

G4 Servo Motor PWM App Note

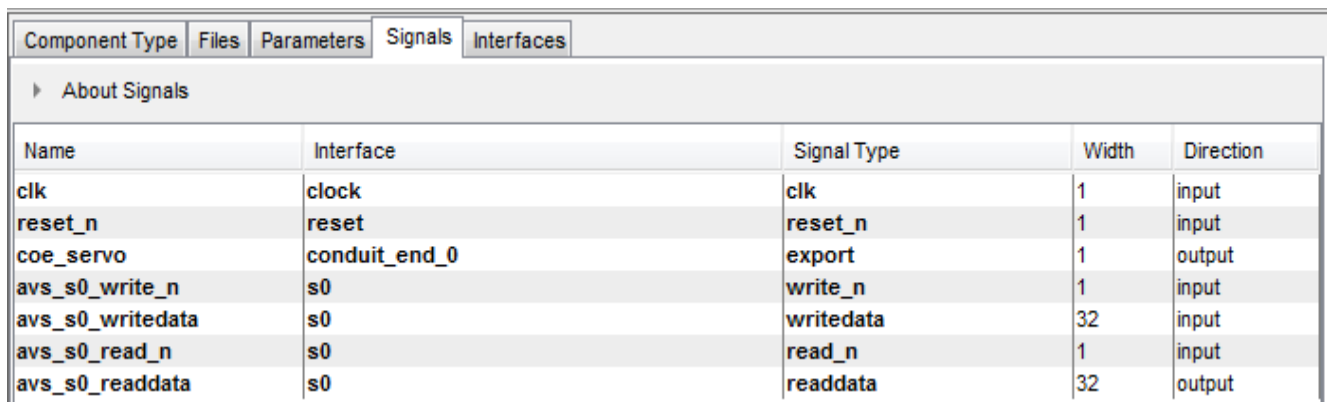
Ryan Corpuz, Hang Peng, Jingjing Liang

Introduction

Servo motors are controlled using a pulse width modulator (PWM). A PWM generates a signal width varying width, and in the case of servos, each pulse is separated by 20 ms. The goal of this application note is to show you how to control a servo motor by directly writing specific pulse widths to the PWM. This driver is optimized for all Hitec Servos which use a 1.5 ms pulse signal for its neutral position. This driver is designed around a 50 MHz clock speed. This means that our servo motor PWM outputs '1' for 75000 clocks. Detail for this will be discussed later in this application note.

Getting Started with this Servo Motor PWM

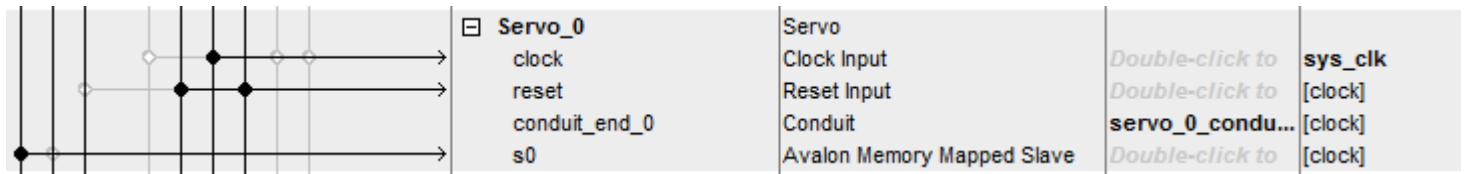
1. Download servo_pwm.vhd and add it to your Quartus II project.
2. Go to Tools>Qsys and load your Qsys file.
3. Create a new component (double-click *New Component* under the Component library tab). In the Name and Display name fields, name your servo PWM. It's best to keep these two names the same.
4. Under the Files tab, under Synthesis Files, add the servo_pwm.vhd file by clicking '+' and navigate to the file.
5. Once added, click on Analyze Synthesis Files. No errors should occur at this point.
6. Under the Signals tab, ensure it resembles the following:



Name	Interface	Signal Type	Width	Direction
clk	clock	clk	1	input
reset_n	reset	reset_n	1	input
coe_servo	conduit_end_0	export	1	output
avs_s0_write_n	s0	write_n	1	input
avs_s0_writedata	s0	writedata	32	input
avs_s0_read_n	s0	read_n	1	input
avs_s0_readdata	s0	readdata	32	output

7. If there are errors such as "Error: s0: Interface must have an associated reset", remedy it by associating "reset" for Associated Reset under "'s0" (Avalon Memory Mapped Slave)" under the Interfaces tab.
8. Now the component by double clicking your new servo PWM component and click finish.
9. For organization, I named my PWM as "Servo_0".
10. Connect your servo PWM by connecting:
 - a. "clock" to your system clock
 - b. "reset" to your system's reset(s)
 - c. "s0" to the data_master

11. For your conduit (In my case, conduit_end_0), double click the export line. The connections should look similar to the following:



12. Don't forget to assign base addresses (Under the System Tab on the top, click "Assign Base Addresses").

13. Now, under the Generation Tab, generate your Qsys system.

14. Once completed, ensure that your new Qsys file is added into your Quartus II project.

15. Next we need to add the PWM to the project's top level file. Add a GPIO pin to output your pulse width signals to the servos. Define a GPIO pin, in this case we will use GPIO pin GPIO_0(0). Should look as follows:

```
GPIO_0 :OUT std_logic_vector (0 downto 0);
```

16. Next, we need to define the servo PWM conduit under. In our case, our conduit is called servo_0_conduit_end_0_export. Therefore the declaration should look as follows:

```
servo_0_conduit_end_0_export :OUT std_logic;
servo_0_conduit_end_0_export => GPIO_0(0),
```

17. Next we must connect our conduit to our GPIO pin. This should look as follows:

18. If all looks well compile your new design. Once that finishes with no errors, load your design to your DE2 board.

Calculating Pulse Counter for Pulse Width Signals for Servo Motors

For the specific use of Hitec Servos, the neutral position is at 1.5 ms pulse width which equates to a pulse counter of 75000 clocks at a clock rate of 50 MHz. The formula for this is as follows:

$$Pulse\ Counter = Clock\ Rate \times Desired\ Pulse\ Width$$

For the case of a desired 1.5 ms pulse width, and running at a 50 MHz clock:

$$Pulse\ Counter = 50\ MHz \times 1.5\ ms \\ = 75000$$

Therefore if we wish to write a 1.5 ms pulse width signal to our servo, we must send 75000 to the PWM.

Ensure that you have reviewed your servo's specifications for its maximum and minimum pulse widths.

*Note: If you are using a different clock rate, you must change the constant values in the servo_pwm.vhd file. Example, the counter for the refresh is set to 1000000, which is 20

ms at 50 MHz. If you are using a clock cycle at 100 MHz, the new refresh counter should be 2000000.

Controlling Your Servo

1. Setup your servo according to the specifications provided by Hitec. Ensure that the signal line is attached to GPIO_0 pin 0.
2. Create a new Hello MicroC/OS-II project with the updated socpinfo file.
3. To write a new pulse width signal to your PWM, simply use:
IOWR_32DIRECT(SERVO_0_BASE, 0, pulse_counter)
 - a. SERVO_0_BASE is the base address for your conduit; check your system.h file to ensure this is the correct address name.
 - b. pulse_counter is the new pulse width to write to your servo.
 - c. Add a long enough delay after this command to ensure the servo does the full rotation
 - d. NOTE: Be sure to include the "io.h" and "system.h" header files.
4. Build and run your project. You should now see your servo moving according to the pulse counter you wrote.

Since each type of Hitec servos are slightly different, it takes a series of trial and error runs to figure out the maximum/minimum pulse widths for the servo as well as finding out what pulse width will move the servo exactly 1 degree of rotation.

A simple test to see if your servos are moving is to add the following in your highest priority task:

```
void task1(void* pdata)
{
    while (1)
    {
        printf("Hello from task1\n");
        IOWR_32DIRECT(SERVO_0_BASE,0,75000);
        OSTimeDlyHMSM(0, 0,2, 0);
        IOWR_32DIRECT(SERVO_0_BASE,0,30000);
        OSTimeDlyHMSM(0, 0, 2, 0);
        IOWR_32DIRECT(SERVO_0_BASE,0,100000);
        OSTimeDlyHMSM(0, 0, 2, 0);
    }
}
```

The provided archived Quartus project is a fully implemented version of the above application note. After compiling this provided project, create a new microC project in Eclipse, and replace your main file with the main.c file provided. Running it will give you servo rotations, or connecting the GPIO_0(0) pin to the oscilloscope, changing pulse widths will be observed.