CMPE2150 SA 09

Servo Motors

Taylor, Moore, and Armstrong

01 Nov 2024

Table of contents

This exercise gets you to investigate the basic characteristics and operation of the S0025P servo motor in your CNT Year 2 Kit.

Note that the specifications indicate two conflicting values for timing, neither of which appear to work: $1520\mu s$ is approximately $667Hz$, not the $333Hz$ indicated; and attempting to operate this servo at either of those frequencies has proven to be anything but optimal, if the motor operates at all.

We'll start by assuming our servo motors require a duty cycle range from 3% to 12%, with a 90° midpoint of 7.5%. You may need to adjust this for your particular motor. Instead, we'll operate it at a fairly typical servo frequency of $50Hz$ ($20ms$ pulse).

As you may not yet have studied the PWM module of your MC9S12XDP512 microcontroller, here's a quick tutorial.

Operating the microcontroller with a $20MHz$ bus frequency allows for cleaner timing than using the basic 8 MHz bus, so the following assumes you've set up your 9S12 to operate with a 20 MHz bus. The PWM module contains eight 8-bit PWM channels, with the option to combine two different pairs of channels into 16-bit PWM channels. The following lines of code will combine channels 6 and 7 into a 16-bit channel called "67", sets the polarity to positive pulses, and makes one of the variables in the calculation 2^0, or 1:

```
PWMCTL |= 0b10000000; //concatenate channels 6 and 7 for 16 bits
PWMCLK |= 0b10000000; //Use scaled clock (helps get 3 ms period
PWMPOL |= 0b10000000; //Not inverted duty cycle
PWMPRCLK &= 0b10001111; //CLK_B prescaler set to 0 (bits 6, 5, 4)
```

We'll get lots of resolution if we make the period of our PWM signal 10,000 cycles of the PWM clock:

```
PWMPER67 = 10000; //lots of resolution, clean number: the duty cycle can be set using percent x
```

Now, let's set PWMSCLB, the PWM clock for this combined channel, so that the period is $20ms$ (for $50Hz$). Here's the calculation:

$$Period = BusClockPeriod * 2^{CLK_B} \cdot 2 \cdot PWMSCLB \cdot PWMPER67$$

Since we know the desired PWM period ($20ms$), the Bus Clock Period, CLK_B (from above) and `PWMPER67` (also from above), we can rearrange this to get what we need to put into `PWMSCLB`.

`PWMSCLB = _____ ;`

We want to start with the servo motor in the middle of its range, i.e. at a duty cycle of 7.5%.

`PWMDTY67 = _____ ;`

Lastly, we need to turn on the PWM channel:

```
PWME |= 0b10000000;
```

Putting all of that together, you should have your microcontroller set up to deliver a $20ms$ pulse with a 7.5% duty cycle, and the motor should move to the middle of its available range, or 90°.

Use your $6V_{DC}$ battery pack to power the servo motor PWM Channel "67" is available at Pin 109 of the 9S12 microcontroller Make sure you establish a common ground between the microcontroller and the motor's power supply

Now it's up to you to create the remaining part of the program to do the following:

1. Allow approximately 10 ms between adjustments so that the motor's position changes relatively slowly
2. First, increment the duty cycle by 0.05% per adjustment until it reaches the upper limit
3. Next, decrement the duty cycle by 0.05% per adjustment until it reaches the lower limit: 0.05% adjustment of the duty cycle requires the addition or subtraction of _____ to or from `PWMDTY67`.

4. Make the process repeat endlessly.

To make the motion of the servo obvious, place the single arm extension available in the servo motor kit on the output shaft.

If your servo motor doesn't cover 180° motion or covers more than 180° of motion, or if your middle value doesn't place the motor at 90°, adjust the duty cycle values to calibrate the range of motion.

Have your instructor grade your work out of seven available marks—two for the working circuit, five for your software. If your instructor grades your work in class, enter the assigned grade below. Otherwise, upload two things: a short video of the servo running, and the `main.c` file from your software.

Answers

- 20
- 750
- 5