Freescale Semiconductor, Inc.
Application Note

Document Number: AN5046
Rev. 2, 11/2015

Ripple Control Receiver Using Kinetis M

by: Martin Sebest

1 Introduction

Electrical energy consumption is not constant during the day. There is different electrical energy consumption in the morning, in the evening, and at night. There are peaks in electricity consumption during the day which are not desirable, but there are efforts to reduce these peaks in electricity consumption. This is the reason why load management is utilized.

Electric companies motivate electricity consumers to turn their electrical appliances off during peak times, and to turn their electrical appliances on during the drops in electricity consumption. The motivation for doing this is usually a lower tariff for electrical energy during the daily drops in electrical energy consumption. And this is the reason why ripple control is utilized.

Ripple control is a form of load control, and it is already used in many countries around the world. With a ripple control receiver, it is possible to change the tariff for electrical energy, turn the electrical appliance off or on from the grid, and much more.

Contents

Ripple control communication is based on superimposing a higher frequency signal onto the 50 Hz or 60 Hz mains power signal. The amplitude of the superimposed signal is usually around 5 % of the nominal phase voltage, and the frequency of the ripple signal is usually ranging from 150 to 1600 Hz.
The description of the ripple control communication protocol is contained in Section 3, "Ripple control communication protocol." The amplitude of the superimposed signal may vary due to various interference sources, for example, rectifiers, thyristor controls, welders, and so on. The distance between the ripple control signal transmitter and the receiver is another factor which may have an impact on the ripple control signal quality.

The main purpose of this application note is to describe how the ripple control communication is working (communication protocol), and how the ripple control library (rcolib) can be implemented in a ripple control receiver.

2 Block diagram

Figure 1 shows a simple block diagram of the ripple control principle. The source of electrical energy is the power plant. Electrical energy is distributed from the power plant to various kinds of electrical appliances by a wire. The consumers of electrical energy are electrical appliances, which can be situated in different locations around the country. Before each final location (house, company), there is a power meter, which measures the amount of electrical energy consumed. This is the basic principle of the distribution of electrical energy.

Among other things, the modern power meter is able to calculate the price of electrical energy consumed. The price is a simple product of the consumed amount of electrical energy in kWh and the unit tariff for one kWh. If we want to apply different tariffs for electrical energy in different parts of the day, we can use ripple control.

For a ripple control communication to be established, a ripple control transmitter and a ripple control receiver are needed. The role of the ripple control transmitter is to generate the ripple control information and to transmit this information to the grid. On the other hand, the ripple control receiver is needed for receiving the ripple control information from the ripple control transmitter.

As mentioned above, the role of the ripple control transmitter is to generate the ripple control information according to the communication protocol, and to transmit this information to the grid. The ripple control transmitter is suitable to be placed near to the final consumers. How the ripple control transmitter generates and transmits the ripple control message to the grid is not the purpose of this application note. It is mentioned only to clarify that there is more than one way of generating the ripple control message in the transmitter and superimposing it onto the main power signal.

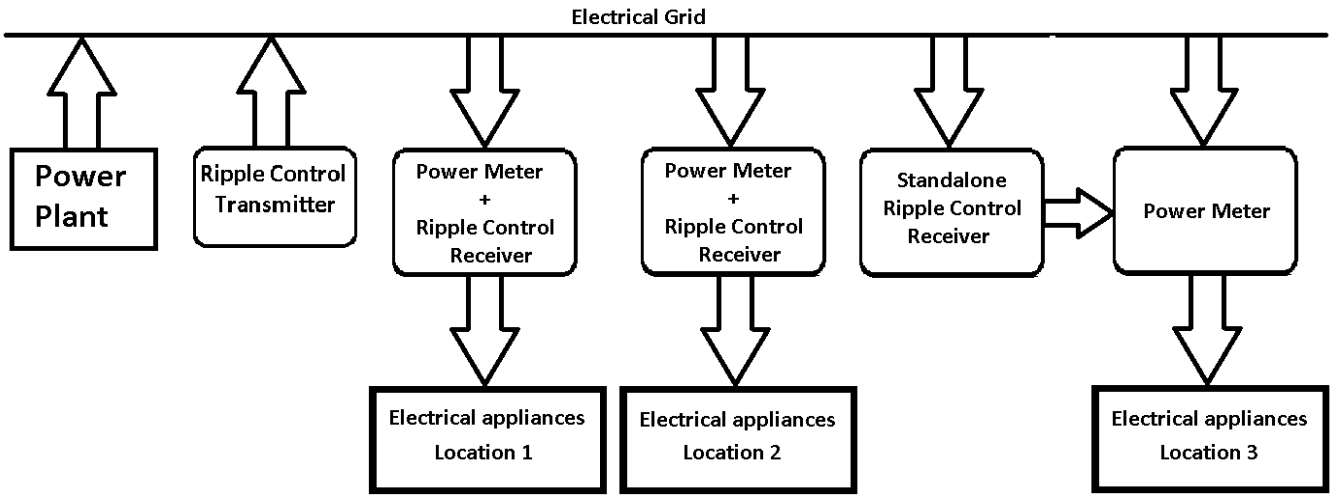

Figure 1. Block diagram of the ripple control principle

The presence of the ripple control receiver is needed in order to receive and apply the ripple control information from the ripple control transmitter. The role of the ripple control receiver is to correctly receive and decode the ripple control information sent from the ripple control transmitter. How the received ripple control message can be applied may also vary. It depends on the specific application, whether we want to only change the tariff for electrical energy, or to disconnect the electrical appliances from the grid, or, whether we want to address power meters only in a specific location. There are many ways in which the ripple control receiver can affect the power meter.

Figure 1 illustrates only two of the many ways in which the ripple control receiver can affect the power meter or electrical appliances connected to the grid. For example, one possibility is that the power meter includes the ripple control receiver function. This is a smart and compact solution – the power meter with an embedded ripple control receiving function assembled in one box. A standalone ripple control receiver is another possibility of how a ripple control receiver can affect one or more power meters or other electrical appliances.

The ripple control communication protocol is described in Section 3, “Ripple control communication protocol.”

3 Ripple control communication protocol

The ripple control communication protocol is described in Figure 2. The communication starts with a start bit. After the start bit, a start pause must follow. The start bit and start pause are followed by the data bits. After each data bit, there must be a data pause. The time lengths of individual parts of the ripple control communication protocol (start bit, start pause, data bit, data pause) may vary. The number of data bits may vary as well. This means that there may be a lot of specific ripple control communication protocols. The most common ripple control communication protocols are described in Table 1.

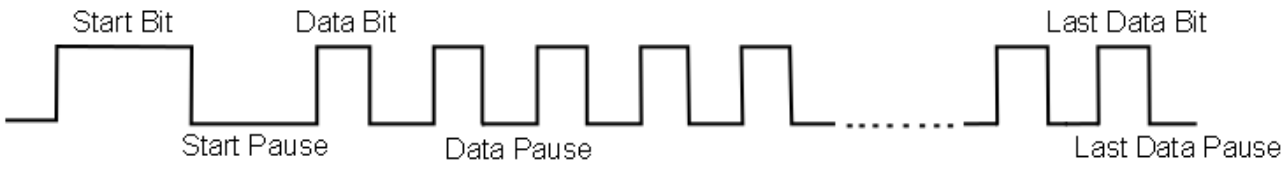

Figure 2. Ripple control communication protocol

The ripple control communication protocol seems to be very similar to the UART communication protocol, but it is not quite the same. A detailed explanation of how the ripple control communication works in a real application is included in Section 3.1, "Detailed explanation of ripple control communication."

3.1 Detailed explanation of ripple control communication

This section describes the ripple control communication in a real application. The waveform of the first harmonic of the power signal is depicted in the first graph of Figure 3. The amplitude of the power signal is 325 V_P (or 230 V_{RSM}), and the frequency of the power signal is 50 Hz.

The ripple control signal is displayed in the second graph of Figure 3. The amplitude of the ripple control signal is 5 % from the first harmonic waveform, and is about 16.2 V_P (or 11.5 V_{RMS}). The frequency of the ripple control signal is, in our case, 216.6 Hz. In a real application, the frequency of a ripple control signal may be different. Which frequency will be used depends on the distributor of electrical energy. Generally, the carrier frequency of the ripple control signal may be in the interval from 110 Hz to 1600 Hz.

As shown in the second graph of Figure 3, the ripple control signal consists of a start bit, start pause, data bit, and a data pause interval. The time duration of individual parts of the ripple control signal is different. For example, the time duration of the start bit is not the same as the time duration of the data bit. Generally, the start bit is two or more times longer than the data bit, as shown in Table 3.

This ripple control signal is superimposed onto the first harmonic of the power signal, as shown in the third graph of Figure 3. The ripple control signal is transmitted to receivers via the power lines. The ripple control signal does not affect the electrical appliances that are connected to the grid. The time duration of the whole ripple control message is less than one minute in most cases.

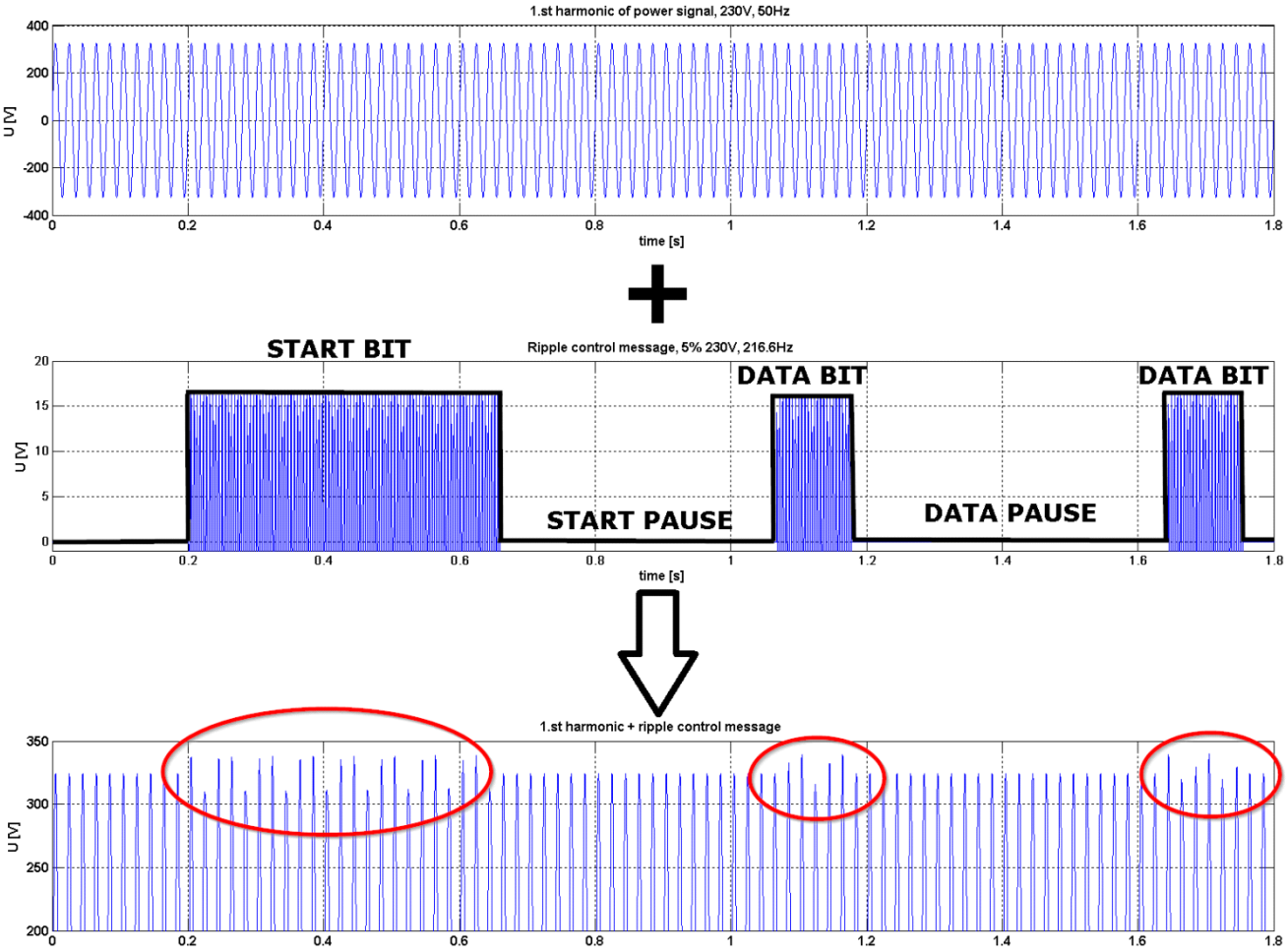

Figure 3. Ripple control communication

3.2 Common ripple control protocols

The most common ripple control communication protocols are described in Table 1.

Table 1. The most common ripple control communication protocols

Manufacturer	Protocol Name	Start Impulse		Data Impulse		Number of bits
		Impulse [ms]	Pause [ms]	Impulse [ms]	Pause [ms]	
ABB	Ricontic b	880	560	320	320	50
	Ricontic s	1600	1360	640	1360	50
Landys & Gyr	Landis & Gyr	460	407	150	427	50
	Semagyr 50	460	407	110	467	50
	Semagyr 50 a	460	387	150	427	50
	Semagyr 50 b	450	695	150	425	50

Table 1. The most common ripple control communication protocols (continued)

Manufacturer	Protocol Name	Start Impulse		Data Impulse		Number of bits
		Impulse [ms]	Pause [ms]	Impulse [ms]	Pause [ms]	
	Semagyr …	460	695	110	467	50
	Semagyr 52	1320	400	320	400	50
	Semagyr 56	2640	800	640	800	50
	RWE	1560	1515	150	427	46
	RWE (mod)	1560	431	150	427	50
Schlumberger	Pulsadis (EdF)	1000	2750	1000	1500	40
	Pulsadis (EdF mod.)	2000	1750	1000	1500	40
	Pulsadis	2000	1050	300	450	31
	Pulsadis / MVM	2000	1000	500	500	50
Siemens	TELENERG 50	1650	600	400	600	50
	TELENERG 29	1650	600	400	600	29
ZPA	ZPA	2330	2990	1000	330	44

More detailed information on the ripple control communication protocols mentioned in Table 1, or about many others, can be found on many web pages; for example: www.rundsteuerung.de/.

4 Ripple control library (Rcolib) – practical implementation

This section describes how the ripple control signal can be received using the ripple control C-code library (C-code).

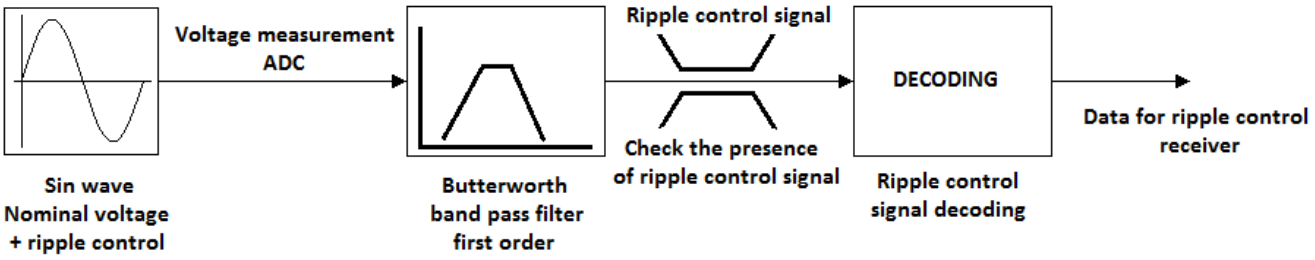

Figure 4. Ripple control signal decoding

A simple block diagram is depicted in Figure 4. It shows how the ripple control message can be decoded from the power-signal waveform, on which the ripple control signal is superimposed. In the beginning, we need to measure the phase voltage where the ripple control message can occur.

The measured samples of the phase voltage are filtered by the band-pass filter. The output from the band-pass filter should be the ripple control signal. When the presence of a ripple control signal is detected, the decoding process can start. The decoding process should provide valid ripple control data, which should be the same as the transmitted ripple control data.

All these steps are included in the rcolib library (ripple control library). This library is described in Section 4.1, "Rcolib library."

4.1 Rcolib library

The rcolib library consists of a couple of functions with a unique application programming interface (API) for receiving the ripple control data from the ripple control signal.

4.1.1 Function API summary

The API functions defined in the rcolib library are described in this section. They are as follows:

- void **RcolibInit** (tRCOLIB_DATA *const ptr)
  Initializes the ripple control function.
- void **RcolibDecode** (register frac16 u, tRCOLIB_DATA *const ptr)
  Decodes the ripple control message from the phase voltage waveform.
- int **RcolibDecodeInfo** (void)
  Monitors the status of the ripple control communication.
- long long **RcolibMsgRead** (void)
  Reads the ripple control message.

4.1.2 RcolibInit

This function initializes all the variables necessary for ripple control data receiving, for example, the filter constants.

4.1.2.1 Syntax

```
#include "rcolib.h"

void     RcolibInit (tRCOLIB_DATA *const ptr);
```

4.1.2.2 Arguments

Table 2. RcolibInit function arguments

Type	Name	Direction	Description
tRCOLIB_DATA	ptr	In	Pointer to ripple control library data structure

4.1.2.3 Returns

This function does not return any arguments.

4.1.2.4 Description

RcolibInit initializes all necessary filter constants and decoding variables for a successful ripple control message reception. The appropriate threshold value depends mainly on the amplitude of the ripple control signal, and also on the carrier frequency of the ripple control signal. It is recommended to set the threshold value very high when using the RcolibInit in an application for the first time. The best threshold value can be set by using the RcolibCalibrate function which is mentioned below.

4.1.3 RcolibDecode

This function decodes the ripple control message from the phase voltage waveform. This function must be called at a constant frequency.

4.1.3.1 Syntax

```
#include "rcolib.h"

void     RcolibDecode (register frac16 u, tRCOLIB_DATA *const ptr);
```

4.1.3.2 Arguments

Table 3. RcolibDecode function arguments

Type	Name	Direction	Description
register	u	in	Instantaneous phase voltage sample in a 16-bit fractional data format
tRCOLIB_ DATA	ptr	In	Pointer to ripple control library data structure

4.1.3.3 Returns

This function does not return any arguments.

4.1.3.4 Description

This function checks for the presence of a ripple control signal and decodes the ripple control signal. The decoding method is briefly explained in Figure 5.

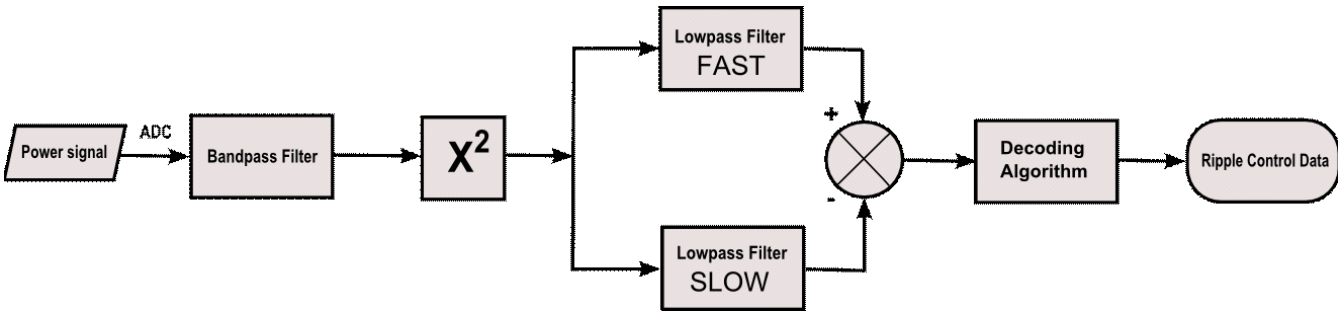

Figure 5. Decoding process with the rcolib

In the beginning, the phase voltage which contains ripple control information must be measured by the ADC converter, and the measured samples are sent through a band-pass filter. The output from the band-pass filter should contain only a ripple control signal generated by the ripple control transmitter. Next, the filtered ripple control signal is “rectified” by being multiplied by itself. This rectified signal is sent through the fast and slow low-pass filter with very different cut-off frequencies. The fast filter has a "quicker" response on input change in comparison to slow filter. It means, there exists difference or error between fast and slow low-pass filter outputs when ripple signal is introduced. According to difference in low pass filters outputs the ripple control message is recognized in decoding algorithm.

4.1.4 RcolibDecodeInfo

This function monitors the status of the ripple control receiver.

4.1.4.1 Syntax

```
#include “rcolib.h”

int     RcolibDecodeInfo (void);
```

4.1.4.2 Arguments

None.

4.1.4.3 Returns

This function returns the actual status of the ripple control decoding.

Table 4. Return options of the RcolibDecodeInfo function

Status	Description
RCOLIB_MSG_READY	Ripple control message is successfully received and ready to be read.
RCOLIB_MSG_PENDING	Ripple control message decoding process is pending.
RCOLIB_MSG_IDLE	Ripple control receiver is idle. Receiver is waiting for start bit.

4.1.4.4 Description

This function is intended to provide the actual status of the ripple control receiver. If the ripple control receiver is idle, the RcolibDecodeInfo returns RCOLIB_MSG_IDLE. The RCOLIB_MSG_IDLE status is valid until the ripple control receiver does not detect a new start bit of a new ripple control message. When the decoding process is in progress, the RcolibDecodeInfo returns RCOLIB_MSG_PENDING. If a ripple control message is successfully received, the status will be RCOLIB_MSG_READY. All statuses can be visualized on the display.

4.1.5 RcolibMsgRead

This function reads the received ripple control message.

4.1.5.1 Syntax

```
#include "rcolib.h"
long long     RcolibMsgRead (void);
```

4.1.5.2 Arguments

None.

4.1.5.3 Returns

This function returns all data bits from the last ripple control message received.

4.1.5.4 Description

This function can be used to read the received ripple control meesage.

4.2 Rcolib library performance

Table 5 shows the MCU computational requirements for using the rcolib library on the ARM® Cortex®-M0+ core (MKM34Z128MCU).

Table 5. Computing performance of the whole rcolib library

Function name	CM0+ core		CM0+ core with MMAU	
	Code size	Clock cycles	Code size	Clock cycles
RcolibInit	154 bytes	1248	154 bytes	1248
RcolibDecode	512 bytes	960	566 bytes	768
RcolibDecodeInfo	24 bytes	40	24 bytes	40
RcolibMsgRead	10 bytes	38	10 bytes	38

4.3 Configuration tool

The configuration tool is intended to easily set up the ripple control algorithm (see Figure 6). The configuration tool is intended to setup filter coefficients according to sampling and ripple control signal frequency and to generate the C-header file that contains the source code with parameters describing the behavior of the ripple control algorithm. The typical usage of the configuration tool is explained in the following section.

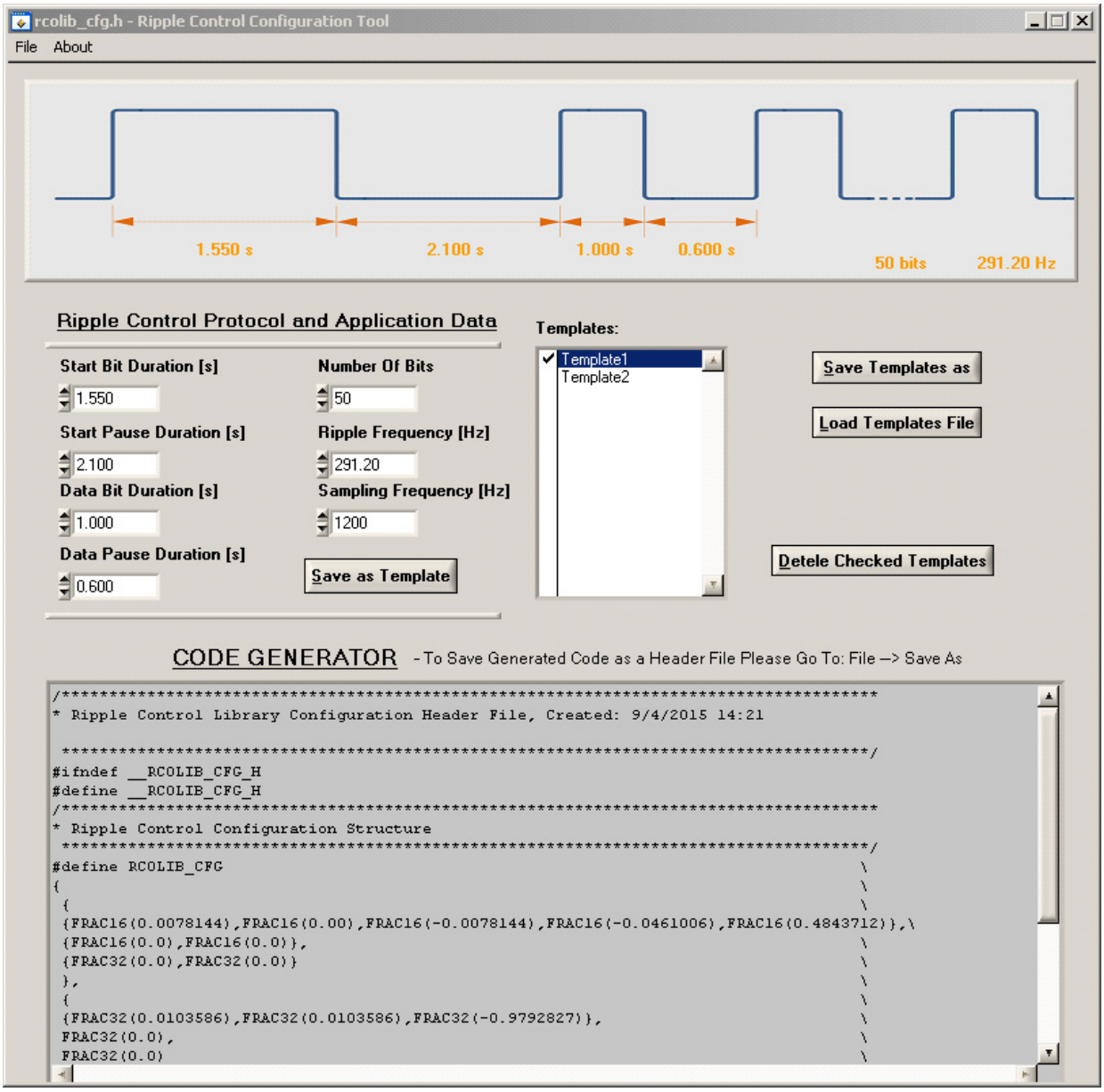

Figure 6. Ripple control library configuration tool

4.3.1 Using the configuration tool

The ripple control configuration tool contains seven important parameters that need to be set according to the requirements of the final application. The following six parameters are linked to the ripple control communication protocol:

- Start bit duration in seconds
- Start pause duration in seconds
- Data bit duration in seconds
- Data pause duration in seconds
- Number of bits of used ripple control protocol
- Ripple frequency in hertz

The final parameter that needs to be set is the sampling frequency in hertz.

The next feature of the ripple control configuration tool is the possibility to create the template from currently entered data (Save as Template button). All created templates can be saved as binary files on the computer's hard drive (Save Templates As button) and then loaded again after the next configuration tool starts (Load Templates File button). Unwanted templates can be deleted from the list of templates. Unwanted templates must be checked and can be deleted (Detele Checked Templates button).

NOTE

The generated header file can only be considered correct and be included into the application project if all seven parameters of the ripple control configuration tool are set.

The preview window of the generated header file is shown at the bottom of the ripple control configuration tool. To save the generated header file use the *Save As* function by clicking on *File* and then *Save As*.

5 Receiving a ripple control message from the grid

For testing purposes, the ripple control library was added into a single-phase Kinetis-M power meter. This single-phase power meter was connected to the grid and waited for a ripple control message to be transmitted from a ripple control transmitter, placed somewhere far away from the single-phase power meter with ripple control receiver functionality. The ripple control receiver was initialized for receiving a ripple control message according to the ZPA protocol (see Table 1).

Over a couple of hours, dozens of ripple control messages were successfully received. The figures below (Figure 7 and Figure 8) serve as clear evidence of successfully received ripple control messages.
In Figure 7, there is one ripple control message received on September 24th, 2015, at 3:51 p.m.
The received ripple control message begins with a start bit and that start bit is followed by 44 data bits.
The received ripple control message is displayed in a hexadecimal format for better readability.

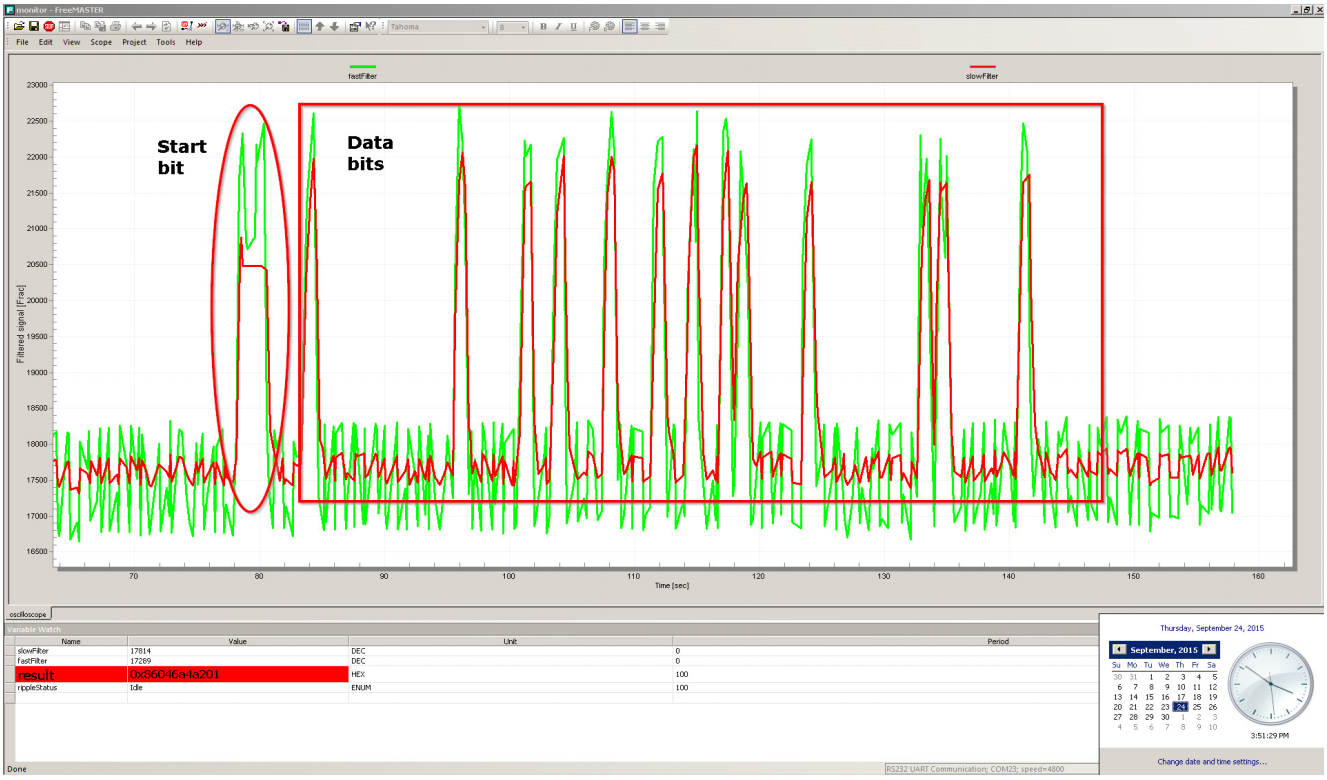

Figure 7. Ripple control message received on September 24th, 2015, at 3:51 PM in Roznov pod Radhostem

Figure 8 shows the single-phase power meter used as a ripple control receiver. On its display, there is the same ripple control message as displayed in Figure 7, but only nine digits can be displayed on the single-phase power meter display. The optically isolated RS232 interface is used for communication between FreeMASTER and the single-phase power meter with ripple control receiver functionality.

Figure 8. Single-phase power meter with ripple control receiver functionality displays a received ripple control message on September 24th, 2015, at 3:51 p.m. in Roznov pod Radhostem in hexadecimal format.

6 Summary

This application note describes the basic principles of ripple control communication. The purpose of this application note is to explain the ripple control communication protocol and the method of how the ripple

control message can be received. The C-code ripple control library (rcolib) was created to successfully receive ripple control messages from the grid.

Section 4, “Ripple control library (Rcolib) – practical implementation” describes the API of the rcolib library. A flowchart depicting one of the possible usages of the rcolib API is shown in this chapter. The header file of the ripple control library is attached in Section Appendix A, “C-Header file of Rcolib and the example of the header file generated by the configuration tool is attached in Section Appendix B, “C-Header file generated by the configuration tool.

Section 5, “Receiving a ripple control message from the grid” presents the functionality of the rcolib library in a real application. A single-phase power meter, expanded by the rcolib library was used as a ripple control receiver. The ZPA communication protocol is used in the Roznov pod Radhostem region. Over a couple of days, lots of ripple control messages were successfully received.

7 References

1. Ripple Control Information, Load Management, en.wikipedia.org/wiki/Load_management
2. Used Ripple Control Protocols, www.rundsteuerung.de/
3. IIR Filter Design, iowahills.com/A4IIRBilinearTransform.html
4. Freescale Semiconductor, “Kinetis M Series MCUs”, freescale.com/products/arm-processors/kinetis-cortex-m/m-series:KINETIS_M_SERIES?cof=0&am=0

8 Revision history

Table 6. Revision history

Revision Number	Date	Substantive Changes
0	11/2014	Initial release
1	10/2015	Added section 4.3, “Configuration tool. Updated 4, “Ripple control library (Rcolib) – practical implementation. Updated 4.1.3.4, “Description. Table 2 updated. Figure 5 changed. Figure 7changed. Appendix A, “C-Header file of Rcolib updated. Appendix B, “C-Header file generated by the configuration tool added. Appendix C, “Test example updated.
2	11/2015	Updated text in typedef struct page 17 of Appendix A, “C-Header file of Rcolib

Appendix A C-Header file of Rcolib

```
/*****************************************************************************
*
* Copyright 2015 Freescale Semiconductor, Inc.
*
* This software is owned or controlled by Freescale Semiconductor.
* Use of this software is governed by the Freescale License
* distributed with this Material.
* See the LICENSE file distributed for more details.
*
*****************************************************************************//*!
*
 * @file      RCOLIB.h
 * @author    B49954
 * @version   2.1.0.0
 * @date      October-16-2015
 * @brief     Header file containing common data types, macros and list of
 *            exported functions supporting receive ripple control message.
 *****************************************************************************/
#ifndef __RCOLIB_H
#define __RCOLIB_H


/*****************************************************************************
 * user data type & macro definitions                                        *
 *****************************************************************************/
#define RCOLIB_MSG_READY    2          /*!< Output if message is ready        */
#define RCOLIB_MSG_PENDING  1          /*!< Output if message is pending      */
#define RCOLIB_MSG_IDLE     0          /*!< Output if message is idle         */
/*****************************************************************************
 * exported function prototypes                                              *
 *****************************************************************************/
#ifdef __cplusplus
extern "C" {
#endif
```

```
/*!< Data protocol structure definition                                    */

typedef struct
{
startBitDur;      /* Start bit duration               */
startPauseDur;    /* Start pause duration             */
dataBitDur;       /* Data bit duration                */
dataPauseDur;     /* Data pause bit duration          */
numberOfBits;     /* Number of bits                   */
ripFreq;          /* Ripple frequency                 */
smplFreq;         /* Sample Frequency                 */
startBitDtcNull;  /* Start bit detected nulled        */
startBitDtcSet;   /* Start bit detected set           */
msgDur;           /* Message Duration                 */
}ProtocolData;

/*!< Bandpass filter structure definition                                  */
/*!< filter implementation: y=c(0)*x+c(1)*x(0)+c(2)*x(1)-c(3)*y(0)-c(4)*y(2)  */
typedef struct
{
  frac16    c[5];                       /* Filter coefficients            */
  frac16    x[2];                       /* Filter input array             */
  frac32    y[2];                       /* Filter output array            */
} IIR1Data;

/*!< Lowpass filter structure definition                                   */
/*!< filter implementation: y=c(0)*x+c(1)*x(0)-c(2)*y(0)                   */
typedef struct
{
  frac32    c[3];                       /* Filter coefficients            */
  frac32    x;                          /* Filter input array             */
  frac32    y;                          /* Filter output array            */
} IIR2Data;
```

```
/*!< filter implementation: y=c(0)*x+c(1)*x(0)-c(2)*y(0)                    */
typedef struct
{
  frac32    c[3];                       /* Filter coefficients              */
  frac32    x;                          /* Filter input array               */
  frac32    y;                          /* Filter output array              */
} IIR2DataSlow;

/*!< Ripple control library data structure definition                       */
typedef struct
{
  IIR1Data        iir1;                 /* Bandpass filter data             */
  IIR2Data        iir2;                 /* Lowpass filter data              */
  IIR2DataSlow    iir2slow;             /* Slow low pass filter data 1st ord */
  ProtocolData    prtcl;                /* Protocol data                    */
} tRCOLIB_DATA;

/*****************************************************************************//*!
 * @brief   Ripple control function initialization.
 * @param   ptr        Pointer to tRCOLIB_DATA strucure
 * @remarks Implemented in C-language. This function initializes all necessary
 *          variables for ripple control message receiving.
 *****************************************************************************/
extern void       RcolibInit (tRCOLIB_DATA *const ptr);


/*****************************************************************************//*!
 * @brief   Decodes ripple control message from phase voltage waveform.
 * @param   u       Instantaneous phase voltage sample in 16-bit fractional
 *                  data format.
 * @param   ptr        Pointer to tRCOLIB_DATA strucure
 * @remarks Implemented in C-language. This function must be called at constant
 *          frequency.
 *****************************************************************************/
extern void       RcolibDecode   (register frac16 u, tRCOLIB_DATA *const ptr);
```

```
/*************************************************************************//*!
 * @brief   Monitor the status of ripple control communication.
 * @return  Function returns: @ref RCOLIB_MSG_READY if ripple control message is
 *          decoded and ready to be read, @ref RCOLIB_MSG_PENDING if ripple control
 *          message reception is in progress or @ref RCOLIB_MSG_IDLE if idle..
 * @remarks Implemented in C-language. This function can be used to find out
 *          the actual status of ripple control receiving.
 *************************************************************************/
extern int        RcolibDecodeInfo (void);


/*************************************************************************//*!
 * @brief   Read ripple control message.
 * @return  Function returns all data bits from ripple control message.
 * @remarks Implemented in C-language. This function can be used to read the
 *          ripple control information.
 *************************************************************************/
extern long long  RcolibMsgRead  (void);

/*************************************************************************//*!
 * @example   rcolib_test.c
 * This example demonstrates use of ripple control library in typical application
 * with input phase voltage waveform generated numerically.
 *************************************************************************/

#ifdef __cplusplus
}
#endif

#endif /* __RCOLIB_H */
```

Appendix B C-Header file generated by the configuration tool

```
/*****************************************************************************
* Ripple Control Library Configuration Header File, Created: 9/4/2015 14:15
 *****************************************************************************/
#ifndef __RCOLIB_CFG_H
#define __RCOLIB_CFG_H
/*****************************************************************************
* Ripple Control Configuration Structure
 *****************************************************************************/
#define RCOLIB_CFG                                                              \
{                                                                               \
 {                                                                              \
 {FRAC16(0.0079006),FRAC16(0.00),FRAC16(-0.0079006),FRAC16(-0.0866869),FRAC16(0.4841988)},\
 {FRAC16(0.0),FRAC16(0.0)},                                                     \
 {FRAC32(0.0),FRAC32(0.0)}                                                      \
 },                                                                             \
 {                                                                              \
 {FRAC32(0.0103586),FRAC32(0.0103586),FRAC32(-0.9792827)},                      \
 FRAC32(0.0),                                                                   \
 FRAC32(0.0)                                                                    \
 },                                                                             \
 {                                                                              \
 {FRAC32(0.0026098),FRAC32(0.0026098),FRAC32(-0.9947803)},                      \
 FRAC32(0.0),                                                                   \
 FRAC32(0.00001)                                                                \
 },                                                                             \
 {                                                                              \
  2.33000, 2.99000, 1.00000, 0.33000,44,  283.30,1200,1831,2516,73812}          \
 }
#endif /*__RCOLIB_CFG_H */
```

Appendix C Test example

```
/*****************************************************************************
*
* Copyright 2015 Freescale Semiconductor, Inc.
*
* This software is owned or controlled by Freescale Semiconductor.
* Use of this software is governed by the Freescale License
* distributed with this Material.
* See the LICENSE file distributed for more details.
*
*****************************************************************************//*!
*
 * @file      rcolib_test.c
 * @author    B49954
 * @version   2.1.0.0
 * @date      October-16-2015
 * @brief     C - Source module with software which can serve as a demonstration
 *            of rcolib library functionality
 ******************************************************************************
 * rcolib_test.c
 *****************************************************************************/

#include "drivers.h"
#include "freemaster.h"
#include <math.h>
#include "rcolib_cfq.h"
#include "rcolib.h"

#ifndef CALCFREQ
    #define CALCFREQ    1200.000       /* Sample frequency in Hz              */
#endif
#define TIMESTEP(s) (int)((s)*CALCFREQ)/* Converts seconds to number of steps */
#define RIP_OMEGA   (1.780E+3)                  /* 2*Pi*f_rip  (283.3Hz)     */
#define RIP_AMPL    (8.131727984E-3)           /* (0.01*230*sqrt(2))/400    */
#define LINE_OMEGA  (3.141592653E+2)               /* 2*Pi*f_line (50Hz)       */
```

```
#define LINE_AMPL    (8.131727984E-1)              /* (230*sqrt(2))/400        */
#define ANGLE        (7.853981634E-1)              /* (45/180)*Pi              */
#pragma diag_suppress = Pa082
/* 3.490 s is time for the center of first data bit after start bit detection */
/* then every 1.330s another data bit is expected                             */
/* it is true only for ZPA protocol                                           */

/*! DataReadyTm is created just for testing the ripple control library        */
static const int  DataReadyTm[]= {  TIMESTEP(3.490),  TIMESTEP(4.820), //2
                  TIMESTEP(6.150),  TIMESTEP(7.480),  TIMESTEP(8.810), //5
                  TIMESTEP(10.140), TIMESTEP(11.470), TIMESTEP(12.800),//8
                  TIMESTEP(14.130), TIMESTEP(15.460), TIMESTEP(16.790),//11
                  TIMESTEP(18.120), TIMESTEP(19.450), TIMESTEP(20.780),//14
                  TIMESTEP(22.110), TIMESTEP(23.440), TIMESTEP(24.770),//17
                  TIMESTEP(26.100), TIMESTEP(27.430), TIMESTEP(28.760),//20
                  TIMESTEP(30.090), TIMESTEP(31.420), TIMESTEP(32.750),//23
                  TIMESTEP(34.080), TIMESTEP(35.410), TIMESTEP(36.740),//26
                  TIMESTEP(38.070), TIMESTEP(39.400), TIMESTEP(40.730),//29
                  TIMESTEP(42.060), TIMESTEP(43.390), TIMESTEP(44.720),//32
                  TIMESTEP(46.050), TIMESTEP(47.380), TIMESTEP(48.710),//35
                  TIMESTEP(50.040), TIMESTEP(51.370), TIMESTEP(52.700),//38
                  TIMESTEP(54.030), TIMESTEP(55.360), TIMESTEP(56.690),//41
                  TIMESTEP(58.020), TIMESTEP(59.350), TIMESTEP(60.680),//44
                   };  /* Data time steps                                     */

/*! Variables                                                                 */
static volatile frac16    u16_sample;        /* Voltage Sample               */
static volatile double    time = 0.000,U_ANGLE = (0.0/180.0)*3.1415927, x=0;
static volatile double    U_MAX=400.000;     /* Voltage Scale                */
static volatile frac32    thr1=FRAC32(0.99); /* Initial threshold value      */
static volatile int       Cnt=-1*TIMESTEP(2.330),Rip=0;
/* Cnt is initialized to -2976 due to start bit and Rip initialized to 0     */
static volatile long long buffer;   /* Result buffer                         */
static volatile int  pit_change = 0;/* PIT flag                              */
volatile int status=0;              /* Receiver status                       */
```

```
static int i=0;                     /* Data bit counter                        */
static int wait=1000;               /* Delay for ripple signal generation      */

/*!< Ripple control data structure definition                                  */
static tRCOLIB_DATA  rcolib = RCOLIB_CFG;
/*****************************************************************************
   * PIT - 1200Hz                                                             *
*****************************************************************************/
void pit_callback (PIT_CALLBACK_TYPE type);
/* PIT callback definition */
void pit_callback (PIT_CALLBACK_TYPE type)
{
  if (type == PIT0CH0_CALLBACK) {pit_change = 1 ; }

}

void main (void)
{
  SIM_Init  (SIM_MODULE_ALL_PERIPH_ON_CONFIG);

  /* route system clock to PTF7                                               */
  SIM_SelClkout(CLKOUT_SRC1);
  PORT_Init (PORTF,PORT_MODULE_ALT3_MODE,PIN7);

  /* clock mode 2:1:1, 48MHz                                                  */
  SIM_SetClkMode (SYSCLK_MODE1);
  SIM_SetClkDiv  (SYSCLK_DIV1);
  FLL_Init  (FLL_MODULE_FEE_48MHZ_CONFIG);


  /* PIT initialization - 1200Hz                                              */
  PIT_InstallCallback (PRI_LVL0,pit_callback);
  PIT_Init (PIT0,CH0, PIT_CH_TMR_EN_CONFIG, 24000000/1200);
  PIT_Enable (PIT0,CH0);
```

```
/* initialize UART and FreeMASTER                                          */
PORT_Init (PORTI, PORT_MODULE_ALT2_MODE, PIN6|PIN7);
UART_Init (UART2, UART_MODULE_POLLMODE_CONFIG(115200,24e6));
UART_InstallCallback (PRI_LVL2, (UART_CALLBACK)FMSTR_Isr);
FMSTR_Init();

/* initialize PORT C                                                       */
PORT_Init (PORTK, PORT_MODULE_ALT1_MODE, PIN2);
GPIO_Init (GPIOK, GPIO_OUT_LOGIC1_MODE, PIN2);
PORT_Init (PORTK, PORT_MODULE_ALT1_MODE, PIN3);
GPIO_Init (GPIOK, GPIO_OUT_LOGIC1_MODE, PIN3);
PORT_Init (PORTG, PORT_MODULE_ALT1_MODE, PIN0);
GPIO_Init (GPIOG, GPIO_OUT_LOGIC0_MODE, PIN0);

/* initialize LEDs                                                         */
PORT_Init (PORTJ, PORT_MODULE_LED_MODE, PIN3); /* green led              */
PORT_Init (PORTJ, PORT_MODULE_LED_MODE, PIN4); /* red led                */
PORT_Init (PORTD, PORT_MODULE_LED_MODE, PIN0); /* orange led             */
GPIO_Init (GPIOJ, GPIO_OUT_LOGIC0_MODE, PIN3); /* green led              */
GPIO_Init (GPIOJ, GPIO_OUT_LOGIC0_MODE, PIN4); /* red led                */
GPIO_Init (GPIOD, GPIO_OUT_LOGIC0_MODE, PIN0); /* orange led             */

 EnableInterrupts();

/*************************************************************************
 * Ripple control detection - initialization                             *
 *************************************************************************/
 RcolibInit(&rcolib);

/* ZPA, 283.3Hz ripple frequency, 1200Hz sample frequency                 */

 while(1)
{
/*************************************************************************
 * PIT - 1200Hz                                                          *
```

```
 ****************************************************************************/
 while (pit_change == 0); /* Waiting for PIT callback, every 1/1200 s       */
 pit_change = 0;

 time = time+(1.0/CALCFREQ);

/****************************************************************************
 * Calculate phase voltage waveform with ripple control information         *
 ****************************************************************************/
 x = (Rip)?(sin(RIP_OMEGA*time+U_ANGLE)*RIP_AMPL):0.0;
 u16_sample = FRAC16(((sin(LINE_OMEGA*time+U_ANGLE)*LINE_AMPL)+x));
 /****************************************************************************
 * Ripple control message generation - example                              *
 ****************************************************************************/

 if(wait==0)
 {
  if(Cnt <0)                                  /* Generating start bit       */
   {
     Rip=1;                                   /* Start bit                  */
   }

  Cnt++;
  if (Cnt == (DataReadyTm[i]-TIMESTEP(0.50))) /* Data bit start             */
  {
    Rip=1;
  }

  if (Cnt == (DataReadyTm[i]+TIMESTEP(0.50))) /* Data bit end               */
  {
    Rip=0;                                    /*Data Pause                  */
    i++;
  }

 /* toggle Rip from 1 to 0. It represents end of start bit.                 */
```

```
 if (Cnt == TIMESTEP(0.000)) /* Cnt is initialized to -2.330*1200= -2976   */
 {
   Rip = 0;                  /* Due to start pause bit                     */
 }
}
else
{
  wait--; /*Decrementing wait*/
}

/*****************************************************************************
 *  Ripple control detection - call in task when phase voltage is measured *
 *****************************************************************************/
RcolibDecode (u16_sample, &rcolib);
/*****************************************************************************
 *  Ripple control message ready and read - call whenever needed, but      *
 *  at slow task for example when updating LCD screens                     *
 *****************************************************************************/
status=RcolibDecodeInfo ();

 /*****************************************************************************
  * In variable "status" is saved the status of ripple control receiver    *
  * "0" if idle, "1" if pending, "2" if message is succesfuly received and *
  * ready to be read.                                                      *
  *****************************************************************************/

switch(status)
 {
   case 0:
      GPIO_Set  (GPIOJ, PIN4);
      GPIO_Set  (GPIOD, PIN0);
      GPIO_Clr  (GPIOJ, PIN3); /*Green Led if idle                         */
      break;

   case 1:
```

```
        GPIO_Set   (GPIOJ, PIN3);
        GPIO_Set   (GPIOD, PIN0);
        GPIO_Clr   (GPIOJ, PIN4); /*Red Led if pending                          */
        break;

      case 2:
        GPIO_Set   (GPIOJ, PIN3);
        GPIO_Set   (GPIOJ, PIN4);
        GPIO_Clr   (GPIOD, PIN0); /*Orange Led if message is ready            */
        break;

    default:
        break;
    }

   if (status == RCOLIB_MSG_READY)
     {
      buffer = RcolibMsgRead(); /* Result word is saved into buffer          */
      /* Variables need to be initialized again, before next packet receiving*/
      Cnt=-1*TIMESTEP(2.330);  /*Cnt is initialized to -2976 due to start bit*/
     Rip=0;                  /* Rip is initialized to 1 due to start bit   */
      time = 0;                /* Time is nulled                            */
      i=0;                     /* Data bit counter nulled                   */
      wait=1000;               /* Set wait for another calibration function  */
     }
   }
}
```

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Document Number: AN5046
Rev. 2
11/2015