October 13, 2015 at 21:55

1. Introduction.

This converts the voltage from a potentiometer to a PWM output for heavier loads. The load then just need a switching device like a transistor. Since the switch does not operate in a partialy on state, it disipates very little power.

2. Operation The potentiometer is connected across VCC, be it 3. 3 or 5 Volts. The wiper of the pot can then have any voltage from 0 to VCC. The wiper is connected to the multiplexer. The MUX then selects this input and connects it to the ADC. The ADC is configured to use VCC as its reference voltage and to provide only 8 bits (0 – 255). For those reasons the ADC resolves 0 to VCC to an integer over the range of 0 to 255. That integer is fed to the PWM timer. The loop alternates between channels.

Extensive use was made of the datasheet, Atmel "Atmel ATtiny25, ATtiny45, ATtiny85 Datasheet" Rev. 2586QAVR08/2013 (Tue 06 Aug 2013 03:19:12 PM EDT) and "AVR130: Setup and Use the AVR Timers" Rev. 2505AAVR02/02.

⟨ Include 4 ⟩
⟨ Volts Table 17 ⟩

3. `"F_CPU"` is used to convey the Trinket clock rate.

```
#define F_CPU 8000000UL
```

4. ⟨ Include 4 ⟩ ≡

```
#include <avr/io.h>      /* need some port access */
#include <avr/interrupt.h>      /* have need of an interrupt */
#include <avr/sleep.h>      /* have need of sleep */
#include <stdlib.h>
#include <stdint.h>
  int main(void)
      {
      ⟨ Initialize ADC 12 ⟩
      ⟨ Initialize Timer 14 ⟩
      uint8_t pot = 0;
```

This code is used in section 2.

5.

```
  sei( );
```

6. Rather than burning loops, waiting for something to happen, the "sleep" mode is used. The specific type of sleep is 'idle'. In idle, execution stops but timers continue and ADC conversion begins.

```
  for ( ; ; )
  {
```

7. The selected pot toggles at the beginning of each loop.

```
  pot = (pot ≡ 0) ? 1 : 0;
```

8. Next a potentiometer is selected by setting the MUX.

```
if (pot ≡ 0) ADMUX = (ADMUX & ^#F0_U) | ^#03_U;      /* ADC3 at PB3 */
else ADMUX = (ADMUX & ^#F0_U) | ^#02_U;      /* ADC2 at PB4 */
```

9. Now we wait in "idle" for an ADC conversion (7.1.1). The counter timer, and so PWM, will continue during sleep.

```
sleep_mode( );
```

10. If execution arrives here, some interrupt has been detected. Usualy we would check for which interrupt but we will assume that it's the ADC.

Next the pot position, as determined by the ADC, is converted to a PWM value and written to the respective timer. PWM is naturally current but the table returns the PWM that would result in a voltage response..

```
if (pot ≡ 0) OCR0A = volts[ADCH];      /* PB0 PWM */
else OCR0B = volts[ADCH];      /* PB1 PWM */
}      /* end for */
return 0; }      /* end main() */
```

11. These are the configuration blocks.

12. ⟨ Initialize ADC 12 ⟩ ≡

```
{
  ADMUX |= (1 ≪ ADLAR);      /* Left adjust for an 8 bit result */
  ADCSRA |= (1 ≪ ADPS2);      /* 500 kHz ADC clock */
  ADCSRA |= (1 ≪ ADIE);      /* Interrupt on completion */
  ADCSRA |= (1 ≪ ADEN);      /* Enable ADC (prescaler starts up) */      /* DIDR0 17.13.5 */
  DIDR0 |= (1 ≪ ADC2D);      /* Disable digital on ADC2 */
  DIDR0 |= (1 ≪ ADC3D);      /* Disable digital on ADC3 */
}
```

This code is used in section 4.

13. Timer Counter 0 is configured for “Phase Correct” PWM which, according to the datasheet, is preferred for motor control. OC0A and OC0B are set to clear on a match which creates a non-inverting PWM.

14. ⟨ Initialize Timer 14 ⟩ ≡

```
{     /* 15.9.1 TCCR0A  Timer/Counter Control Register A */
  TCCR0A |= (1 ≪ WGM00);      /* Phase correct, mode 1 of PWM (table 15-9) */
  TCCR0A |= (1 ≪ COM0A1);      /* Set/Clear on Comparator A match (table 15-4) */
  TCCR0A &= ∼(1 ≪ COM0A0);      /* Set on Comparator A match (table 15-4) */
  TCCR0A |= (1 ≪ COM0B1);      /* Set/Clear on Comparator B match (table 15-7) */
  TCCR0A &= ∼(1 ≪ COM0B0);      /* Set on Comparator B match (table 15-7) */
    /* 15.9.2 TCCR0B  Timer/Counter Control Register B */
  TCCR0B |= (1 ≪ CS01);      /* Prescaler set to clk/8 (table 15-9) */
    /* 14.4.9 DDRD  The Port D Data Direction Register */
  DDRB |= (1 ≪ DDB0);      /* Data direction to output (sec 14.3.3) */
  DDRB |= (1 ≪ DDB1);      /* Data direction to output (sec 14.3.3) */
}
```

This code is used in section 4.

15. Interrupt Handling.

```
ISR(ADC_vect)
{
}
```

16. This m file code: *offset* = 9; *scale* = 0.97; **for** *power* = 1: 255*volts* = 255 ∗ (((*power* ∗ *scale* + *offset*)/255)⊕2); **if** (*volts* > #ff) *volts* = #ff; **endif** *printf* ("0x%0x", *round*(*volts*)); **if** (*rem*(*power*, 12) ≡ 0 ∨ *power* ≡ 255) *printf*(",\n"); **else** *printf*(", "); **endif** *end* was run in Octave to produce the following PWM power to PWM voltage table.

17. ⟨ Volts Table 17 ⟩ ≡

```
const uint8_t volts[256] = {#0, #0, #1, #1, #1, #1, #1, #1, #1, #2, #2, #2, #2, #2, #2, #3, #3, #3, #3, #3,
    #4, #4, #4, #4, #5, #5, #5, #5, #6, #6, #6, #7, #7, #7, #8, #8, #8, #9, #9, #9, #a, #a, #a, #b, #b, #c, #c, #d,
    #d, #d, #e, #e, #f, #f, #10, #10, #11, #11, #12, #12, #13, #13, #14, #14, #15, #15, #16, #17, #17, #18,
    #18, #19, #1a, #1a, #1b, #1b, #1c, #1d, #1d, #1e, #1f, #1f, #20, #21, #21, #22, #23, #24, #24, #25, #26,
    #27, #27, #28, #29, #2a, #2a, #2b, #2c, #2d, #2e, #2f, #2f, #30, #31, #32, #33, #34, #34, #35, #36, #37,
    #38, #39, #3a, #3b, #3c, #3d, #3e, #3f, #40, #41, #42, #43, #44, #45, #46, #47, #48, #49, #4a, #4b, #4c,
    #4d, #4e, #4f, #50, #51, #52, #53, #54, #56, #57, #58, #59, #5a, #5b, #5c, #5e, #5f, #60, #61, #62, #64,
    #65, #66, #67, #68, #6a, #6b, #6c, #6e, #6f, #70, #71, #73, #74, #75, #77, #78, #79, #7b, #7c, #7d, #7f,
    #80, #81, #83, #84, #86, #87, #88, #8a, #8b, #8d, #8e, #90, #91, #93, #94, #95, #97, #98, #9a, #9b, #9d,
    #9f, #a0, #a2, #a3, #a5, #a6, #a8, #a9, #ab, #ad, #ae, #b0, #b1, #b3, #b5, #b6, #b8, #ba, #bb, #bd, #bf,
    #c0, #c2, #c4, #c5, #c7, #c9, #cb, #cc, #ce, #d0, #d1, #d3, #d5, #d7, #d9, #da, #dc, #de, #e0, #e2, #e3,
    #e5, #e7, #e9, #eb, #ed, #ef, #f0, #f2, #f4, #f6, #f8, #fa, #fc, #fe, #ff, #ff};
```

This code is used in section 2.