

Servo Setup with the TMX320F28035

Jerrod Kempf

ECE 480

Design Team 8 – RC Car Drone

11-8-2013

Abstract: This application note will detail the incorporation of a servo into a system that is run off a microcontroller. The coding of the microcontroller and power supply considerations are outlined in detail so that a system can fully incorporate a servo in its functioning.

Servo Fundamentals

A servo is a simple type of motor with 3 input lines, a supply line, a ground line and a control line. The position of the motor is determined by the signal that is received on the control line. For most servos, this control line requires a pulse width modulation signal with a frequency of about 50 Hz, although there is a large tolerance. In cases where the frequency is not exactly 50 Hz, the actual position is determined not by the duty cycle, but rather the duration of a pulse. The gear will be approximately in the center position if the duration of the pulse is 1.5 ms. For example, if a period of 20 ms is used for the pulse width modulation, this will correspond to a duty cycle of around 7.5%. To turn the gear 90 degrees, the pulse width can be modified to be 1 ms or 2 ms, depending on the desired direction of the turn. In many applications this full 180 degree range of motion is not necessary and the pulse width will not vary across the full range. Other less common ranges are available such as 90 degrees and 360 degrees which require a similar input signal.

Specific Application – TMX320F28035 System Incorporation

The following several pages will detail how to configure an existing project ran off of a microcontroller to also incorporate the use of a servo. This servo will allow the system to enable a moving mechanism which can be used for turning a steering rod, activating a robotic limb, rotating an auger or many other functions. Since the servo requires a specific input signal in order to function, we must provide this signal from some source. The code on the following page in Figure 1 illustrates the programming required to output a pulse width of 1.5 ms on one of the pins of the TMX320F28035 microcontroller which is a C2000 family microcontroller created by Texas Instruments. The code utilizes the PWM6A pin to output the desired pulse width.

```
// Module EPWM6A Configuration. Uses "Asymmetrical" Up-Count Mode.
EALLOW ;
SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1 ;
// ePWM6
SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1 ; // Enable TBCLK
GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 0x1 ; // Select EPWM6A (0=GPIO, 1=EPWM6A, 2=Resv, 3=ADC SOC-
B)
EPwm6Regs.TBCTL.bit.HSPCLKDIV = 0x0 ; // Set TBCLK = SYSCLKOUT
EPwm6Regs.TBCTL.bit.CLKDIV = 0x7 ; // Set TBCLK = SYSCLKOUT/128 = 468.75kHz
EPwm6Regs.TBCTL.bit.PRDL = 0x0 ; // Period Register loaded from Shadow Register
EPwm6Regs.TBPHS.half.TBPHS = 0 ; // Set Phase Register to zero
EPwm6Regs.TBCTL.bit.PHSEN = 0x0 ; // Phase loading disabled
EPwm6Regs.TBCTL.bit.CTRMODE = 0x0 ; // Up-Count Asymmetrical Mode
EPwm6Regs.TBCTL.bit.SYNCSEL = 0x3 ; // Synchronization Output disabled
EPwm6Regs.TBPRD = 9375 ; // 9375 for 50Hz pwm with TBCLK = SYSCLKOUT/128 = 468.75kHz
EPwm6Regs.AQCTLA.bit.ZRO = 0x2 ; // Force EPWM6A HIGH at start of pwm interval (when counter = 0)
EPwm6Regs.AQCTLA.bit.CAU = 0x1 ; // Force EPWM6A LOW when counter = CMPA.
EPwm6Regs.CMPCTL.bit.SHDWAMODE = 0x0 ; // Shadow CMPA
EPwm6Regs.CMPCTL.bit.SHDWBMODE = 0x0 ; // Shadow CMPB
EPwm6Regs.CMPCTL.bit.LOADAMODE = 0x0 ; // Shadow load on counter = 0
EPwm6Regs.CMPCTL.bit.LOADBMODE = 0x0 ; // Shadow load on counter = 0
EPwm6Regs.CMPA.half.CMPA = 703 ; // 703 FOR ~7.5% DUTY CYCLE ADJUST: between 0 and
EPwm6Regs.TBPRD = 9375
```

Figure 1 – TMXF28035 Code

The result of this code is a pulse width modulation output on pin 65 of the 80 pin package. The first step of the code is to set the EALLOW bit high which will enable CPU writes to certain pins which are otherwise protected to prevent random or unintended writes. From here, we must go through clock setup including the enabling of clocks related to the pulse width modulation circuitry. Pin 65 must also have EPWM6A set as its function which is done by setting the GPIO10 bit in the GPAMUX1 register to 1. One major hurdle is that the main clock on the microcontroller runs at 60 MHz, however we need our pulse width modulation frequency to be at 50 Hz. To resolve this, we can reduce the TBCLK speed by a factor of (HSPCLKDIV * CLKDIV). In the code above, this value has been set to 1*128 to allow a TBCLK setting of 468.75 kHz.

The generation of the pulse width modulation can be below in figure 2.

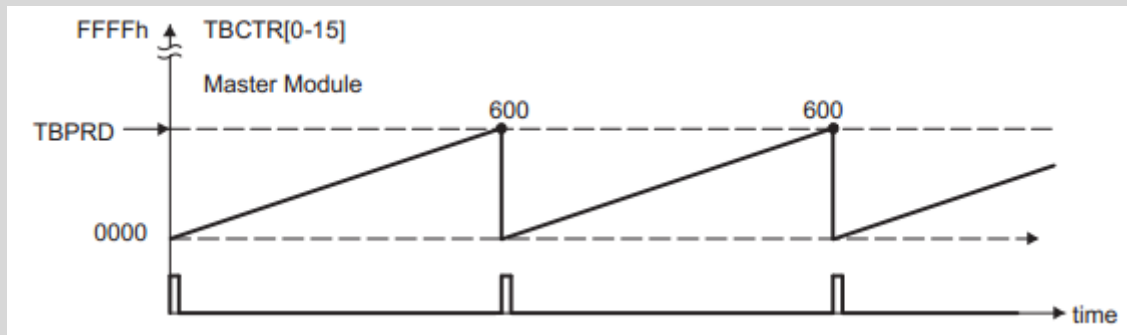


Figure 2 – PWM Scheme [1]

By using upcount for the triangle wave (a scheme that counts from 0 up to a value and then resets to 0 creating an asymmetric triangle wave) and counting up to a TBPRD value of 9375 instead of 600, we can achieve an effective frequency for the triangle wave signal of $468.75 \text{ kHz} / 9375 = 50 \text{ Hz}$. From here we need to specify a duty cycle. For the servo position to be in the center we should have a duty cycle of around 7.5% to allow for a 1.5ms pulse width. This is achieved by setting the CMPA bit to 703. This is because the pulse width output will remain high until the counter reaches the value of CMPA. In other words, the signal will be high for counter values from 0 to CMPA and low for counter values of CMPA + 1 to 9375. This gives us a duty cycle of $703/9375$ or about 7.5% when CMPA is set to 703. From here if we wish to change the position of the servo we simply have to change the value in CMPA.

Power Sourcing

Another major factor when using a servo in an application is the power supply that it is being run off of. Servo datasheets often have multiple operating voltages listed, typically at 4.8 volts and 6 volts although any intermediate voltage is also acceptable. Running at a higher voltage will allow the servo to turn faster and with higher torque which may be required depending on the project parameters although the tradeoff of this is that it will use more power. Regardless a voltage other than the microcontroller voltage of 3.3 volts is required. One way of resolving this is to have a system voltage created from a higher voltage battery of 7.4 volts which is readily available from most

suppliers. From here, regulators can be used to supply the voltage needed to both the microcontroller and the servo. Figure 3 below shows a circuit that is capable of providing 5 volts to the servo supply.

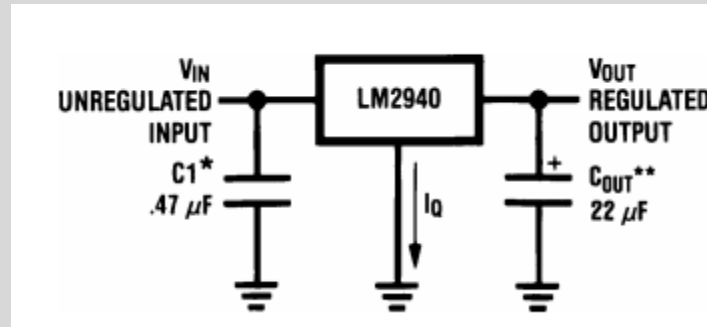


Figure 3 – Regulator Circuit [2]

In this circuit, the capacitors at the input and output are required to ensure that the output is stable. The input voltage must be larger than the intended output voltage of 5 volts plus the dropout voltage which is typically .5 volts for the LM2940.

When choosing a regulator for a servo it is also important to look at the current that can be supplied at the output. Servos, when turning, can draw very large amounts of current (up to 1A for higher torque servos) which is another reason why powering through a microcontroller is not possible. In this case the LM2940 is able to provide the required current. A similar method can be used to power the microcontroller with an LM3940 regulator which supplies an output voltage of 3.3 volts.

At this point we have successfully incorporated a servo into our system being run from the TMX320F28035 microcontroller. A block diagram of the overall setup can be seen below in Figure 4.

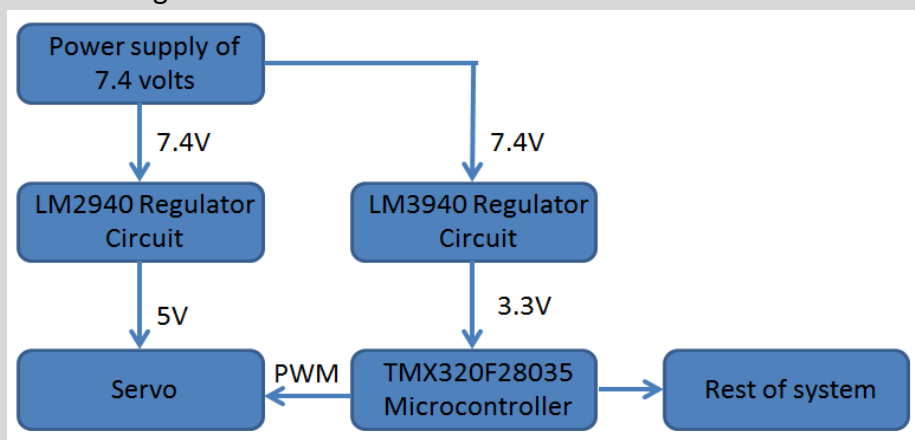


Figure 4 – Block Diagram of Proposed Circuit

To review, the previous circuit modification has taken a system that was being controlled by the TMX320F28035 microcontroller and configured it to enable the use of a servo. Depending on the application this servo could be used for anything from steering a wheeled vehicle to bending a robotic limb.

References

1. Texas Instruments, “TMS320x2802x, 2803x Piccolo Enhanced Pulse Width Modulator (ePWM) Module” TMX320F28035 Reference Guide, Dec. 2008 [Revised Mar. 2011]
2. Texas Instruments, “LM2940-N/LM2940C 1A Low Dropout Regulator” LM2940 datasheet, Mar. 2000 [Revised Apr. 2013].