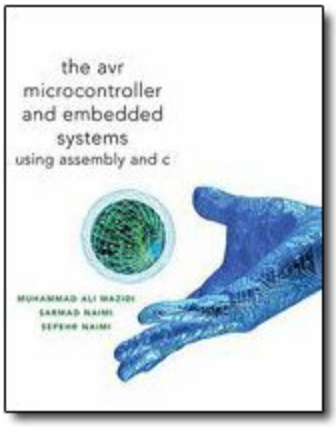
PWM Control With C++

Timer/Counter and PWM

This article is the second of a two part series on the AdaFruit Motor Shield. The first article focused on a Software implementation of the Serial Peripheral Interface (SPI). This second article in the series covers Fast Pulse Width Modulation.



The AVR Microcontroller and Embedded Systems using Assembly and C)
by Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi
Chapter 5: Arithmetic, Logic Instructions, and Programs
    Section 5.4: Rotate and Shift Instructions and Data Serialization
Chapter 7: AVR Programming in C
    Section 7.5 Data Serialization in C
Chapter 11: AVR Serial Port Programming in Assembly and C
Section 17.1 SPI Bus Protocol

ATMEL 8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash
http://www.atmel.com/dyn/resources/prod_documents/doc8161.pdfasd Chapter 14 “16-bit

Timer/Counter 1 with PWM”

Table of Contents

References

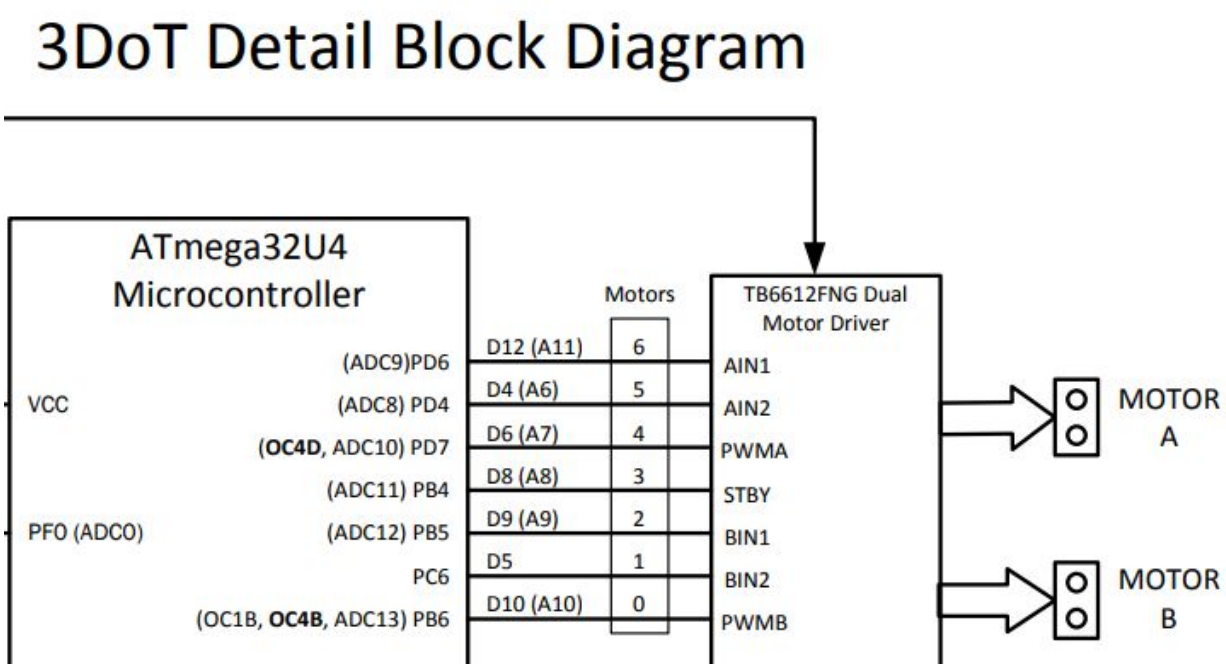
1. ATMEL 8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash http://www.atmel.com/dyn/resources/prod_documents/doc8161.pdfasd Chapter 18 “SPI - Serial Peripheral Interface”
2. ATmega328P Serial Communications (located in the EE346 Lectures folder)
3. ATmega32U4 Datasheet (Link HERE)
4. TB6612FNG Datasheet

Prerequisite

It is assumed that students can configure GPIO on their own by reading data sheet information, and an understanding of setting Pins as inputs and outputs is needed.
It is also assumed they have been exposed to basic C++ flow control methods/ functions for the examples at the end.

Speed Control *or* Fast PWM

The speed of the DC motors is controlled using pulse-width-modulation (PWM). The idea of PWM is to control the power to a motor using a high-frequency square wave. When the square wave signal is high the motor is powered ON, and when the signal is low the power is turned OFF. The speed of the motor is controlled by the fraction of time the controlling signal is ON (duty cycle = Th/Tp %, where Th = time high and Tp = clock period). The Arduino can generate square waves for PWM on digital pins 3, 5, 6, 9, 10, 11.



Before diving into the code let's examine our Pin allocation. We will be using the 3Dot for reference which uses the TB6612FNG dual motor driver. The image from the block diagram shows that we have configured 2 motors where each motor has 3 control pins and a standby (STBY) pin for the entire IC. If STBY is set low then the motors are not powered regardless of the state of the other two pins. The pins attached to AIN1/2 and BIN1/2 are digital outputs that control the rotation of the motor. The PWM inputs clearly must connect to a pin capable of a PWM output to control the speed of the motor.

While the wiring is easy to understand, the code requires some explanation. It directly accesses internal registers of the ATmega processor. The datasheet provides the information we need, so let's dive in.

The ATmega32U4 processor has 4 timer/counter (TC) modules that can be used to generate a PWM signal. They are numbered 0, 1, 3, and 4. I will use TCx convention from now on. Looking at the block diagram again we see the PWMA and PWMB are associated with OC4D and OC4B respectively. OC4X denotes an output compare with TC4. The 32U4 has a more extensive timer system than the 328p and this timer (TC4) has 3 OCRs attached to it. Conveniently for us, both of these utilize the same timer making configuration a little more straightforward.

The TC modules can be configured in various operating modes. We will investigate the "Fast PWM", for our purposes. In "Fast PWM" the output pin is set high (1) each time the counter transitions from the MAX value to BOTTOM value of the TCx. The line is then set low (0) when the counter reaches the described comparison value, this generates the PWM and therefore acts as the speed setting.

Take note though that TC4 in this microcontroller is actually a 10-bit timer. Meaning, if desired, a 1024 bit resolution could be achieved for more refined motor control. The OCRx however, is only 8-bits long and will only compare to the low 8-bits of the timer giving an easy 256 bit resolution which is consistent with arduino implementation. This means we can safely ignore the high 2 -bits of the timer as far as PWM generation is concerned.

Read page 160 of the data sheet for more informations on using the timer in 10-bit.

The TC4 is a monster! It has **5** configuration registers attached to it.

For any PWM you need:

1. Enable Fast PWM Mode
2. An appropriate timer frequency and...
3. OCR configuration

Only the bits required for Fast PWM will be covered, many configuration bits will be skipped over to keep the lecture consistent and digestible.

Enable PWM:

To enable PWM we will use Table 15-18:

Table 15-18. Waveform Generation Mode Bit Description

PWM4x	WGM41..40	Timer/Counter Mode of Operation	TOP	Update of OCR4x at	TOV4 Flag Set on
0	xx	Normal	OCR4C	Immediate	TOP
1	00	Fast PWM	OCR4C	TOP	TOP
1	01	Phase and Frequency Correct PWM	OCR4C	BOTTOM	BOTTOM
1	10	PWM6 / Single-slope	OCR4C	TOP	TOP
1	11	PWM6 / Dual-slope	OCR4C	BOTTOM	BOTTOM

We need to set the bits to match row #2.

PWM4x means we have to find a unique configuration bit for OC4D, and OC4D. The WGM41:WGM0 pair is for all of TC4.

WGM Bits are in TCCR4D:

TCCR4D – Timer/Counter4 Control Register D

Bit	7	6	5	4	3	2	1	0	
	FPIE4	FPEN4	FPNC4	FPES4	FPAC4	FPF4	WGM41	WGM40	TCCR4D
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	

PWM B and D bits are found in TCCR4A and TCCR4C respectively.

TCCR4A – Timer/Counter4 Control Register A

Bit	7	6	5	4	3	2	1	0	
	COM4A1	COM4A0	COM4B1	COM4B0	FOC4A	FOC4B	PWM4A	PWM4B	TCCR4A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

TCCR4C – Timer/Counter4 Control Register C

Bit	7	6	5	4	3	2	1	0	
	COM4A1S	COM4A0S	COM4B1S	COMAB0S	COM4D1	COM4D0	FOC4D	PWM4D	TCCR4C
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	

We will come back to these registers later for additional configuration.

NOTE: Upon further reading the arduino defaults to a 125kHz- Phase correct mode at startup. WGM1:0 = 0:1. Normally this level of detail is not needed because the embedded arduino startup function “init()” found in “wiring.c” will initialize all the prescalers for you. This discussion is provided to enable you to build PWM systems on your own without arduino at all in the future.

Frequency:

The arduino functions hide this from us, but the PWM signal we can use is limited by the motor driver. Look at the TB3312FNG datasheet, from page 3 on the table “ Operating Ranges.”

Operating Range (Ta=-20 ~ 85℃)

Characteristics	Symbol	Min	Typ.	Max	Unit	Remarks
Supply voltage	Vcc	2.7	3	5.5	V	
	VM	4.5	5	13.5	V	
Output current (H-SW)	Iout	---	---	1.0	A	VM 5V
		---	---	0.4		5V > VM 4.5V
Switching frequency	fPWM	---	---	100	kHz	

The PWM frequency is limited to less than 100 kHz. Since we have an 8Mhz F_{IO} we can determine the prescaler needed to be within those limits.

$$\frac{8M}{.1M} = 80$$

This shows we need a minimum prescaler of 80, the closest power of 2 to that is 2^(7) or 128. This configuration is found in TCCR4B.

15.12.2 TCCR4B – Timer/Counter4 Control Register B

Bit	7	6	5	4	3	2	1	0	
	PWM4X	PSR4	DTPS41	DTPS40	CS43	CS42	CS41	CS40	TCCR4B
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	

Looking at table 15-14:

Table 15-14. Timer/Counter4 Prescaler Select

CS43	CS42	CS41	CS40	Asynchronous Clocking Mode	Synchronous Clocking Mode
0	0	0	0	T/C4 stopped	T/C4 stopped
0	0	0	1	PCK	CK
0	0	1	0	PCK/2	CK/2
0	0	1	1	PCK/4	CK/4
0	1	0	0	PCK/8	CK/8
0	1	0	1	PCK/16	CK/16
0	1	1	0	PCK/32	CK/32
0	1	1	1	PCK/64	CK/64
1	0	0	0	PCK/128	CK/128
1	0	0	1	PCK/256	CK/256
1	0	1	0	PCK/512	CK/512
1	0	1	1	PCK/1024	CK/1024
1	1	0	0	PCK/2048	CK/2048
1	1	0	1	PCK/4096	CK/4096
1	1	1	0	PCK/8192	CK/8192
1	1	1	1	PCK/16384	CK/16384

The Stop condition provides a Timer Enable/Disable function.

We will desire CS43-CS40 bit values of 0b1000 to set our 128 prescaler.

Be aware that if power is a big consideration that the timer is **disabled** until the prescaler is set. Leaving the prescaler off as late as possible can save some power. In addition note that if building a project in the arduino environment that **all** the timers are operating by default.

OCR Configuration

The comparison registers are identified as OCR4D and OCR4B. The OCR configuration bits will gives us control over how we want the pin to change when the timer hits the provided compare value.

We will need TCCR4A and TCCR4C again for these bits.

TCCR4A – Timer/Counter4 Control Register A

Bit	7	6	5	4	3	2	1	0	
	COM4A1	COM4A0	COM4B1	COM4B0	FOC4A	FOC4B	PWM4A	PWM4B	TCCR4A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

TCCR4C – Timer/Counter4 Control Register C

Bit	7	6	5	4	3	2	1	0	
	COM4A1S	COM4A0S	COM4B1S	COMAB0S	COM4D1	COM4D0	FOC4D	PWM4D	TCCR4C
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	

The COMnX (n is a letter, x is a bit number) bits do different things for different PWM modes. We will be looking at the Fast PWM table.

This table / description is for COMAx but it the same for COMDx and COMBx in Fast PWM.

- **Bits 7, 6 - COM4A1, COM4A0: Comparator A Output Mode, Bits 1 and 0**

These bits control the behavior of the Waveform Output (OCW4A) and the connection of the Output Compare pin (OC4A). If one or both of the COM4A1:0 bits are set, the OC4A output overrides the normal port functionality of the I/O pin it is connected to. The complementary $\overline{OC4B}$ output is connected only in PWM modes when the COM4A1:0 bits are set to "01". Note that the Data Direction Register (DDR) bit corresponding to the OC4A and $\overline{OC4A}$ pins must be set in order to enable the output driver.

The function of the COM4A1:0 bits depends on the PWM4A, WGM40 and WGM41 bit settings. The table shows the COM4A1:0 bit functionality when the PWM4A bit is set to Normal Mode (non-PWM).

The table shows the COM4A1:0 bit functionality when the PWM4A, WGM40 and WGM41 bits are set to fast PWM mode.

Table 15-6. Compare Output Mode, Fast PWM Mode

COM4A1..0	OCW4A Behavior	OC4A	$\overline{OC4A}$
00	Normal port operation	Disconnected	Disconnected
01	Cleared on Compare Match. Set when TCNT4 = 0x000.	Connected	Connected
10	Cleared on Compare Match. Set when TCNT4 = 0x000.	Connected	Disconnected
11	Set on Compare Match. Cleared when TCNT4 = 0x000.	Connected	Disconnected

The "notted" pins do exactly what it says. If OC4B is set, then OC4B not pin is the opposite. We will not be using them as they reside on separate pins that are not used for our motors PWM. For example, !OC4D is on Pin PD6.

The description also says that for the PWM to work we need to set DDRD of the ports as outputs. So we will be enable DDRD - PD7 and DDRB - PB6 from looking at the block diagram.

Our configuration for the motors will use COM4X1..0 = 0b10.

The last part of the configuration is setting the OCR threshold which will generate the PWM signal. We will be working from OCR4D and OCR4B located here:

OCR4B – Timer/Counter4 Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	OCR4B
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	

OCR4D – Timer/Counter4 Output Compare Register D

The output compare register D is an 8-bit read/write register.

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	OCR4D
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	

Write directly to these registers to augment your PWM duty cycle.

Example Outline:

Handling PWM in C++ will be divided into two functions(or sections).

1. Configure the timer and OCR
2. Set OCR value as needed to change PWM

```
void RobotPWM(){
// Configure Timer4 for PWMs
// Motor A on PD7 (OC4D)
// Motor B on PB6 ( OC4B)
// Ignore 10-bit mode for ease of use

//Need to configure Timer4 for fast PWM
// PWM4D and PWM4B set with WGM4 1:0 = 0b00
//Setting WGM = 00
TCCR4D &= ~(_BV(WGM41) | _BV(WGM40));
// Set PD7 and PB6 as outputs
// I have also added digital pins since they are part of the same
system
// If i want the PWM then I want the digitals also

//MOTOR PINS
//Motor A PD6,4,7
DDRD |=  _BV(PD7) | _BV(PD6) | _BV(PD4);  // 0xD0
// Default to Low output
PORTD &= ~(_BV(PD7) | _BV(PD6) | _BV(PD4));

//Motor B PB5,6 and  and STBY = PB4
DDRB  |=    _BV(PB6) | _BV(PB5) | _BV(PD4) ;          //0x70
PORTB &=    ~(_BV(PB6) | _BV(PB5 | _BV(PD4)));

//And for motor B PC6 BIN2
DDRC  |=    _BV(PB6) ;          //0x40
PORTC &=    ~(_BV(PB6));


//Setting PWM4B and COMB
TCCR4A |= (_BV(COM4B1)| _BV(PWM4B));
TCCR4A &= ~(_BV(COM4B1));
//Setting PWM4D and COMD
```

```
TCCR4C |= (_BV(COM4D1)| _BV(PWM4D));
TCCR4A &= ~(_BV(COM4D0));

//SetPrescaler - turn on timer
//Assumes *Mhz external with defult fuses (making Fio = 1Mhz)
//TB66612FNG says wants PWM Freq <= 100k
// Fio = 8Mhz
// 8M / 128 = 62.5kHz

//CS4 4:0 = 0b1000;
TCCR4B |= _BV(CS43);
TCCR4B &= ~(_BV(CS42)| _BV(CS41)|_BV(CS40)); // Clear prior settings
from arduino.
}
```

With the configuration ready we can make a function to set our PWM by change the OCR threshold.

```
void setPWM(char Pin, uint8_t val){

    // Only using 8-bit mode so value is from 0-255 as normal.
    // This assumes proper values passed, if block should protect
from bad inputs
    //if (Pin == 'A'| Pin == 'B') (Pin == 'A' ) ? (OCR4D = val) :
(OCR4B = val) ;
    // looking at more stuff this ternary probably won't work ):

// wanted something more elegant that  switch, but it works.
// would have to make own methods like arduino has it set up to be
really clean.
  switch (Pin){
  case 'A':                // Due to ASCII, this is case sensitive,
change if you wish.
    OCR4D = val;
    break;

  case 'B':
    OCR4B = val;

    break;

default:
```

```
  Serial.println("Invalid Motor Pin");

  break;
  }
}
```

Additional Notes:
This is a great starter to understand building the PWM implementation from the ground up. Ideally to avoid pin confusion(like with the Switch case) I would make a Motor object where the pin is specific to that motor. This would also help make the code more readable.