AN205557

F²MC-16FX Family, Reset

This application note describes the functionality of the Reset circuit, Reset types and gives some hardware and software examples.

Contents

1 Introduction....1
1.1 Key Features....1
2 Reset....1
2.1 Outline....1
2.2 Mode Pins and Reset....2
2.3 Registers....2
2.4 Clock Stop Detection and Reset....4
2.5 Low Voltage Detection and Reset....4
3 Electrical Considerations....5
3.1 Power-On Voltage....5
4 Reset Examples....5
4.1 External Pin Circuit....5
4.2 Software Reset....6
4.3 Clock Stop Detection Reset....7
5 Additional Information....10
6 Document History....11

1 Introduction

This application note describes the functionality of the Reset circuit, Reset types and gives some hardware and software examples.

1.1 Key Features

- Reset of MCU by external Hardware and Software
- Internal Hardware Resets: Watchdog, Clock Stop Detection and Low Voltage Detection
- Reset cause readable in dedicated Register

2 Reset

The Basic Functionality Of The Reset

2.1 Outline

The Reset is the highest priority “interrupt”. It has its own vector and cannot be masked.

There are several types of Reset available:

- Power (on/low voltage) Reset
- External Reset (`RSTX`)
- Clock Stop Detection Reset
- Software Reset
- Watchdog Timer Reset
- Low Voltage Detection Reset

2.2 Mode Pins and Reset

After reset the internal boot ROM program is executed which reads out the states of the mode pins (MD2-MD0) of the MCU. Depending on the mode pins setting the MCU activates the appropriate operation mode.

The following table shows the different operation modes resulting from the pin setting:

Table 1. Mode Pin Settings

Mode Pin Setting			Operation Mode	User Program Start Vector Location	Configuration of external Bus for Start Vector Read Access	Remarks
MD2	MD1	MD0				
0	0	0	External Vector Mode 0	External	8 bit, address/data multiplexed	Available only for MCU with External Bus Interface
0	0	1	External Vector Mode 1	External	16 bit, address/data multiplexed	Available only for MCU with External Bus Interface
0	1	0	Serial Communication Mode	-	-	Used for Serial Flash Programming
0	1	1	Internal Vector Mode (Normal Run Mode)	Internal	-	Available for all Devices
1	0	0	*(Reserved)*	-	-	Test Mode
1	0	1	*(Reserved)*	-	-	Test Mode
1	1	0	External Vector Mode 2	External	8 bit, address/data non-multiplexed	Available only for MCU with Non-multiplexed External Bus Interface
1	1	1	Parallel Flash Programming Mode	-	-	Available only for Flash Devices

2.3 Registers

2.3.1 Reset Configuration Register (RCR)

This register controls the software, low voltage detection and clock stop detection reset.

Table 2. RCR

Bit No.	Name	Explanation	Initial Value	Value	Operation
7	–	-	X	–	Reserved; always write “0”
6	–	-	X	–	Reserved; always write “0”
5	SCSDI[1]	Sub Clock Stop Detection Interval Select	0	0	Sub Clock Stop Detection: 384 - 512 RC Clock Cycles
				1	Sub Clock Stop Detection: 24 - 32 RC Clock Cycles
4	MCSDI	Main Clock Stop Detection Interval Select	0	0	Main Clock Stop Detection: 6 - 8 RC Clock Cycles
				1	Main Clock Stop Detection: 3 - 4 RC Clock Cycles

[1] This bit is not available for all devices of 16FX family.

Bit No.	Name	Explanation	Initial Value	Value	Operation
3	CSDRE	Clock Stop Detection Reset	0	0	Disable Clock Stop Detection Reset
				1	Enable Clock Stop Detection Reset
2	LVDE[1]	Low Voltage Detection Select	1	0	Disable Low Voltage Detection
				1	Enable Low Voltage Detection
1	LVRE[1]	Low Voltage Detection Reset	1	0	Disable Low Voltage Detection Reset
				1	Enable Low Voltage Detection Reset
0	SRSTG	Software Reset Generation	0	0	Write: No Effect Read: always “0”
				1	Write: Generate Software Reset

2.3.2 Reset Cause and Clock Status Register (RCCSR/RCCSRC)

This register is accessible at two different addresses. A read access of RCCSRC at the address 0x40B clears all the bits of the register where as read access of RCCSR at the address 0x40D does not clear any bits of the same. A write access to these is always ignored.

Table 3. RCCSR/RCCSRC

Bit No.	Name	Explanation	Initial Value	Value	Operation
15	SCMF[2]	Sub Clock Missing Flag	X	0	No missing Sub Clock
				1	Missing Sub Clock detected
14	MCMF	Main Clock Missing Flag	X	0	No missing Main Clock
				1	Missing Main Clock detected
13	WRST	Watchdog Timer Reset Cause Bit	X	0	No Watchdog Timer Reset
				1	Watchdog Timer Reset was generated
12	SRST	Software Reset Cause Bit	X	0	No Software Reset
				1	Software Reset was generated
11	SCRST[2]	Sub Clock Stop Detection Reset Cause Bit	X	0	No Sub Clock missing Reset
				1	Sub Clock missing Reset was generated
10	MCRST	Main Clock Stop Detection Reset Cause Bit	X	0	No Main Clock missing Reset
				1	Main Clock missing Reset was generated
9	ERST	External Reset Cause Bit	X	0	No External Reset detected
				1	External Reset detected
8	PRST	Power Reset Cause Bit	X	0	No Power Reset detected
				1	Power Reset detected

[2] This bit is not available for all devices of 16FX family.

2.4 Clock Stop Detection and Reset

The clock stop detection circuitry monitors the Main Clock and Sub Clock. If any of these specified clocks is not available for the configured interval while the clock stop detection reset is enabled and any of these clocks are used for system clocks (CLKS1 or CLKS2) or for the watchdog timer, then the corresponding clock stop detection reset cause bit (bit SCRST or MCRST of RCCSR) gets set and the reset is generated.

In the above situation, if the Main Clock is not used for system clocks (CLKS1 or CLKS2) or for the watchdog timer, then instead of generating clock stop detection reset only corresponding clock missing flag bits (bit MCMF of RCCSR) will get set. Also, if the Sub Clock is not used for system clocks (CLKS1 or CLKS2) or for the watchdog timer, then instead of generating clock stop detection reset only corresponding clock missing flag bits (bit SCMF of RCCSR) will get set.

To enable the clock stop detection reset function, CSDRE bit of RCR register needs to be written with a value of '1'. However, it should be noted that while writing to the CSDRE bit, the System Clock 1 (CLKS1) should be set to RC clock. Also the RC clock should not be disabled when the clock stop detection reset is enabled, otherwise reset is generated. This is because reference and source of clock for the clock stop detection circuitry is the RC Clock.

During the stop mode (since the RC clock is stopped) and during the Main/Sub clock stabilization time (to avoid accidental assertion of reset), the clock stop detection reset function is disabled. That means the clock stop detection reset will not be generated even if the clock is failed while the MCU is in the stop mode or it is waking up from the stop mode (i.e. while Main/Sub clock stabilization time). Hence it should be noted that before entering the stop mode, both CLKS1 and CLKS2 should be changed to RC clock. Once the MCU wakes up from stop mode, CLKS1 and CLKS2 should be restored to the required settings. While this switchover (i.e. during the stabilization time of Main/Sub clock), the failure in the Main/Sub clock can be determined using some pre-defined timeout.

The low voltage reset is enabled by default on all 16FX family MCUs. Depending on the device, it can be disabled. Please check if your device support disabling the low voltage reset in the ordering information in the datasheet.

If the column "Persistent Low Voltage Reset" shows "No" for your device, then the low voltage detection and the low voltage detection reset can be disabled by clearing bits LVDE and LVRE of RCR register respectively. But if the column "Persistent Low Voltage Reset" shows "Yes" for your device, then the low voltage detection and the low voltage detection reset cannot be disabled by any means. This means LVDE and LVRE bits of RCR register can not be cleared.

It should be noted that the *start.asm* file does not handle a failed clock situation. It will be stuck in waiting for the PLL Clock or Main Clock in case of failure in the Main Clock. Hence it is advisable that the user should add the intelligence in *start.asm* to check the RCCSR register determine if all (Main/Sub/PLL) clock are running fine. In case one or all of the clocks are failed then it should jump to a subroutine which reports such failure (may be via display, LED or buzzer). Such subroutine should be inserted before section 6.11 (Wait for clocks to stabilise) in *start.asm*.

2.5 Low Voltage Detection and Reset

If the low voltage detector and low voltage reset both are enabled and the Vcc falls below internal generated reference voltage, then the PRST flag of RCCSR register is set. After recovery of power supply voltage, the power reset extension counter deactivates the power reset signal if the power supply remained stable for at least 700 RC clock cycles.

It should be noted that if low voltage detection is to be disabled (LVDE =0), low voltage reset should also be disabled (LVRE= 0) at the same time or before. If the low voltage reset has to be enabled, the low voltage detector should be enabled and stabilized before.

While going in the standby mode, the low voltage detector and reset functionality should be disabled, if the current drawn by the same is not within the tolerable limit.

3 Electrical Considerations

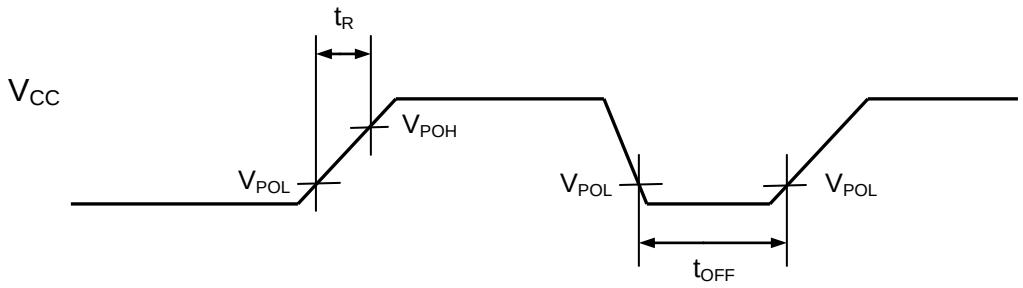
This Chapter Shows Electrical Constraints

3.1 Power-On Voltage

The voltage rise during power-on between V_{POL} and V_{POH} is restricted to a minimum and maximum time (t_R). Also the time between power-down ($V_{CC} < V_{POL}$) is restricted to a minimum time (t_{OFF}).

Please see the corresponding data sheet for correct values.

Figure 1. Power-On and –Off Timing Diagram



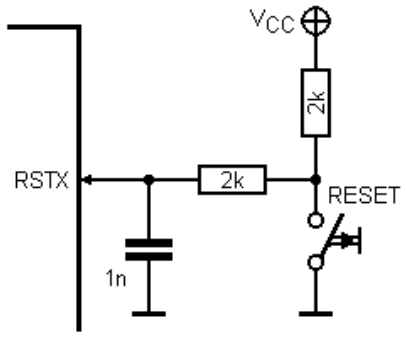
4 Reset Examples

Examples for Reset

4.1 External Pin Circuit

The following example schematic shows a possible external circuit connected to the RSTX pin with a push button.

Figure 2. External Pin Circuit Example Schematic



The 2 kΩ pull up resistor provides a logical “high” level on the RSTX pin, if the button is not pushed. The 1 nF capacitor is used for debouncing and filtering EMI noise. Increasing and decreasing the value by factor 10 may be counterproductive to this. The 2 kΩ series resistor limits current from possible line spikes, over- and under-shots, caused of line inductivity and capacity, which can lead to latch up effects.

4.2 Software Reset

The following sample code shows how to use the Software Reset. The uppermost 4 Bytes of the internal RAM are used to check, how often the code was executed after Reset.

```
void main(void)
{
    unsigned char temp:

        PDR09 = 0x00;
        DDR09 = 0xFF;       /* set Port09 to output 0x00 */


    /* Check first call of Main via RAM pattern               */
    /* => unequal to 0x1234567 after power-on or second call  */

    if (*(unsigned long __far*) (__far) 0x7FFC != 0x1234567)
    {
        /* first call:          */
        /*  => set RAM pattern  */
        *(unsigned long __far*) (__far) 0x7FFC = 0x1234567;

        /* clear all Reset cause bits */
        temp = RCCSRC;

        /* set Software Reset         */
        RCR = 0x01;

        /* no further execution here! */
    }
    else
    {
        /* second call:                        */
        /*  => was it really a Software Reset? */
        if ((RCCSR & 0x10) == 0x10)
        {
            /* Set Port09 to 0x55 and clear RAM pattern */
            PDR09 = 0x55;
            *(unsigned long __far*) (__far) 0x7FFC = 0x00000000ul;
        }
    }
        while(1);
}
```

4.3 Clock Stop Detection Reset

The below software sample code example demonstrates to set the clock stop detection reset. It also specifies how to go in stop mode (while clock stop detection reset is enabled) and resume from the stop mode.

Function `Set_CSDR()` sets the clock stop detection reset. Before doing so it changes the CLKS1 to RC clock.

Function `Set_Clocks()` sets the CLKS1 to Main Clock and CLKS2 to Sub Clock (this can be application specific).

```
#define TIMEOUT  50000          /* User configurable */
void Set_CSDR(void)
{
       while (CKMR_RCM != 1);  /* Check if RC Clock is ready */
       CKSR_SC1S = 0;          /* Set the CLKS1 to RC Clock */
       while (CKMR_SC1M != 0); /* Check if CLKS1 is transitioned to RC Clock */
       RCR_CSDRE =  1;         /* Set Clock Stop Detection Reset */
}
void Set_Clocks(void)
{
       while (CKMR_MCM != 1);   /* Check if Main Clock is ready */
       CKSR_SC1S = 1;           /* Set the CLKS1 to Main Clock */
       while (CKMR_SC1M != 1);  /* Check if CLKS1 transitioned to Main Clock */

       while (CKMR_SCM != 1);  /* Check if Sub Clock is ready */
       CKSR_SC2S = 3;          /* Set the CLKS2 to Sub Clock */
       while (CKMR_SC2M != 3); /* Check if CLKS2 is transitioned to Sub Clock */
}
void Stop_InCSDR(void)
{
       unsigned char clock_stat;
       int i = 0;
        clock_stat = CKSR;      /* Store the clock status */
       CKSR = 0xB0;            /* Set the CLKS1 & CLKS2 to RC Clock */
       while (CKMR_SC1M != 0); /* Check if CLKS1 is transitioned to RC Clock */
       while (CKMR_SC2M != 0); /* Check if CLKS2 is transitioned to RC Clock */
       SMCR = 0x03;            /* Stop Mode */

       while (CKMR_MCM != 1 || i < TIMEOUT ) /* Waked up */
       {                                     /* Wait till Main Clock is Ready */
              __asm(" NOP");                 /* or timeout */
              __asm(" NOP");
                  i++;
       }

       if ( 0x01 == (clock_stat & 0x03))       /* if CLKS1 = CLKMC */
       {
              if (CKMR_MCM != 1)
              {
                     /* Declare timeout of Main clock */
              }
              else
              {
                     CKSR_SC1S = 1;              /* Set the CLKS1 to Main Clock */
                     while (CKMR_SC1M != 1);     /* Check if CLKS1 transitioned to
                                                    Main Clock */
       }
```

Function `Stop_InCSDR()` stores the current status of `CKSR` register, sets CLKS1 and CLKS2 back to RC Clock and goes to stop mode. After the MCU wakes up from the stop mode (via an external interrupt), it restores the clock setting as it was before going to STOP mode. While restoring the clock settings, if any of the clocks (CLKMC/CLKPLL/CLKSC) are not stabilized within the pre-defined time, then timeout is declared.

```
    else if ( 0x02 == (clock_stat & 0x03))       /* if CLKS1 = CLKPLL */
    {
            i = 0;
            while (CKMR_PCM != 1 || i < TIMEOUT )
            {
                    __asm(" NOP");         /* Wait till PLL Clock is Ready */
                    __asm(" NOP");         /* or timeout */
                    i++;
            }

            if (CKMR_PCM != 1)
            {
                    /* Declare timeout of PLL clock */
            }
            else
            {
                    CKSR_SC1S = 2;           /* Set the CLKS1 to PLL Clock */
                    while (CKMR_SC1M != 2); /* Check if CLKS1 is transitioned to
                                               PLL Clock */
    }

    }
    else if ( 0x03 == (clock_stat & 0x03)) /* if CLKS1 = CLKSC */
    {
            i = 0;
            while (CKMR_SCM != 1 || i < TIMEOUT )
            {
                    __asm(" NOP");         /* Wait till Sub Clock is Ready */
                    __asm(" NOP");         /* or timeout */
                    i++;
            }

            if (CKMR_SCM != 1)
            {
                    /* Declare timeout of Sub clock */
            }
            else
            {
                    CKSR_SC1S = 3;           /* Set the CLKS1 to Sub Clock */
                    while (CKMR_SC1M != 3); /* Check if CLKS1 is transitioned to
                                               Sub Clock */
            }

    }

    if ( 0x04 == (clock_stat & 0x0C))     /* if CLKS2 = CLKMC */
    {
            if (CKMR_MCM != 1)
            {
                    /* Declare timeout of Main clock */
            }
            else
            {
                    CKSR_SC2S = 1;           /* Set the CLKS2 to Main Clock */
                    while (CKMR_SC2M != 1); /* Check if CLKS2 transitioned to
                                               Main Clock */
    }
```

```
        else if ( 0x08 == (clock_stat & 0x0C))  /* if CLKS2 = CLKPLL */
        {
                i = 0;
                while (CKMR_PCM != 1 || i < TIMEOUT )
                {
                    __asm(" NOP");   /* Wait till PLL Clock is Ready */
                    __asm(" NOP");   /* or timeout */
                    i++;
                }

                if (CKMR_PCM != 1)
                {
                       /* Declare timeout of PLL clock */
                }
                else
                {
                    CKSR_SC2S = 2;          /* Set the CLKS2 to PLL Clock */
                    while (CKMR_SC2M != 2); /* Check if CLKS2 is transitioned
                                               to PLL Clock */
                }

        }
        else if ( 0x03 == (clock_stat & 0x03))  /* if CLKS2 = CLKSC */
        {
                i = 0;
                while (CKMR_SCM != 1 || i < TIMEOUT )
                {
                    __asm(" NOP");   /* Wait till Sub Clock is Ready */
                    __asm(" NOP");   /* or timeout */
                    i++;
                }

                if (CKMR_SCM != 1)
                {
                       /* Declare timeout of Sub clock */
                }
                else
                {
                    CKSR_SC2S = 3;            /* Set the CLKS2 to Sub Clock */
                    while (CKMR_SC2M != 3);   /* Check if CLKS2 is transitioned
                                                 to Sub Clock */
                }

        }
}
```

```
void main(void)
{
       set_il(7);        /* allow all levels */
       __EI();           /* globally enable interrupts */

       Set_CSDR();      /* Set clock stop detection reset */
       Set_Clocks();    /* Configure CLKS1 and CLKS2 */

       . . .

       . . .

       Stop_InCSDR();   /* Go to stop mode */

}
```

5 Additional Information

Information about Cypress Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software example related to this application note is:

96340_sw_reset

It can be found on the following Internet page:

http://www.cypress.com/16lx

6 Document History

Document Title: AN205557 - F²MC-16FX Family, Reset

Document Number: 002-05557

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	07/17/2006	Initial Release
			03/07/2007	Added information regarding Clock Stop, Low Voltage
			08/03/2007	Added information regarding low voltage option; correct typos
*A	5082091	NOFL	02/29/2016	Migrated Spansion Application Note from MCU-AN-300219-E-V12 to Cypress format
*B	5866748	AESATP12	08/30/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2006-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.