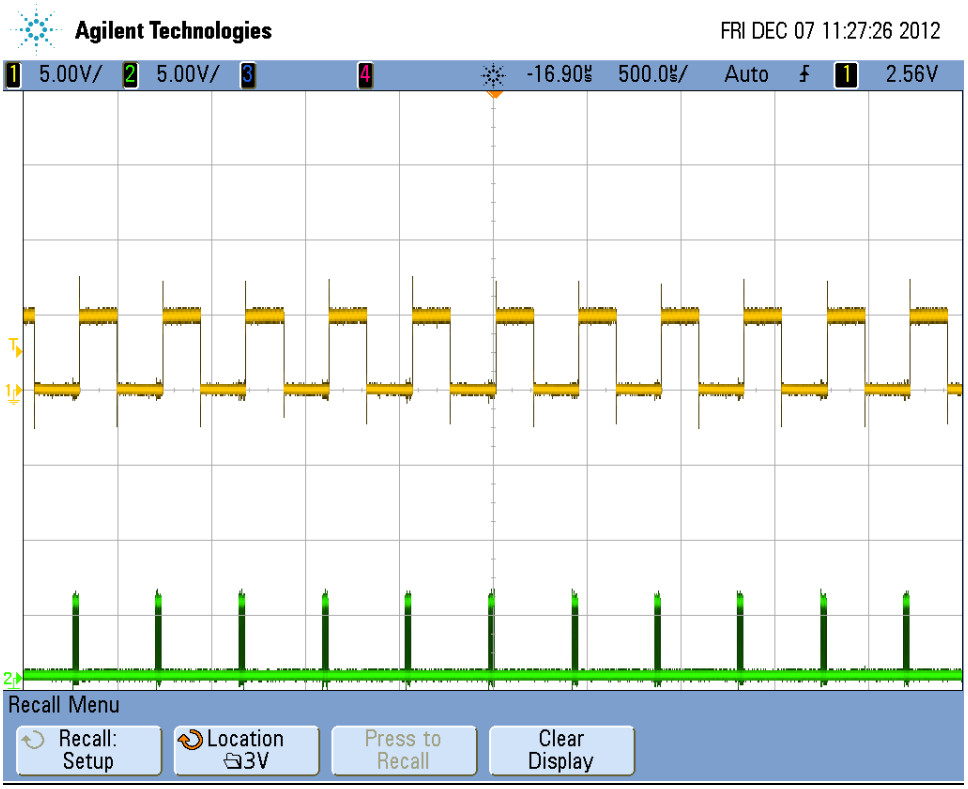

Figure 9: 2.25kHz square wave at 4.9V vs. SPI's SCK

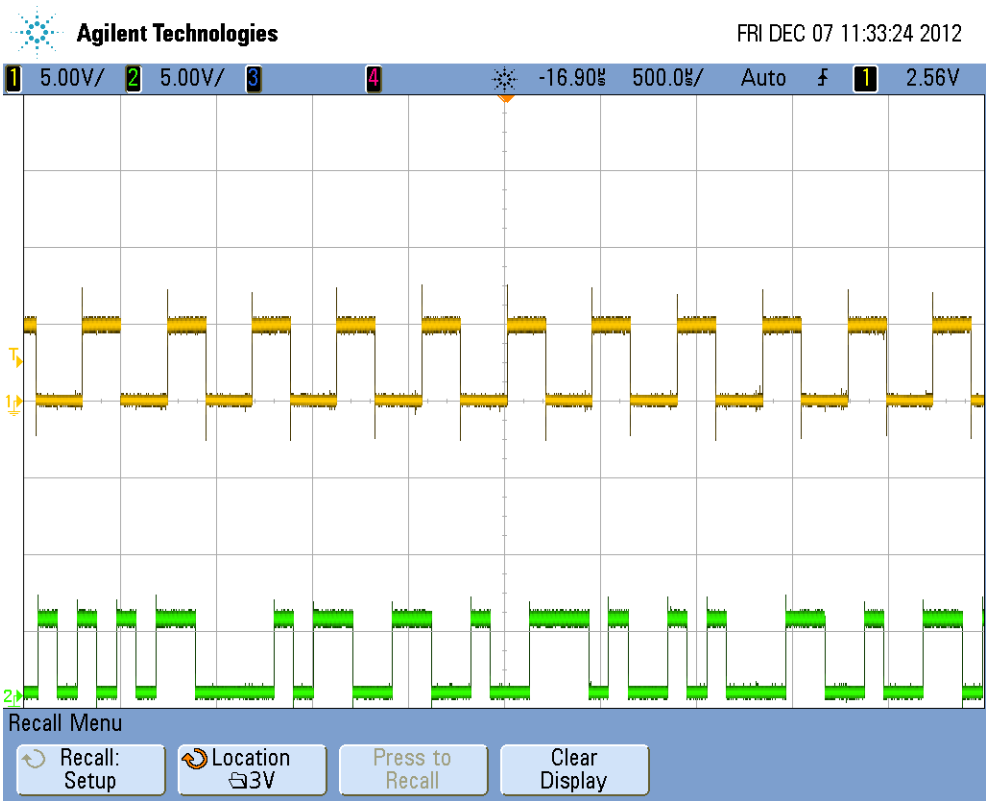

Figure 10: 2.25kHz square wave vs. data being transmitted by USART

Waveforms at 2V:

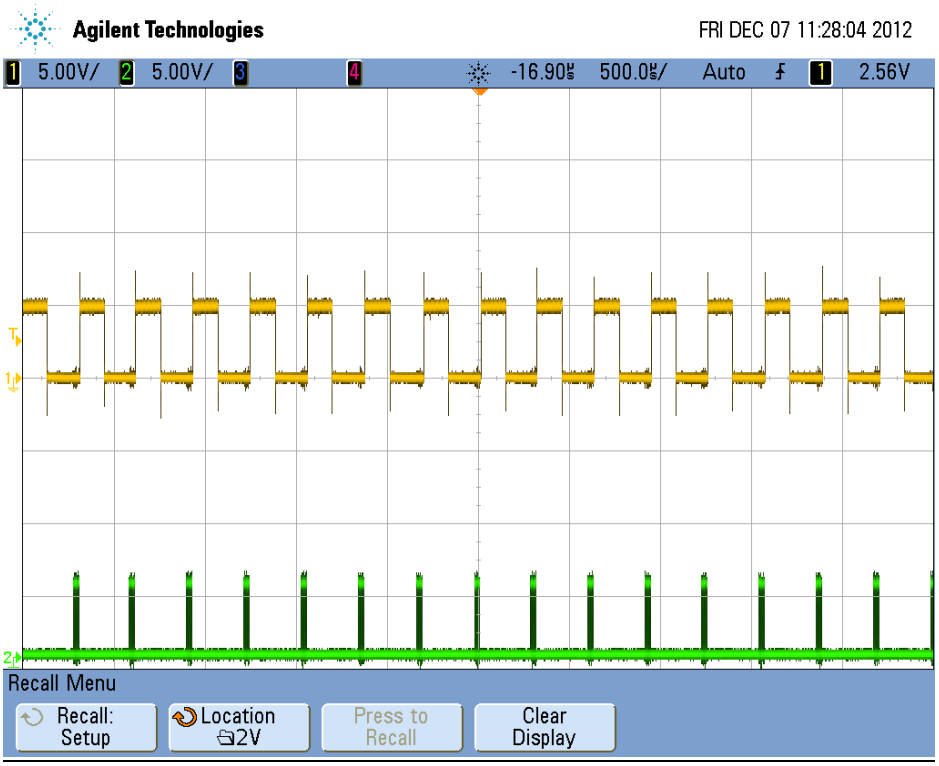

Figure 11: 3.2kHz square wave at 4.9V vs. SPI's SCK

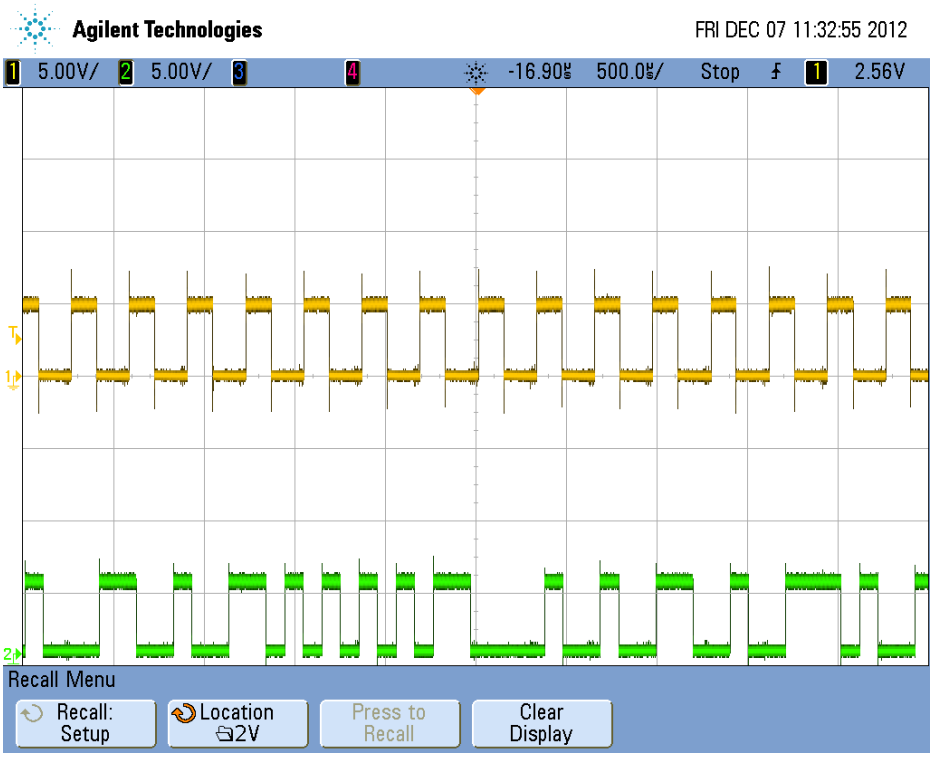

Figure 12: 3.2kHz square wave vs. data being transmitted by USART

Waveforms at 1V:

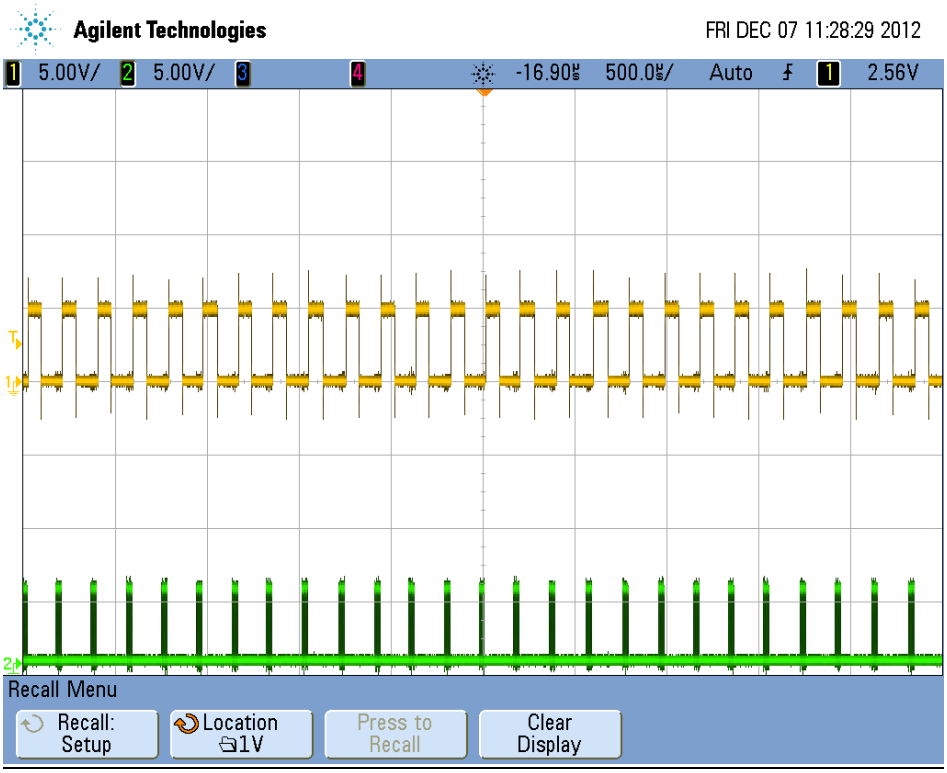

Figure 13: 5kHz square wave at 4.9V vs. SPI's SCK

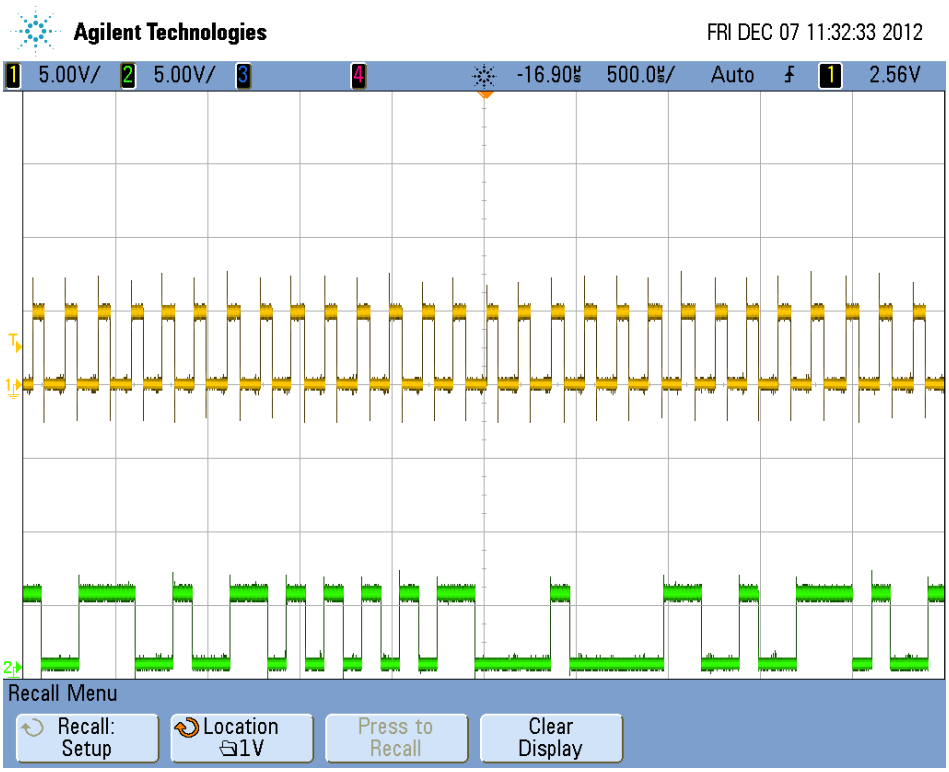

Figure 14: 5kHz square wave vs. data being transmitted by USART

Waveforms at 0V:

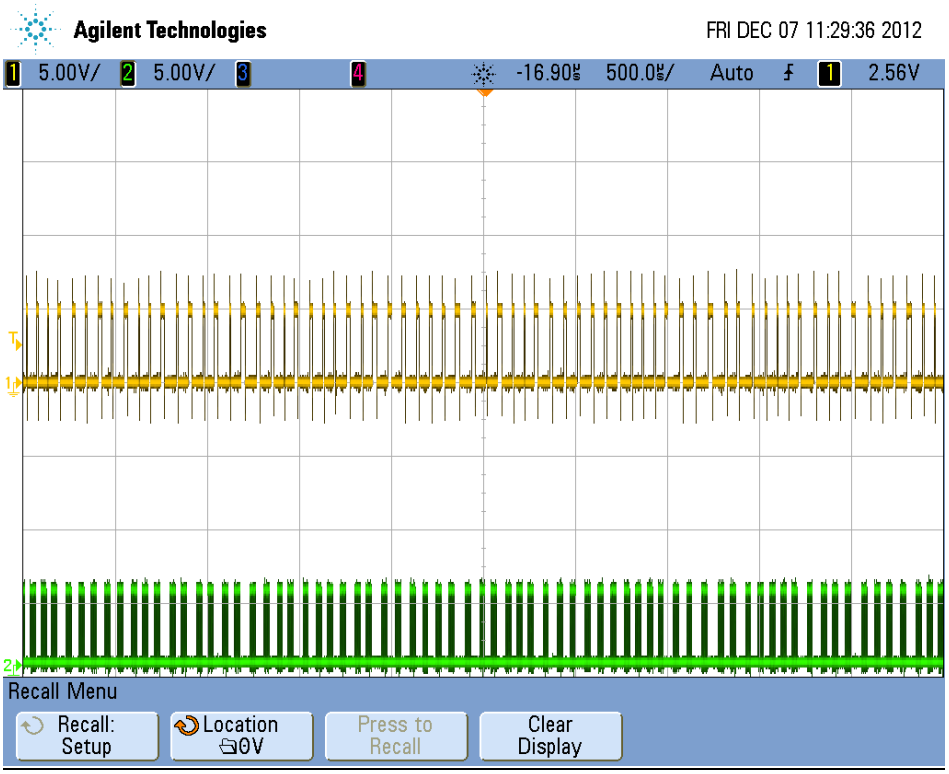

Figure 15: 13kHz square wave at 4.9V vs. SPI's SCK

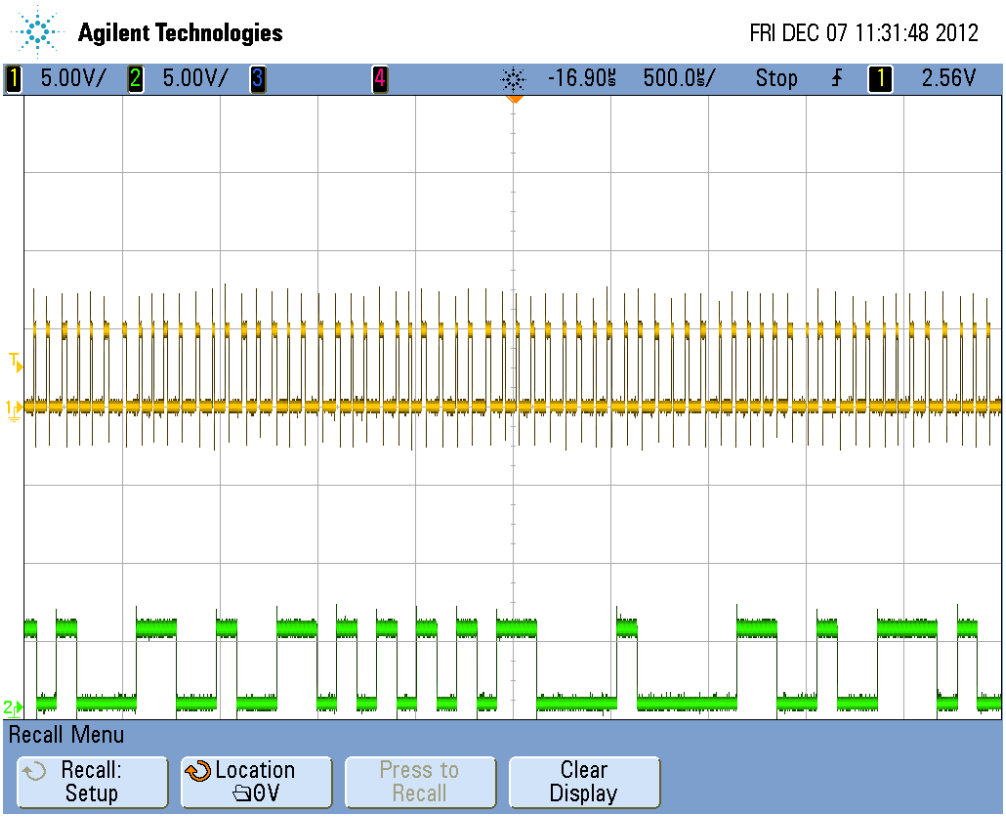

Figure 16: 13kHz square wave vs. data being transmitted by USART

As the potentiometers resistance was decreased, the voltage displayed on the terminal increased and the frequency of the outputted square wave was increased. These changes were happening in real time because of the ADC. We can also see from the waveform that as the frequency of our square wave increased, the frequency of the SCK wave was increased which in result means that the data being sent through SPI was sent faster. Finally we can see that as the frequency of the square wave was increased, the frequency of the data sent out through the ATMEGA8515's USART was not affected. This is because even though the data is being transmitted from one microprocessor to another faster, the USART is still checking for data to send always at the same speed.

THEORY

The two new features that were explored through this procedure were SPI and ADC. To set up these two functions of the microprocessor properly, many things were taken into account:

ADC:

ADC Multiplexer Selection Register – ADMUX:

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:6 –REFS1:0: Reference Selection Bits

These bits select the voltage reference for the ADC using the following table:

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

REFS0 was set to 1 because the reference voltage used for the conversion was an external source (VCC – 5V). The external reference voltage was connected to the AREF pin through a 0.1uF capacitor. To help with noise and accuracy in the conversion, a low pass filter (inductor and capacitor network) was connected to the AVCC pin.

Bits 3:0 – MUX3:0: Analog Channel Selection Bits

The value of these bits selects which analog inputs are connected to the ADC. The following table shows the values to select each analog input:

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5

In our case, since we used ADC0, we didn't set any bits.

ADC Control and Status Register A – ADCSRA:

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – ADEN: ADC Enable

Writing this bit to 1 enables the ADC. If this bit is ever turned off while a conversion is in progress, it will terminate the conversion.

Bit 6 – ADSC: ADC Start Conversion

Writing this bit to 1 starts the conversion. Since were using free running mode, we wrote a 1 to this bit in the beginning to start the first conversion. The first conversion takes 25 ADC clock cycles because it performs initialization of the ADC. All the other conversions that follow this conversion only take 13 clock cycles.

Bit 5 – ADFR: ADC Free Running Select:

Setting this bit to 1 allows the ADC to operate in a free running mode. In this mode, the ADC samples and updates the data register continuously.

Bit 4 – ADIF: ADC Interrupt Flag

This bit is set when an ADC conversion completes and the data Registers are updated. ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set.

Bit 3 – ADIE: ADC Interrupt Enable

When this bit is written to 1 and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

Bit 2:0 – ADPS2:0: ADC Prescaler Select Bits

These bits determine the division factor between the XTAL frequency and the input clock to the ADC. The following table shows the prescaler selections:

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The division factor that was chosen was two, which was done by setting ADPS0 to 1. With the frequency of the microprocessor being 4 MHz and a division factor of 2, the input clock to the ADC was 2MHz.

The ADC Data Register – ADCL and ADCH:

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When an ADC conversion is complete, the result is found in these two registers. When ADCL is read, the ADC Data Register is not updated until ADCH is read. Because we only used 8 bits, the result is left adjusted and it is sufficient to read just the ADCH. The ADLAR bit in ADMUX and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

ADC9:0: ADC Conversion result

These bits represent the result from the conversion.

SPI:

The Serial Peripheral Interface Bus is a synchronous serial data link standard that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. The ATMEGA8 was configured as the master and sent the data received from the ADC conversion to the ATMEGA8515 which was configured as slave. The configuration is as follows:

SPI Control Register - SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**	**SPCR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 6 – SPE: SPI Enable

When this bit is set to 1, SPI is enabled. This bit must be set to enable any SPI operations.

Bit 4 – MSTR: Master/Slave Select

This bit selects Master SPI mode when written to one (ATMEGA8), and Slave SPI mode when written logic zero (ATMEGA8515). If SS is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set.

Bits 1, 0 –SPR1, SPR0: SPI Clock Rate Select 1 and 0

These two bits control the SCK rate of the device configured as master and have no effect on the slave. The relationship between SCK and the Oscillator Clock frequency is shown in the following table:

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

SPR0 was set to one and so since the speed of the microprocessor was set to 4MHz, the SCK frequency was 250kHz (4MHz/16).

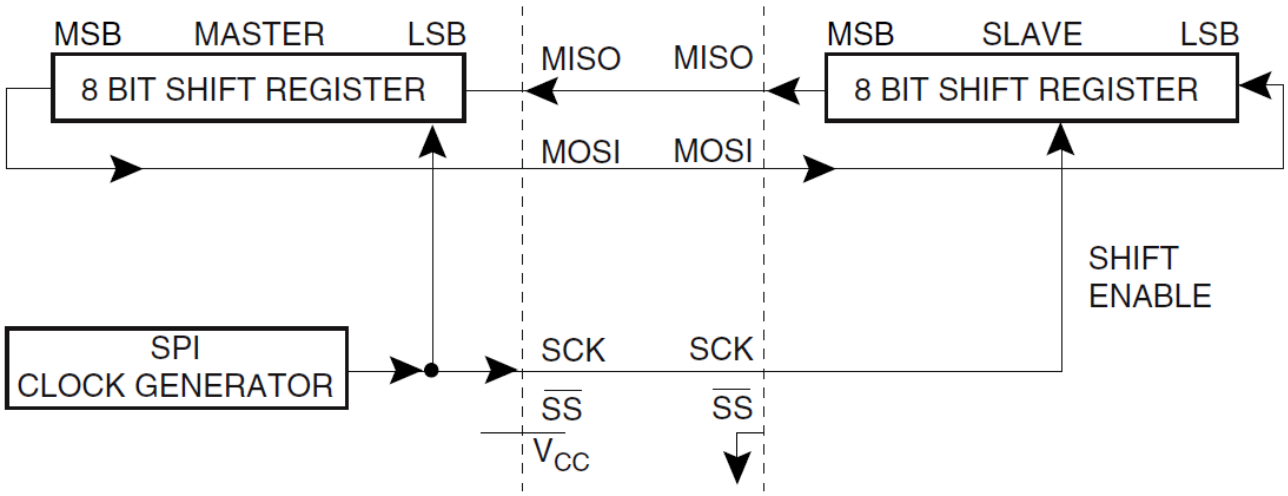

Figure 17: SPI master – Slave interconnection

CODE

ATMEGA8 – ADC, MASTER SPI

```
.include "m8def.inc"

.org 0
RESET: rjmp init
.org 0x00E
acd_int: rjmp adcstart  //ADC interrupt vector

init:
      ldi r17, high(RAMEND)
      out sph, r17
      ldi r17, low(RAMEND)
      out spl, r17
      ldi r17, 0x80                 //global interrupt enable (bit7 - I flag)
      out $3f, r17
      ldi r17, 0xff
      out DDRD, r17
      ldi r18, 0
      ldi r19, 0xff
      ldi r20, 0x00
      ldi r21, 0
      ldi r22, 0

SPI_MasterInit:
      ; Set MOSI and SCK output, all others input
      ldi r17,(1<<DDB3)|(1<<DDB5)
      out DDRB,r17

      ; Enable SPI, Master, set clock rate fck/16
      ldi r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
      out SPCR,r17
```

```
adcinit:
      ;enable ADC, free running, set division factor to 2, start conversion,
interrupt flag, interrupt enable
      ldi r17,(1<<ADEN)|(1<<ADFR)|(1<<ADPS0)|(1<<ADSC)|(1<<ADIF)|(1<<ADIE)
      out ADCSRA,r17

      ;result is left adjusted, using external refernece
      ldi r17,(1<<ADLAR)|(1<<REFS0)
      out ADMUX,r17
      sei

main:
      out PORTD, r19    //square wave high
      rcall delay
      out PORTD, r20    //square wave low
      rcall delay
      rcall SPI_MasterTransmit
      rjmp main

SPI_MasterTransmit:
      cli
      ; Start transmission of data (r16)
      out SPDR,r16
      Wait_Transmit:
      ; Wait for transmission complete
      sbis SPSR,SPIF
      rjmp Wait_Transmit
      sei
      ret

adcstart:
      ;read output from ADC
      in r16,ADCH
      reti

delay:      //delay is based on data received from ADC conversion
      cli
      inc r21
      cp r21, r16
      brne delay
      ldi r21, 0
      sei
      ret
```

ATMEGA8515 – SLAVE SPI, USART

```
.include "m8515def.inc"

init:
      ldi r17, high(RAMEND)
      out sph, r17
      ldi r17, low(RAMEND)
      out spl, r17
      ldi r18, 48
      ldi r19, 86
      ldi r20, 46
      ldi r21, 0
      ldi r22, 0
      ldi r24, 51             //1v is equal to 51 in the adc register
      ldi r25, 47             //converts to to ascii
      ldi r27, 0
      ldi r28, 5
      ldi r29, 0x0D           //carriage return

SPI_SlaveInit:
      ; Set MISO output, all others input
      ldi r17,(1<<DDB6)
      out DDRB,r17

      ; Enable SPI
      ldi r17,(1<<SPE)
      out SPCR,r17

USART_Init:
      ; Set baud rate 9.6kHz
      ldi r16, 0x00
      ldi r17, 0x19
      out UBRRH, r16
      out UBRRL, r17
      ; Enable receiver and transmitter
      ldi r16, (1<<RXEN)|(1<<TXEN)
      out UCSRB,r16
      ; Set frame format: 8data, 1stop bit, no parity
      ldi r16, (1<<URSEL)|(0<<UPM0)|(0<<USBS)|(3<<UCSZ0)
      out UCSRC,r16

main:
      rcall SPI_SlaveReceive
      rcall disp
      rcall usart_transmit
      rjmp main

SPI_SlaveReceive:
      ; Wait for reception complete
      sbis SPSR,SPIF
      rjmp SPI_SlaveReceive
      ; Read received data and return
      in r26,SPDR
      ret
```

```
usart_transmit:
j:    //transmits carriage return
      sbis UCSRA,UDRE
      rjmp j
      out UDR, r29

k:    //transmits the voltage value
      sbis UCSRA,UDRE
      rjmp k
      out UDR,r21
      ldi r21, 0

l:    //transmits a decimal
      sbis UCSRA,UDRE
      rjmp l
      out UDR, r20

m:    //transmits the 100mv
      sbis UCSRA,UDRE
      rjmp m
      cpi r27, 58
      brne cont
      ldi r27, 57
cont:
      out UDR,r27
      ldi r27, 0
      ldi r22, 0

n:    //transmits the letter v
      sbis UCSRA,UDRE
      rjmp n
      out UDR, r19
      ret

disp:
loop:
      inc r21
      add r22, r24      //add 51
      cp r22, r26       //compare 1v with the received value
      brlo loop         //if not equal, check if equal with 2v, etc
      sub r22, r24
      sub r26, r22      //subtract the voltage from the value in the register
      add r21, r25      //add 47
      ldi r22, 0
loop1:
      inc r27
      add r22, r28      //add 5
      cp r22, r26       //compare 100mv with remainder of value
      brlo loop1        //if not equal check if equal to 200mv, etc
      add r27, r25      //add 47
      ret
```

CONCLUSION

Two new features of the ATMEGA8 and ATMEGA8515 microprocessors were learned in this experiment. Serial Peripheral Interface (SPI) was a feature that was on both the ATMEGA8 and the ATMEGA8515. Enabling SPI on both these devices permitted serial communication between themselves. The ATMEGA8 was configured as master and to send data to the ATMEGA8515 which was configured as slave. After SPI was established on both of the devices, the ADC feature of the ATMEGA8 was implemented and used as the data to be sent across these devices. The data was then sent out through the atmega8515'S USART which was sent to the computers terminal to display the voltage that is being inputted into the ADC converter in real time.