Lab-Prep 5 – Assembly Timers and ADC in ATMEL
Marcus Pham - Sept 2018

Revision for IO on the Arduino Nano

1. Remember to add the header file (.include “m328Pdef.inc”) at the top, and to initialise the stack pointer if needed:
   LDI R16, low(RAMEND)

   OUT SPL, R16
   LDI R16, high(RAMEND)

   OUT SPH, R16

2. Set up IO directions:

   a. DDR(B/C/D) is a special register that stores an 8-bit value representing the directions for the pins on the corresponding port. (Most significant first).
      Outputs = 1.
      Inputs = 0.

   b. To set the register, place the 8-bit value you’d like to set inside a general purpose register (eg. R16).

   c. Then set the port to the corresponding value using OUT.
      eg. To set Port B to all inputs, except B4 and B0 which are outputs:
      LDI R16, 0b00010001
      OUT DDRB, R16

3. To set pins use OUT and PORT(B/C/D), after setting up DDR.
   eg. To set Pins 4 and 0 in port B to high (internal pull-up):
   LDI R16, 0b00010001
   OUT PORTB, R16

4. To read from pins use IN and PIN(B/C/D)
   eg. To read from Pins 5 and 1 on port B:
   IN R16, PINB

Creating PWM signal, method 1 – Blocking

1. One method to create a PWM signal is to set the desired pin to high and then run a loop of commands (NOP) to delay the next line of code, before setting to low and repeating the process.

2. First thing you will need is the clock speed of the controller that you are using. For the Arduino nano (ATMEGA328P), the clock speed is 16MHz.

3. From this you can calculate the time needed for the signal to be high then create a loop that blocks the next execution a certain amount of cycles.

Creating a PWM signal, method 2 – Interrupts and Timers (Non-blocking)

However, there will be times where you still need to process other inputs and outputs whilst creating a signal. A blocking method would stop all other execution and hence make you unable to read in external inputs or do any other commands.
One way around this is to use the timers on the microcontroller.

The documentation for use of timers is available in the **full ATMEGA328P datasheet from page 93.**

1. To use timers, again you must set the 2 corresponding registers to set the timer. For our lab we will be using a 16 bit timer (TCCR1A/B). Related information is available from **page 131**

2. We are using channel A, and hence need to set the corresponding bits:

```
LDS     R16, TCCR1A ; Storing the current value of the timer register to R16
ORI     R16, (1<<COM1A1)|(1<<WGM11) ; Sets COM1A1 and WGM11, leaving the rest
STS     TCCR1A, R16 ; Stores the new configuration back to the timer register

LDS     R16, TTCR1B ; Storing the current value of the timer register
ORI     R16, (1<<WGM13)|(1<<WGM12)|(1<<CS11)|(1<<CS10) ; setting pins
STS     TCCR1B, R16 ; Storing configuration
```

   These steps set your timers to :
   - Clear OC1A/B on match (set to low, for the low section of our PWM)
   - Mode 14: Fast PWM, with ICR1 as the top.
   - Prescaler of clk/64

3. To set period, we can now set ICR1 to reflect the entire period (eq. on page 125)

```
LDI     R17, HIGH(<Value of ICR1>)
LDI     R16, LOW(<Value of ICR1>)
STS     ICR1H, R17
STS     ICR1L, R16
```

   The calculation for the values:

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

4. To set the high sections of our period we set OCR1A (Same equation as above)
   ```
   LDI     R16, <Value of OCR1>
   LDI     R17, <Value of OCR1>
   STS     OCR1AH,  R17
   STS     OCR1AL, R16
   ```

Analog to Digital Conversion (ADC)

Similar to the timers, we must set some registers to allow for ADC. From **page 234.**

1. Set ADMUX and ADCSRA:
   ```
   LDI     R16, (1<<REFS0)|(1<<ADLAR)|(0<<MUX3)
   STS     ADMUX, R16
   LDI     R16, (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
   STS     ADCSRA, R16
   ```
   This sets our reference to Vcc (should out a capacitor), Left adjust.
   Also sets MUX3-0 as 0, giving us ADC0 for use.
   Additionally, it sets the ADC control and status register A.
   This setting will set the enable, and select the prescaler mode (in this case, 128)

2. To run the ADC, you must set the start conversion pin (ADSC) on ADCSRA register
   ```
   LDS     R16, ADCSRA
   ORI     R16, (1<<ADSC)
   STS     ADCSRA, R16
   ```

3. As the conversion takes time, we must wait until the conversion is complete. This is signalled by bit 6 (ADSC), the start conversion pin becoming 0.
   So inside a loop we check that pin before using the number:
   ```
   adc_loop:
   LDS     R16, ADCSRA
   SBRC    R16, 6 ; could have used ADSC
   RJMP    adc_loop
   ```

4. So if the conversion is complete, it will break out of the loop and we can now read the value

   The value is stored in ADCH (and ADCL). This value is dependant on if we selected most significant bit first or not (ADLAR from before). This is detailed on page 250.
   ```
   LDS     R16, ADCL
   LDS     R17, ADCH
   ```