Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI's products or services with statements different from or beyond the parameters stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:
Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright ©2011, Texas Instruments Incorporated
Trademarks
TMS320 is the trademark of Texas Instruments Incorporated.
All other trademark mentioned herein are property of their respective companies

Acronyms
xDAIS : eXpress DSP Algorithm Interface Standard
IALG : Algorithm interface defines a framework independent interface for the creation of
algorithm instance objects
STB : Software Test Bench
QMATH: Fixed Point Mathematical computation
CcA : C-Callable Assembly
FIR : Finite Impulse Response Filter
IIR : Infinite Impulse Response Filter
FFT : Fast Fourier Transform
Contents

1. Introduction .................................................................................................................. 1

2. SIN Generator .............................................................................................................. 1
   2.1. Standard THD Sin generator .................................................................................... 1
   2.2. Low THD sin generator ......................................................................................... 1
   2.3. High precision sin generator .................................................................................. 1

3. Signal Generator Modules ............................................................................................. 5
   3.1. SGENT_1: Single Channel SIN Generator (Table look-up) .................................. 5
   3.2. SGENT_2: Dual Channel SIN Generator (Table look-up) .................................... 10
   3.3. SGENT_3: 3φ SIN Generator (Table look-up) ....................................................... 15
   3.4. SGENT_3D: Dual 3φ SIN Generator (Table look-up) ........................................... 20
   3.5. SGENTI_1: Single Channel SIN Generator (Table look-up and Linear Interpolation)... 26
   3.6. SGENTI_2: Dual Channel SIN Generator (Table look-up and Linear Interpolation)... 31
   3.7. SGENTI_3: 3φ SIN Generator (Table look-up and Linear interpolation) ................. 36
   3.8. SGENTI_3D: Dual 3φ SIN Generator (Table look-up and Linear Interpolation) ....... 41
   3.9. SGENHP_1: High Precision SIN Generator (Table look-up and Linear Interpolation)... 47
   3.10. SGENHP_2: High Precision SIN Generator (Table look-up and Linear Interpolation).. 52
   3.11. RMPGEN: Ramp Generator ................................................................................. 57
   3.12. TZDLGEN: Trapezoidal generator ..................................................................... 62
   3.13. PROFILE: Profile generator ................................................................................. 67

4. Revision History ........................................................................................................... 75
1. Introduction

The signal generator module repository contains SIN generation, ramp generation and trapezoidal generation modules. The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency. This strategy is adopted for all signal generator modules

NOTE:
The source project for the library currently has two build options: STD and STD_FPU32. Both build configurations are implemented using IQ math, however, the STD_FPU32 config ensures that the library can be used in another project that has “fpu32” run time support option turned on. Each build configuration generates its own libray: C28x_SGEN_Lib_fixed.lib and C28x_SGEN_Lib_fpu32.lib respectively. The examples were built using the fpu32 enabled library(C28x_SGEN_Lib_fpu32.lib) but can easily be modified to work with fixed point operations by
a) Turning off fpu32 support from the compiler’s Run Time Support options (project properties)
b) Replacing the fpu32 enabled library i.e. C28x_SGEN_Lib_fpu32.lib with the standard fixed point version, C28x_SGEN_Lib_fixed.lib, in the linker’s File Search Path option

Each example has a javascript file :“AddWatchWindowVars.js” which the user can run from the scripting console to setup the watch window variables and graphs for each project.

2. SIN Generation

The signal generator repository contains comprehensive set of SIN generation modules viz., Single channel, Dual channel, 3φ and dual 3φ SIN generator to cater to various application requirements. Single and Dual channel modules are available in three forms viz., Standard THD version (SGENT_xx, Low THD version (SGENTI_xx) and High precision version (SGENHP_xx). The 3φ and dual 3φ SIN generator are available in two forms viz., Standard THD version (SGENT_xx) and Low THD version (SGENTI_xx).

2.1. Standard THD sin generator (SGENT_1, SGENT_2, SGENT_3& SGENT_3D)
The standard THD sine generators are implemented using direct table look-up technique and it uses 16-bit modulo counter. Although a 16-bit counter is used, the upper byte (8-bits) is used to index the 256-point look-up table and hence to obtain the SIN value. Thus, by changing how quickly values overflow from lower byte (i.e., manipulating step value) the frequency of the sine wave can be changed. Modulo counter ignores the overflow or carry out of 16-bit counter and retains only the remainder. The graph shown in page 2 exemplifies the error of the SIN output obtained using direct table look-up technique with respect to the floating point results.

2.2. Low THD sin generator (SGENTI_1, SGENTI_2, SGENTI_3& SGENTI_3D)
The low THD sin generators are implemented using Table look-up and linear interpolation technique and it uses 16-bit modulo counter. The upper byte (8-bits) is used to index the 256-point look-up table and lower byte (8-bits) used to interpolate between the look-up table entries. The graph shown in page 3 exemplifies the error of the SIN output obtained using table look-up with linear interpolation technique with respect to the floating point results.

2.3. High Precision sin generator (SGENHP_1 & SGENHP_2)

The high precision sin generators are implemented using Table look-up and linear interpolation technique and it uses 32-bit modulo counter. The high precision modules allow precise frequency control because of the fact that it uses 32-bit value for frequency input and also uses 32-bit modulo counter. The upper byte (8-bits) is used to index the 256-point look-up table and the 15-bits following the upper byte are used to interpolate between the look-up table entries. The graph shown in page 4 exemplifies the error of the SIN output obtained with this implementation with respect to the floating-point results.
Standard THD sin generator (SGENT_1, SGENT_2, SGENT_3 & SGENT_3D) error graph

SIN(x) Obtained by C-float (Q15 format)

SIN(x) Obtained by Direct table look-up technique (in Q15 format)

Error with respect to Floating point
Low THD sin generator (SGENTI_1, SGENTI_2, SGENTI_3 & SGENTI_3D) error graph

SIN(x) Obtained by C-float (Q15 format)

SIN(x) Obtained by table look-up with linear interpolation (in Q15 format)

Error with respect to Floating point
High Precision sin generator (SGENHP_1 & SGENHP_2) error graph

SIN(x) Obtained by C-float (Q15 format)

SIN(x) Obtained by table look-up with linear interpolation (in Q15 format)

Error with respect to Floating point
SGENT_1

Description
This module generates single channel digital SIN signal using direct table look-up technique.

Availability
C-Callable Assembly (CcA)

Module Properties
Type: Target Independent, Application Dependent
Target Devices: x28xx
C-Callable Assembly Files: sgt1c.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>16 +257words +</td>
<td>257 Look-up Table entries</td>
</tr>
<tr>
<td></td>
<td>cinit</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENT_1 structure consumes 8 words in the data memory and 11 words in the cinit section
^ Each instance of SGENT_1 module consumes 8 words in Data memory.

©Texas Instruments Inc., September 2011
C/C-Callable ASM Interface

Object Definition

The structure of SGENT_1 object is defined by the following structure definition:

```c
typedef struct {
  unsigned int freq;
  unsigned int step_max;
  unsigned int alpha;
  int gain;
  int offset;
  int out;
  void (*calc)(void *);
} SGENT_1;
```

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between $[0,F_{MAX}]$ normalized to $[0,1]$</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase $[0,2\pi]$</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
</tbody>
</table>
|      | step_max | $F_{MAX} = \frac{step\_max \times F_s}{65536}$.

The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.

Output | out | SIN Output | Q15 | 8000-7FFF |

Special Constants and Data types

**SGENT_1**
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type SGENT_1

**SGENT_1_handle**
User defined Data type of pointer to SGENT_1 Module

**SGENT_1_DEFAULTS**
Structure symbolic constant to Initialize SGENT_1 Module. This provides the initial values to the terminal variables as well as method pointers.

Methods

```c
void (*calc)(void *);
```

This function implements the single channel digital SIN signal generation using direct table look-up technique.
Module Usage

Instantiation
The following example instances empty signal generator object
SGENT_1 sgen;

Initialization
To Instance pre-initialized object
SGENT_1 sgen = SGENT_1_DEFAULTS;

Invoking the computation function
sgen.calc(&sgen);

Example
The following pseudo code exemplifies, 50Hz single channel digital SIN signal generation
using SGENT_1 module.

#include <sgen.h>

SGENT_1 sgen = SGENT_1_DEFAULTS;
int x1;
main()
{
  sgen.offset=0;
  sgen.gain=0x7fff; /* gain=1 in Q15 */
  sgen.freq=5369; /* freq = (Required Freq/ Max Freq)*2^15 */
    /* = (50/305.17)*2^15 = 5369 */
  sgen.step_max=1000; /* Max Freq = (step_max * sampling freq)/65536 */
    /* Max Freq = (1000*20k)/65536 = 305.17 */
  }

void interrupt isr_20khz()
{
  sgen.calc(&sgen);
  x1 = sgen.out;
}

Note: Edit Linker Command file, to place the look-up table in Program memory.

SINTBL > PROG PAGE 0
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The standard THD sine generators are implemented using direct table look-up technique and it uses 16-bit modulo counter. Although a 16-bit counter is used, the upper byte (8-bits) is used to index the 256-point look-up table and hence to obtain the SIN value. Thus, by changing how quickly values overflow from lower byte (i.e., manipulating step value) the frequency of the sine wave can be changed. Modulo counter ignores the overflow or carry out of 16-bit counter and retains only the remainder. The graph shown in page 2 exemplifies the error of the SIN output obtained using direct table look-up technique with respect to the floating point results.

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[ T = \frac{2^{16}}{\text{step}} \times T_{\text{ISR}} = \frac{65536}{\text{step}} \times T_{\text{ISR}} \]  \hspace{1cm} (1)

The frequency of the generated SIN wave is reciprocal of the time, hence

\[ F = \frac{\text{step}}{65536} \times F_{\text{ISR}} \]  \hspace{1cm} (2)

Where \( F_{\text{ISR}} = \frac{1}{T_{\text{ISR}}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the "step" value used to increment the modulo-counter and the ISR execution rate. The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (2)

\[ \text{step} = \frac{500 \times 65536}{20000} = 1638.4 \]
Background Information

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz.

The frequency resolution is $\frac{F_{\text{MAX}}}{\text{step}_\text{max}}$, hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency $f$, initialize the “freq” element of the SIN generator module to $\frac{f}{f_{\text{MAX}}} \times 2^{15}$. Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the SIN generator module.
**SGENT_2**

**Dual Channel SIN Generator (Table look-up)**

**Description**
This module generates dual channel digital SIN signal with phase control using direct table look-up technique.

**Availability**
C-Callable Assembly (CcA)

**Module Properties**
**Type**: Target Independent, Application Dependent

**Target Devices**: x28xx

**C-Callable Assembly files**: sg2c.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>24 words + 257 + cinit*</td>
<td>257 Look-up Table entries</td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENT_2 structure consumes 10 words in the data memory and 13 words in the cinit section

^ Each instance of SGENT_2 module consumes 10 words in Data memory.

©Texas Instruments Inc., September 2011
C/C-Callable ASM Interface

Object Definition
The structure of SGENT_2 object is defined by the following structure definition

typedef struct {
    unsigned int freq;
    unsigned int step_max;
    unsigned int alpha;
    int gain;
    int offset;
    int out1;
    unsigned int phase;
    int out2;
    void (*calc)(void *);
} SGENT_2;

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between ([0,F_{MAX})] normalized to ([0,1])</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase ([0,2\pi])</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>phase</td>
<td>Phase angle between the two SIN outputs ([-\pi,\pi]) is normalized to ([-1,1])</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>(F_{MAX} = \frac{step_max \times F_s}{65536}). The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
</tbody>
</table>

| Output | out1     | SIN Output 1 \(\rightarrow \sin(\theta)\)        | Q15    | 8000-7FFF  |
|        | out2     | SIN Output 2 \(\rightarrow \sin(\theta + phase)\) | Q15    | 8000-7FFF  |

Special Constants and Data types

SGENT_2
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type SGENT_2

SGENT_2_handle
User defined Data type of pointer to SGENT_2 Module

SGENT_2_DEFAULTS
C/C-Callable ASM Interface

Structure symbolic constant to Initialize SGENT_2 Module. This provides the initial values to the terminal variables as well as method pointers.

Methods

void (*calc)(void *);
This function implements the dual channel digital SIN signal generation with phase control using direct table look-up technique.

Module Usage

Instantiation
The following example instances empty signal generator object
SGENT_2 sgen;

Initialization
To Instance pre-initialized object
SGENT_2 sgen = SGENT_2_DEFAULTS;

Invoking the computation function
sgen.calc(&sgen);

Example
The following pseudo code exemplifies, two 50Hz digital SIN signal generation with 90deg phase shift using SGENT_2 module.

#include <sgen.h>
SGENT_2 sgen=SGENT_2_DEFAULTS;
Int x1, x2;
main()
{
  sgen.offset=0;
  sgen.gain=0x7fff;  /* gain = 1 in Q15 */
  sgen.freq=5369;   /* freq = (Required Freq/Max Freq)*2^15 */
  sgen.step_max=1000; /* Max Freq= (step_max * sampling freq)/65536 */
  sgen.phase=4000h  /* Phase = (required Phase)/180 in Q15 */
}

void interrupt isr_20khz()
{
  sgen.calc(&sgen);
  x1=sgen.out1;
  x2=sgen.out2;
}

Note: Edit Linker Command file, to place the look-up table in Program memory.

SINTBL > PROG PAGE 0
The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The standard THD sine generators are implemented using direct table look-up technique and it uses 16-bit modulo counter. Although a 16-bit counter is used, the upper byte (8-bits) is used to index the 256-point look-up table and hence to obtain the SIN value. Thus, by changing how quickly values overflow from lower byte (i.e., manipulating step value) the frequency of the sine wave can be changed. Modulo counter ignores the overflow or carry out of 16-bit counter and retains only the remainder. The graph shown in page 2 exemplifies the error of the SIN output obtained using direct table look-up technique with respect to the floating point results.

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[
T = \frac{2^{16}}{step} \times T_{ISR} = \frac{65536}{step} \times T_{ISR} \tag{1}
\]

The frequency of the generated SIN wave is reciprocal of the time, hence

\[
F = \frac{step}{65536} \times F_{ISR} \tag{2}
\]

Where \( F_{ISR} = \frac{1}{T_{ISR}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate. The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (2)

\[
step = \frac{500 \times 65536}{20000} = 1638.4
\]
Background Information

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz

The frequency resolution is $\frac{F_{\text{MAX}}}{\text{step}_\text{max}}$, hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency $f$, initialize the “freq” element of the SIN generator module to $\frac{f}{f_{\text{MAX}}} \times 2^{15}$. Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the SIN generator module.
SGENT_3

Three Phase SIN Generator (Table look-up)

Description
This module generates 3-phase digital SIN signal with fixed 120° phase shift between the channels using direct table look-up technique.

Availability
C-Callable Assembly (CcA)

Module Properties
Type: Target Independent, Application Dependent

Target Devices: x28xx

C-Callable Assembly files: sgt3c.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>34 words + 257 +</td>
<td>257 Look-up Table entries</td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENT_3 structure consumes 10 words in the data memory and 13 words in the cinit section
^ Each instance of SGENT_3 module consumes 10 words in Data memory.
C/C-Callable ASM Interface

Object Definition

The structure of SGENT_3 object is defined by the following structure definition:

typedef struct {
    unsigned int freq;
    unsigned int step_max;
    unsigned int alpha;
    int gain;
    int offset;
    int out1;
    int out2;
    int out3;
    void (*calc)(void *);
} SGENT_3;

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between (0,F_{MAX})</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>normalized to ([0,1])</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase ([0,2\pi])</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>(F_{MAX} = \frac{step_max \times F_s}{65536}))</td>
<td>Q0</td>
<td>0000-FFFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is set to 1000 to</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>generate the maximum frequency of 305.17Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>using 20KHz sampling loop.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>out1</td>
<td>SIN Output 1 (\rightarrow \sin(\theta))</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out2</td>
<td>SIN Output 2 (\rightarrow \sin(\theta + 120^\circ))</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out3</td>
<td>SIN Output 3 (\rightarrow \sin(\theta + 240^\circ))</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>

Special Constants and Data types

SGENT_3

The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type SGENT_3

SGENT_3_handle

User defined Data type of pointer to SGENT_3 Module

SGENT_3_DEFAULTS
Structure symbolic constant to Initialize SGENT_3 Module. This provides the initial values to the terminal variables as well as method pointers.

**Methods**

```c
void (*calc)(void *);
```

This function implements 3-phase digital SIN signal generation with fixed 120° phase shift between the channels using direct table look-up technique.

**Module Usage**

**Instantiation**
The following example instances empty generic signal generator object

```c
SGENT_3 sgen;
```

**Initialization**
To Instance pre-initialized object

```c
SGENT_3 sgen = SGENT_3_DEFAULTS;
```

**Invoking the computation function**

```c
sgen.calc(&sgen);
```

**Example**
The following pseudo code exemplifies, 3-phase digital SIN signal (50Hz) generation using SGENT_3 module.

```c
#include <sgen.h>

SGENT_3 sgen=SGENT_3_DEFAULTS;

Int x1, x2, x3;
main ( )
{
    sgen.offset=0;
    sgen.gain=0x7fff; /* gain = 1 in Q15 */
    sgen.freq=5369; /* freq  = (Required Freq/Max Freq)*2^15 */
    /*  = (50/305.17)*2^15 = 5369 */
    sgen.step_max=1000; /* Max Freq= (step_max * sampling freq)/65536 */
    /* Max Freq = (1000*20k)/65536 = 305.17 */
    sgen.phase=4000h /* Phase = (required Phase)/180 in Q15 */
    /* = (+90/180) in Q15 = 4000h */
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x1=sgen.out1;
    x2=sgen.out2;
    x3=sgen.out3;
}
```

**Note:** Edit Linker Command file, to place the look-up table in Program memory.

| SINTBL | > PROG PAGE 0 |

©Texas Instruments Inc., September 2011
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The standard THD sine generators are implemented using direct table look-up technique and it uses 16-bit modulo counter. Although a 16-bit counter is used, the upper byte (8-bits) is used to index the 256-point look-up table and hence to obtain the SIN value. Thus, by changing how quickly values overflow from lower byte (i.e., manipulating step value) the frequency of the sine wave can be changed. Modulo counter ignores the overflow or carry out of 16-bit counter and retains only the remainder. The graph shown in page 2 exemplifies the error of the SIN output obtained using direct table look-up technique with respect to the floating point results.

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[
T = \frac{2^{16}}{\text{step}} \times T_{\text{ISR}} = \frac{65536}{\text{step}} \times T_{\text{ISR}}
\]  

(1)

The frequency of the generated SIN wave is reciprocal of the time, hence

\[
F = \frac{\text{step}}{65536} \times F_{\text{ISR}}
\]

(2)

Where \( F_{\text{ISR}} = \frac{1}{T_{\text{ISR}}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate. The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (2)

\[
\text{step} = \frac{500 \times 65536}{20000} = 1638.4
\]
Background Information

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz.

The frequency resolution is \( \frac{F_{\text{MAX}}}{\text{step\_max}} \), hence the “step\_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency \( f \), initialize the “freq” element of the SIN generator module to \( \frac{f}{f_{\text{MAX}}} \times 2^{15} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step\_max” and input as Q15 number to the SIN generator module.
**SGENT_3D**

**Dual, three Phase SIN Generator (Table look-up)**

**Description**
This module generates dual, 3-phase digital SIN signal with fixed 120° phase shift between the channels and phase control between the two three phase signals using direct table look-up technique.

```
phase
freq
gain
offset
step_max

SGENT_3D

out11
out12
out13

out21
out22
out23
```

**Availability**
C-Callable Assembly (CcA)

**Module Properties**
**Type:** Target Independent, Application Dependent

**Target Devices:** x28xx

**C-Callable Assembly files:** sgt3dc.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size¹</td>
<td>63 words + 257 +</td>
<td>257 Look-up Table entries cinit*</td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENT_3D structure consumes 14 words in the data memory and 17 words in the cinit section

³ Each instance of SGENT_3D module consumes 14 words in Data memory.
## C/C-Callable ASM Interface

### Object Definition
The structure of SGENT_3D object is defined by the following structure definition:

```c
typedef struct {
    unsigned int freq;
    unsigned int step_max;
    unsigned int alpha;
    int gain;
    int offset;
    int out11;
    int out12;
    int out13;
    unsigned int phase;
    int out21;
    int out22;
    int out23;
    void (*calc)(void *);
} SGENT_3D;
```

### Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>freq</td>
<td>Frequency in hertz between $[0, F_{\text{max}}]$ normalized to $[0, 1]$.</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase $[0, 2\pi]$</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>$F_{\text{max}} = \frac{\text{step}_\text{max} \times F_s}{65536}$.</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>out11</td>
<td>SIN Output 11 $\rightarrow \sin(\theta)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out12</td>
<td>SIN Output 12 $\rightarrow \sin(\theta + 120^\circ)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out13</td>
<td>SIN Output 13 $\rightarrow \sin(\theta + 240^\circ)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out21</td>
<td>SIN Output 21 $\rightarrow \sin(\theta + \text{phase})$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out22</td>
<td>SIN Output 22 $\rightarrow \sin(\theta + 120^\circ + \text{phase})$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out23</td>
<td>SIN Output 23 $\rightarrow \sin(\theta + 240^\circ + \text{phase})$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>
Special Constants and Data types

**SGENT_3D**
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type `SGENT_3D`.

**SGENT_3D_handle**
User defined Data type of pointer to SGENT_3D Module

**SGENT_3_DEFAULTS**
Structure symbolic constant to Initialize SGENT_3D Module. This provides the initial values to the terminal variables as well as method pointers.

**Methods**
```c
void (*calc)(void *);
```
This function implements two, 3-phase digital SIN signal generation with phase control between the three phase signals using direct table look-up technique.

**Module Usage**

**Instantiation**
The following example instances empty generic signal generator object
```c
SGENT_3D sgen;
```

**Initialization**
To Instance pre-initialized object
```c
SGENT_3D sgen = SGENT_3D_DEFAULTS;
```

**Invoking the computation function**
```c
sgen.calc(&sgen);
```
Example
The following pseudo code exemplifies, dual 3-phase digital SIN signal (50Hz) generation with 90deg phase shift between the three phase signals using SGENT_3D module.

```c
#include <sgen.h>

SGENT_3D sgen=SGENT_3D_DEFAULTS;

Int x11, x12, x13, x21, x22, x23;
main ( )
{
    sgen.offset=0; /* gain = 1 in Q15 */
    sgen.gain=0x7fff; /* freq = (Required Freq/Max Freq)*2^15 */
    sgen.freq=5369; /* = (50/305.17)*2^15 = 5369 */
    sgen.step_max=1000; /* Max Freq= (step_max * sampling freq)/65536 */
    sgen.phase=4000h /* Phase = (required Phase)/180 in Q15 */
        /* = (+90/180) in Q15 = 4000h */
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x11=sgen.out11;
    x12=sgen.out12;
    x13=sgen.out13;
    x21=sgen.out21;
    x22=sgen.out22;
    x23=sgen.out23;
}

Note: Edit Linker Command file, to place the look-up table in Program memory.
```

SINTBL > PROG PAGE 0
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The standard THD sine generators are implemented using direct table look-up technique and it uses 16-bit modulo counter. Although a 16-bit counter is used, the upper byte (8-bits) is used to index the 256-point look-up table and hence to obtain the SIN value. Thus, by changing how quickly values overflow from lower byte (i.e., manipulating step value) the frequency of the sine wave can be changed. Modulo counter ignores the overflow or carry out of 16-bit counter and retains only the remainder. The graph shown in page 2 exemplifies the error of the SIN output obtained using direct table look-up technique with respect to the floating point results.

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[ T = \frac{2^{16}}{\text{step}} \times T_{\text{ISR}} = \frac{65536}{\text{step}} \times T_{\text{ISR}} \]  

(1)

The frequency of the generated SIN wave is reciprocal of the time, hence

\[ F = \frac{\text{step}}{65536} \times F_{\text{ISR}} \]  

(2)

Where \( F_{\text{ISR}} = \frac{1}{T_{\text{ISR}}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate. The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (2)

\[ \text{step} = \frac{500 \times 65536}{20000} = 1638.4 \]
Background Information

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz.

The frequency resolution is \( \frac{F_{\text{MAX}}}{\text{step}_\text{max}} \), hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency \( f \), initialize the “freq” element of the SIN generator module to \( \frac{f}{f_{\text{MAX}}} \times 2^{15} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the SIN generator module.
SGENTI_1

Single Channel SIN Generator (Table look-up and interpolation)

Description
This module generates single channel digital SIN signal using table look-up and linear interpolation technique.

Availability
C-Callable Assembly (CcA)

Module Properties
Type: Target Independent, Application Dependent
Target Devices: x28xx
C-Callable Assembly Files: sgti1c.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>26 words + 257 + cinit*</td>
<td>257 Look-up Table entries</td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENTI_1 structure consumes 8 words in the data memory and 11 words in the cinit section
^ Each instance of SGENTI_1 module consumes 8 words in Data memory.
C/C-Callable ASM Interface

Object Definition
The structure of SGENTI_1 object is defined by the following structure definition

```c
typedef struct {
    unsigned int freq;
    unsigned int step_max;
    unsigned int alpha;
    int gain;
    int offset;
    int out;
    void (*calc)(void *);
} SGENTI_1;
```

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between ([0,F_{\text{MAX}}]) normalized to ([0,1])</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase ([0,2\pi])</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>[F_{\text{MAX}} = \frac{\text{step } _ \text{ max} \times F_{\text{s}}}{65536}]. The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
<tr>
<td>Output</td>
<td>out</td>
<td>SIN Output</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>

Special Constants and Data types

**SGENTI_1**
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type SGENTI_1

**SGENTI_1_handle**
User defined Data type of pointer to SGENTI_1 Module

**SGENTI_1_DEFAULTS**
Structure symbolic constant to Initialize SGENTI_1 Module. This provides the initial values to the terminal variables as well as method pointers.

Methods
void (*calc)(void *);
This function implements the single channel digital SIN signal generation using table look-up and linear interpolation technique.
Module Usage

Instantiation
The following example instances empty signal generator object
SGENTI_1 sgen;

Initialization
To Instance pre-initialized object
SGENTI_1 sgen = SGENTI_1_DEFAULTS;

Invoking the computation function
sgen.calc(&sgen);

Example
The following pseudo code exemplifies, 50Hz single channel digital SIN signal generation using SGENTI_1 module.

#include <sgen.h>
SGENTI_1 sgen=SGENTI_1_DEFAULTS;
int x1;
main ()
{
    sgen.offset=0;
    sgen.gain=0x7fff;  /* gain=1 in Q15 */
    sgen.freq=5369;   /* freq = (Required Freq/Max Freq)*2^15 */
    sgen.step_max=1000; /* Max Freq= (step_max * sampling freq)/65536 */
    sgen.freq=5369;   /* freq = (Required Freq/Max Freq)*2^15 */
    sgen.step_max=1000; /* Max Freq= (step_max * sampling freq)/65536 */
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x1=sgen.out;
}

Note: Edit Linker Command file, to place the look-up table in Program memory.

SINTBL > PROG PAGE 0
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The low THD sin generators are implemented using Table look-up and linear interpolation technique and it uses 16-bit modulo counter. The upper byte (8-bits) is used to index the 256-point look-up table and lower byte (8-bits) used to interpolate between the look-up table entries.

\[
y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \times (x - x_1)
\]

(1)

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[
T = 2^{16}_{\text{step}} \times T_{\text{ISR}} = \frac{65536}{\text{step}} \times T_{\text{ISR}}
\]

(2)

The frequency of the generated SIN wave is reciprocal of the time, hence

\[
F = \frac{\text{step}}{65536} \times F_{\text{ISR}}
\]

(3)

Where \( F_{\text{ISR}} = \frac{1}{T_{\text{ISR}}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate.

The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.
Background Information

Assuming that the application requires the maximum frequency of 500Hz using 20KHz ISR loop. Then the step value to generate 500Hz is determined using equation (3)

\[ \text{step} = \frac{500 \times 65536}{20000} = 1638.4 \]

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz.

The frequency resolution is \( \frac{F_{\text{MAX}}}{\text{step}_\text{max}} \), hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency \( f \), initialize the “freq” element of the SIN generator module to \( \frac{f}{f_{\text{MAX}}} \times 2^{15} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the SIN generator module.
**SGENTI_2**  
*Dual Channel SIN Generator (Table look-up and Interpolation)*

**Description**  
This module generates dual channel digital SIN signal with phase control using table look-up and linear interpolation technique.

```
SGENTI_2
```

**Availability**  
C-Callable Assembly (CcA)

**Module Properties**  
**Type:** Target Independent, Application Dependent

**Target Devices:** x28xx

**C-Callable Assembly files:** sgti2c.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>45 words + 257</td>
<td>257 Look-up Table entries</td>
</tr>
<tr>
<td></td>
<td>+ cinit*</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENTI_2 structure consumes 10 words in the data memory and 13 words in the *cinit* section

▲ Each instance of SGENTI_2 module consumes 10 words in Data memory.
C/C-Callable ASM Interface

Object Definition
The structure of SGENTI_2 object is defined by the following structure definition:

typedef struct {
    unsigned int freq;
    unsigned int step_max;
    unsigned int alpha;
    int gain;
    int offset;
    int out1;
    unsigned int phase;
    int out2;
    void (*calc)(void *);
} SGENTI_2;

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between $[0, F_{MAX}]$ normalized to $[0, 1]$</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase $[0, 2\pi]$</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>phase</td>
<td>Phase angle between the two SIN outputs $[-\pi, +\pi]$ is normalized to $[-1, +1]$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>$F_{MAX} = \frac{step_max \times F_s}{65536}$. The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
<tr>
<td>Output</td>
<td>out1</td>
<td>SIN Output 1 $\rightarrow \sin(\theta)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out2</td>
<td>SIN Output 2 $\rightarrow \sin(\theta + phase)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>

Special Constants and Data types

SGENTI_2
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type SGENTI_2

SGENTI_2_handle
User defined Data type of pointer to SGENTI_2 Module

SGENTI_2_DEFAULTS
C/C-Callable ASM Interface

Structure symbolic constant to Initialize SGENTI_2 Module. This provides the initial values to the terminal variables as well as method pointers.

Methods

void (*calc)(void *);
This function implements the dual channel digital SIN signal generation with phase control using table look-up and linear interpolation technique.

Module Usage

Instantiation
The following example instances empty signal generator object
SGENTI_2 sgen;

Initialization
To Instance pre-initialized object
SGENTI_2 sgen = SGENTI_2_DEFAULTS;

Invoking the computation function
sgen.calc(&sgen);

Example
The following pseudo code exemplifies, two 50Hz digital SIN signal generation with 90deg phase shift using SGENTI_2 module.

#include <sgen.h>
SGENTI_2 sgen = SGENTI_2_DEFAULTS;

Int x1, x2;
main ( )
{
    sgen.offset=0;
    sgen.gain=0x7fff; /* gain = 1 in Q15 */
    sgen.freq=5369; /* freq = (Required Freq/Max Freq)*2^15 */
    /* freq = (50/305.17)*2^15 = 5369 */
    sgen.step_max=1000; /* Max Freq = (step_max * sampling freq)/65536 */
    /* Max Freq = (1000*20k)/65536 = 305.17 */
    sgen.phase=4000h; /* Phase = (required Phase)/180 in Q15 */
    /* Phase = (+90/180) in Q15 = 4000h */
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x1=sgen.out1;
    x2=sgen.out2;
}

Note: Edit Linker Command file, to place the look-up table in Program memory.

SINTBL > PROG PAGE 0
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The low THD sin generators are implemented using Table look-up and linear interpolation technique and it uses 16-bit modulo counter. The upper byte (8-bits) is used to index the 256-point look-up table and lower byte (8-bits) used to interpolate between the look-up table entries.

\[
y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \times (x - x_1)
\]

(1)

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[
T = \frac{2^{16}}{step} \times T_{ISR} = \frac{65536}{step} \times T_{ISR}
\]

(2)

The frequency of the generated SIN wave is reciprocal of the time, hence

\[
F = \frac{step}{65536} \times F_{ISR}
\]

(3)

Where \( F_{ISR} = \frac{1}{T_{ISR}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate.

The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.
Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (3)

\[
step = \frac{500 \times 65536}{20000} = 1638.4
\]

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz.

The frequency resolution is \( \frac{F_{MAX}}{step \_ max} \), hence the “step_max” should be high to get good frequency resolution. It should be set to at-least “100” for reasonable frequency resolution.

To generate SIN signal of frequency \( f \), initialize the “freq” element of the SIN generator module to \( \frac{f}{f_{MAX}} \times 2^{15} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the SIN generator module.
**SGENTI_3**

*Three Phase SIN Generator (Table look-up and interpolation)*

**Description**  
This module generates 3-phase digital SIN signal with fixed 120° phase shift between the channels using table look-up and linear interpolation technique.

![Diagram of SGENTI_3](image)

**Availability**  
C-Callable Assembly (CcA)

**Module Properties**  
**Type:** Target Independent, Application Dependent

**Target Devices:** x28xx

**C-Callable Assembly files:** sgti3c.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>66 words + 257 +</td>
<td>257 Look-up Table entries</td>
</tr>
<tr>
<td></td>
<td>cinit*</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENTI_3 structure consumes 10 words in the data memory and 13 words in the cinit section

^ Each instance of SGENTI_3 module consumes 10 words in Data memory.
C/C-Callable ASM Interface

Object Definition

The structure of SGENTI_3 object is defined by the following structure definition:

```c
typedef struct {
    unsigned int freq;
    unsigned int step_max;
    unsigned int alpha;
    int gain;
    int offset;
    int out1;
    int out2;
    int out3;
    void (*calc)(void *);
} SGENTI_3;
```

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between $0,F_{MAX}$ normalized to $[0,1]$</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase $[0,2\pi]$</td>
<td>Q16</td>
<td>0-FFFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>$F_{MAX} = \frac{step_ max \times F_s}{65536}$. The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
<tr>
<td>Output</td>
<td>out1</td>
<td>SIN Output 1 $\rightarrow \sin(\theta)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out2</td>
<td>SIN Output 2 $\rightarrow \sin(\theta + 120^\circ)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out3</td>
<td>SIN Output 3 $\rightarrow \sin(\theta + 240^\circ)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>

Special Constants and Data types

**SGENTI_3**
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type SGENTI_3

**SGENTI_3_handle**
User defined Data type of pointer to SGENTI_3 Module

**SGENTI_3_DEFAULTS**
Structure symbolic constant to Initialize SGENTI_3 Module. This provides the initial values to the terminal variables as well as method pointers.

**Methods**

```c
void (*calc)(void *);
```

This function implements 3-phase digital SIN signal generation with fixed 120° phase shift between the channels using table look-up and linear interpolation technique.

**Module Usage**

**Instantiation**

The following example instances empty generic signal generator object

```c
SGENTI_3 sgen;
```

**Initialization**

To Instance pre-initialized object

```c
SGENTI_3 sgen = SGENTI_3_DEFAULTS;
```

**Invoking the computation function**

```c
sgen.calc(&sgen);
```

**Example**

The following pseudo code exemplifies, 3-phase digital SIN signal (50Hz) generation using SGENTI_3 module.

```c
#include <sgen.h>

SGENTI_3 sgen=SGENTI_3_DEFAULTS;

int x1, x2, x3;
main ( )
{
    sgen.offset=0;
    sgen.gain=0x7fff; /* gain = 1 in Q15 */
    sgen.freq=5369; /* freq = (Required Freq/Max Freq)*2^15 */
    /* freq = (50/305.17)*2^15 = 5369 */
    sgen.step_max=1000; /* Max Freq= (step_max * sampling freq)/65536 */
    /* Max Freq = (1000*20k)/65536 = 305.17 */
    sgen.phase=4000h /* Phase = (required Phase)/180 in Q15 */
    /* Phase = (+90/180) in Q15 = 4000h */
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x1=sgen.out1;
    x2=sgen.out2;
    x3=sgen.out3;
}
```

**Note:** Edit Linker Command file, to place the look-up table in Program memory.

```
SINTBL > PROG PAGE 0
```
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The low THD sin generators are implemented using Table look-up and linear interpolation technique and it uses 16-bit modulo counter. The upper byte (8-bits) is used to index the 256-point look-up table and lower byte (8-bits) used to interpolate between the look-up table entries.

\[ y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \times (x - x_1) \]  

(1)

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[ T = 2^{16}_{\text{step}} \times T_{\text{ISR}} = \frac{65536}{\text{step}} \times T_{\text{ISR}} \]  

(2)

The frequency of the generated SIN wave is reciprocal of the time, hence

\[ F = \frac{\text{step}}{65536} \times F_{\text{ISR}} \]  

(3)

Where \( F_{\text{ISR}} = \frac{1}{T_{\text{ISR}}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate.

The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.
Background Information

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (3)

\[
\text{step} = \frac{500 \times 65536}{20000} = 1638.4
\]

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz.

The frequency resolution is \( \frac{F_{\text{MAX}}}{\text{step} _{\text{max}}} \), hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency \( f \), initialize the “freq” element of the SIN generator module to \( \frac{f}{f_{\text{MAX}}} \times 2^{15} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the SIN generator module.
**SGENTI_3D**

*Dual, three Phase SIN Generator (Table look-up and interpolation)*

**Description**
This module generates dual, 3-phase digital SIN signal with fixed 120° phase shift between the channels and phase control between the two three phase signals using table look-up and linear interpolation technique.

![Diagram](image)

**Availability**
C-Callable Assembly (CcA)

**Module Properties**
**Type:** Target Independent, Application Dependent

**Target Devices:** x28xx

**C-Callable Assembly Files:** sg3dc.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>128 words + 257</td>
<td>257 Look-up Table entries + cinit*</td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENTI_3D structure consumes 14 words in the data memory and 17 words in the cinit section

^ Each instance of SGENTI_3D module consumes 14 words in Data memory.
C/C-Callable ASM Interface

Object Definition

The structure of SGENTI_3D object is defined by the following structure definition:

```c
typedef struct {
    unsigned int freq;
    unsigned int step_max;
    unsigned int alpha;
    int gain;
    int offset;
    int out11;
    int out12;
    int out13;
    unsigned int phase;
    int out21;
    int out22;
    int out23;
    void (*calc)(void *);
} SGENTI_3D;
```

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between $[0,F_{\text{MAX}}]$ normalized to $[0,1]$</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase $[0,2\pi]$</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>$F_{\text{MAX}} = \frac{\text{step}_{\text{max}} \times F_s}{65536}$</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
</tbody>
</table>

The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.

| Output | out11  | SIN Output 11 $\rightarrow \sin(\theta)$ | Q15    | 8000-7FFF   |
|        | out12  | SIN Output 12 $\rightarrow \sin(\theta + 120^\circ)$ | Q15    | 8000-7FFF   |
|        | out13  | SIN Output 13 $\rightarrow \sin(\theta + 240^\circ)$ | Q15    | 8000-7FFF   |
|        | out21  | SIN Output 21 $\rightarrow \sin(\theta + \text{phase})$ | Q15    | 8000-7FFF   |
|        | out22  | SIN Output 22 $\rightarrow \sin(\theta + 120^\circ + \text{phase})$ | Q15    | 8000-7FFF   |
|        | out23  | SIN Output 23 $\rightarrow \sin(\theta + 240^\circ + \text{phase})$ | Q15    | 8000-7FFF   |
Special Constants and Data types

**SGENTI_3D**
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type **SGENTI_3D**

**SGENTI_3D_handle**
User defined Data type of pointer to **SGENTI_3D** Module

**SGENTI_3_DEFAULTS**
Structure symbolic constant to Initialize **SGENTI_3D** Module. This provides the initial values to the terminal variables as well as method pointers.

**Methods**
void (*calc)(void *);
This function implements two, 3-phase digital SIN signal generation with phase control between the three phase signals using table look-up and linear interpolation technique.

**Module Usage**

**Instantiation**
The following example instances empty generic signal generator object

```
SGENTI_3D sgen;
```

**Initialization**
To Instance pre-initialized object

```
SGENTI_3D sgen = SGENTI_3D_DEFAULTS;
```

**Invoking the computation function**

```
sgen.calc(&sgen);
```
C/C-Callable ASM Interface

Example
The following pseudo code exemplifies, dual 3-phase digital SIN signal (50Hz) generation with 90deg phase shift between the three phase signals using SGENTI_3D module.

```
#include <sgen.h>

SGENTI_3D sgen=SGENTI_3D_DEFAULTS;

Int x11, x12, x13, x21, x22, x23;
main ( )
{
    sgen.offset=0;
    sgen.gain=0x7fff; /* gain = 1 in Q15 */
    sgen.freq=5369;  /* freq  = (Required Freq /Max Freq)*2^15 */
    /*          = (50/305.17)*2^15 = 5369 */
    sgen.step_max=1000; /* Max Freq= (step_max  * sampling freq)/65536 */
    /*         = (1000*20k)/65536 = 305.17 */
    sgen.phase=4000h /* Phase = (required Phase)/180 in Q15 */
    /*       = (+90/180) in Q15 = 4000h */
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x11=sgen.out11;
    x12=sgen.out12;
    x13=sgen.out13;
    x21=sgen.out21;
    x22=sgen.out22;
    x23=sgen.out23;
}

Note: Edit Linker Command file, to place the look-up table in Program memory.

SINTBL > PROG PAGE 0
```
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The low THD sin generators are implemented using Table look-up and linear interpolation technique and it uses 16-bit modulo counter. The upper byte (8-bits) is used to index the 256-point look-up table and lower byte (8-bits) used to interpolate between the look-up table entries.

\[
y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \times (x - x_1)
\]  

(1)

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[
T = \frac{2^{16}}{step} \times T_{ISR} = \frac{65536}{step} \times T_{ISR}
\]  

(2)

The frequency of the generated SIN wave is reciprocal of the time, hence

\[
F = \frac{step}{65536} \times F_{ISR}
\]  

(3)

Where \( F_{ISR} = \frac{1}{T_{ISR}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate.

The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.
Background Information

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (3)

$$step = \frac{500 \times 65536}{20000} = 1638.4$$

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz.

The frequency resolution is $$\frac{F_{MAX}}{step \_ max}$$, hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency $$f$$, initialize the “freq” element of the SIN generator module to $$\frac{f}{f_{MAX}} \times 2^{15}.$$ Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the SIN generator module.
SGENHP_1

Description
This module generates single channel digital SIN signal using table look-up and linear interpolation technique and it uses 32-bit integration counter for high precision SIN generation.

Availability
C-Callable Assembly (CcA)

Module Properties
Type: Target Independent, Application Dependent

Target Devices: x28xx

C-Callable Assembly files: sghp1c.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size(^1)</td>
<td>34 words + 257 +</td>
<td>257 Look-up Table entries</td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words(^1)</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENHP_1 structure consumes 12 words in the data memory and 15 words in the \textit{cinit} section

\(^1\) Each instance of SGENHP_1 module consumes 12 words in Data memory.
C/C-Callable ASM Interface

Object Definition

The structure of SGENHP_1 object is defined by the following structure definition:

```c
typedef struct {
    void (*calc)(void *);
    unsigned long int freq;
    unsigned long int step_max;
    unsigned long int alpha;
    int gain;
    int offset;
    int out;
} SGENHP_1;
```

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between ([0,F_{MAX}]) normalized to ([0,1])</td>
<td>Q31</td>
<td>0-7FFFFFFFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase ([0,2\pi])</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>(F_{MAX} = \frac{step_max \times F_S}{2^{32}})</td>
<td>Q0</td>
<td>0-7FFFFFFFF</td>
</tr>
</tbody>
</table>

The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.

Output

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>out</td>
<td>SIN Output</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>

Special Constants and Data types

SGENHP_1

The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type SGENHP_1

SGENHP_1_handle

User defined Data type of pointer to SGENHP_1 Module

SGENHP_1_DEFAULTS

Structure symbolic constant to Initialize SGENHP_1 Module. This provides the initial values to the terminal variables as well as method pointers.

Methods

```c
void (*calc)(void *);
```
This function implement the single channel digital SIN signal generation using table look-up and linear interpolation technique and it uses 32-bit integrator to generate high precision SIN signal.

Module Usage

Instantiation
The following example instances empty signal generator object
SGENHP_1 sgen;

Initialization
To Instance pre-initialized object
SGENHP_1 sgen = SGENHP_1_DEFAULTS;

Invoking the computation function
sgen.calc(&sgen);

Example
The following pseudo code exemplifies, 50Hz single channel digital SIN signal generation using SGENHP_1 module.

#include <sgen.h>

SGENHP_1 sgen=SGENHP_1_DEFAULTS;
int x1;
main ( )
{
    sgen.offset=0;
    sgen.gain=0x7fff; /* gain=1 in Q15 */
    sgen.freq=0x14F8CF92; /* freq = (Required Freq/Max Freq)*2^31 */
    sgen.freq = (50/305.17)*2^31 = 0x14f8cf92 */
    sgen.step_max=0x3E7FB26; /* Max Freq= (step_max * sampling freq)/2^32 */
    sgen.step_max=(0x3E7FB26*20k)/2^32 = 305.17 */
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x1=sgen.out;
}

Note: Edit Linker Command file, to place the look-up table in Program memory.

SINTBL > PROG PAGE 0
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The high precision sin generators are implemented using Table look-up and linear interpolation technique and it uses 32-bit modulo counter. The upper byte (8-bits) is used to index the 256-point look-up table and the 15-bits following the upper byte are used to interpolate between the look-up table entries.

\[ y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \times (x - x_1) \]  

\[ T = \frac{2^{32}}{\text{step}} \times T_{\text{ISR}} \]  

The frequency of the generated SIN wave is reciprocal of the time, hence

\[ F = \frac{\text{step}}{2^{32}} \times F_{\text{ISR}} \]

Where \( F_{\text{ISR}} = \frac{1}{T_{\text{ISR}}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate.

The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.
Background Information

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (3)

\[
\text{step} = \frac{500 \times 2^{32}}{20000} = 107374182.4
\]

This step value of 107374182 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution given by equation (4)

The frequency resolution is \(
\frac{F_{\text{MAX}}}{\text{step}_\text{max}}
\) (4)

Hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency \( f \), initialize the “freq” element of the SIN generator module to \( \frac{f}{f_{\text{MAX}}} \times 2^{31} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q31 number to the SIN generator module.

Since the frequency control variable is represented in Q31 format, we can precisely generate the required frequency.
SGENHP\_2

**Description**
This module generates single channel digital SIN signal using table look-up and linear interpolation technique with phase control and it uses 32-bit integration counter for high precision SIN generation.

![SGENHP_2 Diagram](image)

**Availability**
C-Callable Assembly (CcA)

**Module Properties**
Type: Target Independent, Application Dependent

Target Devices: x28xx

C-Callable Assembly files: sghp2c.asm, sintb360.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>57 words + 257 + 257 Look-up Table entries</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized SGENHP\_2 structure consumes 14 words in the data memory and 17 words in the cinit section

* Each instance of SGENHP\_2 module consumes 14 words in Data memory.
C/C-Callable ASM Interface

Object Definition

The structure of SGENHP_2 object is defined by the following structure definition:

```c
typedef struct {
    unsigned long int freq;
    unsigned long int step_max;
    unsigned long int alpha;
    int gain;
    int offset;
    int out1;
    int out2;
    unsigned long int phase;
    void (*calc)(void *);
} SGENHP_2;
```

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between $[0,F_{MAX}]$</td>
<td>Q31</td>
<td>0-7FFFFFFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>normalized to $[0,1]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the SIN signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase $[0,2\pi]$</td>
<td>Q32</td>
<td>0-FFFFFFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the SIN signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>phase</td>
<td>Phase angle between the two SIN outputs $[-\pi,\pi]$ is normalized to $[-1,1]$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>$F_{MAX} = \frac{step_max \times F_s}{2^{32}}$.</td>
<td>Q0</td>
<td>0-7FFFFFFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>out1</td>
<td>SIN Output 1 $\rightarrow \sin(\theta)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>out2</td>
<td>SIN Output 2 $\rightarrow \sin(\theta + phase)$</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>

Special Constants and Data types

**SGENHP_2**

The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type SGENHP_2

**SGENHP_2_handle**

User defined Data type of pointer to SGENHP_2 Module

**SGENHP_2_DEFAULTS**
C/C-Callable ASM Interface

Structure symbolic constant to Initialize SGENHP_2 Module. This provides the initial values to the terminal variables as well as method pointers.

Methods

void (*calc)(void *);
This function implements the dual channel digital SIN signal generation using table look-up and linear interpolation technique with phase control and it uses 32-bit integrator to generate high precision SIN signal.

Module Usage

Instantiation
The following example instances empty signal generator object
SGENHP_2  sgen;

Initialization
To Instance pre-initialized object
SGENHP_2 sgen = SGENHP_2_DEFAULTS;

Invoking the computation function
sgen.calc(&sgen);

Example
The following pseudo code exemplifies, two 50Hz digital SIN signal generation with 90deg phase shift using SGENHP_2 module.

#include <sgen.h>
SGENHP_2 sgen=SGENHP_2_DEFAULTS;

int x1, x2;
main( )
{
    sgen.offset=0;
    sgen.gain=0x7fff; /* gain=1 in Q15 */
    sgen.freq=0x14F8CF92; /* freq = (Required Freq/Max Freq)*2^31 */
        /* = (50/305.17)*2^31 = 0x14f8cf92 */
    sgen.step_max=0x3E7FB26; /* Max Freq= (step_max * sampling freq)/2^32 */
        /* = (0x3E7FB26*20k)/2^32 = 305.17 */
    sgen.phase=0x40000000; /* Phase= (required Phase)/180 in Q31 */
        /* = (+90/180) in Q31 = 40000000h */
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x1=sgen.out1;
    x2=sgen.out2;
}

Note: Edit Linker Command file, to place the look-up table in Program memory.

| SINTBL | > PROG PAGE 0 |

©Texas Instruments Inc., September 2011 55
Background Information

The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The high precision sin generators are implemented using Table look-up and linear interpolation technique and it uses 32-bit modulo counter. The upper byte (8-bits) is used to index the 256-point look-up table and the 15-bits following the upper byte are used to interpolate between the look-up table entries.

\[ y = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \times (x - x_1) \]  

The amount of time it takes for the 32-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[ T = \frac{2^{32}}{step} \times T_{ISR} \]  

The frequency of the generated SIN wave is reciprocal of the time, hence

\[ F = \frac{\text{step}}{2^{32}} \times F_{ISR} \]  

Where \( F_{ISR} = \frac{1}{T_{ISR}} \) is the ISR invocation frequency.

Thus the actual frequency of the SIN wave is determined by the “step” value used to increment the modulo-counter and the ISR execution rate.

The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.
Background Information

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (3)

\[
step = \frac{500 \times 2^{32}}{20000} = 107374182.4
\]

This step value of 107374182 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution given by equation (4)

The frequency resolution is \( F_{\text{MAX}} \frac{step}{step_{\text{max}}} \) (4)

Hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate SIN signal of frequency \( f \), initialize the “freq” element of the SIN generator module to \( \frac{f}{f_{\text{MAX}}} \times 2^{31} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q31 number to the SIN generator module.

Since the frequency control variable is represented in Q31 format, we can precisely generate the required frequency.
Ramp Generator

Description
This module generates ramp output (Positive or Negative ramp) of adjustable gain, frequency and DC offset.

Availability
C-Callable Assembly (CcA)

Module Properties
Type: Target Independent, Application Dependent

Target Devices: x28xx

C-Callable Assembly Files: rampgc.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>12 words + cinit*</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized RMPGEN structure consumes 8 words in the data memory and 11 words in the cinit section
▲ Each instance of RMPGEN module consumes 8 words in Data memory.
C/C-Callable ASM Interface

Object Definition
The structure of RMPGEN object is defined by the following structure definition:

```c
typedef struct {
    int freq;
    unsigned int step_max;
    unsigned int alpha;
    int gain;
    int offset;
    int out;
    void (*calc)(void *);
} RMPGEN;
```

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between ([-F_{MAX}, F_{MAX}) normalized to ([-1,1]). The positive frequency input generates ramp up (+ve Ramp) and negative frequency input generates ramp down output (-ve Ramp)</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the ramp signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase ([0,2\pi])</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the ramp signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>(F_{MAX} = \frac{step_max \times F_s}{65536}). The default value is set to 1000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop.</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
<tr>
<td>Output</td>
<td>out</td>
<td>SIN Output</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>

Special Constants and Data types

**RMPGEN**
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type RMPGEN.

**RMPGEN_handle**
User defined Data type of pointer to RMPGEN Module

**SGENTI_1_DEFAULTS**
Structure symbolic constant to Initialize RMPGEN Module. This provides the initial values to the terminal variables as well as method pointers.

**Methods**
- `void (*calc)(void *)`;
  This function generates ramp output with adjustable gain, frequency and DC offset.
Module Usage

Instantiation
The following example instances empty signal generator object
RMPGEN  sgen;

Initialization
To Instance pre-initialized object
RMPGEN sgen = RMPGEN_DEFAULTS;

Invoking the computation function
sgen.calc(&sgen);

Example
The following pseudo code exemplifies, 50Hz negative ramp signal generation using
RMPGEN module.

#include <sgen.h>

RMPGEN sgen=RMPGEN_DEFAULTS;
int x1;
main ( )
{
   sgen.offset=0;
   sgen.gain=0x7fff;  /* gain=1 in Q15  */
   sgen.freq=-5369;  /* freq = (Required Freq/Max Freq)*2^15 */
                   /*  = (50/305.17)*2^15 = 5369 */
                   /*  Negate freq input for -ve ramp  */
   sgen.step_max=1000; /* Max Freq= (step_max * sampling freq)/65536 */
                       /*  Max Freq = (1000*20k)/65536 = 305.17 */

}

void interrupt isr_20khz()
{
   sgen.calc(&sgen);
   x1=sgen.out;
}
The signal generator modules are implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The amount of time it takes for the 16-bit modulo counter to overflow, assuming that the counter is incremented in ISR.

\[ T = \frac{2^{16}}{step} \times T_{ISR} = \frac{65536}{step} \times T_{ISR} \] 

The frequency of the generated ramp signal is reciprocal of the time, hence

\[ F = \frac{step}{65536} \times F_{ISR} \] 

Where \( F_{ISR} = \frac{1}{T_{ISR}} \) is the ISR invocation frequency.

Thus the actual frequency of the ramp is determined by the “step” value used to increment the modulo-counter and the ISR execution rate. The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop. Then the step value to generate 500Hz is determined using equation (2)

\[ step = \frac{500 \times 65536}{20000} = 1638.4 \]

This step value of 1638 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 0.305Hz

The frequency resolution is \( \frac{F_{MAX}}{step_{max}} \), hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.
To generate RAMP signal of frequency \( f \), initialize the “freq” element of the ramp generator module to \( \frac{f}{f_{MAX}} \times 2^{15} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the ramp generator module. The negative frequency input generates negative ramp output.

**Positive Ramp Output:**

![Positive Ramp Output Diagram]

**Negative Ramp Output:**

![Negative Ramp Output Diagram]
This module generates trapezoidal output of adjustable gain, frequency and DC offset. Input pre-scalar is provided to generate very low frequency output.

**Availability**

C-Callable Assembly (CcA)

**Module Properties**

**Type:** Target Independent, Application Dependent

**Target Devices:** x28xx

**C-Callable Assembly Files:** tzdlgc.asm, sgen.h

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>89 words + cinit*</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words*</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized TZDLGEN structure consumes 14 words in the data memory and 17 words in the *cinit* section

^ Each instance of TZDLGEN module consumes 14 words in Data memory.
C/C-Callable ASM Interface

Object Definition
The structure of TZDLGEN object is defined by the following structure definition

typedef struct {
    unsigned int skip_cntr;
    unsigned int prescalar;
    unsigned int freq;
    unsigned int step_max;
    unsigned int task;
    unsigned int alpha;
    int gain;
    int offset;
    int out;
    void (*init)(void *);
    void (*calc)(void *);
} TZDLGEN;

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>freq</td>
<td>Frequency in hertz between ([0,F_{MAX}]) normalized to ([0,1])</td>
<td>Q15</td>
<td>0000-7FFF</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the trapezoidal signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase ([0,2\pi])</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the trapezoidal signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>prescalar</td>
<td>Prescalar for modulo counter, used to generate very low frequency.</td>
<td>Q0</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>(F_{MAX} = \frac{step_max \times F_s}{4 \times 65536 \times \text{prescalar}})</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
</tbody>
</table>

The default value is set to 4000 to generate the maximum frequency of 305.17Hz using 20KHz sampling loop and unity prescalar.

| Output | out | Trapezoidal Output | Q15 | 8000-7FFF |

Special Constants and Data types

TZDLGEN
The module definition is created as a data type. This makes it convenient to instance an interface to the signal generator module. To create multiple instances of the module simply declare variables of type TZDLGEN.

TZDLGEN_handle
User defined Data type of pointer to TZDLGEN module

TZDLGEN_DEFAULTS
Structure symbolic constant to Initialize TZDLGEN Module. This provides the initial values to the terminal variables as well as method pointers.

Methods

void (*init)(void *);
This function initializes the trapezoidal module.

void (*calc)(void *);
This function generates trapezoidal output with adjustable gain, frequency and DC offset.

Module Usage

Instantiation
The following example instances empty signal generator object
TZDLGEN sgen;

Initialization
To instance pre-initialized object
TZDLGEN sgen = TZDLGEN_DEFAULTS;

Invoking the computation function
sgen.calc(&sgen);

Example
The following pseudo code exemplifies, 50Hz trapezoidal signal generation, using TZDLGEN module.

#include <sgen.h>

TZDLGEN sgen=TZDLGEN_DEFAULTS;
int x1;
main()
{
    sgen.prescalar=1;
    sgen.freq=5369;  /* freq = (Required Freq/Max Freq)*2^15 */
    /* = (50/305.17)*2^15 = 5369 */
    sgen.step_max=4000;  /* Max Freq= (step_max*Fs)/(4*65536*prescalar) */
    /* Max Freq = (4000*20k)/(4*65536*1) = 305.17 */
    sgen.gain=0x7fff;  /* ~1 in Q15 format */
    sgen.offset=0;
    sgen.init(&tgen);
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x1=sgen.out;
}
Background Information

The trapezoidal module is implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The trapezoidal signal consists of four states viz., MIN, RAMP UP, MAX and RAMP DOWN. The module is initialized to “MIN” state by the initialization routine and state switching is performed during the overflow of the modulo counter. Hence, the modulo counter overflows four times to complete one cycle of trapezoidal output. As a result the frequency of the generated signal is reciprocal of the time it takes for 4 overflows of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The modular counter have software prescalar to reduce the increment rate in order to generate very low frequency trapezoidal signal. The prescalar essentially increases the time to overflow, thereby reducing the frequency. The amount of time it takes for the 16-bit modulo counter to overflow 4 times, for a given prescalar is given by

\[ T = \frac{4 \times 2^{16} \times \text{prescalar}}{\text{step}} \times T_{\text{ISR}} = \frac{4 \times 65536 \times \text{prescalar}}{\text{step}} \times T_{\text{ISR}} \]  

(1)

The frequency of the generated trapezoidal signal is reciprocal of the time, hence

\[ F = \frac{\text{step}}{4 \times 65536 \times \text{prescalar}} \times F_{\text{ISR}} \]  

(2)

Where \( F_{\text{ISR}} = \frac{1}{T_{\text{ISR}}} \) is the ISR invocation frequency.
Background Information

Thus, the actual frequency of the trapezoidal signal is determined by the incremental step value, prescalar and ISR execution rate. The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop and unity prescalar. Then the step value to generate 500Hz is determined using equation (2)

\[
step = \frac{500 \times 4 \times 65536 \times 1}{20000} = 6553.6
\]

This step value of 6553 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 76.3mHz

The frequency resolution is \[\frac{F_{\text{MAX}}}{\text{step_max}}\], hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate trapezoidal signal of frequency \(f\), initialize the “freq” element of the ramp generator module to \[\frac{f}{f_{\text{MAX}}} \times 2^{15}\]. Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the trapezoidal generator module.

**Prescalar:**
From equation (2), the minimum frequency is generated for unity prescalar, if the step increment is “1”.

\[
F_{\text{MIN}} = \frac{1}{4 \times 65536 \times 1} \times F_{\text{ISR}} = \frac{20000}{4 \times 65536} = 76\text{mHz}
\]

Hence, the minimum frequency is 76mHz or 13.1sec time period for 20Khz ISR.

If the pre-scalar value is set to 2, then the minimum frequency is 38mHz or 26.2sec time period for 20K ISR.

\[
F_{\text{MIN}} = \frac{1}{4 \times 65536 \times 2} \times F_{\text{ISR}} = \frac{20000}{4 \times 65536 \times 2} = 38\text{mHz}
\]

Thus, the pre-scalar is essentially used to generate very low frequency signal by increasing the value.
**Description**

This module generates profiling signal of adjustable gain, frequency and DC offset. It can generate continuous or triggered single shot output. Input pre-scalar is provided to generate very low frequency.

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size$^\dagger$</td>
<td>156 words</td>
<td>+</td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words$^*$</td>
<td></td>
</tr>
<tr>
<td>XDAIS ready</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>XDAIS component</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

* Each pre-initialized PROFILE structure consumes 20 words in the data memory and 23 words in the cinit section

$^\dagger$ Each instance of PROFILE module consumes 20 words in Data memory.
C/C-Callable ASM Interface

Object Definition
The structure of PROFILE object is defined by the following structure definition
typedef struct {
    int mode;
    int trig;
    unsigned int skip_cntr;
    unsigned int prescalar;
    unsigned int freq;
    unsigned int step_max;
    int t_rmpup;
    int t_max;
    int t_rmpdn;
    int t_min;
    unsigned int task;
    unsigned int alpha;
    int gain;
    int offset;
    int out;
    void (*init)(void *);
    void (*calc)(void *);
} PROFILE;

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>mode</td>
<td>1: Continuous Signal generation</td>
<td>Binary</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Single Shot operation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trig</td>
<td>Trigger input for single shot operation</td>
<td>Binary</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td>prescalar</td>
<td>Prescalar for modulo counter, used to</td>
<td>Q0</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>generate very low frequency.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>freq</td>
<td>Frequency in hertz between ([0,F_{MAX}])</td>
<td>Q15</td>
<td>0000-7FFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>normalized to ([0,1])</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>step_max</td>
<td>(F_{MAX} = \frac{step_max \times F_S}{4 \times 65536 \times \text{prescalar}}).</td>
<td>Q0</td>
<td>0000-7FFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is set to 4000 to generate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>the maximum frequency of 305.17Hz using</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20KHz sampling loop and unity prescalar.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>DC offset in the trapezoidal signal</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>Initial phase ([0,2\pi])</td>
<td>Q16</td>
<td>0-FFFF</td>
</tr>
<tr>
<td></td>
<td>gain</td>
<td>Gain of the trapezoidal signal</td>
<td>Q15</td>
<td>0-7FFF</td>
</tr>
<tr>
<td></td>
<td>t_min</td>
<td>Ratio of minimum state time with respect to the</td>
<td>Q8</td>
<td>0000-0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time period ((T)) in Q8 format</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t_rmpup</td>
<td>Ratio of ramp up state time with respect to the</td>
<td>Q8</td>
<td>0000-0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time period ((T)) in Q8 format</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t_max</td>
<td>Ratio of max state time with respect to the</td>
<td>Q8</td>
<td>0000-0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time period ((T)) in Q8 format</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t_rmpdn</td>
<td>Ratio of ramp down state time with respect to the</td>
<td>Q8</td>
<td>0000-0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time period ((T)) in Q8 format</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>out</td>
<td>Profile Output</td>
<td>Q15</td>
<td>8000-7FFF</td>
</tr>
</tbody>
</table>
Methods

void (*init)(void *);
This function initializes the profile module.

void (*calc)(void *);
This function generates profile signal of adjustable gain, frequency and DC offset. It provides option for single shot or continuous signal generation.

Module Usage

Initialization

The following example instances empty signal generator object
PROFILE sgen;

Example

The following pseudo code exemplifies 50Hz profile signal generation with the following properties using PROFILE module (Assuming 20KHz sampling frequency).
1) Min State time: 10% of T. 2) Ramp up state time: 20% of T.
3) Max state time: 30% of T. 4) Ramp down state time: 40% of T.

#include <sgen.h>
PROFILE sgen=PROFILE_DEFAULTS;
int x1;
main ()
{
    /* Signal Generator module initialization */
    sgen.mode=1; /* Set Mode bit for Continuous signal Generation */
    sgen.prescalar=1;
    sgen.freq=5369; /* freq = (Required Freq/Max Freq)*2^15 */
    /* = (50/305.17)*2^15 = 5369 */
    sgen.step_max=4000; /* Max Freq = (step_max * sampling freq)/(4*65536) */
    /* Max Freq = (4000*20k)/(4*65536) = 305.17 */
    sgen.gain=0x7fff; /* ~1 in Q15 format */
    sgen.offset=0;
    sgen.t_rmpup=51; /* 20% of T, 0.2 in Q8 */
    sgen.t_max=77; /* 30% of T, 0.3 in Q8 */
    sgen.t_rmpdn=102; /* 40% of T, 0.4 in Q8 */
    sgen.t_min=25; /* 10% of T, 0.1 in Q8 */
    sgen.init(&sgen);
}

void interrupt isr_20khz()
{
    sgen.calc(&sgen);
    x1=sgen.out;
}
Background Information

The profile signal generator is implemented using modulo arithmetic counter (i.e. Any overflow is ignored and only the remainder is kept) to precisely control the frequency. The profile signal consists of four states viz., MIN, RAMP UP, MAX and RAMP DOWN. The module is initialized to “MIN” state by the initialization routine and state switching is performed during the overflow of the modulo counter. Hence, the modulo counter overflows four times to complete one cycle of profile output. As a result the frequency of the generated signal is reciprocal of the time it takes for 4 overflows of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency.

The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

The step value is calculated using the following equation

\[ \text{step} = \frac{\text{freq} \times \text{step } \_ \text{max}}{\text{freq}} \]  \hspace{1cm} (1)

Where, \( \text{freq} = 0 : 1 \) in Q15 format

Adding the same step value continuously to the modulo counter provides equal time period (\( \frac{T}{4} \) or 25% of T) for each state of the profile viz., MIN, RAMP UP, MAX and RAMP DOWN resulting in trapezoidal output as given below. Profile generator requires the traversal time for each state modifiable. This is done by providing 4 parameters describing the MIN state time, RAMP UP state time, MAX state time and RAMP DOWN state time. The time input for each state is specified as the ratio with respect to the Profile time period (T) in Q8 format.
The modular counter have software pre-scalar to reduce the increment rate in order to generate very low frequency trapezoidal signal. The pre-scalar essentially increases the time to overflow, thereby reducing the frequency. The amount of time it takes for the 16-bit modulo counter to overflow 4 times, for a given pre-scalar is given by

\[ T = \frac{4 \times 2^{16} \times \text{prescalar}}{\text{step}} \times T_{\text{ISR}} = \frac{4 \times 65536 \times \text{prescalar}}{\text{step}} \times T_{\text{ISR}} \]  

(2)

The frequency of the generated trapezoidal signal is reciprocal of the time, hence

\[ F = \frac{\text{step}}{4 \times 65536 \times \text{prescalar}} \times F_{\text{ISR}} \]  

(3)

Where \( F_{\text{ISR}} = \frac{1}{T_{\text{ISR}}} \) is the ISR invocation frequency.

Thus, the actual frequency of the trapezoidal signal is determined by the incremental step value, pre-scalar and ISR execution rate. The signal generator modules use the normalized control variable to modulate the frequency instead of directly commanding the step value. The frequency control variable is normalized with respect to the maximum frequency.

Assuming that the application requires the maximum frequency of 500Hz using 20Khz ISR loop and unity pre-scalar. Then the step value to generate 500Hz is determined using equation (3)

\[ \text{step} = \frac{500 \times 4 \times 65536 \times 1}{20000} = 6553.6 \]

This step value of 6553 is used to initialize the “step_max” element of the signal generator module. The normalized control variable “freq” helps to control the frequency from 0 to 500Hz by varying it between 0 to 1 (Q15 format) with the frequency resolution of 76.3mHz

The frequency resolution is \( \frac{F_{\text{MAX}}}{\text{step_max}} \), hence the “step_max” should be high to get good frequency resolution. It should be set to at least “100” for reasonable frequency resolution.

To generate trapezoidal signal of frequency \( f \), initialize the “freq” element of the trapezoidal generator module to \( \frac{f}{f_{\text{MAX}}} \times 2^{15} \). Thus the required frequency is normalized with respect to the maximum frequency as set by “step_max” and input as Q15 number to the trapezoidal generator module.
Background Information

**Pre-scalar**

From equation (2), the minimum frequency is generated for unity pre-scalar, if the step increment is “1”.

\[
F_{\text{MIN}} = \frac{1}{4 \times 65536 \times 1} \times F_{\text{ISR}} = \frac{20000}{4 \times 65536} = 76\text{mHz}
\]

Hence, the minimum frequency is 76mHz or 13.1sec time period for 20Khz ISR.

If the pre-scalar value is set to 2, then the minimum frequency is 38mHz or 26.2sec time period for 20K ISR.

\[
F_{\text{MIN}} = \frac{1}{4 \times 65536 \times 2} \times F_{\text{ISR}} = \frac{20000}{4 \times 65536 \times 2} = 38\text{mHz}
\]

Thus, the pre-scalar is essentially used to generate very low frequency signal by increasing the value.

**Time Specification**

The state traversal time for MIN, RAMP_UP, MAX and RAMP DOWN are input to the module as ratio with respect to the profile time period in Q8 format and it is given below. The sum of the time parameter expressed as ratio with respect to the profile time period, must be unity. Otherwise the generated frequency will be different from the one set by the “freq” input.

\[
t_{\text{min}} = \left| \frac{T_1}{T} \times 2^8 \right| \quad (4)
\]

\[
t_{\text{rmpup}} = \left| \frac{T_2}{T} \times 2^8 \right| \quad (5)
\]

\[
t_{\text{max}} = \left| \frac{T_3}{T} \times 2^8 \right| \quad (6)
\]

\[
t_{\text{rmpdn}} = \left| \frac{T_4}{T} \times 2^8 \right| \quad (7)
\]
To generate the profile given in the last page, initialize the time parameters as given below,

1) \( t_{\text{min}} = \frac{0.2T}{T} \times 2^8 = 51 \)
2) \( t_{\text{rmpup}} = \frac{0.1T}{T} \times 2^8 = 26 \)
3) \( t_{\text{max}} = \frac{0.3T}{T} \times 2^8 = 77 \)
4) \( t_{\text{rmpdn}} = \frac{0.4T}{T} \times 2^8 = 102 \)

The step value in equation (1), if added to modulo counter will provide \( \frac{T}{4} \) or 25% of \( T \) for each state. To get the required traversal time for each state, the step value is scaled up/down before adding to modulo counter. The step value is scaled up, if the state time parameter is less the 0.25 the scaled down if the state time parameter is greater then 0.25.

Actual step value added during ramp up phase is give by the following equation

\[
= step \times 0.25
\]

Similarly for the other state of the profile output, actual step values are calculated based on time parameter and added to modulo counter.

**Single shot & Continuous profile generation:**
The profile generator operates in two modes viz., Single shot mode and Continuous mode. It is selected by using “mode” element. Setting the “mode” element will generate continuous profile output, resetting to “zero” will allow the module to work in single shot mode. In single shot mode, the profile generator generates one cycle of profile, if “trig” element is set to “1”. Trigger element is automatically reset to “zero”, after a cycle of profile output.

![Wave Generation Diagram](image)

**Wave Generation:**
The profile generator can be used to generate ramp and triangular waveform by appropriately initializing the state time parameters

Positive ramp generation:
1) \( t_{\text{rmpup}}=256 \) (1 in Q8)
2) \( t_{\text{max}}=0 \)
3) \( t_{\text{min}}=0 \)
4) \( t_{\text{rmpdn}}=0 \)

Negative ramp generation:
1) \( t_{\text{rmpup}}=0 \)
2) \( t_{\text{max}}=0 \)
3) \( t_{\text{min}}=0 \)
4) \( t_{\text{rmpdn}}=256 \) (1 in Q8)

Triangular wave generation:
1) \( t_{\text{rmpup}}=128 \) (0.5 in Q8)
2) \( t_{\text{max}}=0 \)
3) \( t_{\text{min}}=0 \)
4) \( t_{\text{rmpdn}}=128 \) (0.5 in Q8)
4. Revision History

**V1.00 – Nov 2010**
- First Release

**V1.01 – Sep 2011**
- Minor Revision
- Created two build configurations for source library project
  - STD (fixed point only): C28x_SGEN_Lib_fixed.lib
  - STD_FPU32 (floating point support): C28x_SGEN_Lib_fpu32.lib
- Created single linker command file; linked in to all projects
- Added support scripts to set up watch variables and graphs