Example Objective
This example shows how to generate a sine wave of 60 Hz using an 8-bit DAC, 16-bit counter as time base, and 64 point lookup table (LUT) in PSoC® 1.

Overview
This example demonstrates a simple method of generating a sine wave of 60Hz in PSoC 1 using a 64 point look up table (LUT), a DAC, and a time base. The output frequency can be varied by changing the output frequency of the time base and the number of samples per cycle.

User Module List and Placement
The following table lists user modules in the project and the hardware resources used by each user module.

<table>
<thead>
<tr>
<th>User Module</th>
<th>Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter16_1</td>
<td>DBB00 and DBB01</td>
</tr>
<tr>
<td>DAC8_1</td>
<td>ASC10 and ASD20</td>
</tr>
</tbody>
</table>

User Module Parameter Settings
The following tables show the user module parameter settings for each user module in the project.

<table>
<thead>
<tr>
<th>User Module Name: Counter16_1</th>
<th>Parameter</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clock</td>
<td>VC1</td>
<td>Not applicable. Refer Note [1]</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>High</td>
<td>Enables the counter.</td>
</tr>
<tr>
<td></td>
<td>Period</td>
<td>6249</td>
<td>A period of 6249 results in a divider of 6250. The output frequency from the counter is 24 MHz / 6250 = 3.84 kHz.</td>
</tr>
<tr>
<td></td>
<td>CompareValue</td>
<td>3125</td>
<td>Sets the duty cycle to 50%.</td>
</tr>
<tr>
<td></td>
<td>CompareType</td>
<td>Less Than Or Equal</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>InterruptType</td>
<td>Terminal Count</td>
<td>Generates an interrupt on terminal count.</td>
</tr>
<tr>
<td></td>
<td>ClockSync</td>
<td>Use SysClk Direct</td>
<td>24 MHz clock is fed to the Counter UM. This setting overrides the Clock parameter.</td>
</tr>
</tbody>
</table>

Note [1]: When the ClockSync is set to “Use SysClk Direct”, the clock input to the counter is set to SysClk irrespective of the Clock parameter.
User Module Name : DAC8_1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalogBus</td>
<td>AnalogOutBus_0</td>
<td>Connects the DAC output to P0 [3] via analog output buffers.</td>
</tr>
<tr>
<td>ClockPhase</td>
<td>Normal</td>
<td>Auto-zero cycle occurs during Phase1 and output of DAC is valid in Phase 2.</td>
</tr>
<tr>
<td>DataFormat</td>
<td>OffsetBinary</td>
<td>Offset-binary values are positive numbers, with the lowest output voltage represented by zero and the highest by 254.</td>
</tr>
</tbody>
</table>

**Note:** The most important parameter for a DAC is the column clock. The column clock should be set within the value specified in the user module data sheet. The maximum column clock for the DAC is different for different power levels. Refer the user module data sheet and configure the column clock according to the desired operating power.

### Global Resources

<table>
<thead>
<tr>
<th>Important Global Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Supply Voltage</td>
</tr>
<tr>
<td>SysClk Source</td>
</tr>
<tr>
<td>CPU_Clock</td>
</tr>
<tr>
<td>VC1=SysClk/N</td>
</tr>
<tr>
<td>Analog Power</td>
</tr>
<tr>
<td>Ref Mux</td>
</tr>
</tbody>
</table>

**Note:** The table lists the global resources that are specific to the project. Other parameters are left at their default value or configured as required.

### Pin Configuration

<table>
<thead>
<tr>
<th>Pin</th>
<th>Select</th>
<th>Drive</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0[3]</td>
<td>AnalogOutBuf_0</td>
<td>High Z Analog</td>
</tr>
</tbody>
</table>

### Hardware Connections

This project does not require any external components. When the PSoC device is programmed and powered, the output signal is observed on P0 [3] using an oscilloscope.

### Operation

For a sampled system, the sample rate required to produce the desired output frequency is given by formula:

\[
\text{Sample Rate} = \text{Output Frequency} \times \text{No. Of Samples}
\]

This example generates a 60 Hz sine wave using 64 samples. For an output frequency of 60 Hz and 64 samples, the desired sample rate is 3.84 ksp. Using the above formula, any desired output frequency may be generated by varying the sample rate or the number of samples.

A 64 point LUT is created which resembles a sine wave. The formula to create the samples of the LUT is:

\[
\text{Value}_n = \left( \sin \left[ n \times \frac{360}{64} \right] \times \text{Full Scale} \right) + \text{Zero}
\]

Where,

- \( n = \) Sample number (0 to 63)
- Full Scale = ± Full scale count for the DAC, which is 127 for DAC8
- Zero = Value at which DAC produces 0V, which is 127 for DAC8
Using the formula given earlier, the following LUT is created and stored as a ROM array in `main.c`.

```c
const char SineTable64[] = {
127, 139, 152, 164, 176, 187, 198, 208, 217, 225, 233, 244, 249, 252, 253, 254, 253, 252, 249, 244, 239, 233, 225, 217, 208, 198, 187, 176, 164, 152, 139, 127, 115, 102, 90, 78, 67, 56, 46, 37, 29, 21, 15, 10, 5, 2, 1, 0, 1, 2, 5, 10, 15, 21, 29, 37, 46, 56, 67, 78, 90, 102, 115
};
```

The maximum output frequency that can be achieved using the 8 bit DAC can be calculated as:

\[
\text{Maximum Output Frequency} = \frac{\text{Maximum Sample rate}}{\text{No. Of Samples}}
\]

Maximum Sample rate for an 8-bit DAC is 125 Kbps.

Maximum No. of samples = 256.

Maximum Output Frequency that can be achieved will be 488 Hz.

**Firmware**

On reset, all hardware settings from the device configuration are loaded into the device and `main.c` is executed. The following operations are performed in `main.c`:

- Counter16 is started
- Counter16 interrupt is enabled
- DAC8 is started at high power
- Global interrupts are enabled. Any interrupt in the code will be serviced only if the Global Interrupt is enabled.
- The pointer to the LUT is initialized to 0
- An infinite loop is entered. Henceforth, all the operations take place inside the Counter's ISR.

Reading from the LUT and writing to the DAC8 take place inside the Counter's ISR. The function Counter_ISR in `main.c` performs the operations. This function is declared as an interrupt handler by using the following code.

```c
#pragma interrupt_handler Counter_ISR;
```

The #pragma interrupt_handler is use to make a “C” function as an ISR function.

The ISR will be:

```c
void Counter_ISR(void)
{
    // Update the DAC with the value in lookup table pointed the variable Pointer
    DAC8_1_WriteBlind(SineTable64[Pointer]);

    // Increment pointer
    Pointer++;

    // If the Pointer is incremented by 2, the effective number of samples will be
    // 32 and the output frequency will be doubled. If Pointer is incremented by 4,
    // number of samples will be 16 and the output frequency will be quadrupled.

    // Reset Pointer if greater than or equal to 64
    if (Pointer >= 64) Pointer = 0;
}
```

To execute the function on interrupt, a jump to this function must be done inside the ISR. This is done by placing the following code inside the `_Counter16_1_ISR` inside the `Counter16_1.INT.asm` file.
When the counter interrupt occurs, the boot.asm redirects the interrupt to the _Counter16_1_ISR. From here the control is transferred to the interrupt handler in main.c. The following operations take place inside the ISR.

- The value from the LUT that corresponds to the pointer is read
- The DAC is updated with this value
- The Pointer is incremented
- If Pointer = 64, Pointer is reset to 0, so that the next cycle starts from the first sample in the LUT.
Some Design Considerations

- The same 64 point LUT can be used for smaller tables such as 32 samples, 16 samples, and 8 samples. For 32 samples, inside the ISR, increment the Pointer by 2 instead of 1. For 16 samples, increment the Pointer by 4.

- Higher the number of samples, smoother the output wave and lesser the harmonics. But as the desired output frequency becomes higher, the sample rate also becomes high. When the time taken to execute the Counter's ISR becomes greater than the sample time, CPU load becomes 100%. At this point, the number of samples must be reduced to further increase the output frequency.

- To smoothen the output signal when sample’s per cycle is less add an RC Filter with corner frequency (fc) above fundamental frequency and below the sampling frequency.

  Example: Fundamental Frequency = 60 Hz, Sampling Freq = 120 Hz. Let us consider a corner frequency = 80 Hz.

  Then we can design an RC LPF with R = 2 KΩ and C = 1 uf.

Upgrade Information

When the project is opened with a later version of PSoC Designer™, you receive a project upgrade notification. When the project is upgraded, the boot.tpl in the project is moved to the backup folder and a new boot.tpl is placed. Also, depending on the upgrade, some of the user module library code may be replaced. To make sure that the project works after the upgrade, open the Counter16_1INT.asm file and check that the ljmp _Counter_ISR instruction is present in the _Counter16_1_ISR function inside the user code marker area. If this instruction is not present, manually add the instruction. Save and compile the project.