SIEMENS

Microcontrollers

ApNote AP1658

☐ or ☑ additional file
AP165801.EXE available

Implementation of the power management features with the Keil compiler (SAB C161RI BB)

This Application Note describes the implementation of power management features by means of the C166 Keil compiler (COMPILER C166 V3.05a, Linker L166 V3.05, Obj.-Hex. Converter HEX86 FILE CONVERTER V2.0). The power management makes that microcontroller very useful in power sensitive telecommunication applications.

Author: Wolfgang Boelderl-Ermel / HL COM WN SE

AP1658 ApNote - Revision History		
Actual Revision : Rel.01	Previous Revison: None	
Page of actual Rel.	Page of prev. Rel.	Subjects changes since last release)

1. Introduction

The microcontrollers of the Siemens 16-bit family have been designed to meet the high performance requirements of real-time embedded control applications. The architecture of this family has been optimized for high instruction throughput and minimum response time to external interrupts.

The core of the 16-bit family has been developed with a modular family concept in mind. All family members execute an efficient control-optimized instruction set. This allows an easy and quick implementation of new family members with different internal memory sizes and technologies, different sets of on-chip peripherals and/or different numbers of IO pins.

The C161RI device is further enhanced by a flexible power management. This power management mechanism provides effective means to control the power that is consumed in a certain state of the controller and thus allows the minimization of the overall power consumption with respect to a given application.

To implement these features it can be made use of both assembler programming and high level language programming (Keil C). In this application note the main focus lays on a pure C-language solution. For the assembler solution only hints are given.

2. Implementation

To make use of the power management at the C161RI it is important to know how we use the unlock sequence. The correct execution of this unlock sequence is very important for that feature. To program the power management the registers SYSCON2 and SYSCON3 have to be used.
The power management control registers SYSCON2 and SYSCON3 control functions and modes which are critical for the C161RI's operation. For this reason they are locked (except for bitfield SYSRLS in register SYSCON2) after the execution of EINIT (like register SYSCON) so these vital system functions cannot be changed inadvertently e.g. by software errors.
However, as these registers control the power management they need to be accessed during operation to select the appropriate mode. The system control software gets this access via a special unlock sequence which allows one single write access to either SYSCON2 or SYSCON3 when executed properly. This provides a maximum of security.
Note: Of course SYSCON2 and SYSCON3 may be read at any time without restrictions.
The unlock sequence is executed by writing defined values to bitfield SYSRLS using defined instructions (see table below). The instructions of the unlock sequence (including the intended write access) must be secured with an EXTR instruction (switch to ESFR space and lock interrupts).
Note: The unlock sequence provides no write access to register SYSCON.

The basic routine to unlock the power management consists of the following assembler instructions:

```
EXTR    #xH                 ;Switch to ESFR space and unlock sequence
BFLDL   SYSCON2,#0FH,#09H   ;Unlock sequence, step 1 (1001B)
MOV     SYSCON2,#0003H      ;Unlock sequence, step 2 (0011B)
BSET    SYSCON2.2           ;Unlock sequence, step 3 (0111B)
```

The x in the EXTR instruction indicates the number of those instructions which should be accessed in the ESFR range.

A typical error in executing the unlock sequence occurs if the wrong hardware address is put into the header file, where SYSCON2 is declared. For the C161RI BB step it must be the hardware address F1D0h. (The correct instruction is usually in the header file called reg161RI.h. Syntax: sfr SYSCON2 = 0xF1D0;) This address is located in the ESFR range. Therefore it has to be taken care that the access to this register is correct. Accesses to this address are usually performed by the short address and this short address appears both in the SFR area and in the ESFR area. The C161 will realize the right area by means of the EXTR instruction.

In addition it has to be taken care about the right bit address of the SYSCON2.2 bit. In our case we have declared a name for this bit called UNLOCK. Like the SYSCON2 address it has to be defined in the header file reg161RI.reg. (Syntax: bit UNLOCK = SYSCON2^2; Attention: the syntax of the Tasking compiler could be different here.)

For advanced power management the register SYSCON3 has to be used. A similar code to SYSCON2 has to be implemented in the header file reg161RI.h - again the hardware address has to be checked. (Syntax: sfr SYSCON3 = 0xF1D4;)

To add the unlock sequence to the source code it is now made use of intrinsic functions. These functions are delivered together with the Keil compiler. The declaration of these functions can be found in the header file intrins.h. This file has to be included in the source code. The intrinsic functions help the C-language programmer to use C161RI-specific assembler instructions.

To execute the unlock sequence in the right way it is proposed to use the functions _bfld_() and _nop_(). After the unlock sequence the "power management instruction" will be executed. In our example the CPU has to be slowed down. The code has to be implemented in the following way:

```
_bfld_(SYSCON2,0x000F,0x0009);          /* execute unlock sequence */
SYSCON2   =    0x03;
UNLOCK    =    1;
_bfld_(SYSCON2, 0x7F00, 0x2900);        /* slow down the CPU */
```

Note1: The command SYSCON2 = 0x03; writes only to the bits SYSRLS in the register SYSCON2. (Register SYSCON2 is not unlocked yet.)
Note 2: In the intrinsic function _bfld_() only constant values may be used.

A look at the generated assembler code (use compiler option CD) in the list file shows the following result:

```
Hex code    Assembler instruction
D1B0          EXTR        #04H
0AE80F09      BFLDL       SYSCON2,#0FH,#09H      ;execute unlock sequence
E6E80300      MOV         SYSCON2,#03H
2FE8          BSET        UNLOCK
1AE8297F      BFLDH       SYSCON2,#07FH,#029H    ;slow down CPU executed
```

Note: The command EXTR is added automatically by the compiler. It calculates the correct number of instructions which use ESFR registers.

An advanced technique to avoid problems with the unlock sequence is to initialize the bitfield SYSRLS with 0, before executing the unlock sequence. It is strongly recommended to add the following instructions:

```
bfld_(SYSCON2,0x000F,0x0000); /* initialize bitfield SYSRLS */
_nop_();
```

It is very important to add one _nop_() instruction after that initialization. If it is missed the compiler will be confused and generate the following result (together with unlock sequence and the instruction for slow down the CPU):

```
D1B0        EXTR    #04H
0AE80F00    BFLDL   SYSCON2,#0FH,#00H       ;initialize bitfield SYSRLS
0AE80F09    BFLDL   SYSCON2,#0FH,#09H       ;execute unlock sequence
E6E80300    MOV     SYSCON2,#03H
2FE8        BSET    UNLOCK
D180        EXTR    #01H                    ;!!! error !!!
1AE8297F    BFLDH   SYSCON2,#07FH,#029H     ;slow down CPU not executed
```

This sequence will cause an error which is not obvious at once. The included instruction EXTR #01H interrupts the correct execution of the unlock sequence. So, adding one _nop_() instruction at the right place in the C-source code solves this problem. The right code is listed below:

```
bfld_(SYSCON2,0x000F,0x0000);       /* initialize bitfield SYSRLS */
_nop_();                    /* Do not miss the NOP! */
_bfld_(SYSCON2,0x000F,0x0009);          /* execute unlock sequence */
SYSCON2     =       0x03;
UNLOCK      =       1;
_bfld_(SYSCON2, 0x7F00, 0x2900);   /* slow down the CPU */
```

This C-code will be compiled to:

```
D180        EXTR    #01H
0AE80F00    BFLDL   SYSCON2,#0FH,#00H    ;initialize bitfield SYSRLS
CC00        NOP
D1B0        EXTR    #04H
0AE80F09    BFLDL   SYSCON2,#0FH,#09H    ;execute correct unlock sequence
E6E80300    MOV     SYSCON2,#03H
2FE8        BSET    UNLOCK
1AE8297F    BFLDH   SYSCON2,#07FH,#029H     ;slow down CPU will be executed
```

After executing an instruction dealing with SYSCON2 and SYSCON3 an additional check of the right content of SYSCON2 and SYSCON3 is recommended (See enclosed source code list file), e.g.

```
if ((SYSCON2 & 0x7F00) != 0x0100)
                return 0;
```

The function will return with an error indication if the unlocksequence plus power management instruction have failed.

An alternative to implement the necessary code in high level language is to use in-line assembler. The problem is that the Keil compiler is not very convenient with large projects using in-line assembler.
To add in-line assembler in a source file the compiler option ASM/ENDASM has to be used.

```
#pragma asm
;in-line assembler instructions
#pragma endasm
```

Together with ASM/ENDASM always the option SRC has to be used (see Keil Compiler User´s guide). This option forces the C-compiler to generate only assembler code out of the C source code. This code has to be translated by the assembler separately.

3. Appendix

```
/*****************************************************************************/
/*   File    : pdwn16.C                                                      */
/*   Date    : 31.07.1998                                                    */
/*   Version : V4.3                                                          */
/*                                                                           */
/*   Copyright (c) 1988-1998, SIEMENS AG.  All rights reserved.              */
/*                                                                           */
/*   Purpose  :  Driver functions for the power management                   */
/*               for 16 Bit microcontroller C161RI.                          */
/*               Hint: take care of the stepping code of C161RI.             */
/*               Use only BB steps.                                          */
/*****************************************************************************/

/* ============================== */
/* Includes                       */
/* ============================== */

#include "reg161.h"
#include <intrins.h>

/* ============================== */
/* Local Macros & Definitions     */
/* ============================== */
#define unlocksequence;
_bfld_(SYSCON2,0x000F,0x0000);_nop_();_bfld_(SYSCON2,0x000F,0x0009);SYSCON2
=0x03;UNLOCK=1

/* ============================== */
/* Global function definition     */
/* ============================== */

/*****************************************************************************************/
/*  Function: SDD16_on()                                                                 */
/*  Parms   : unsigned char speed: factor for the slow down divider in this              */
/*            function must be 0, 1, 2, 3, 4, 5, 10, 31                                  */
/*            The speed will be reduced according to the follwing rule:                  */
/*             e.g. 2: from 2+1=3 clock cycles 2 will be dropped => clock/3              */
/*            (0: 0+1=1 clock cycles 0 will be dropped => clock/1                        */
/*             So, the range will be clock/1 .. clock/32                                 */
/*            Attention: always check the code generated by the compiler!!!              */
/*  Purpose : Slow down the C161 RI. Attention: CPU, Peripherals and                     */
/*             Interfaces will slow down. So, change the time base for                   */
/*             timers and for interefaces.                                               */
/*            Returns 1, if the action was successful. Else 0.                           */
/*****************************************************************************************/
```

```
unsigned char SDD16_on(unsigned char speed)
{
        if (speed == 0)
        {
            unlocksequence;

             /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
            _bfld_(SYSCON2, 0x7F00, 0x0100+(0x0400*0));
            if ((SYSCON2 & 0x7F00) != 0x0100)
                return 0;
            return 1;
        };

        if (speed == 1)
        {
            unlocksequence;

             /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
            _bfld_(SYSCON2, 0x7F00, 0x0100+(0x0400*1));
            if ((SYSCON2 & 0x7F00) != 0x0500)
                return 0;
            return 1;
        };

        if (speed == 2)
        {
            unlocksequence;

             /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
            _bfld_(SYSCON2, 0x7F00, 0x0100+(0x0400*2));
            if ((SYSCON2 & 0x7F00) != 0x0900)
                return 0;
            return 1;
        };

        if (speed == 3)
        {
            unlocksequence;

             /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
            _bfld_(SYSCON2, 0x7F00, 0x0100+(0x0400*3));
            if ((SYSCON2 & 0x7F00) != 0x0D00)
                return 0;
            return 1;
        };
```

```
        if (speed == 4)
        {
            unlocksequence;

             /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
            _bfld_(SYSCON2, 0x7F00, 0x0100+(0x0400*4));
            if ((SYSCON2 & 0x7F00) != 0x1100)
                return 0;
            return 1;
        };

        if (speed == 5)
        {
            unlocksequence;

             /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
            _bfld_(SYSCON2, 0x7F00, 0x0100+(0x0400*5));
            if ((SYSCON2 & 0x7F00) != 0x1500)
                return 0;
            return 1;
        };

        if (speed == 10)
        {
            unlocksequence;

             /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
            _bfld_(SYSCON2, 0x7F00, 0x0100+(0x0400*10));
            if ((SYSCON2 & 0x7F00) != 0x2900)
                return 0;
            return 1;
        };

        if (speed == 31)
        {
            unlocksequence;

             /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
            _bfld_(SYSCON2, 0x7F00, 0x0100+(0x0400*31));
            if ((SYSCON2 & 0x7F00) != 0x7D00)
                return 0;
            return 1;
        };
        return 0;

}
```

```
/*****************************************************************************************/
/*  Function: SDD16_off()                                                                */
/*  Parms   : none                                                                       */
/*  Purpose : Switch off the slow down divider of the C161 RI.      */
/*            Attention: CPU, Peripherals and Interfaces will change their      */
/*            speed. So, change the time base for timers and for            */
/*            interefaces.                                                      */
/*           Attention: always check the code generated by the compiler!!!*/
/*           Returns 1, if the action was successful. Else 0.          */
/*****************************************************************************************/

unsigned char SDD16_off(void)
{
      unlocksequence;

       /* now SYSCON2.0 .. SYSCON2.14 is ready to be written */
      _bfld_(SYSCON2, 0x7F00, 0x0000);
      if ((SYSCON2 & 0x7F00) != 0x0000)
            return 0;
      return 1;
}

/*****************************************************************************************/
/*  Function: ADC16_off()                                                                */
/*  Parms   : none                                                                       */
/*  Purpose : Switch off the ADC of the C161 RI to save system power          */
/*            (power management). The complete ADC can be started again,      */
/*            if the ADC16_on() function is executed. (This means, that    */
/*            an external device which wants to send an signal to be        */
/*            AD-converted has firstly to call by an interrupt.             */
/*            The interrupt service has to call that ADC16_on() function.       */
/*            Be aware of time critical analog signals!)                    */
/*           Returns 1, if the action was successful. Else 0.          */
/*****************************************************************************************/

unsigned char ADC16_off(void)
{
      unlocksequence;

       /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
      _bfld_(SYSCON3, 0x0001, 0x0001);
      if ((SYSCON3 & 0x0001) != 0x0001)
            return 0;
      return 1;
}

/*****************************************************************************************/
```

```
/*  Function: ADC16_on()                                                */
/*  Parms   : none                                                      */
/*  Purpose : Switch on the ADC of the C161 RI (power management).      */
/*            (see ADC16_off)                                           */
/*          Returns 1, if the action was successful. Else 0.            */
/*******************************************************************************************/

unsigned char ADC16_on(void)
{
      unlocksequence;

       /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
      _bfld_(SYSCON3, 0x0001, 0x0000);
      if ((SYSCON3 & 0x0001) != 0x0000)
            return 0;
      return 1;
}

/*******************************************************************************************/
/*  Function: ASC016_off()                                              */
/*  Parms   : none                                                      */
/*  Purpose : Switch off the ASC0 of the C161 RI to save system power   */
/*            (power management). The complete ASC0 can be started again,  */
/*            if the ASC016_on() function is executed. (This means, that  */
/*            an incoming byte may only be received, if the ASC0 is     */
/*            switched on fast enough by an interrupt handler.)         */
/*          Returns 1, if the action was successful. Else 0.            */
/*******************************************************************************************/

unsigned char ASC016_off(void)
{
      unlocksequence;

       /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
      _bfld_(SYSCON3, 0x0002, 0x0002);
      if ((SYSCON3 & 0x0002) != 0x0002)
            return 0;
      return 1;
}

/*******************************************************************************************/
/*  Function: ASC016_on()                                               */
/*  Parms   : none                                                      */
/*  Purpose : Switch on the ASC0 of the C161 RI (power management). */
/*            (see ASC016_off)                                          */
/*          Returns 1, if the action was successful. Else 0.            */
/*******************************************************************************************/
```

```
unsigned char ASC016_on(void)
{
     unlocksequence;

      /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
     _bfld_(SYSCON3, 0x0002, 0x0000);
     if ((SYSCON3 & 0x0002) != 0x0000)
          return 0;
     return 1;
}

/*****************************************************************************/
/*  Function: SSC16_off()                                                    */
/*  Parms   : none                                                           */
/*  Purpose : Switch off the SSC of the C161 RI to save system power */
/*            (power management). The complete SSC can be started again, */
/*            if the SSC16_on() function is executed. (This means, that      */
/*            an incoming byte may only be received, if the SSC0 is      */
/*            switched on fast enough by an interrupt handler.)              */
/*          Returns 1, if the action was successful. Else 0.          */
/*****************************************************************************/

unsigned char SSC16_off(void)
{
     unlocksequence;

      /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
     _bfld_(SYSCON3, 0x0004, 0x0004);
     if ((SYSCON3 & 0x0004) != 0x0004)
          return 0;
     return 1;
}

/*****************************************************************************/
/*  Function: SSC16_on()                                                     */
/*  Parms   : none                                                           */
/*  Purpose : Switch on the SSC of the C161 RI (power management).     */
/*            (see SSC16_off)                                                */
/*          Returns 1, if the action was successful. Else 0.          */
/*****************************************************************************/

unsigned char SSC16_on(void)
{
     unlocksequence;

      /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
```

```
    _bfld_(SYSCON3, 0x0004, 0x0000);
    if ((SYSCON3 & 0x0004) != 0x0000)
        return 0;
    return 1;
}

/*****************************************************************************/
/*  Function: GPT16_off()                                                    */
/*  Parms   : none                                                           */
/*  Purpose : Switch off the GPT (Timer Unit) of the C161 RI to save         */
/*            system power (power management)                                */
/*          Returns 1, if the action was successful. Else 0.                 */
/*****************************************************************************/

unsigned char GPT16_off(void)
{
    unlocksequence;

     /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
    _bfld_(SYSCON3, 0x0008, 0x0008);
    if ((SYSCON3 & 0x0008) != 0x0008)
        return 0;
    return 1;
}

/*****************************************************************************/
/*  Function: GPT16_on()                                                     */
/*  Parms   : none                                                           */
/*  Purpose : Switch on the GPT (Timer Unit) of the C161 RI                  */
/*            (power management). (see GPT16_off)                            */
/*          Returns 1, if the action was successful. Else 0.                 */
/*****************************************************************************/

unsigned char GPT16_on(void)
{
    unlocksequence;

     /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
    _bfld_(SYSCON3, 0x0008, 0x0000);
    if ((SYSCON3 & 0x0008) != 0x0000)
        return 0;
    return 1;
}

/*****************************************************************************/
/*  Function: I2C16_off()                                                    */
/*  Parms   : none                                                           */
```

```
/*  Purpose : Switch off the I2C of the C161 RI to save system power       */
/*            (power management). The complete I2C can be started again,   */
/*            if the I2C16_on() function is executed. (This means, that    */
/*            an incoming byte may only be received, if the I2C is         */
/*            switched on fast enough by an interrupt handler.)            */
/*          Returns 1, if the action was successful. Else 0.           */
/*****************************************************************************/

unsigned char I2C16_off(void)
{
      unlocksequence;

       /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
      _bfld_(SYSCON3, 0x0800, 0x0800);
      if ((SYSCON3 & 0x0800) != 0x0800)
            return 0;
      return 1;
}

/*****************************************************************************************/
/*  Function: I2C16_on()                                                    */
/*  Parms   : none                                                      */
/*  Purpose : Switch on the I2C of the C161 RI (power management).      */
/*            (see I2C16_off)                                           */
/*          Returns 1, if the action was successful. Else 0.           */
/*****************************************************************************************/

unsigned char I2C16_on(void)
{
      unlocksequence;

       /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
      _bfld_(SYSCON3, 0x0800, 0x0000);
      if ((SYSCON3 & 0x0008) != 0x0000)
            return 0;
      return 1;
}

/*****************************************************************************************/
/*  Function: PCD16_off()                                                   */
/*  Parms   : none                                                      */
/*  Purpose : Switch off the PCD (Peripheral Clock) of the C161 RI    */
/*            to save system power (power management)                   */
/*          Returns 1, if the action was successful. Else 0.           */
/*****************************************************************************************/

unsigned char PCD16_off(void)
```

```
{
      unlocksequence;

       /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
      _bfld_(SYSCON3, 0x8000, 0x8000);
      if ((SYSCON3 & 0x8000) != 0x8000)
            return 0;
      return 1;
}

/*****************************************************************************/
/*  Function: PCD16_on()                                                     */
/*  Parms   : none                                                           */
/*  Purpose : Switch on the PCD (Peripheral Clock) of the C161 RI    */
/*            (power management). (see PCD16_off)                            */
/*          Returns 1, if the action was successful. Else 0.           */
/*****************************************************************************/

unsigned char PCD16_on(void)
{
      unlocksequence;

       /* now SYSCON3.0 .. SYSCON3.15 is ready to be written */
      _bfld_(SYSCON3, 0x8000, 0x8000);
      if ((SYSCON3 & 0x8000) != 0x8000)
            return 0;
      return 1;
}
/*****************************************************************************/
/*    File     :  PDWN16.H                                                   */
/*    Date     :  31.07.1998                                                 */
/*    Version  :  V4.3                                                       */
/*                                                                           */
/*    Copyright (c) 1988-1998, SIEMENS AG.  All rights reserved.      */
/*                                                                           */
/*    Purpose  :  Headerfile for PDWN16.C                                    */
/*****************************************************************************/
/* ============================== */
/* Extern function declaration        */
/* ============================== */

#ifndef _H_POWER
#define _H_POWER
#include "reg161RI.h"

extern unsigned char SDD16_on(unsigned char speed);
extern unsigned char SDD16_off(void);
```

```
extern unsigned char ADC16_off(void);
extern unsigned char ADC16_on(void);
extern unsigned char ASC016_off(void);
extern unsigned char ASC016_on(void);
extern unsigned char SSC16_off(void);
extern unsigned char SSC16_on(void);
extern unsigned char GPT16_off(void);
extern unsigned char GPT16_on(void);
extern unsigned char I2C16_off(void);
extern unsigned char I2C16_on(void);
extern unsigned char PCD16_off(void);
extern unsigned char PCD16_on(void);
#endif
```