Cerebot 32MX4 ™
Servo Control Reference Design

Revision: October 7, 2009

215 E Main Suite D | Pullman, WA 99163
(509) 334 6306 Voice and Fax

Overview

The Cerebot 32MX4 Servo Control Reference Design was written to provide students a technique to use the eight servo ports located on the Cerebot 32MX4. The technique described here requires the use of only one timer along with an output compare peripheral.

The source code that accompanies this document is written to drive all eight of the servo ports with independent signals for each. However, the source can be modified to support less than eight servos if the user does not need to support all eight of the ports and would like to reduce the amount of processor time devoted to timer interrupts.

The demonstration code requires:

- the Cerebot 32MX4 board
- Microchip's MPLAB Software Development Suite
- A USB A to micro USB cable

While reading this document and the source code, refer to the following additional documents:

- Digilent Cerebot 32MX4 Board Reference Manual. (Doc: 502-173)
- Digilent Cerebot32MX4 Schematic (Doc: 500-173)
- Microchip PIC32MX3XX/4XX Family Data Sheet (DS61143F)
- Microchip PIC32MX Family Reference Manual (DS61132B)
- Microchip Section 8. Interrupts (DS61108D)
- Microchip Section 12. I/O Ports (DS61120D)
- Microchip Section 14. Timers (DS61105D)
- Microchip Section 16. Output Compare (DS61111D)

Interface Variables

The purpose of this reference design is to provide a technique for driving up to eight servos directly from the board, while offering a simple method of defining the desired position of each servo. The source includes functions that set up the processor's peripherals that drive the servos. These peripherals do not require any end-user modification, however there is a function that generates position data for the servos that you will likely want to remove and replace with your own. This function is called UpdateServoPos() and is placed in the Timer2 interrupt service routine. When creating your own routines that modify servo position data, a better idea would be to place it in the main loop. UpdateServoPos() was placed in the ISR for sake of automation.

Variable name	Description
cntPulseRate	Number of clock cycles that occur between servo addresses. This number should be between 20K and 30K. Servo Refresh Rate (Hz) = $1/((cntPulseRate*10^{-7})*numServos)$
cntWdtInit	Initial servo position upon start up.
cntWdtIdle	Optimal center of servo travel
cntWdtMin	Range minimum
cntWdtMax	Range maximum
cntWdtDelta	Demo specific variable: value used to define the amount of movement during each sequential movement statement.
cmsUpdateInterval	Demo specific variable: number of refresh cycles between servo position updates. Update Freq.(Hz) = $1/(cntPulseRate*10^{-7}*$ numServos*cmsUpdateInterval)
cntWdt(n), where n is the servo number	This is the data that should be modified by the end user. This data holds the position value for each servo. cntWdtMin ≤ cntWdt(n) ≤ cntWdtMax

Program Customization

To control the individual servo positions, the user needs to create a function that loads values into the cntWdt(n) variables, keeping in mind the range of maximum and minimum values for those variables.

Feel free to enable serial communication to set the cntWdt(n) values, or use some of the servo channels as inputs using an algorithm to drive the remaining servo channels. An example of this would be a servo mixing application that would time a set of incoming servo pulses to drive another servo.

Functional Description

The RC style servo requires a pulse-width modulated signal. The pulse has to occur at a semi-regular rate around 50 Hz. With experimentation you may find that the frequency of the pulse can have up to a 20Hz variance. This means that the servo may reliably respond to signals between 30 and 70 Hz. A frequency outside this range will cause the servo to operate in an unpredictable manner and may inadvertently damage itself.

The position of the servo is determined by the pulse *width*. The standard range of response for a servo is between 1 ms and 2 ms, with a nominal center of 1.5 ms. This range will typically result in a rotational range of about 90°. As with the pulse frequency, the pulse width can also be extended outside this range through experimentation. We have found that most servos can operate between 0.5 ms to 2.4 ms producing a rotational range of about 180°. When operating outside the standard range, take care to ensure that your servos will respond in a safe reliable manner. Some servos have physical limitations to rotating beyond their designed range, and when presented with a pulse outside this range, it may proceed by destroying itself.

Functions

Reference the source code while reading this document to better understand the functions described here.

DeviceInit():

This function sets up ports as outputs that are tied to ground as well as setting up the system clock. It also sets the ports connected to the LEDs as output and initially keeps them turned off. Explicitly configuring the LEDs is repetitive as it was done by the previous instructions, but this step will ensure that the LEDs will still be set up correctly after future development has been completed. The servo signal pins are set up as outputs with the pins initially tied to ground.

AppInit():

This function initializes the global variables. It also sets up the Timer2 and Output Compare2 peripherals and interrupts.

Remaining Functions:

The StartServoPulse() and EndServoPulse() functions control the logic levels on the servo signal pins and do so in an ordered way. Figure 1 shows that as the interrupt service routine calls these two functions, the servos have their pins addressed in an ordered manner.

The Timer2 interrupt will call the StartServoPulse() function which will bring the logic level of one servo high until the EndServoPulse() function is called by the Output Compare2 interrupt, which then lowers the logic level of the signal and points the next interrupt at the next servo in the chain.

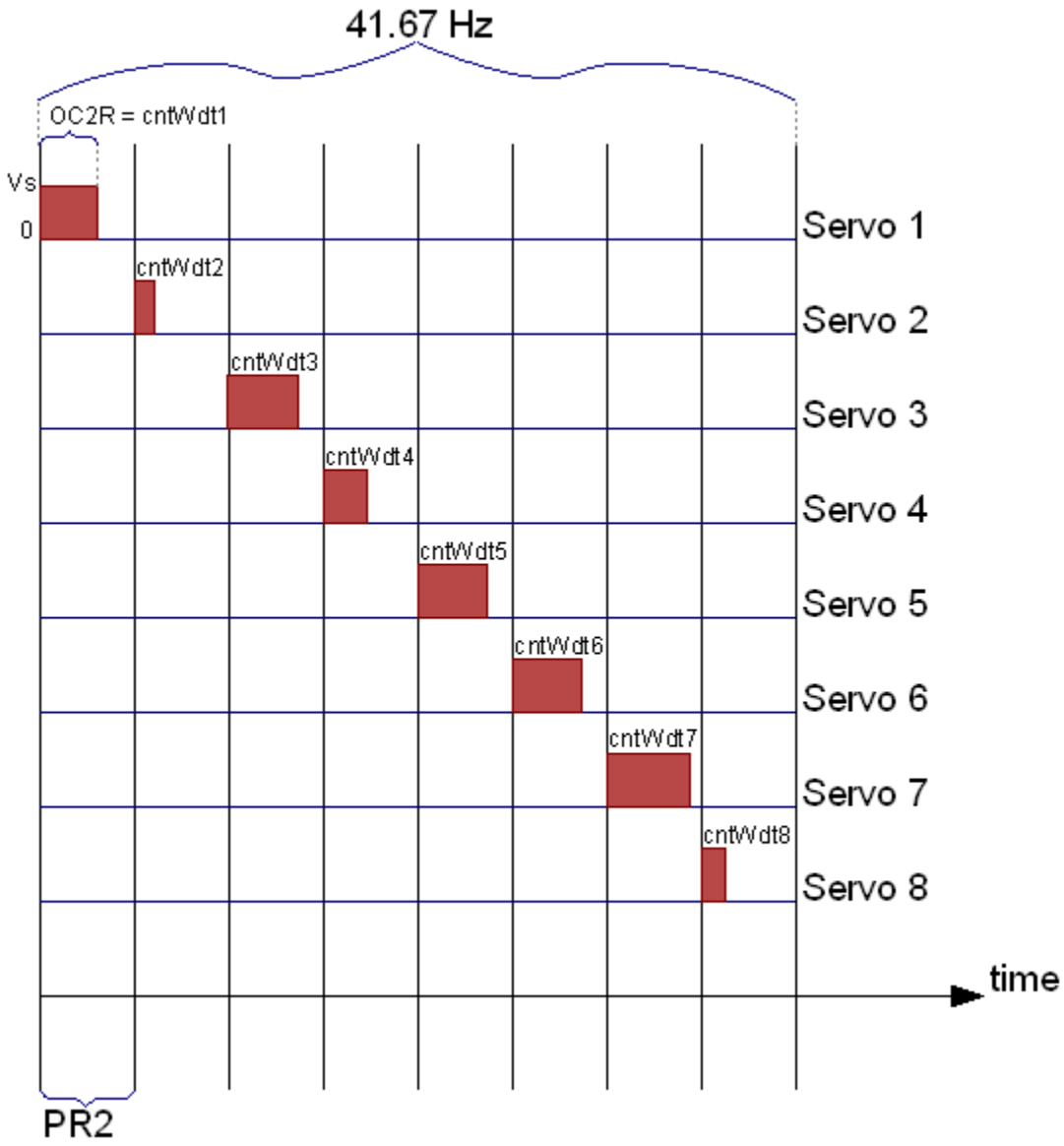

Figure 1 – Servo Timing Illustration

By the time all eight servos have been addressed, it is time to address the first servo again. If you are not using all eight servos, make sure you change cntPulseRate to reflect any changes you have made. If you are removing servos, you will be *increasing* the value stored in cntPulseRate.

In the main loop, the program calls a wait function and blinks LD1 on and off. This gives a visual confirmation that the program is running.