Казанский национальный исследовательский технический университет им. А.Н. Туполева

MSP430 Family

October 23, 2016, Bulat Valeev

Lection 5. Debugging.

Answer for the previous task

Challenges

Reliability of the code

Ways to control code in the MCU

- Simulation
- "LED debug"
- Debug with UART

Watchdog timer

Find a bug in the code

Task in the class

Hometask

Initialization

The resulting initialization code is presented here:

```
int main(void){
Init(); /Initialization
while(1){
        if (flag_1){
                flag_1=0;//do something
                flag_2=1;
        }
        if (flag_2){
                flag_2=0;//do something
                flag_1=1;
        }
}
return;
}
```

Control loop

The resulting timer code is presented here:

```
int main(void){
Init(); /Initialization
while(1){
        if (flag_1){
                flag_1=0;//do something
                flag_2=1;
        }
        if (flag_2){
                flag_2=0;//do something
                flag_1=1;
        }
}
return;
}
```

UART loop

The resulting UART code is presented here:

```
int main(void){
Init(); /Initialization
while(1){
        if (flag_1){
                flag_1=0;//do something
                flag_2=1;
        }
        if (flag_2){
                flag_2=0;//do something
                flag_1=1;
        }
}
return;
}
```

Challenges

What you should know at the end of the day.

- Learn how provide stable and reliable software work
- Learn debug algorithms.
- Understand how control code implementaton on the MCU

Reliability

The code, which you will write, must work predictable and provide necessary tasks.
Hardware errors must not change significantly reliability of the system. Examples of the special cases which will produce problems in the software:

Brown-out without capacitors protection

Pulse noise, with error clock tick

Noised button pin

Forgotten errata notes

What if code is not working?

So your code has bug, what you should do?
Strategy is very simple:

Localize and correct.

How predict error in the code?

Harder task is make reliable code which will avoid all unspecified behaviour in the system.
You must understand all physical dependencies in the hardware and be careful with errara, to predict everything.
The simplest cases are watchdog timer usage, critical strings protection.

Ways to control code in the MCU

There are a lot of ways to find error and control code execution on the chip:

Code analysis

Simulation

On-chip simulation

"LED-debug"

Debug messages by UART

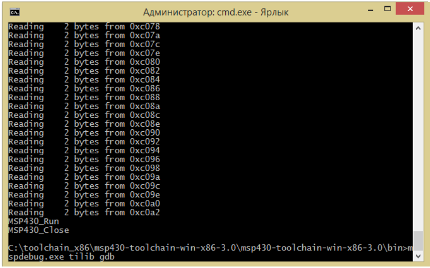
etc.

Simulation

Many IDE's allow to execute code in the laptop and control variables in the chip. This option allows to find error without any hardware tracing. The most usable case for simulation is complicated, expensive system, of system where you have no access to the hardware.
The disadvantages are high complexity, price of the IDE with simulator.

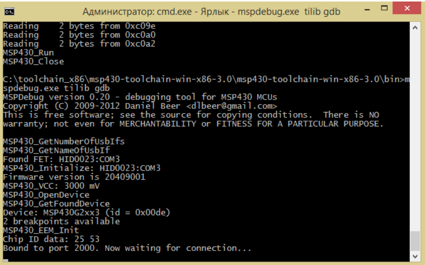
Example eclipse

Here presented example eclipse session with MSP430 MCU.



```
Reading     2 bytes from 0xc078
Reading     2 bytes from 0xc07a
Reading     2 bytes from 0xc07c
Reading     2 bytes from 0xc07e
Reading     2 bytes from 0xc080
Reading     2 bytes from 0xc082
Reading     2 bytes from 0xc084
Reading     2 bytes from 0xc086
Reading     2 bytes from 0xc088
Reading     2 bytes from 0xc08a
Reading     2 bytes from 0xc08c
Reading     2 bytes from 0xc08e
Reading     2 bytes from 0xc090
Reading     2 bytes from 0xc092
Reading     2 bytes from 0xc094
Reading     2 bytes from 0xc096
Reading     2 bytes from 0xc098
Reading     2 bytes from 0xc09a
Reading     2 bytes from 0xc09c
Reading     2 bytes from 0xc09e
Reading     2 bytes from 0xc0a0
Reading     2 bytes from 0xc0a2
MSP430_Run
MSP430_Close

C:\toolchain_x86\msp430-toolchain-win-x86-3.0\msp430-toolchain-win-x86-3.0\bin>m
spdebug.exe tilib gdb
```



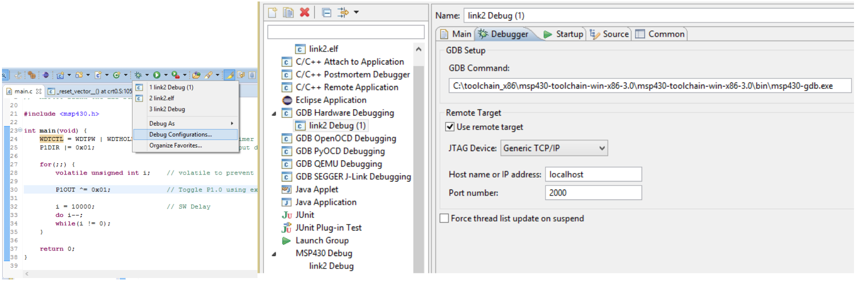
```
Reading     2 bytes from 0xc09e
Reading     2 bytes from 0xc0a0
Reading     2 bytes from 0xc0a2
MSP430_Run
MSP430_Close

C:\toolchain_x86\msp430-toolchain-win-x86-3.0\msp430-toolchain-win-x86-3.0\bin>m
spdebug.exe tilib gdb
MSPDebug version 0.20 - debugging tool for MSP430 MCUs
Copyright (C) 2009-2012 Daniel Beer <dlbeer@gmail.com>
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

MSP430_GetNumberOfUsbIfs
MSP430_GetNameOfUsbIf
Found FET: HID0023:COM3
MSP430_Initialize: HID0023:COM3
Firmware version is 20409001
MSP430_VCC: 3000 mV
MSP430_OpenDevice
MSP430_GetFoundDevice
Device: MSP430G2xx3 (id = 0x00de)
2 breakpoints available
MSP430_EEM_Init
Chip ID data: 25 53
Bound to port 2000. Now waiting for connection...
```

Example eclipse

Here presented example eclipse session with MSP430 MCU.



```
#include <msp430.h>

int main(void) {
    WDTCTL = WDTPW | WDTHOL                        imer
    P1DIR |= 0x01;                                  put d

    for(;;) {
        volatile unsigned int i;    // volatile to prevent

        P1OUT ^= 0x01;              // Toggle P1.0 using ex

        i = 10000;                  // SW Delay
        do i--;
        while(i != 0);
    }

    return 0;
}
```

Name: link2 Debug (1)

GDB Setup

GDB Command:

C:\toolchain_x86\msp430-toolchain-win-x86-3.0\msp430-toolchain-win-x86-3.0\bin\msp430-gdb.exe

Remote Target

☑ Use remote target

JTAG Device: Generic TCP/IP

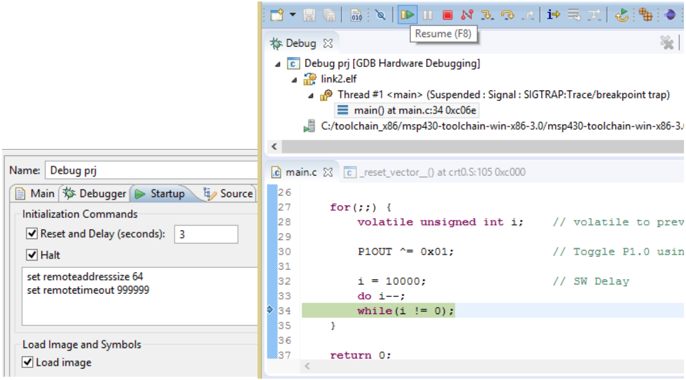
Host name or IP address: localhost

Port number: 2000

□ Force thread list update on suspend

Example eclipse

Here presented example eclipse session with MSP430 MCU.



Debug prj [GDB Hardware Debugging]
link2.elf
Thread #1 <main> (Suspended : Signal : SIGTRAP:Trace/breakpoint trap)
main() at main.c:34 0xc06e
C:/toolchain_x86/msp430-toolchain-win-x86-3.0/msp430-toolchain-win-x86-3.

Name: Debug prj

Initialization Commands

☑ Reset and Delay (seconds): 3

☑ Halt

```
set remoteaddresssize 64
set remotetimeout 999999
```

Load Image and Symbols

☑ Load image

```

    for(;;) {
        volatile unsigned int i;    // volatile to prev

        P1OUT ^= 0x01;              // Toggle P1.0 usin

        i = 10000;                  // SW Delay
        do i--;
        while(i != 0);
    }

    return 0;
```

"LED debug"

"LED debug" is control code execution way with simple LED blink in certain point of the code. Very simple way to find an error. Disadvantages: applicable only for slow, simple algorithms with hardware access and free pins.

Debug with UART

Debug with UART messages is very powerful way to control code execution in the chip.

It is simple, relatively fast way to trace code.

Disadvantages: applicable only with access to hardware.

Watchdog timer

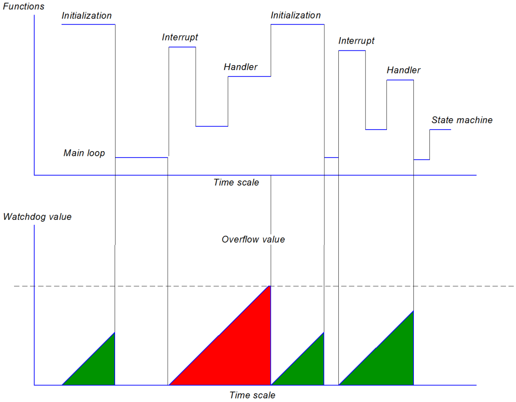
Watchdog is timer which has only one interrupt, and this interrupt lead to the chip reset. The interval between chip reset is relatively high.

Watchdog timer will reset chip automatically in case if there are infinite loops or points, where software will be locked.

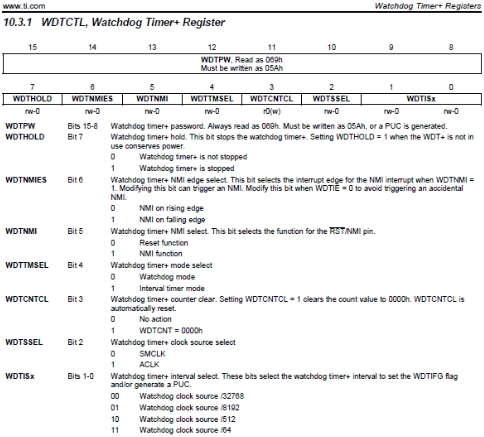
Important notes

You also must block the watchdog interrupt execution in case if code works well.
The code usually has function which execute periodically and clear the watchdog timer without interrupt execution. This function will not executed in case if there is any lock loop in the software. So infinite loop in the code will lead to the watchdog interrupt execution.

Example of the watchdog execution



Important notes



www.ti.com — *Watchdog Timer+ Registers*

10.3.1 WDTCTL, Watchdog Timer+ Register

15	14	13	12	11	10	9	8
WDTPW, Read as 069h Must be written as 05Ah							

7	6	5	4	3	2	1	0
WDTHOLD	**WDTNMIES**	**WDTNMI**	**WDTTMSEL**	**WDTCNTCL**	**WDTSSEL**	**WDTISx**	
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-0	rw-0	rw-0

WDTPW Bits 15-8 Watchdog timer+ password. Always read as 069h. Must be written as 05Ah, or a PUC is generated.

WDTHOLD Bit 7 Watchdog timer+ hold. This bit stops the watchdog timer+. Setting WDTHOLD = 1 when the WDT+ is not in use conserves power.

0 Watchdog timer+ is not stopped

1 Watchdog timer+ is stopped

WDTNMIES Bit 6 Watchdog timer+ NMI edge select. This bit selects the interrupt edge for the NMI interrupt when WDTNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when WDTIE = 0 to avoid triggering an accidental NMI.

0 NMI on rising edge

1 NMI on falling edge

WDTNMI Bit 5 Watchdog timer+ NMI select. This bit selects the function for the $\overline{\text{RST}}$/NMI pin.

0 Reset function

1 NMI function

WDTTMSEL Bit 4 Watchdog timer+ mode select

0 Watchdog mode

1 Interval timer mode

WDTCNTCL Bit 3 Watchdog timer+ counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset.

0 No action

1 WDTCNT = 0000h

WDTSSEL Bit 2 Watchdog timer+ clock source select

0 SMCLK

1 ACLK

WDTISx Bits 1-0 Watchdog timer+ interval select. These bits select the watchdog timer+ interval to set the WDTIFG flag and/or generate a PUC.

00 Watchdog clock source /32768

01 Watchdog clock source /8192

10 Watchdog clock source /512

11 Watchdog clock source /64

Atomic operations

The atomic operations are explained previously. You must use this in each potentially important point in the code.

CRUCIAL: Always use atomic operations.

Find a bug in the code

Find the possible errors in presented here code:

```
int main(void){
Init(); /Initialization
while(1){
        if (flag_1){
                flag_1=0;//do something
                flag_2=1;
        }
        if (flag_2){
                flag_2=0;//do something
                flag_1=1;
        }
}
return;
}
```

Task in the class: Code improvement

Use project lab 5 in the workspace and improve code to trace with UART possible errors in the code execution. Use watchdog timer to make code more reliable.

Result

Here presented example of the result

```
int main(void){
Init(); /Initialization
while(1){
        if (flag_1){
                flag_1=0;//do something
                flag_2=1;
        }
        if (flag_2){
                flag_2=0;//do something
                flag_1=1;
        }
}
return;
}
```

Hometask

Use your ADC project to improve the code with tracing via Led.
Use watchdog timer to make code more reliable.
Deadline is xx.xx.xxxx.

Thanks for your attention

Reference slide

- http://easyelectronics.ru/avr-uchebnyj-kurs-programmirovanie-na-si-atomarnye-operacii.html
- http://easyelectronics.ru/avr-uchebnyj-kurs-operacionnaya-sistema-dispetcher-zadach.html