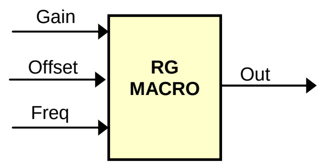
RAMP_GEN	***Ramp Generator***

Description This module generates ramp output of adjustable gain, frequency and dc offset.



Availability This IQ module is available in one interface format:

1) The C interface version

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

Target Devices: 28x Fixed Point or Piccolo

C Version File Names: rampgen.h

IQmath library files for C: IQmathLib.h, IQmath.lib

C Interface

Object Definition

The structure of RAMPGEN object is defined by following structure definition

```
typedef struct { _iq  Freq;                 // Input: Ramp frequency
                 _iq  StepAngleMax;         // Parameter: Maximum step angle
                 _iq  Angle;                // Variable: Step angle
                 _iq  Gain;                 // Input: Ramp gain
                 _iq  Out;                  // Output: Ramp signal
                 _iq  Offset;               // Input: Ramp offset
               } RAMPGEN;

typedef RAMPGEN *RAMPGEN_handle;
```

Item	Name	Description	Format*	Range(Hex)
Inputs	Freq	Ramp frequency	GLOBAL_Q	80000000-7FFFFFFF
	Gain	Ramp gain	GLOBAL_Q	80000000-7FFFFFFF
	Offset	Ramp offset	GLOBAL_Q	80000000-7FFFFFFF
Outputs	Out	Ramp signal	GLOBAL_Q	80000000-7FFFFFFF
RAMPGEN parameter	StepAngleMax	sv_freq_max = fb*T	GLOBAL_Q	80000000-7FFFFFFF
Internal	Angle	Step angle	GLOBAL_Q	80000000-7FFFFFFF

*GLOBAL_Q valued between 1 and 30 is defined in the IQmathLib.h header file.

Special Constants and Data types

RAMPGEN

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

RAMPGEN_handle

User defined Data type of pointer to RAMPGEN module

RAMPGEN_DEFAULTS

Structure symbolic constant to initialize RAMPGEN module. This provides the initial values to the terminal variables as well as method pointers.

Methods

RC_MACRO(RAMPGEN_handle);

This definition implements one method viz., the ramp generator computation macro. The input argument to this macro is the module handle.

Module Usage

Instantiation
The following example instances two RAMPGEN objects
RAMPGEN rg1, rg2;

Initialization
To Instance pre-initialized objects
RAMPGEN rg1 = RAMPGEN_DEFAULTS;
RAMPGEN rg2 = RAMPGEN_DEFAULTS;

Invoking the computation macro
RC_MACRO(rg1);
RC_MACRO(rg2);

Example
The following pseudo code provides the information about the module usage.

```
main()
{

}

void interrupt periodic_interrupt_isr()
{
    rg1.Freq = freq1;           // Pass inputs to rg1
    rg1.Gain = gain1;           // Pass inputs to rg1
    rg1.Offset = offset1;       // Pass inputs to rg1

    rg2.Freq = freq2            // Pass inputs to rg2
    rg2.Gain = gain2;           // Pass inputs to rg2
    rg2.Offset = offset2;       // Pass inputs to rg2

    RC_MACRO(rg1);              // Call compute macro for rg1
    RC_MACRO(rg2);              // Call compute macro for rg1

    out1 = rg1.Out;             // Access the outputs of rg1
    out2 = rg2.Out;             // Access the outputs of rg1
}
```

Technical Background

In this implementation the frequency of the ramp output is controlled by a precision frequency generation algorithm which relies on the modulo nature (i.e. wrap-around) of finite length variables in 28xx. One such variable, called *StepAngleMax* (a data memory location in 28xx) in this implementation, is used as a variable to determine the minimum period (1/frequency) of the ramp signal. Adding a fixed step value to the *Angle* variable causes the value in *Angle* to cycle at a constant rate.

$$Angle = Angle + StepAngleMax \times Freq$$

At the end limit the value in *Angle* simply wraps around and continues at the next modulo value given by the step size.

For a given step size, the frequency of the ramp output (in Hz) is given by:

$$f = \frac{\text{StepAngle} \times \text{fs}}{2^m}$$

where f_s = sampling loop frequency in Hz and m = # bits in the auto wrapper variable *Angle*.

From the above equation it is clear that a *StepAngle* value of 1 gives a frequency of 0.3052Hz when m=16 and f_S=20kHz. This defines the frequency setting resolution of the

For IQmath implementation, the maximum step size in per-unit, *StepAngleMax*, for a given base frequency, fb and a defined GLOBAL_Q number is therefore computed as follows:

$$\text{StepAngleMax} = f_b \times T_s \times 2^{\text{GLOBAL\_Q}}$$

Equivalently, by using _IQ() function for converting from a floating-point number to a _iq number, the *StepAngleMax* can also be computed as

$$\text{StepAngleMax} = \text{\_IQ}(f_b \times T_s)$$

where Ts is the sampling period (sec).