PMA71xx / PMA51xx

SmartLEWIS^{TM} MCU

RF Transmitter FSK/ASK 315/434/868/915 MHz
Embedded 8051 Microcontroller with 10 bit ADC
Embedded 125 kHz ASK LF Receiver

Application Note

PMAfob Software Example
Revision 1.2, 2010-10-11

Sense & Control

Edition 2010-10-11

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2010 Infineon Technologies AG
All Rights Reserved.

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Table of Contents

Revision History: 2010-10-11, Revision 1.2

Previous Revision: 1.1

Page	Subjects (major changes since last revision)
6	Path of source code download changed
6, 8	Unique ID can be used as PMA unique ID or customer defined unique ID
12	Error in figure 3 corrected: 0..button pressed, 1..button released
	General description as wireless remote control application, not only RKE
1,8-17	Description of configurable software by long button presses

Trademarks of Infineon Technologies AG

ABM™, BlueMoon™, CONVERGATE™, COSIC™, C166™, FALC™, GEMINAX™, GOLDMOS™, ISAC™, OMNITUNE™, OMNIVIA™, PROSOC™, SEROCCO™, SICOFI™, SIEGET™, SMARTi™, SmartLEWIS™, SMINT™, SOCRATES™, VINAX™, VINETIC™, VOIPRO™, X-GOLD™, XMM™, X-PMU™, XWAY™

Other Trademarks

Microsoft®, Visio®, Windows®, Windows Vista®, Visual Studio®, Win32® of Microsoft Corporation. Linux® of Linus Torvalds. FrameMaker®, Adobe® Reader™, Adobe Audition® of Adobe Systems Incorporated. APOXI®, COMNEON™ of Comneon GmbH & Co. OHG. PrimeCell®, RealView®, ARM®, ARM® Developer Suite™ (ADS), Multi-ICE™, ARM1176JZ-S™, CoreSight™, Embedded Trace Macrocell™ (ETM), Thumb®, ETM9™, AMBA™, ARM7™, ARM9™, ARM7TDMI-S™, ARM926EJ-S™ of ARM Limited. OakDSPCore®, TeakLite® DSP Core, OCEM® of ParthusCeva Inc. IndoorGPS™, GL-20000™, GL-LN-22™ of Global Locate. mipi™ of MIPI Alliance. CAT-iq™ of DECT Forum. MIPS™, MIPS II™, 24KEc™, MIPS32®, 24KEc™ of MIPS Technologies, Inc. Texas Instruments®, PowerPAD™, C62x™, C55x™, VLYNQ™, Telogy Software™, TMS320C62x™, Code Composer Studio™, SSI™ of Texas Instruments Incorporated. Bluetooth® of Bluetooth SIG, Inc. IrDA® of the Infrared Data Association. Java™, SunOS™, Solaris™ of Sun Microsystems, Inc. Philips®, I2C-Bus® of Koninklijke Philips Electronics N.V. Epson® of Seiko Epson Corporation. Seiko® of Kabushiki Kaisha Hattori Seiko Corporation. Panasonic® of Matsushita Electric Industrial Co., Ltd. Murata® of Murata Manufacturing Company. Taiyo Yuden™ of Taiyo Yuden Co., Ltd. TDK® of TDK Electronics Company, Ltd. Motorola® of Motorola, Inc. National Semiconductor®, MICROWIRE™ of National Semiconductor Corporation. IEEE® of The Institute of Electrical and Electronics Engineers, Inc. Samsung®, OneNAND®, UtRAM® of Samsung Corporation. Toshiba® of Toshiba Corporation. Dallas Semiconductor®, 1-Wire® of Dallas Semiconductor Corp. ISO® of the International Organization for Standardization. IEC™ of the International Engineering Consortium. EMV™ of EMVCo, LLC. Zetex® of Zetex Semiconductors. Microtec® of Microtec Research, Inc. Verilog® of Cadence Design Systems, Inc. ANSI® of the American National Standards Institute, Inc. WindRiver® and VxWorks® of Wind River Systems, Inc. Nucleus™ of Mentor Graphics Corporation. OmniVision® of OmniVision Technologies, Inc. Sharp® of Sharp Corporation. Symbian OS® of Symbian Software Ltd. Openwave® of Openwave Systems, Inc. Maxim® of Maxim Integrated Products, Inc. Spansion® of Spansion LLC. Micron®, CellularRAM® of Micron Technology, Inc. RFMD® of RF Micro Devices, Inc. EPCOS® of EPCOS AG. UNIX® of The Open Group. Tektronix® of Tektronix, Inc. Intel® of Intel Corporation. Qimonda® of Qimonda AG. 1GOneNAND® of Samsung Corporation. HyperTerminal® of Hilgraeve, Inc. MATLAB® of The MathWorks, Inc. Red Hat® of Red Hat, Inc. Palladium® of Cadence Design Systems, Inc. SIRIUS Satellite Radio® of SIRIUS Satellite Radio Inc. TOKO® of TOKO Inc., KEIL™

The information in this document is subject to change without notice.

Last Trademarks Update 2009-03-10

List of Figures

1 Introduction

The PMAfob Software Example shows a possible software solution for wireless remote control applications like Remote Keyless Entry (RKE) or Home Automation using PMA51xx or PMA71xx. The available software example files include inline documentation and are developed to enable an easy software development start and fast time to market. The software example files can be downloaded from **http://www.infineon.com/PMA_tooling.** The installer for the source code of the PMAfob Software Example is called *PMAfob_DEMO_Tx_Sources_Vx.y.msi*. The installer is included in the following download packages:

- PMAfob - Software Example
- PMAfob - RKE Demo Software
- PMAfob - Home Automation Demo Software

Furthermore more documentation of the source code done with doxygen is also included in the download package and can be displayed with a standard browser by opening file *./PMAfobSoftwareExample/html/index.html* after running *PMAfob_DEMO_Tx_Sources_Vx.y.msi* installer.

The software example supports the following features:

- Five buttons (Handling of five external wake-ups)
- Button stuck detection
- Debounce buttons
- Secure communication
  - AES[1] or XTEA[2] encryption
  - Rolling code generation
- Battery voltage measurement
- Energy saving by using Power Down Mode
- Unique ID (PMA or user defined)
- Configurable software by longer button presses
  - Change RF Framing
  - Switch encryption ON / OFF and change baudrate
- Change unique key number (PMA or user defined)

This document describes the general file structure, the program flow, the initialization of the emulated EEPROM, the port sampling and the RF-protocol which is used in the PMAfob Software Example. Furthermore the configuration possibilities by longer button presses are illustrated. This document is compatible with source code revision 1.1 (*PMAfob_DEMO_Tx_Sources_V1.1.msi*).

2 File structure

This chapter gives an overview over the file structure of the PMAfob Software Example and describes the functionality implemented in the source files. **Figure 1** shows how the files are linked.

1) Advanced Encryption Standard

2) eXtended Tiny Encryption Algorithm

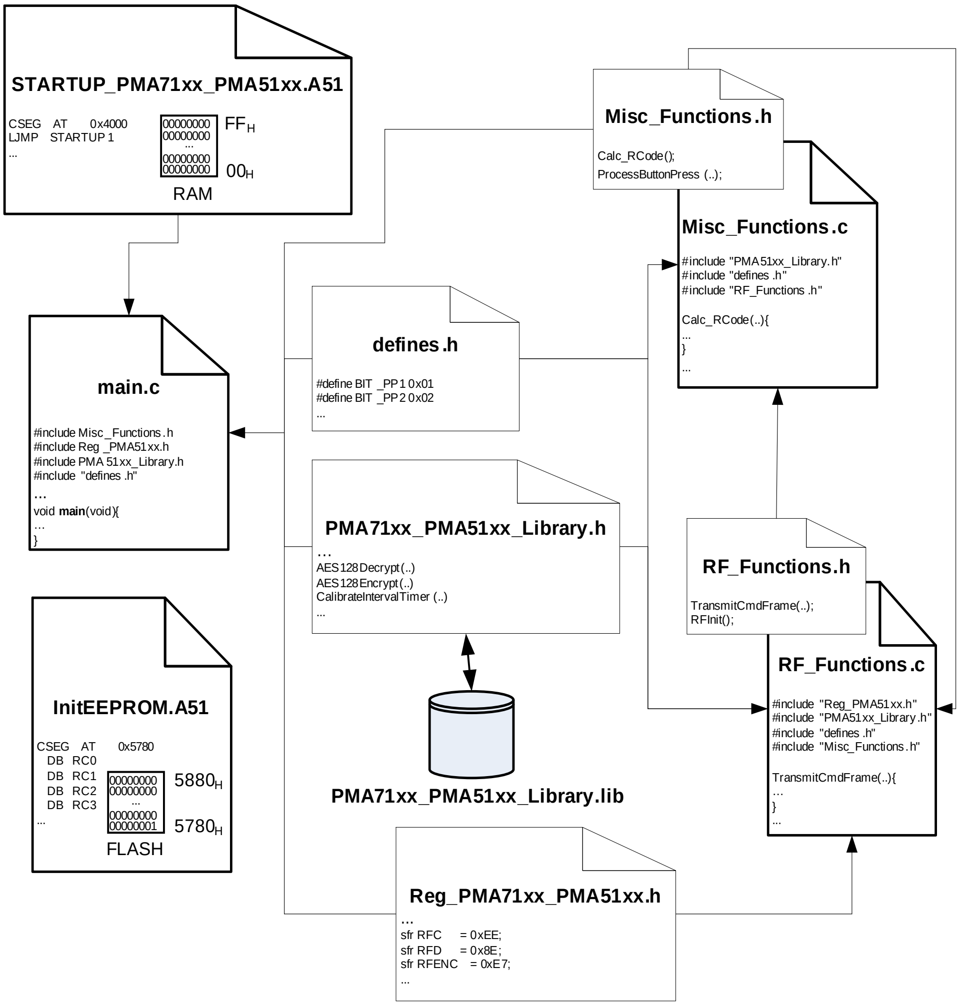


Figure 1 File Structure of PMAfob Software Example

2.1 Header files

In the header files the interfaces to different modules are defined.

2.1.1 Reg_PMA71xx_PMA51xx.h

This is the register definition file for PMA71xx / PMA51xx. Here all SFRs (Special Function Registers) of PMA71xx / PMA51xx are defined. This file has to be added to the project if direct SFR access is needed.

2.1.2 PMA71xx_PMA51xx_Library.h

PMA71xx_PMA51xx_Library.h is the interface to the PMA71xx / PMA51xx Function Library. The prototypes of the Function Library and some declarations for the RF-Transmission are defined here. This file has to be added to the project together with *PMA71xx_PMA51xx_Library.lib* if the PMA71xx / PMA51xx Function Library is intended to be used. All Functions of the Function Library are described in detail in [1].

2.1.3 defines.h

Defines.h includes bit definitions to increase the readability of the code. The enumeration types *Encryption_Type, Encryption_Status* and *eUnique_Key_Nr* are also defined here. The struct *ActualButtonPresses_s* is used to identify the pressed button and count the actual button press duration.

2.1.4 RF_Functions.h

The interface to functions *TransmitCmdFrame_AES()* and *TransmitCmdFrame_XTEA()* is defined in *RF_Functions.h*.

2.1.5 Misc_Functions.h

The interface to functions *Calc_RCode()*, *ProcessButtonPress()* and *XTEA_encipher()* is defined in *Misc_Functions.h*.

2.2 Source files

The source files include the implementation of the start up file, the EEPROM initialization file, the functions used for button press detection, rolling code generation, RF framing and transmission.

2.2.1 STARTUP_PMA71xx_PMA51xx.A51

This file is a modified copy of the standard 8051 startup file *STARTUP.A51* delivered from KEIL™. It has to be added to the project. If not, the standard *STARTUP.A51* is included by the linker. If PMA71xx / PMA51xx starts up from reset the whole idata memory 00_H - FF_H is initialized to 00_H. The lower idata memory 00_H - $7F_H$ can be powered during Power Down or Thermal Shutdown Mode by setting SFR bit CFG2.4 [PDLMB] to zero. If the PMA71xx / PMA51xx wakes up from Power Down or Thermal Shutdown State bit PDLMB is checked. If the lower idata memory is powered during Power Down or Thermal Shutdown State only the higher idata memory block 80_H - FF_H, otherwise the whole idata memory 00_H - FF_H, is initialized to 00_H.

Finally the stack pointer is set and a ljmp to *main()* is executed.

2.2.2 InitEEPROM.A51

The location of the rolling code start value in the FLASH is defined in this file. When the program is loaded down to the device the rolling code is written to FLASH User Data Sector I.

2.2.3 main.c

This file includes the *main()* function of the PMAfob Software Example and is executed after *STARTUP_PMA71xx_PMA51xx.A51*.

The following global variables are used to change the RF-Frame, the encryption, and the unique key number by a long button press:

- *My_Encryption_Type*: AES or XTEA framing can be used. The RF-Frames are defined in **Chapter 6**.
- *My_Encryption_Status*: AES or XTEA (depends on *My_Encryption_Type*) encryption can be switched on/off.
- *My_Unique_Key_Nr*: 4 byte of PMA unique ID or user defined unique ID can be used. The user defined unique ID is set with #define *USER_KEY_NR* in defines.h.

The xdata variables are also defined globally. In the *main()* function the wake-up bit DSR.1 [WUP] is checked to decide whether the device starts up from reset or with a wake-up event from Power Down Mode.

In case of a reset, PP1-PP4 and PP6 are configured to be used as the external wake-ups. Therefore the port direction is set to input, the internal pull-up resistors and the external wake-ups WU0-WU4 are enabled (unmasked). PP8 is also set to input and the pull-up resistor for PP8 is enabled. PP8 is used to check if the port sampling feature has been used (when one sector of the emulated EEPROM has been erased). Finally some variables are initialized and the Interval Timer is set to the maximal wake-up interval of about 524 s.

In the wake-up routine the Watchdog Timer is handled. External wake-ups in combination with the Interval Timer wake-up are used to debounce the buttons, detect button presses and button stuck.

2.2.4 Misc_Functions.c

The functions *Check_SampleArray()*, *Calc_RCode()*, *ProcessButtonPress()* and *XTEA_encipher()* are implemented within this file. A description of the functionality of each function can be found below.

2.2.4.1 Check_SampleArray()

Check_SampleArray() is a basic implementation of checking the sample array which is used to monitor the ports while the EEPROM write function in *Calc_RCode()* is executed. The WU ports are sampled on every 4th write access. PP8 is used to check if the ports were sampled. If the Ports were sampled, the sample array is 1_B on the position of PP8, due to the internal pull-up resistor, otherwise it is 0_B.

2.2.4.2 Calc_RCode()

The previous rolling code is loaded from the emulated EEPROM. According to the setting of the global variable *My_Encryption_Type* in file *main.c* the new rolling code is calculated. If AES Framing has been choosen the previous rolling code is multiplied with a constant long (32 bit) value. Then a constant int (16 bit) value is added. If XTEA Framing has been choosen the rolling code is implemented as a simple 32 bit up-counter. Therefore the previous rolling code is incremented by 1. The result, the actual rolling code, is stored to the emulated EEPROM. While the EEPROM write function is executed the ports are monitored to be able to detect a button pressed in between.

2.2.4.3 ProcessButtonPress()

When a button is pressed the button ID is stored in the battery buffered xdata. *ProcessButtonPress()* checks if the ID of the pressed button is equal to the ID stored in the xdata (button has been pressed before) and increments a counter. If this counter reaches a predefined value *TransmitCmdFrame_AES()* or *TransmitCmdFrame_XTEA()* is called and the button ID is set to zero, so the counter is not increased by a stuck button. Within *TransmitCmdFrame_AES()* or *TransmitCmdFrame_XTEA()* a new rolling code is calculated by *Calc_RCode()*. If a button press has been detected during *Calc_RCode()* a new rolling code is calculated and the appropriate RF-Frame is sent.

2.2.4.4 XTEA_encipher()

The algorithm for encrypting data with XTEA is a public domain implementation by David Wheeler and Roger Needham. It is implemented in C and not optimized for 8051 microcontrollers up to now. 32 rounds are used for encryption.

2.2.5 RF_Functions.c

The functions of *RFInit()*, *RFTransmit()*, *TransmitCmdFrame_AES()* and *TransmitCmdFrame_XTEA()* are implemented in *RF_Functions.c*.

2.2.5.1 RFInit()

First of all a variable of the struct *RF_Config*, see *PMA71xx_PMA51xx_Library.h*, is defined and initialized. Then the Library function *InitRF()* is called to set the appropriate values to all SFRs needed for the RF-Transmission. A detailed description of each element of the struct *RF_Config* can be found in **[1]**.

2.2.5.2 RFTransmit()

This function is used to generate the RF-Framing which is required by the TDA523x receiver. The RF-Frame is described in detail in **Chapter 6**.

2.2.5.3 TransmitCmdFrame_AES()

The payload of the RF-Frame is generated within this function. The structure of this payload can be found in **Chapter 6.1**. A part of the payload is the rolling code and the battery voltage, which is measured using the Library Function *MeasureSupplyVoltage()*. The whole 128 bit payload can be encrypted with AES and embedded into the RF-Frame.

2.2.5.4 TransmitCmdFrame_XTEA()

The payload of the RF-Frame is generated within this function. The structure of this payload can be found in **Chapter 6.2**. A part of the payload is the rolling code and the battery voltage, which is measured using the Library Function *MeasureSupplyVoltage()*. According to the LSB of the rolling code the nibbles of some bytes of the payload are swapped if the RF-Frame is encrypted which is shown in **Chapter 6.2.1**. 64 bit of the 88 bit payload can be encrypted with XTEA and embedded into the RF-Frame.

2.3 Function Library file (PMA71xx_PMA51xx_Library.lib)

This is the Function Library where all functions, defined in PMA71xx_PMA51xx_Library.h, are implemented. All functions of the Function Library are described in detail in **[1]**.

3 Program flow

Figure 2 on Page 11 shows the program flow of the PMAfob Software Example.

The device starts program execution within the startup file. After RAM initialization the wake-up flag in SFR DSR.1 [WUP] is checked to decide whether the device starts up from a reset or because of a wake-up event.

If PMA71xx / PMA51xx starts up from reset PP1-PP4 and PP6 (WU0-WU4) are configured as external wake-up pins. Therefore the port direction must be set to input and the internal pull-up resistors must be activated (assumed that a pressed button generates a LOW on the pin). Also PP8 is set to input and its internal pull-up resistor is activated. This is used for analysing the port sampling array. Finally, the Interval Timer is set to the longest possible wake-up interval of about 524 s and the PMA71xx / PMA51xx is set into Power Down Mode to save energy and waits for a wake-up event.

If PMA71xx / PMA51xx starts up from Power Down Mode with a wake-up event, the wake-up source is checked. Seven wake-up sources are handled by the PMAfob Software Example. These are the Watchdog Timer, the external wake-ups WU0-WU4 and the Interval Timer. The Watchdog Timer wake-up has the highest priority. If a Watchdog Timer wake-up occurs, a software reset is triggered to ensure that all SFRs have a predefined state.

If no Watchdog Timer wake-up preceded, the external wake-ups are checked. When an external wake-up has been detected, the appropriate wake-up source is disabled (masked), a wake-up ID is stored in the xdata and the Interval Timer is set to 50 ms.

In the Interval Timer wake-up routine the pins PP1-PP4 and PP6 are checked to detect whether a button is still pressed. If the external wake-up ID previously stored in the xdata is equal to the currently pressed button a counter is increased. If this counter reaches a predefined value, the button is recognized to be pressed. Then a new rolling code is calculated, the battery voltage is measured, the appropriate command is inserted into the RF-Frame, the RF-Frame is encrypted with AES or XTEA, if encryption is switched on, and transmitted. When the new rolling code is written to the EEPROM the ports are sampled for any action. If a button press was detected during writing to the EEPROM, a new rolling code is calculated and the appropriate RF-Frame is sent. For every button which is detected to be unpressed in the Interval Timer wake-up routine the corresponding wake-up is (re-)enabled. If no button is pressed the Interval Timer is set to the longest possible wake-up interval of about 524 s. A *Key Stuck* is detected by the Interval Timer wake-up routine when a button is pressed for at least 1h and no other button is pressed in between. Some configurations of the software can be changed by a long button press (see **Chapter 7**). Therefore the Interval Timer is used to check if a button is pressed for a longer time then set by #define *SWITCH_DUR* (defines.h).

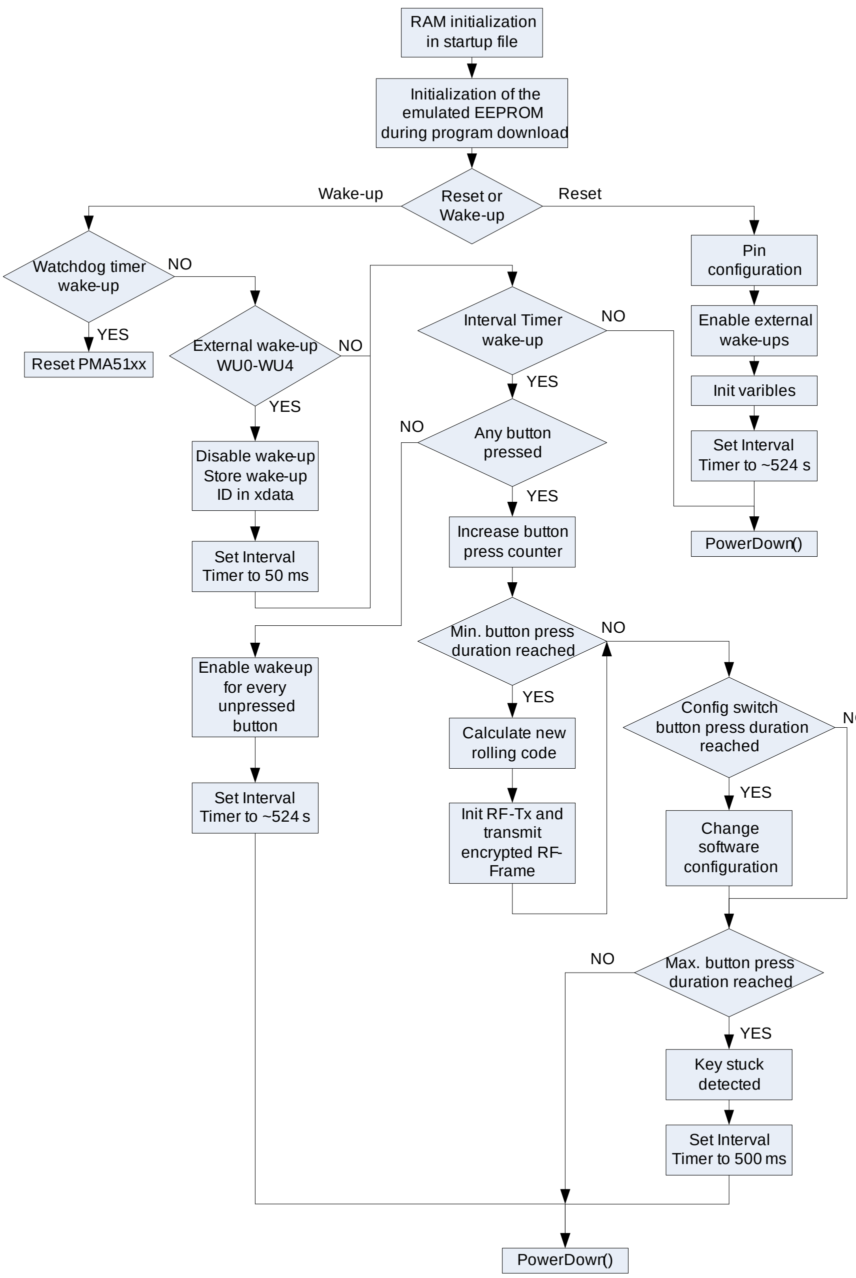


Figure 2 Program flow of the PMAfob Software Example

Figure 3 on Page 12 shows the timing of a button press and how the Interval Timer wake-up interval is varied. While no button is pressed the Interval Timer wakes up the PMA71xx / PMA51xx with the longest possible interval of about 524 s to save energy. If a button is pressed an external wake-up is detected and the Interval Timer is set to 50 ms. The Interval Timer wake-up service routine checks if the button is still pressed every 50 ms and sets the device into Power Down Mode between each measurement. This method is used to debounce the buttons.

If a button is pressed for at least 150 ms (three button checks in the Interval Timer wake-up service routine resulted in a pressed button) a *Button Press* is identified and an RF-Transmission is started. The debounce time of 150 ms can be easily changed by modifying the value of *BUTTON_PRESS_DUR* defined in file *defines.h* and / or changing the Interval Timer settings. As long as the button is pressed the Interval Timer wakes up every 50 ms to check the button. This is done for at most 1 h.

For button presses which are longer then 3 sec, the software configuration is changed according to the pressed button (see **Chapter 7**). The configuration switch time can be easily changed by modifing the value of *SWITCH_DUR* defined in file *defines.h and / or changing the Interval Timer settings*.

If a button is pressed for about 1 h, and no other button is pressed in between, the button is detected to be stuck and the Interval Timer is set to 500 ms to save energy. If the button is released, and no other button is pressed, the Interval Timer is set to about 524 s again.

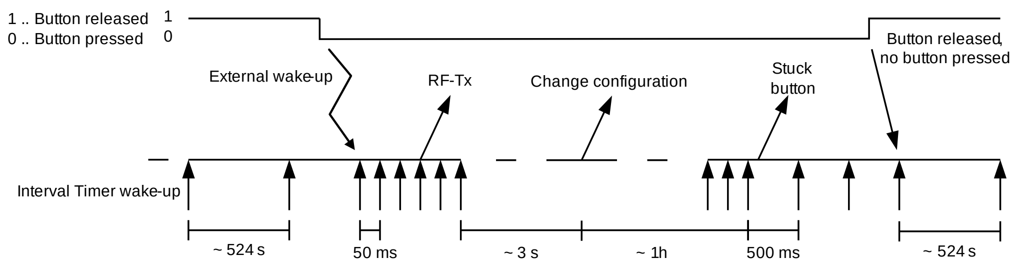


Figure 3 Timing of button press and Interval Timer

4 Initialization of the emulated EEPROM

The PMAfob Software Example stores the rolling code in the emulated EEPROM. This is done to be able to keep the rolling code while the PMA71xx / PMA51xx is set into Power Down Mode. But how should the emulated EEPROM be initialized ?

There are different methods to initialize the PMA71xx / PMA51xx emulated EEPROM:

- With PMA71xx / PMA51xx Function Library function *EEPROM_Init*()
- During program download

One approach would be to initialize the emulated EEPROM in the reset rountine by calling the Library function *EEPROM_Init()*. For a remote control application where a rolling code is used to increase the security level, this method has the major drawback that after a battery replacement a synchronisation between the remote control and the receiver is necessary. The problem is that the rolling code of the remote control is reset, while the rolling code of the receiver remains.

The solution for the problem is to initialize the emulated EEPROM, including the rolling code start value, during program download. If the battery is replaced, the rolling code of the PMAfob remains and no synchronisation is necessary.

Note: Precausion to initialize the emulated EEPROM, including the rolling code start value, during program download

1. *Add InitEEPROM.A51 to the Keil project*
2. *Ensure that User Data Sector I and II are erased before program download*

5 Port Sampling

After a button press is detected a new rolling code is calculated and stored into the emulated EEPROM. With every 4th write access to the emulated EEPROM a User Data Sector has to be erased which takes about 102 ms. To be able to detect button presses during this time, it is possible to sample the wake-up ports. The PMAfob Software Example shows how the port sampling feature can be handled. **Figure 4** illustrates how the port sampling array looks like and how this array is manipulated. The Library function *Wr_EELong(..)* is called within *Calc_RCode(..)*. If a User Data Sector is erased during *Wr_EELong(..)* the wake-up ports are sampled every 5 ms and inserted into the sampling array as shown in **Figure 4**.

The function *Check_SampleArray(..)* analyses the sampling array and declares whether a button was pressed or not. If a button press is detected, the appropriate RF-Frame is sent.

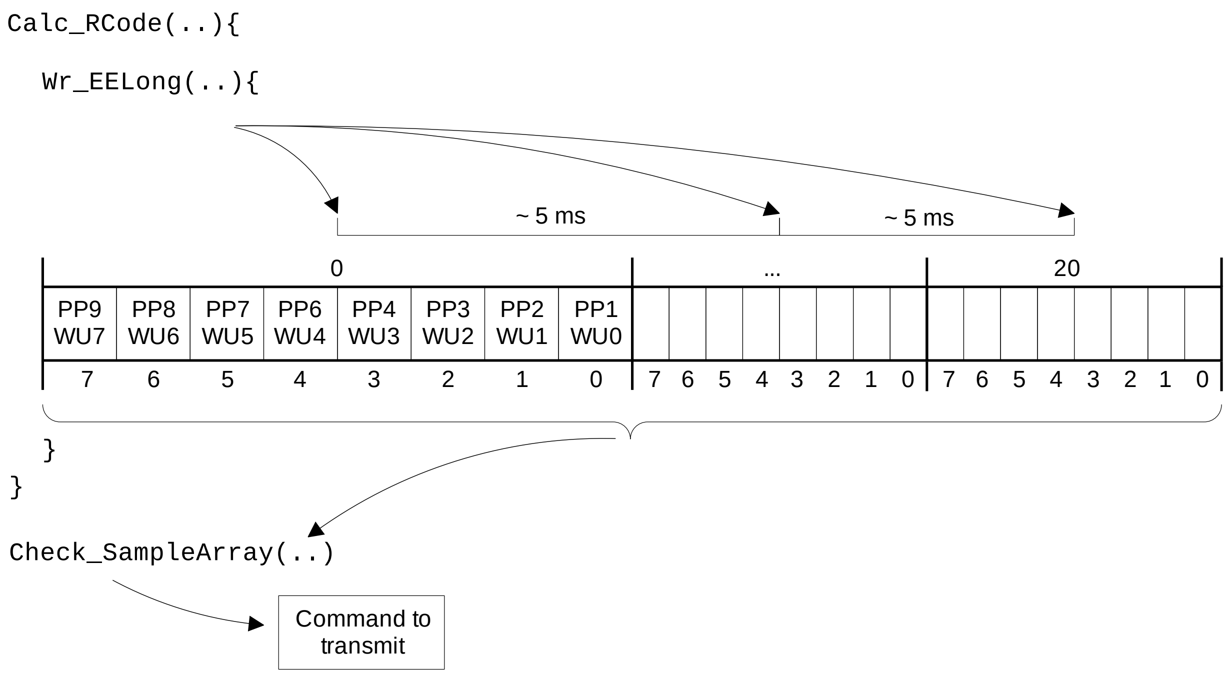


Figure 4 Port sampling during emulated EEPROM write access

6 RF-Protocol

The PMAfob Software Example is designed to be compatible with TDA523x receivers. Therefore a special RF-Protocol has to be used.

The following settings are used for RF-Transmission (set in *RFInit()*, see *RF_Functions.c*):

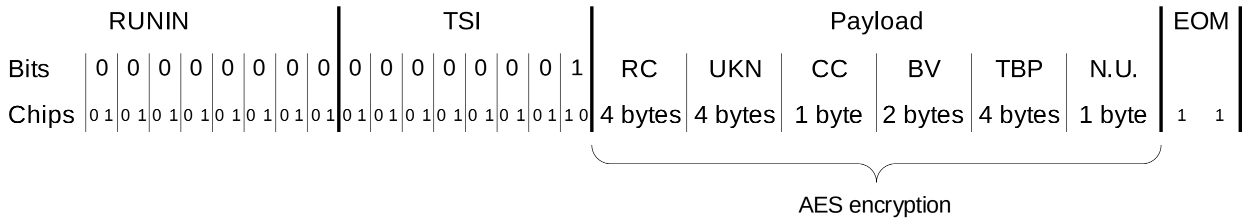
- **Encoding:** Manchester
- **Modulation:** FSK
- **Baudrate:** 9600 bps if encryption is switched on, 4800 bps if encryption is switched off
- **Frequency:** 434 MHz

The RF-Frame starts with a RUNIN sequence of 8 manchester coded data bits (16 chips) which are used by the TDA523x receiver for internal filter setting and frequency adjustment. Then the 16 chips long TSI (Telegram Start Identifier) follows, which is used to synchronize the frame and detect the exact start of a data frame (payload). To detect the EOM (End of Message) a manchester violation (two 1_B chips) is sent.

The payload depends on the setting of the global variable *My_Encryption_Type* defined in file *main.c*.

6.1 Payload for AES Framing

The AES Framing payload consists of 128 bits and includes the rolling code, a unique key number, a command code, the battery voltage and the total button presses. The whole RF-Frame including the (AES encrypted) payload is shown in **Figure 5 on Page 14**.



RUNIN .. Run in sequence(synchronisation)

TSI .. Telegram Start Identifier

RC .. Rolling Code

UKN .. Unique Key Number

CC .. Command Code

BV .. Battery Voltage

TBP .. Total Button Presses

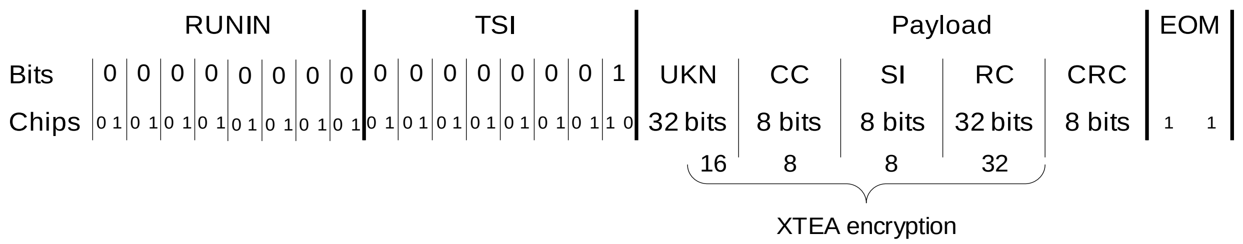
N.U. .. Not Used

EOM .. End of Message

Figure 5 RF-Frame with AES encrypted payload

6.2 Payload for XTEA Framing

64 bits of the 88 bit payload can be encrypted with XTEA. The payload includes the unique key number, the command code, the status information (battery voltage), the rolling code and a CRC checksum. **Figure 6 on Page 14** shows the whole RF-Frame including the payload and describes which part of the payload can be encrypted with XTEA.



RUNIN .. Run in sequence(synchronisation)

TSI .. Telegram Start Identifier

UKN .. Unique Key Number

CC .. Command Code

SI .. Status Information (e.g. Battery voltage)

RC .. Rolling Code

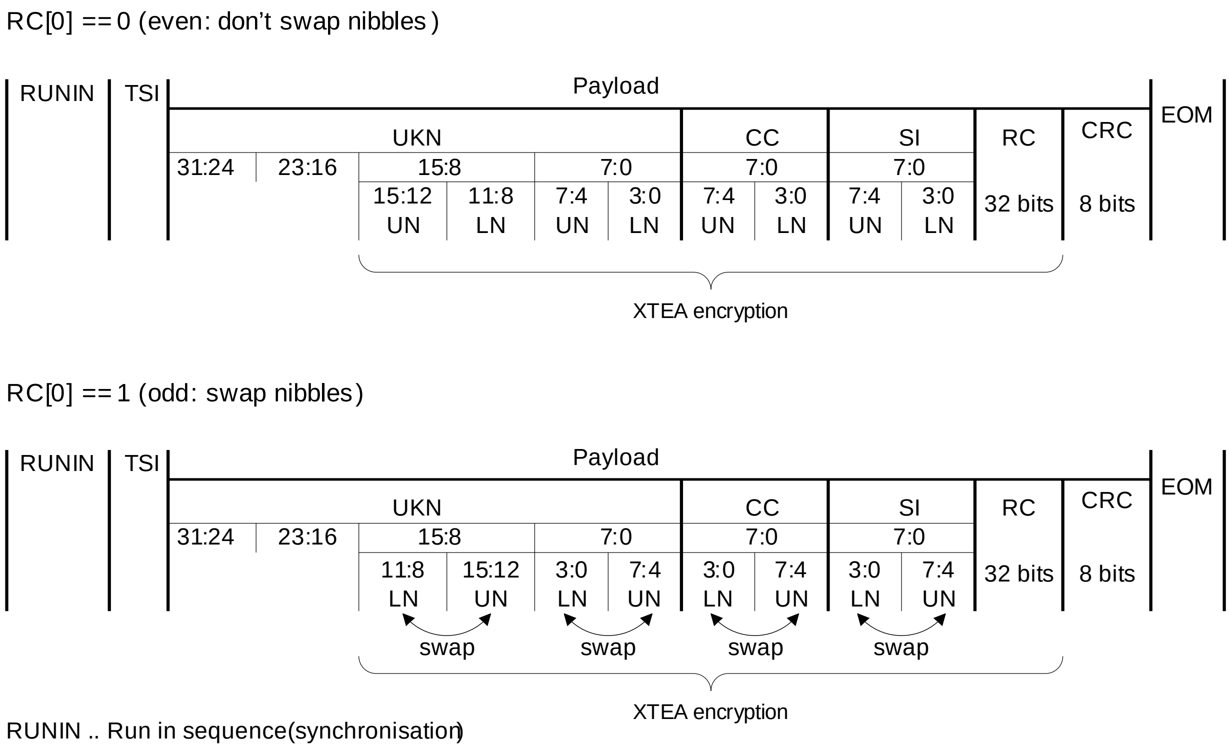
CRC .. checksum over UKN, CC, SI, RC

EOM .. End of Message

Figure 6 RF-Frame with XTEA encrypted payload

6.2.1 Nibble swapping

To increase security the nibbles of the lower 2 bytes of the unique key number, the command code and the status information byte are swapped according to the LSB of the rolling code if encryption is switched on. If the LSB is 1_B the nibbles are swapped. The nibble swapping is shown in **Figure 7 on Page 15**.



RUNIN .. Run in sequence(synchronisation)

TSI .. Telegram Start Identifier

UKN .. Unique Key Number

CC .. Command Code

SI .. Status Information (e.g. Battery voltage)

RC .. Rolling Code

CRC .. checksum over UKN, CC, SI, RC

EOM .. End of Message

LN .. Lower Nibble

UN .. Upper Nibble

Figure 7 Nibble swapping according to the LSB of the rolling code

7 Flexible Software

The software of the PMAfob Software Example can be configured by long button presses without reflashing the device. Therefore one of the GPIOs PP1-PP4 or PP6 has to be connected to GND. 5 buttons (GPIOs) are used to change the RF-Framing, to switch on / off the encryption, to change the datarate and to use 4 byte of the PMA unique ID or the user defined unique ID.

The #define *SWITCH_DUR* (defines.h) is used to set the duration of a button press for configuring the software. The default value of the switch time is about 3 s. The different configurations and the assignments to the PMAfob used in the PMAfob Home Automation and PMAfob RKE Demo are shown in this chapter.

7.1 Reset configuration

After reset, e.g. battery replacement, the software configuration is as following:

- **Framing**: AES
- **Encryption**: ON
- **Unique ID**: User defined unique ID (defines.h: #define *USER_KEY_NR*)
- **Baudrate:** 9600 bps

7.2 Software configuration and GPIO / button assignments

Table 1 shows the implemented software configurations and the GPIO / button assignments to this configurations.

Table 1 Software configuration and GPIO / button assignments

PMAfob Button	GPIO	Software configuration
	PP1	**Switch encryption on / off**: AES or XTEA encryption, depends on the selected framing, is switched off or on. The following configurations are changed if encryption is turned on / off: • **Encryption ON** – **Baudrate**: 9600 bps – **XTEA Framing:** XTEA encryption is used, nibble swapping is turned on – **AES Framing**: AES encryption is used • **Encryption OFF** – **Baudrate**: 4800 bps – **XTEA Framing**: XTEA encryption is not used, nibble swapping is turned off – **AES Framing**: AES encryption is not used
	PP2	**Choose AES Framining**: The AES Framing as described in **Chapter 6.1** is used.

Table 1 Software configuration and GPIO / button assignments

PMAfob Button	GPIO	Software configuration
	PP4	**Choose XTEA Framing**: The XTEA Framing as described in **Chapter 6.2** is used.
	PP3	**Choose user defined unique id**: The user defined unique id (defines.h: #define *USER_KEY_NR*) is used as unique key number.
	PP6	**Choose PMA unique id**: 4 bytes of the PMA unique id are used as unique key number.

References

[1] PMA51xx Function Library Guide

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG