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

Software Framework
Revision 1.2, 2010-06-23
Preliminary

Sense & Control

Edition 2010-06-23

Published by
Infineon Technologies AG
81726 Munich, Germany

Table of Contents

Revision History: 2010-06-23, Revision 1.2

Previous Revision: 1.1

Page	Subjects (major changes since last revision)
6	This document is valid for software example revision V2.x

List of Figures

1 Introduction

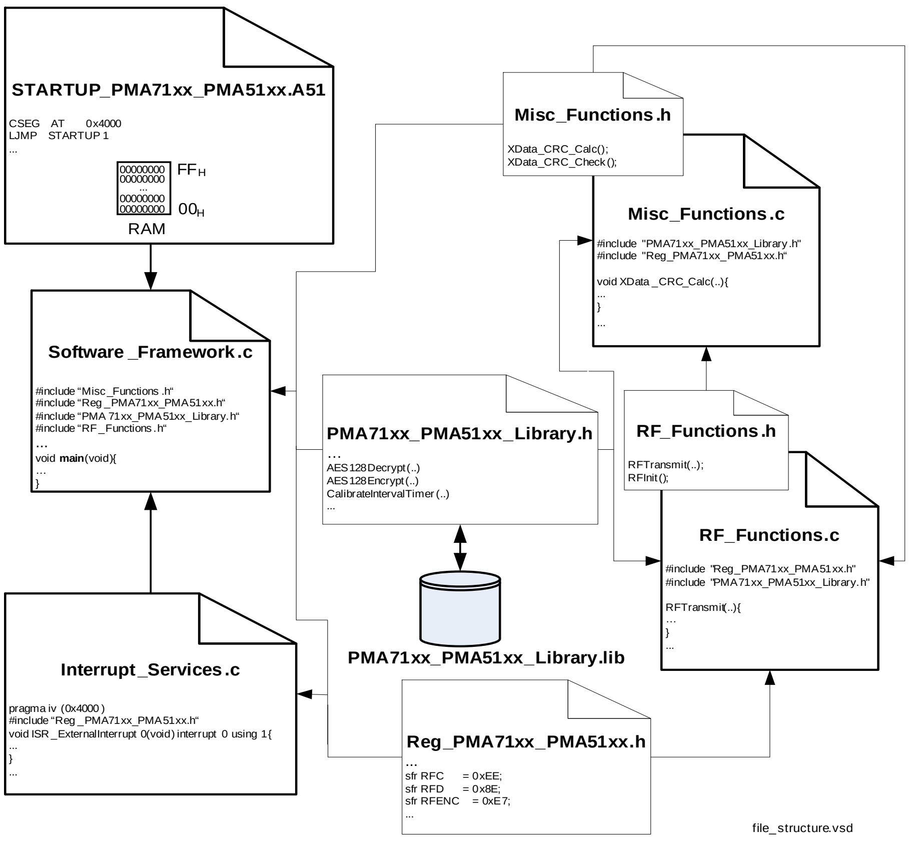
The PMA71xx/PMA51xx Software Framework is an example how software for PMA71xx/PMA51xx can be structured. The available software example files include inline documentation and are developed to enable an easy software development start and fast time to market. The software example files can be downloaded from **http://www.infineon.com/PMA_tooling**. The installer for the source code of the PMA71xx/PMA51xx Software Framework is called *PMA_Software_Framework_V2.x.msi*. The installer is included in the download package of the *PMA Starter Kit* as well as in the download package of the *PMA Evaluation Kit*. Furthermore more documentation of the source code done with doxygen will be copied to the specified location by the *PMA_Software_Framework_V2.x.msi* installer. The default path is *PMA_Software_Framework_V2.x/html*. The documentation can be displayed with a standard browser by opening file *index.html*. The Software Framework has already implemented the following functionality:

- Implementation of PMA71xx/PMA51xx startup file
- Wake-up and reset handling
- Interval Timer usage
- Watchdog Timer handling
- RF-Transmission
- Interrupt handling

This document describes the general file structure, specific features and the recommended program flow and is compatible with source code revision 2.x (*PMA_Software_Framework_V2.x.msi*).

2 File structure

This chapter gives an overview over the structure of the PMA71xx/PMA51xx Software Framework and describes the functionality implemented in the source files. **Figure 1** shows how the files are linked.



```
CSEG   AT      0x4000
LJMP   STARTUP 1
...
```

```
00000000
00000000
...
00000000
00000000
```

```
#include “Misc_Functions .h“
#include “Reg _PMA71xx_PMA51xx.h“
#include “PMA 71xx_PMA51xx_Library. h“
#include “RF_Functions . h“
...
void main(void){
...
}
```

```
XData_CRC_Calc();
XData_CRC_Check ();
```

```
#include  "PMA71xx_PMA51xx_Library .h"
#include  "Reg_PMA71xx_PMA51xx.h“

void XData _CRC_Calc(..){
...
}
...
```

```
...
AES128Decrypt(..)
AES128Encrypt(..)
CalibrateIntervalTimer (..)
...
```

```
RFTransmit(..);
RFInit();
```

```
#include  "Reg_PMA71xx_PMA51xx.h"
#include  "PMA71xx_PMA51xx_Library. h"

RFTransmit(..){
...
}
...
```

```
pragma iv  (0x4000 )
#include “Reg _PMA71xx_PMA51xx.h“
void ISR _ExternalInterrupt  0(void) interrupt  0 using  1{
...
}
...
```

```
...
sfr RFC      = 0xEE;
sfr RFD      = 0x8E;
sfr RFENC    = 0xE7;
...
```

Figure 1 File Structure of PMA71xx/PMA51xx Software Framework

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 Spezial Function Registers (SFRs) 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.

2.1.3 Misc_Functions.h

Interface to functions *Xdata_CRC_Calc()* and *Xdata_CRC_Check()*.

2.1.4 RF_Functions.h

Interface to functions *RFInit()* and *RFTransmit()*.

2.2 Source files

The source files include the implementation of the start up file, the functions defined in the header files, the *main()* function and the interrupt service routines.

2.2.1 STARTUP_PMA71xx_PMA51xx.A51

This file is a modified copy of the standard 8051 startup file *STARTUP.A51* delivered from KEIL^{TM}. 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 initialized and a ljmp to *main()* is executed.

2.2.2 Misc_Functions.c

The functions *Xdata_CRC_Calc()* and *Xdata_CRC_Check()* are implemented within this file. A description of the functionality of each function can be found below.

2.2.2.1 Xdata_CRC_Calc()

Calculates a CRC checksum over xdata content 00_H - $0E_H$ using the Library function *CRC8Calculation()* and stores the CRC checksum into xdata $0F_H$. As the buffer passed to the Library function *CRC8Calculation()* must be in memory space idata, the xdata content has to be copied into a idata buffer.

2.2.2.2 Xdata_CRC_Check()

Checks if the CRC checksum of the xdata content is valid. Therefore the CRC checksum over the whole xdata content 00_H - $0F_H$ is calculated and must be 00_H.

2.2.3 Software_Framework.c

This file includes the *main()* function of the PMA71xx/PMA51xx Software Framework and is executed after *STARTUP_PMA71xx_PMA51xx.A51*. The xdata variables are defined here. They can be located to an absolute position in the xdata memory using the keyword **_at_**, therefore the definition must be global. Additionally some bit definitions are done to increase the readability of the code. In the *main()* function the wake-up bit DSR.1 [WUP] is checked to decide whether the PMA71xx/PMA51xx starts up from reset or with a wake-up event from Power Down Mode.

In case of a reset port 1 and port 3 direction, output and sensitivity registers are set according to the usage of the pins. All port pins, instead of PP2, are set to input and internal pull-down resistors are enabled. PP2 is set to input and the internal pull-up resistor is enabled. Currently all interrupts are disabled but can be easily enabled by setting the global interrupt enable and the appropriate interrupt enable bit. The wake-up mask registers WUM and ExtWUM are used to enable or disable the internal and external wake-up sources. In the Software Framework the internal wake-up for thermal shutdown and the external wake-up WU1 (PP2) is enabled (unmasked). Watchdog - and Interval Timer wake-up are always on in Normal, Programming and Debug Mode. After the wake-up settings the Interval Timer is calibrated. Therefore the system clock is switched to the external crystal clock to achieve higher accuracy. At the end of the reset routine the CRC checksum over the whole xdata content is calculated. If the CRC checksum is not valid, the xdata variables are (re-)initialized to zero.

If the PMA71xx/PMA51xx wakes up from Power Down or Thermal Shutdown State, the wake-up source is checked. If a wake-up has been detected, the appropriate xdata counting variable is incremented, the CRC checksum over xdata content 00_H - $0E_H$ is calculated and written to xdata memory at $0F_H$. In case of a Watchdog Timer - or thermal shutdown wake-up the PMA71xx/PMA51xx is reset to ensure a defined state of the SFRs. If an Interval Timer - or external WU1 wake-up occurred the Watchdog Timer is reset, the RF-Transmitter is initialized through *RFInit()*, the xdata variables are copied into an idata buffer and this buffer is transmitted over RF by calling *RFTransmit()*. After the transmission the PMA71xx/PMA51xx is set into Power Down Mode to save energy and waits for the next wake-up event. The check for all existing wake-up flags is already implemented. To be able to use one of the currently disabled (masked) wake-up sources, the wake-up source has to be enabled and the wake-up has to be handled.

2.2.4 Interrupt_Services.c

Currently no interrupt is enabled. To be able to use an interrupt the appropriate enable bits in *Software_Framework.c* has to be set. In *Interrupt_Services.c* all interrupt service routines are defined and have to be implemented if an interrupt should be used. There is a lot of inline documentation within this file which describes which bits have to be set to enable every single interrupt.

2.2.5 RF_Functions.c

The functions of *RFInit()* and *RFTransmit()* 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()

The RF-Transmission is done by this function. 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 5**.

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.

3 PMA71xx/PMA51xx MCU specific features

The PMA71xx/PMA51xx MCU supports some specific features that are not standard within 8051 micro controllers and which assist the user with the software development. These are listed below and described in the following chapters:

- PMA71xx/PMA51xx Function Library
- Wake-up and reset
- Watchdog Timer
- Interval Timer

3.1 Function Library

The PMA71xx/PMA51xx Function Library is a set of optimized functions developed by Infineon which make it very easy for the user to develop a software application for the PMA71xx/PMA51xx. A detailed description of every function implemented in the PMA71xx/PMA51xx Function Library can be found in **[1]**. By using a PMA71xx/PMA51xx Function Library function the user does not need to set every single bit in a couple of SFRs to get the intended functionality. To initialize the RF-Transmission for example about 85 bits in 11 SFRs have to be set. This can be simply done by only one PMA71xx/PMA51xx Function Library function, *InitRF*(..), as shown in below example.

3.1.1 Code Example

To set up the RF-Transmitter a variable of struct *RF_Config*, which is defined in *PMA71xx_PMA51xx_Library.h*, is initialized and the PMA71xx/PMA51xx Function Library function *InitRF(..)* is called using this variable as parameter.

```
// declare and initialize a variable of struct RF_Config
struct RF_Config idata myRF_Config;
myRF_Config.Quiescent = 0;
myRF_Config.IntMask = 0;
myRF_Config.DutyControl = DUTY_OFF;
myRF_Config.XcapShort = DISABLE;
myRF_Config.Encoding = MANCHESTER;
myRF_Config.dataLength = 7;
myRF_Config.DutyCycle = DUTY_27;
myRF_Config.Modulation = ASK;
myRF_Config.Invert = NO_INVERT;
myRF_Config.Frequency = RF_315MHZ;
myRF_Config.OutStages = STAGE_3;
myRF_Config.xtal0 = 0x28;
myRF_Config.xtal1 = 0x11;
myRF_Config.baudrate = 9600;

// call Library function InitRF(..)
InitRF(&myRF_Config);
```

3.2 Wake-up and reset

After wake-up or reset the PMA71xx/PMA51xx starts program execution at code address 4000_H. To decide whether a wake-up or a reset has occurred the following code example could be used.

3.2.1 Code Example

The SFR bit DSR.1 [WUP] signals if there was a wake-up. If this bit is not set the PMA71xx/PMA51xx must has started up from reset. The PMA71xx/PMA51xx Function Library function *PowerDown*() sets the PMA71xx/PMA51xx into Power Down Mode to save energy.

```
// Include PMA71xx/PMA51xx Function Library
#include "PMA71xx_PMA51xx_Library.h"
// Include the PMA71xx/PMA51xx register file (DSR is defined here)
```

```
#include "Reg_PMA71xx_PMA51xx.h"
#define BIT_WUP 0x02

void main(void){
          // Decision: reset or wake-up entry:
          // ================================
          if ((DSR & BIT_WUP)== 0) {
                    // Reset entry:
                    // ============
          }
          else {
                    // Wake-up entry:
                    // =============
          }
          // Set PMA71xx/PMA51xx into Power Down Mode (Function Library call)
          PowerDown();
}
```

3.3 Watchdog Timer

For operation security a Watchdog Timer is available. The Watchdog Timer represents an up-counter that is started with value 0 and counted up by the 2kHz RC-LP clock. If the Watchdog Timer overflows after approximately 1 s, it causes the wake-up unit to generate a reset pulse in order to reset the system. Thus infinite software loops caused by an unstable system are broken up.

The Watchdog Timer is automatically reset at wake-up- or Idle events.

Note: The up-counter has to be reset periodically by setting the SFR bit CFG2.1 [WDRES] to avoid the Watchdog Timer reset event. The Watchdog Timer mask bit WUM.7 [WDog] is only applicable in Test Mode. In Normal Mode, the Watchdog Timer cannot be disabled. In Programming and Debug Mode, the firmware handler resets the Watchdog Timer in its main polling loop (polling for I^{2}C commands).

3.4 Interval Timer

The Interval Timer is responsible to wake up the PMA71xx/PMA51xx from POWER DOWN state after a specified time interval. In Normal Mode the Interval Timer and the corresponding wake-up is always enabled. As shown in **Chapter 3.4.1** the Interval Timer can be configured easily using the PMA71xx/PMA51xx Function Library function *CalibrateIntervalTimer(..)*.

3.4.1 Code Example

In the reset entry the Interval Timer is calibrated with the PMA71xx/PMA51xx Function Library function *CalibrateIntervalTimer(..)*. To get higher accuracy an external crystal oscillator shall be used.

In the wake-up entry the wake-up source is checked for Interval Timer wake-up. The handling of the Interval Timer wake-up has to be implemented.

The PMA71xx/PMA51xx Function Library function *PowerDown*() sets the PMA71xx/PMA51xx into Power Down Mode to save energy.

```
// Include PMA71xx/PMA51xx Function Library
#include "PMA71xx_PMA51xx_Library.h"
// Include the PMA71xx/PMA51xx register file (DSR is defined here)
#include "Reg_PMA71xx_PMA51xx.h"
#define BIT_WUP 0x02
#define BIT_IT  0x01

void main(void){
          unsigned long idata Xtal_Frequency_2 = 9040000;
          unsigned char WU_Frequency = 10;

          // Decision: reset or wake-up entry:
          // ================================
          if ((DSR & BIT_WUP)== 0) {
                    // Reset entry:
                    // ============
                    // Interval Timer configuration:
                    // ============================
                    // Dependent on accuracy criteria as well as time and voltage drift
                    // recalibration is recommended.

                    // Switch from 12MHz Hi RC oscillator to external crystal
                    // to get higher accuracy. (Function Library call)
                    Switch2XTAL();

                    // Interval Timer is calibrated to wake up the PMA71xx/PMA51xx every 100 ms (f = 10Hz)
                    // (Function Library call)
                    CalibrateIntervalTimer(WU_Frequency, &Xtal_Frequency_2);
          }
          else {
                    // Wake-up entry:
                    // =============

                    // Interval Timer wake-up (be careful WUF is clear on read!)
                    if (WUF & BIT_IT) {
                            // .. handle Interval Timer wake-up
                    }
          }
          // Set PMA71xx/PMA51xx into Power Down Mode (Function Library call)
          PowerDown();
}
```

4 Program flow

Figure 2 on Page 12 shows a flowchart of the Software Framework.

The PMA71xx/PMA51xx 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 PMA71xx/PMA51xx starts up from a reset or because of a wake-up event.

If PMA71xx/PMA51xx starts up from reset all used modules are initialized by the corresponding SFR. To be able to use the external wake-up WU1 Pin PP2 is configured as input and the appropriate wake-up is enabled. Currently all interrupts are disabled and can be easily activated by enabling the global and the according interrupt. The Interval Timer is calibrated and a CRC checksum is calculated over the XRAM. This checksum is used to control if the XRAM data is still valid after a reset triggered by the Watchdog Timer or a thermal shutdown. Finally 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. Four wake-up sources are handled by the Software Framework. This are the Watchdog Timer -, the thermal shutdown -, the Interval Timer -, and the external WU1 wake-up. A counting variable is stored in the XRAM for every implemented wake-up. These variables are incremented in the appropriate wake-up service routine and a CRC checksum is calculated. After Watchdog Timer - and thermal shutdown wake-up a software reset is triggered to ensure that all SFRs have a predefined state. After Interval Timer - or external WU1 wake-up the RF-Transmitter is initialized, the XRAM content is transmitted via RF and the PMA71xx/PMA51xx is set into Power Down Mode.

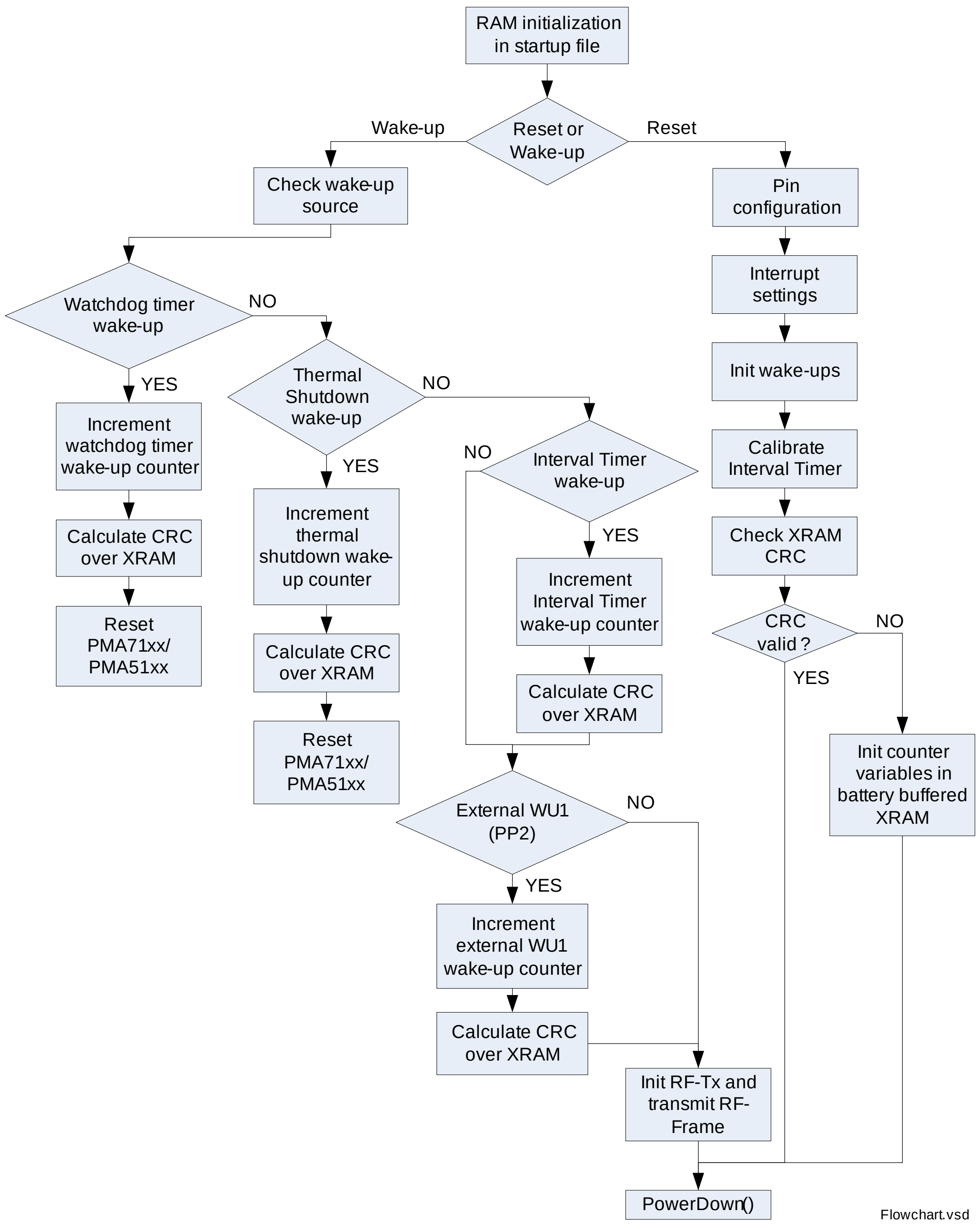


Figure 2 Flowchart of the PMA71xx/PMA51xx Software Framework

5 RF-Protocol

The RF-Transmission of the PMA71xx/PMA51xx Software Framework is designed to be compatible with TDA523x receivers. Therefore a special RF-Protocol has to be used.

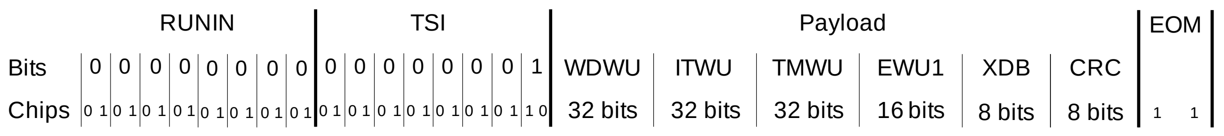
The following settings are used for RF-Transmission (set in *RFInit()*, see *RF_Functions.c*):

- **Encoding:** Manchester
- **Modulation:** FSK
- **Baud rate:** 9600 bps
- **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.

5.1 Payload of the RF-Transmission

The payload of the RF-Transmission is 16 bytes long and consists of simple up counter values for different wake-ups. These wake-ups are Watchdog Timer -, Interval Timer -, thermal shutdown - and external WU1 wake-up. Additionaly there is one byte (*XDB*) from the xdata memory availalbe which is currently not used, but also transmitted via RF. Every time one of these wake-ups occurs the appropriate counter variable is incremented and an RF-Transmission is started. The whole RF-Frame including the TDA523x-Frame and the payload is shown in **Figure 3 on Page 13**.



RUNIN .. Run in sequence (synchronisation)

TSI .. Telegram Start Identifier

WDWU .. Watchdog Timer wake-up counter

ITWU .. Interval Timer wake-up counter

TMWU .. Thermal Shutdown wake-up counter

EWU1 .. External WU1 wake-up counter

XDB .. xdata byte

CRC .. checksum over xdata[0_H:E_H]

EOM .. End of Message

rf_frame.vsd

Figure 3 RF-Frame with TDA523x-Frame and payload

References

[1] PMA Function Library Guide