Mid-Term SOEN 422

Maximilien Malderle

26562906

Question 1:

i)

F_CPU is set to 16,000,000, which corresponds to the clock frequency of the Arduino Nano, this means that there are 16 million cycles per second on the Arduino CPU clock. This needs to be defined in a C program and not in an Arduino sketch, because the Arduino IDE already knows the hardware configuration of the nano, meaning that by selecting the appropriate device, the IDE will initialize the corresponding clock frequency.

ii)

sei() and cli() are used in the context of programming interrupts. They are macros which allow the programmer to clear all interrupts with cli() and to set the interrupt using sei(). The practice here is to cli() so that any interrupts which may be unknown to the programmer can be cleared to make way for the intended interrupt which is set using sei().

iii)

DDRB |= (0 << 3) is intended to set the third bit in the Data Direction Register of PORT B to 0, which corresponds to an input on pin D11. This bit wise OR equals (|=) essentially takes the current register contents and performs an or with the third bit and the 0, which was programmer selected. If the third bit is 1, the operation would be 1 | 0, which results in a 1, so the operation would not perform what the programmer intended. If the bit is already 0, the operation would be 0 | 0, which results in 0, the intended result. The proper manipulation of the DDRB would be DDRB &= ~(1 << 3), which operates an bit wise AND (&=) between DDRB and the negation of the third bit set to one. If the third bit in the register is 0, then it would be evaluated as 0 & 0, which results in a 0, the intended result. If the third bit in the register is 1, then it would be evaluated as 1 & 0, which results in 0 the intended result.

Question 2:

Compiling :

i)

```
avr-gcc –Os –Wall =mmcu=atmega328p myfirmware.cpp –o firm.o
```

ii)

```
avr-objcopy –O ihex –j .text –j .data firm.o firm.hex
```

Uploading:

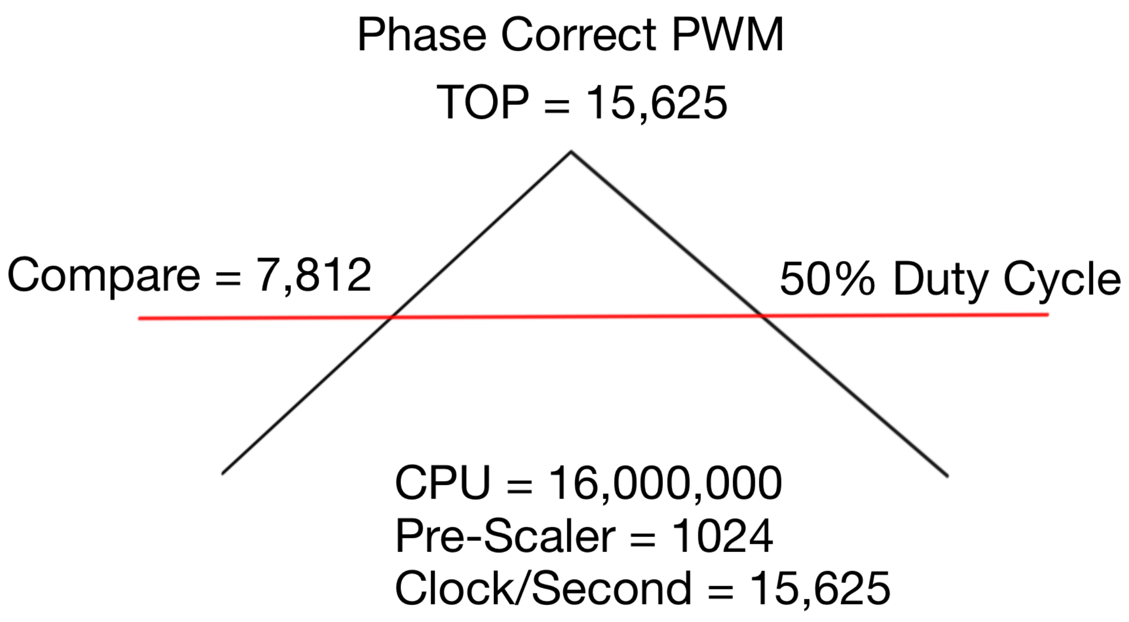
i)

```
avrdude -c arduino -P /dev/cu.wchusbserial1420 -b 57600 -p atmega328p -D -
Uflash:w:firm.hex:i
```

Question 3:

PWM)

The program is designed to have two timers, timer 1 in charge of a 1hz frequency LED, timer 0 in charge of a variable brightness and interval LED.

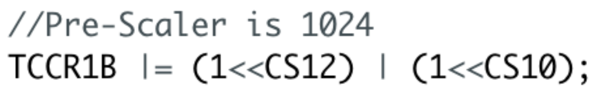


Achieving a 1 Hz flashing frequency (1 blink per second) with a Phase-Correct-PWM requires a 1 second OFF condition and a 1 second ON condition. With the nano the clock has a frequency of 16,000,000 hz, with a pre-scaler of 1024, the clock ticks 15,625 times per second. Setting the compare and match value to 7,812 represents half of the TOP, also referred to a 50% duty cycle. This means that the period between the beginning and the compare is 0.5 seconds and the period between the compare and the TOP is 0.5 seconds. This means that in the period of 1 second the LED will be ON half the time and OFF for the other half, achieving a 1hz frequency.

To achieve this in a practical environment Mode 10 of the PWM options should be selected. Mode 10 allows for a Phase-Correct-PWM with the ICR1 register as the TOP.

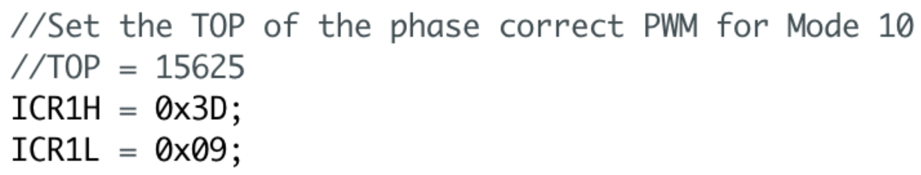
```
//PWM Phase Correct Mode 10 With ICR1 as TOP
TCCR1A |= (1<<WGM11);
TCCR1B |= (1<<WGM13);
```

Setting the pre-scaler to achieve a 15,625 clock/second period



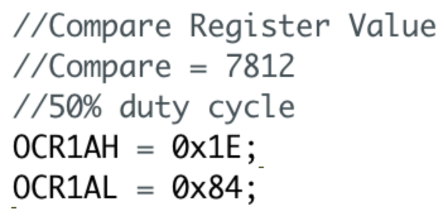
```
//Pre-Scaler is 1024
TCCR1B |= (1<<CS12) | (1<<CS10);
```

Setting the value of the ICR1 register.



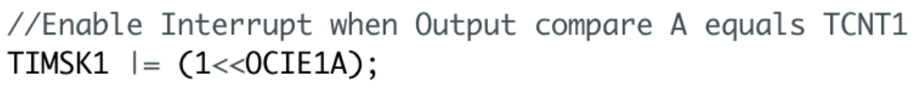
```
//Set the TOP of the phase correct PWM for Mode 10
//TOP = 15625
ICR1H = 0x3D;
ICR1L = 0x09;
```

Setting the compare register value to represent a 50% duty cycle.



```
//Compare Register Value
//Compare = 7812
//50% duty cycle
OCR1AH = 0x1E;
OCR1AL = 0x84;
```

Enabling the Output Compare Interrupt

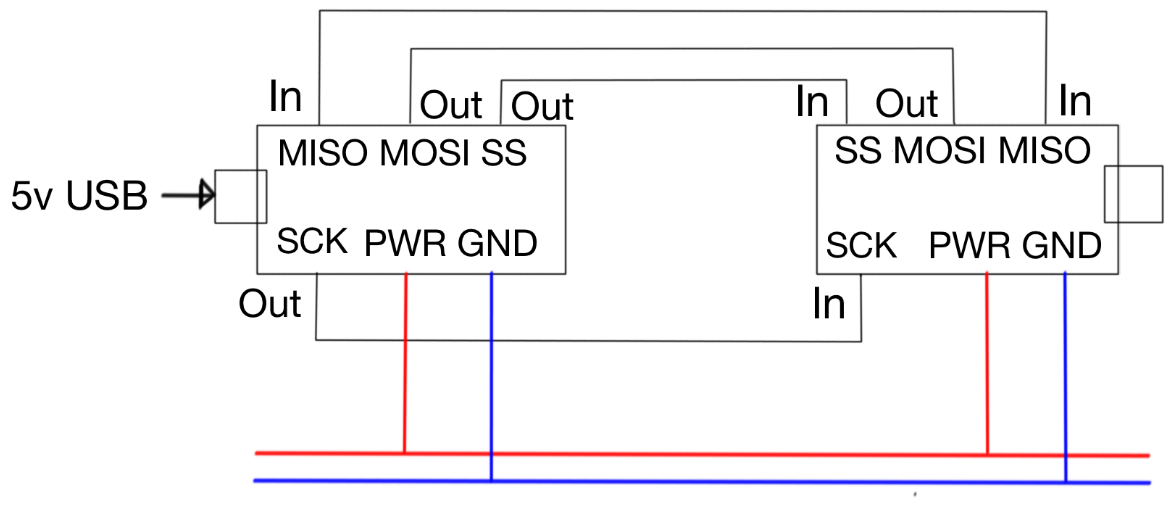


```
//Enable Interrupt when Output compare A equals TCNT1
TIMSK1 |= (1<<OCIE1A);
```

Timer 0 is set up as in Lab 1, it uses the Output compare A register value to set the brightness as a fraction of 5v (255). As for the intervals, limited to multiples of 1hz, are achieved through global counters.

SPI)

Master/Slave Communication Set-up Graph



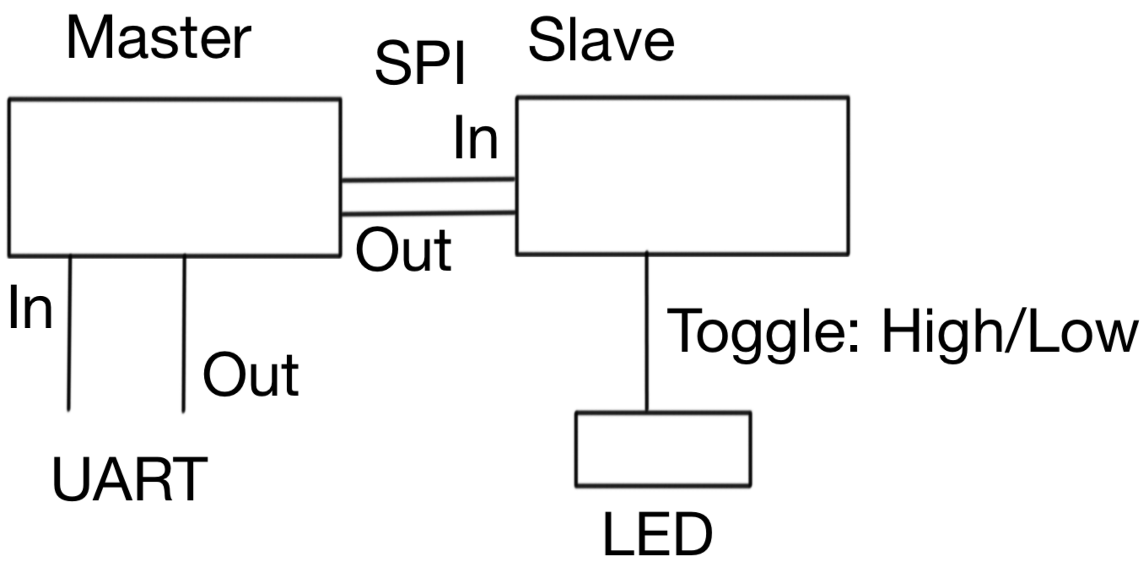
Master:

```
//Set MOSI, SCK and SS as outputs, MISO as input
DDRB |= ((1 << mosi) | (1 << sck) | (1 << ss));

//Set device as SPI master, enable SPI.
SPCR |= ( 1 << MSTR) | (1 << SPE);
```

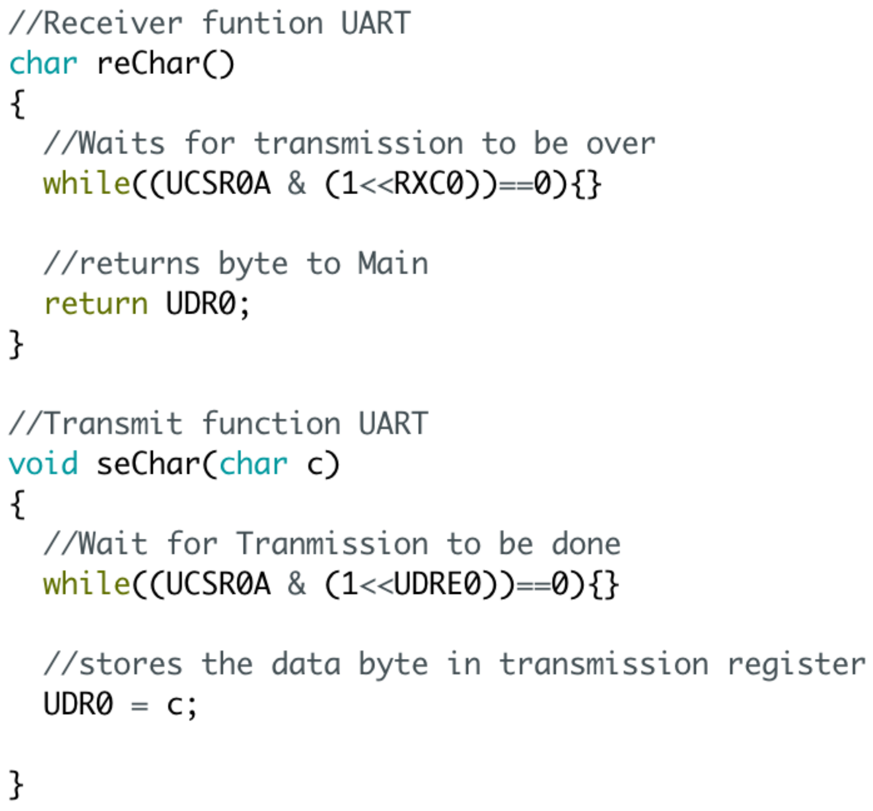
Slave:

```
//Initialization of a Slave nano
// Set MISO as output. MOSI, SCK and SS are set to inputs
DDRB |= (1 << miso);

// Set device as SPI slave just by enabling SPI
SPCR |= (1 << SPE);

//Set Pin A0 PC0 to output
DDRC |= (1<<0);
```

Master/Slave Information Dependency Graph



UART Communication Functions:



```
//Receiver funtion UART
char reChar()
{
  //Waits for transmission to be over
  while((UCSR0A & (1<<RXC0))==0){}

  //returns byte to Main
  return UDR0;
}

//Transmit function UART
void seChar(char c)
{
  //Wait for Tranmission to be done
  while((UCSR0A & (1<<UDRE0))==0){}

  //stores the data byte in transmission register
  UDR0 = c;

}
```

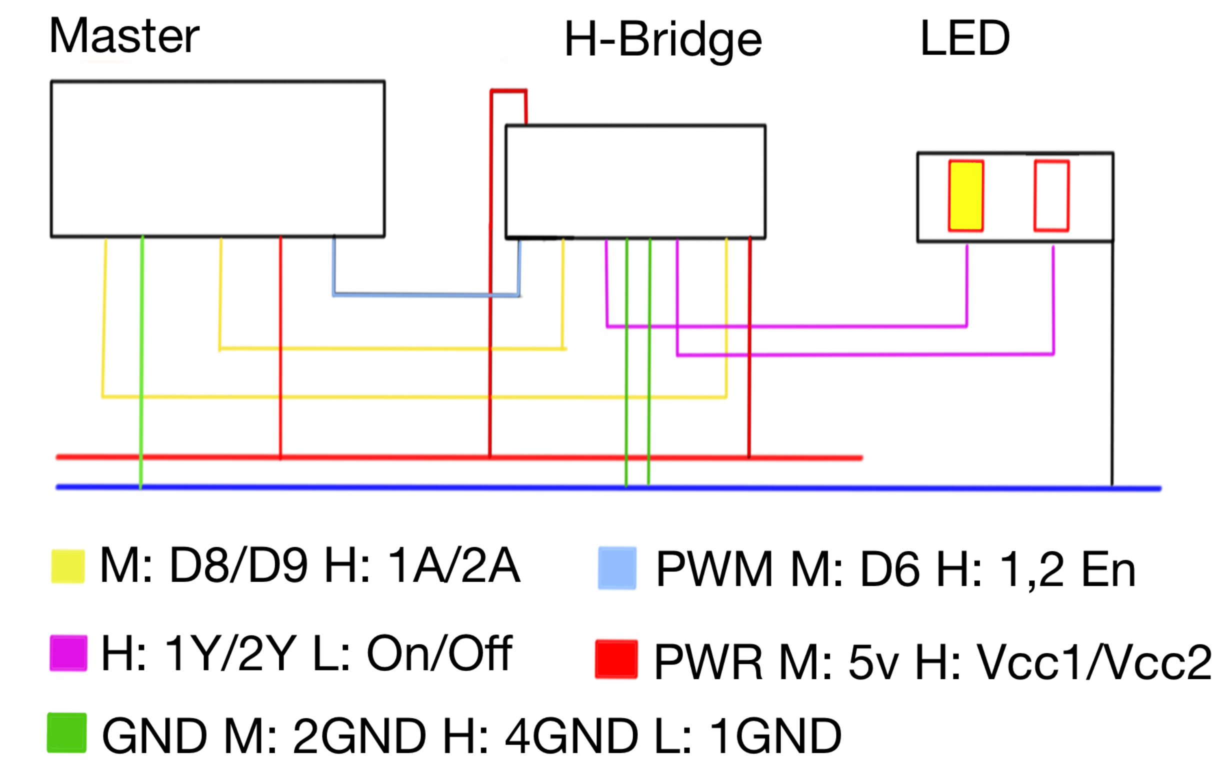
SPI Communication Functions:

```
//Receiving function
uint8_t receiver ()
{
  // Wait until interrupt flag is asserted
  //Is asserted when transmission is complete
  while ((SPSR & (1 << SPIF))==0){}

  //returns the contents of the data register
  return (SPDR);
}

//Transmitting function
void transmit (uint8_t data)
{
  //Load byte of data into data transfer register
  SPDR = data;

  // Wait until interrupt flag is asserted.
  //Is asserted when transmission is complete
  while ((SPSR & (1 << SPIF))==0){}
}
```

Toggle (Within an ISR in the slave):

```
//if Receive is a 1
if((int)rc == 49)
{
  //if LED is ON
  if(PORTC & (1<<0))
  {
    //Transmit byte representing Off
    //Turn the light Off
    transmit('0');
    PORTC &= ~(1<<0);
  }
  else
  {
    //Transmit byte representing On
    //Turn the light On
    transmit('1');
    PORTC |= (1<<0);
  }
}
```

Motor)



The Master is connected to the H-Bridge at three places, two connections are regular ON or OFF and the third is a PWM connection. The ON is linked to the output direction of the H-bridge, which is attached to where the light is ON on the LED array. The OFF is linked to the input direction of the H-Bridge, which is attached to where the light is OFF on the LED array. The PWM allows the output/input of the H-Bridge to be variable. The H-Bridge is useful for driving motors, because with this configuration, a motor can run at different speeds and in different directions. By switching ON to OFF and OFF to ON, the direction of the current flowing to the motor switches, causing the motor to turn in a different direction. Changing the value of the output compare register reduces the provided voltage. When lowering the voltage, a DC motor turns with a lower frequency.

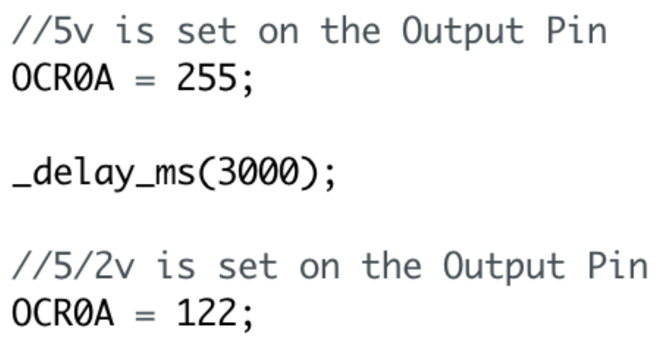
Variables/Pins:

```
//Global Variables
//Pin PB0 or D8
int oneA = 0;

//Pin PB1 or D9
int twoA = 1;

//Pin PD6 or D6
int PWM = 6;
```

Setting the Flows:

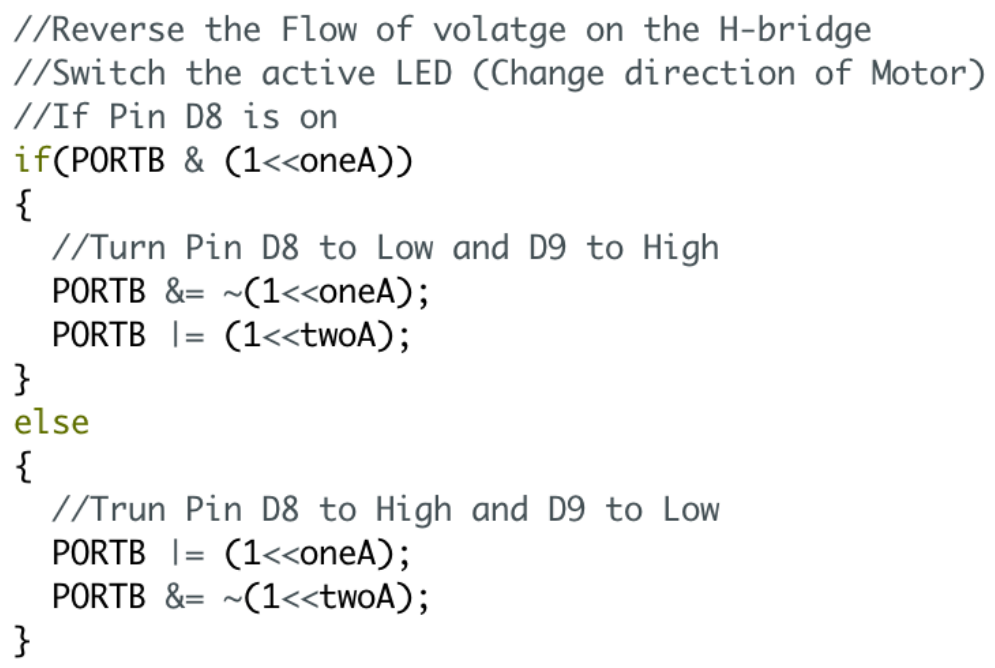
```
//Turn the Pins to Output
DDRB |= (1<<oneA);
DDRB |= (1<<twoA);
DDRD |= (1<<PWM);

//Turn Pin D8 to High
PORTB |= (1<<oneA);

//Turn Pin D9 to Low
PORTB &= ~(1<<twoA);

//Turn Pin D6 to High
PORTD |= (1<<PWM);

//Fast PWM with 255 (0xFF) TOP
//Compare Match Enabled
TCCR0A |= (1<<COM0A1) | (1<<WGM01) | (1<<WGM00);

//Pre-Scaler of 64
TCCR0B |= (1<<CS01) | (1<<CS00);
```

Changing the Compare Register/Simulating Varying Speeds



```
//5v is set on the Output Pin
OCR0A = 255;

_delay_ms(3000);

//5/2v is set on the Output Pin
OCR0A = 122;
```

Varying the HIGH/LOW on pins to Change the Direction of the Motor



```
//Reverse the Flow of volatge on the H-bridge
//Switch the active LED (Change direction of Motor)
//If Pin D8 is on
if(PORTB & (1<<oneA))
{
  //Turn Pin D8 to Low and D9 to High
  PORTB &= ~(1<<oneA);
  PORTB |= (1<<twoA);
}
else
{
  //Trun Pin D8 to High and D9 to Low
  PORTB |= (1<<oneA);
  PORTB &= ~(1<<twoA);
}
```