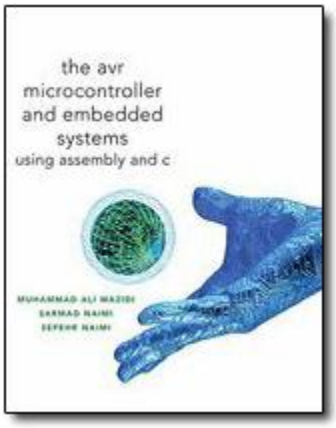
Adafruit Motor Shield - Part 2

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. Adafruit Motor Shield Arduino User Guide pages 4 and 17

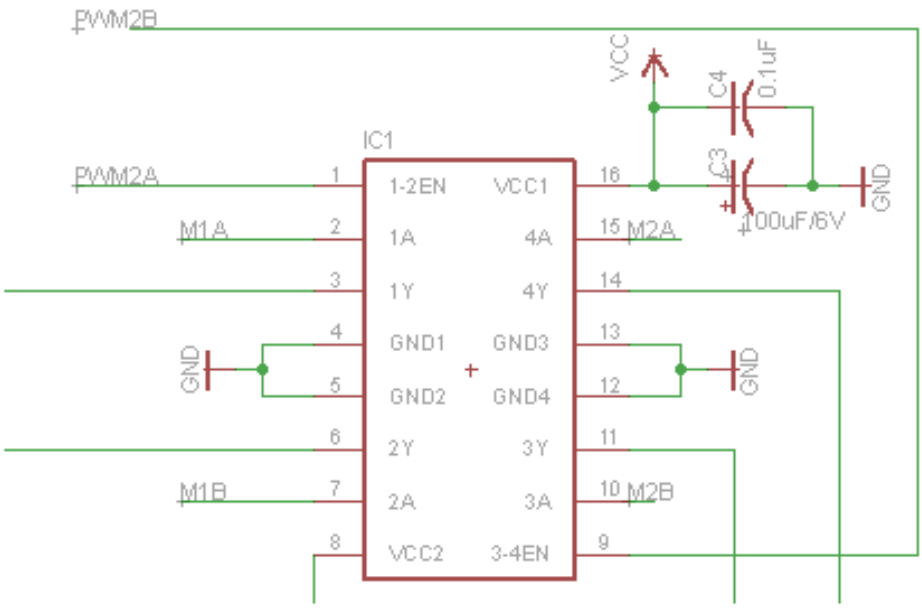
Speed Control *or* Fast PWM

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.

Here is a cross-reference table for mapping the *tower of babel* names used by Atmel, Arduino, Adafruit, and TI Semiconductor

Atmel AVR ATmega328P		Arduino Digital		Adafruit	TI 293D			
Port	bit	name	Pin	Jumper	Name	IC	Pin	Name	Description
D	3	OC2B/INT1	3	J1-4	PWM2B	1	9	3-4EN	
D	5	OC0B/T1	5	J1-6	PWM0B	2	1	1-2EN	
D	6	OC0A/AIN1	6	J1-7	PWM0A	2	9	3-4EN	
B	1	OC1A	9	J3-2	PWM1A				SER1
B	2	SS/OC1B	10	J3-3	PWM1B				SERVO_2
B	3	MOSI/OC2A	11	J3-4	PWM2A	1	1	1-2EN	

The wiring of the PWM pins for speed control is straightforward. On the motor driver ICs, in addition to the two control inputs for each motor, there is an enable input. If it is set low then the motor is not powered regardless of the state of the other two pins. Four of the PWM pins are wired directly to the four enable inputs on the two motor drivers. This detail from the schematic shows two PWM inputs to the driver for motors 1 and 2.

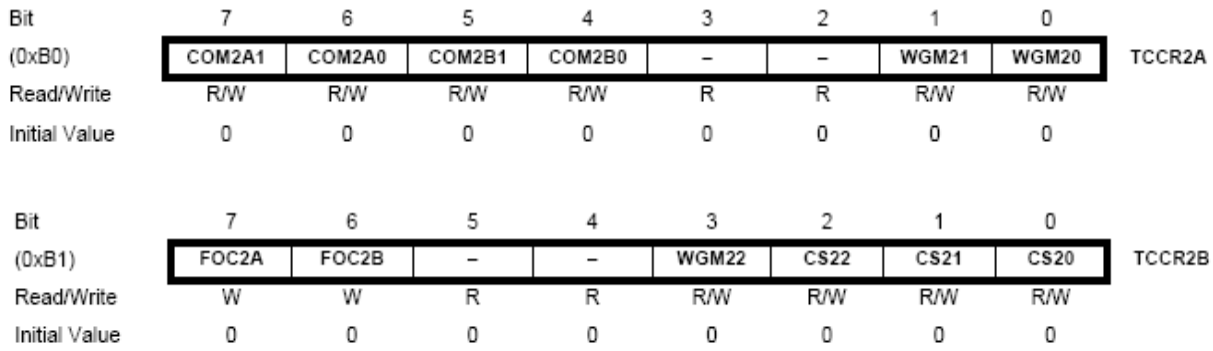


While the wiring is easy to understand, the code requires some explanation. It directly accesses internal registers of the ATmega processor, and the only detailed documentation I could find was in the datasheet. So let's dive in.

The ATmega processor has three timer/counter (TC) modules that can be used to generate a PWM signal. They are numbered 0, 1, and 2, and I'll refer to them as TC0, TC1, TC2. Conveniently the motor shield schematic uses the same numbering, so in the above figure you can see that TC2 is being used to control motors 1 and 2. Basically, the way the TC modules work is that a clock increments a counter on each clock cycle. Each TC has two associated comparison registers and two associated output pins; a pin can be set low or high depending on a comparison between the counter and the corresponding comparison register.

The modules can be configured in various operating modes, but in the "Fast PWM" mode that is used by the library, the output pin is set high (1) each time the counter transitions from FF_{16} (255_{10}) to 00_{16} and set low (0) when the counter reaches the comparison value, which therefore acts as the speed setting. TC0 and TC2 each have an 8-bit counter, which is why the speed range for the motors is 0-255.

Each TC module is controlled by two registers. TC2 is controlled by registers TCCR2A and TCCR2B, which have the following bit layouts:



Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The comparison registers are identified as OCR2A and OCR2B. The AVR headers define assignable macros with names identical to the register names as given in the datasheet. With this background, the following code becomes fairly clear. This is a brief excerpt of the AF_DCMotor constructor (`subroutine initPWM1(freq)`), which is initializing speed control for motor 1:

```
// use PWM from timer2A
TCCR2A |= _BV(COM2A1) | _BV(WGM20) | _BV(WGM21); // fast PWM, turn on OC2A
TCCR2B = freq & 0x7;
OCR2A = 0x00;
DDRB |= _BV(3);
break;
```

This can really only be understood in full detail by looking at the tables in the datasheet, which I don't want to reproduce here. However, here is a brief description of what these specific bit settings mean:

- WGM20 = 1, WGM21 = 1: selects the "fast PWM" mode described above.
- COM2A1 = 1, COM2A0 = 0: in fast PWM mode, causes the output to be set low when the comparison value is reached (non-inverting mode). *If this is correct then the program contains a bug. The above C++ expression will set a bit if it is cleared. It will not clear a bit if it is set. Therefore, if COM2A0 was 1 it will remain 1 -- gch.*
- OCR2A = 0: initializes the comparison value (i.e. the speed setting) to zero.
- Three low-order bits of TCCR2B; chooses the frequency of the PWM signal.
- DDRB bit 3 set high: sets the pin mode for the pin corresponding to output 2A. This is pin 11. This correspondence is defined by the ATmega chip itself and can't be changed.

Timer/Counter 0 (TC0) has analogous control registers, just with "2" replaced by "0" in the names.

Finally, here's the body of the setSpeed() function. All it does is set the comparison register for the appropriate motor:

```
void AF_DCMotor::setSpeed(uint8_t speed) {
  switch (motornum) {
  case 1:
      OCR2A = speed; break;
  case 2:
      OCR2B = speed; break;
  case 3:
      OCR0A = speed; break;
  case 4:
      OCR0B = speed; break;
  }
}
```

One thing I had hoped to understand out of all this but haven't completely figured out is why no frequency control is supported for motors 3 and 4. TC0 has the same 3 bits for controlling the frequency that TC2 has, so it should be possible. It may have to do with the fact that TC0 and TC1 share the same prescaler, and TC1 is used to control the servos; but I'm not sure.

Using the Motor Shield

Power

The Motor Shield provides for two (2) power sources. The first source provides power to the digital logic (74HC595 and digital side of L293D H-Bridge). This power comes from the J1 header of the Arduino. The motor power comes from the EXT_PWR connector on the Motor Shield. You have the option of running the motors supply from the Arduino. This is done by installing the Power Jumper. This is not recommended!

Program

To see how all the subroutines written earlier come together to operate a motor, look at sample scripts provided by Adafruit. Here is a DC motor script example.

```
#include <AFMotor.h>

AF_DCMotor motor(2, MOTOR12_64KHZ); // create motor #2, 64KHz pwm

void setup() {
  Serial.begin(9600); // set up Serial library at 9600 bps
  Serial.println("Motor test!");

  motor.setSpeed(200); // set the speed to 200/255
}

void loop() {
  Serial.print("tick");

  motor.run(FORWARD); // turn it on going forward
  delay(1000);

  Serial.print("tock");
  motor.run(BACKWARD); // the other way
  delay(1000);

  Serial.print("tack");
  motor.run(RELEASE); // stopped
  delay(1000);
}
```

Acknowledgment

This document is based on an article on the Adafruit Motor Shield written by Michael Koehrsen on 5/26/2009. Original material is used by permission.