For a project that I have been working on with the PSoC4 BLE, I required a PWM source with a minimum frequency of 4.4kHz with a resolution of at least 14 bits the higher the better. However with a conventional PWM, with a clock of 24MHz, a 14 bit PWM would give an output frequency of 1.464kHz, which was not enough for my application. Then I remembered having used a high frequency high resolution PWM for an audio DAC application in PSoC1 many years back. This was based on my friend and fellow PSoC1 enthusiast Victor Kremin's (a genius in exploiting PSoC1's flexible analog and digital resources) application note AN2246 PWM Source High Frequency, High Resolution. Why not port this to PSoC4 BLE?

Picture below shows the High Resolution PWM architecture. The concept is to interlace two PWM signals with different duty cycles with a third PWM signal. The third PWM signal could be another conventional PWM, or a PRS PWM. The terminal count output of PWM1 or PWM2 is the clock to the control PWM.
PWM1 and PWM2 have the same period and compare value with one PWM having a compare type Less Than, and the other Less Than or Equal To. This introduces a difference in duty cycle between the two PWMs equal to one clock. The third PWM (or PRS PWM) is used to control a mux to switch the output between these two PWMs. The effective resolution is the sum of resolution of PWM1/PWM2 + the control PWM. For a 14 bit PWM, the resolution of PWM1 and PWM2 would be 8 bits and the control PWM, 6 bits. To set the duty cycle of the output, the most significant 8 bits of the 14 bit compare value is written to PWM1 and PWM2 and the least significant 6 bits is used as compare value for the control PWM.

An important requirement for this setup to work is PWM1 and PWM2 should be started exactly at the same time to have the one clock phase shift. This is achieved by tying the enable inputs of both the PWMs together and turning them on together. In PSoC1 this is done using the output of a LUT. Also, the two to one mux is implemented using three LUTs, two with AND functions and one with OR.

Below picture shows the final implementation that uses four digital blocks (only either the PRS8 in DBB00 or PWM in DCB03 need to be used in the final project), four LUTs, and three global interconnects.

And here is the PSoC4 BLE implementation of a 16 bit High Resolution PWM with a 12MHz input clock and an output frequency of 46.875kHz. I have implemented a conventional PWM for the control PWM, as a PRS PWM in PSoC4 could not use the output of another block as clock.
Below picture shows the PWM1 configuration windows.

PWM1 is configured to have two 8 bit outputs, one channel with compare type Less and the other channel with compare type Less or Equal. PWM mode is set to Hardware Select. This implements a multiplexer between the two PWM channels selected by a hardware signal. This is the cmp_sel signal input in PWM1 which is driven by PWM2. While it was necessary to simultaneously start the two PWM channels in PSoC1, in PSoC4 BLE, as the PWM is one block simultaneous starting is automatically taken care of.

PWM2 is configured for single output, with a period of 255. Below picture shows the configuration windows for PWM2.

The code for the high resolution PWM is straightforward:

```c
void HighResPWM_Start(void)
{
```
The HighResPWM_Start function starts both the PWMs, and the HighResPWM_Stop function stops them.

```c
void HighResPWM_Stop(void) {
    /* Stop PWM1 and PWM2 */
    PWM1_Stop();
    PWM2_Stop();
}
```

The HighResPWM_Start function starts both the PWMs, and the HighResPWM_Stop function stops them.

```c
void HighResPWM_WritePulseWidth(uint16 PulseWidth) {
    /* The most significant 8 bits become compare value for the two PWM channels in PWM1 */
    PWM1_WriteCompare1(HI8(PulseWidth));
    PWM1_WriteCompare2(HI8(PulseWidth));

    /* The least significant 8 bits become compare value for the control PWM */
    PWM2_WriteCompare(LO8(PulseWidth));
}
```

The HighResPWM_WritePulseWidth function writes the most significant 8 bits of the compare value to the two channels of PWM1 and the least significant 8 bits of the compare value to the control PWM. The duty cycle of the HighResPWM is given by:

\[
\text{Duty Cycle \%} = \left( \frac{\text{Compare Value}}{65536} \right) \times 100.
\]

Following pictures show the output of the HighResPWM at 10% and 50% duty cycles.

![HighResPWM Output at 10% Duty Cycle](image1.png)

![HighResPWM Output at 50% Duty Cycle](image2.png)
So, I started with a minimum requirement of a 14 bit PWM with 4.4kHz, and ended up with a 16 bit PWM with 10x the frequency. Following is the digital resource usage of the HighResPWM.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Used</th>
<th>Free</th>
<th>Max</th>
<th>% Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital clock dividers</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>25.00%</td>
</tr>
<tr>
<td>Pins</td>
<td>5</td>
<td>33</td>
<td>38</td>
<td>13.16%</td>
</tr>
<tr>
<td>USB Macrocells</td>
<td>5</td>
<td>27</td>
<td>32</td>
<td>15.63%</td>
</tr>
<tr>
<td>USB Unique Pterms</td>
<td>7</td>
<td>57</td>
<td>64</td>
<td>10.94%</td>
</tr>
<tr>
<td>USB Total Pterms</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USB Dapath Cells</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>50.00%</td>
</tr>
<tr>
<td>USB Status Cells</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>0.00%</td>
</tr>
<tr>
<td>USB Control Cells</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>50.00%</td>
</tr>
</tbody>
</table>

The project file may be found below.
HighFrequencyPWM.cywrk.Archive.zip

Blog:
The PSoC Hacker Blog

Related Categories:
PSoC® 4 BLE (Bluetooth Smart)

Comments

new
Submitted by Tac on Fri, 10/13/2017 - 07:18

Thank you for this informative article. I have a Psoc 5LP and i adapted your code in order to get 24bit Pwm at a maximum frequency of 331Khz clocking the 5LP at 80Mhz. Looks a very good signal on my scope.
Thanks again.

Add new comment

Your name odissy1

Comment *