External interrupts and interrupt priorities

STM8

The external interrupt system in STM8 is quite tricky. The developers gave us the ability to catch interrupts from any pin, but they did not allocate a vector for each pin. As a result, this part of the STM8L (in S- it is somehow better with this) is simply dotted with various crutches and tricks.
Let's figure out how everything is arranged.

In addition to external interrupts, we will also consider setting up interrupt priorities.

We were promised interrupts on all pins. Let's start with them, that is, with the pins.

You can enable or disable an interrupt for a specific pin via the GPIOx_CR2 register (where x is the port for which the interrupt is configured). If the pin is configured as an input, then writing 1 to the corresponding bit of the GPIOx_CR2 register will enable the interrupt for it.

For example, this is how you can configure the button on the STM8L-Discovery (pin C1):

```
GPIOC->DDR |= GPIO_Pin _1; //На вход
GPIOC->CR1 |= GPIO_Pin _1; //Подтягивающий резистор (хотя он тут и не ну
GPIOC->CR2 |= GPIO_Pin _1; //Прерывание разрешено
```

Except for the pin settings, the STM8 external interrupt system looks a little different in different families and lines . First, let's take a closer look at how it is organized in the STM8L15xx. There, the developers' clever tricks reach their utmost concentration, while in other series, everything is arranged more simply. Especially in the STM8S - there, the external interrupt system is very similar to PCINT in the AVR.

The first surprise (for me, after the AVR) is that **there are two groups of interrupts** : port interrupt (EXTIB, EXTID ...) without specifying the pin where interrupts from each port pin arrive (like PCINT); and pin interrupt (EXTI0, EXTI1 ... EXTI7) without specifying the port where interrupts from several ports, but from the same pins, arrive. At first, this is a bit mind-boggling, but then you

Live

Comments Publications

penzet → Sprint Layout in OS X 18 → Software for electronics engineer

Vga → EmBitz 6 → Software for electronics engineer

Vga → Rail-to-rail: ideal operational amplifier or a clever marketing ploy? 1 → Theory, measurements and calculations

Flint → A little more about 1-wire + UART 56 → Hardware connection to the computer.

penzet → Cross-platform terminal - SerIO 3.x 25 → Software for electronics engineer

Gornist → PIP regulator 2 → Algorithms and software solutions

whoim → CC1101, Treatise on Tracing 89 → Blog im. khomin

OlegG → Clock on Bluetooth LE module 8 → Cypress PSoC

Technicum505SU → Nuances of PWM control of a DC motor by a microcontroller 3 → Theory, measurements and calculations

sunjob → DDS synthesizer AD9833 88 → Blog named after grand1987

DIHALT → W801, LCD screen and fly in the ointment 2 → Detail

nictrace → New Arduino-compatible board. 20 → FPGA

sunjob → Connecting the ARM GCC compiler to CLion 1 → Software for electronics engineers

Vga → CRC32: on STM32 as on PC or on PC as on STM32. 58 → STM32

dmitrij999 → Capturing images from a USB camera using STM32 6 → STM32

Gilaks → Expanding the capabilities of a simple MC up to an ADC on 2 or 1 pin. 8 → Theory, measurements and calculations

sva_omsk → Lithium ECAD - Russian PCB CAD 40 → Software for electronics engineer

x893 → W801 - budget controller with Wi-Fi 4 → Details

podkassetnik → Changing the standard instrument cluster lighting of Logan-like cars 5 → Automotive electronics

get used to it.

Each half of the port can be configured to generate 1 interrupt for any pin (EXTID, EXTIG — by port name) or generate a separate interrupt for each pin (EXTI1,EXTI2...). This is configured through the EXTI_CONF1 and 2 registers, or more precisely, through the PxLIS and PxHIS bits in them.

7	6	5	4	3	2	1
PFES	PFLIS	PEHIS	PELIS	PDHIS	PDLIS	PBHI
rw	rw	rw	rw	rw	rw	rw

PxLIS is responsible for the lower 4 bits, and PxHIS — for the higher ones. If the bit is set (1), then this half of the port will generate one interrupt for all pins. And if the bit is cleared (0) — separate interrupts for each pin: EXTI0..3 for the lower half and EXTI4..7 for the higher half.

For some ports, common interrupts (of which there are one per port) are not available . They can only be worked with through pin interrupts. Such deprived ones include ports A and C in STM15x. In STM101, only ports B and D have their own interrupts.

For STM8S, on the contrary, there are only port interrupts, but no EXTI0, EXTI1 . This greatly reduces the configuration options - the edge at which the interrupt is triggered is configured for the entire port. But more on that below.

Since the senior STM8L15x tragically lacks vectors for all ports, switching was invented. For example, **the EXTIB/G vector can process interrupts from port B or G.** The PFES, PHDS, PBGS bits in the EXTI_CONFx registers are responsible for switching vectors between ports. If 1 is written to the PFES bit, the EXTIE/ **F** vector will catch interrupts from port F, otherwise - from port E. The same is true for the PHDS and PBGS bits (they are only in senior models, not on Discovery).

Bit 7 **PFES:** Port F or port E external interrupt select
0: Port E is used for interrupt generation
1: Port F is used for interrupt generation

The edge or level at which the interrupt is triggered is configured through the **EXTI_CRx** registers . There are 4 such registers in STM8L15x, 3 in STM8L101.

7	6	5	4	3	2	1
P3IS[1:0]		P2IS[1:0]		P1IS[1:0]		
rw		rw		rw		

The bits in them are called **PxIS** and are collected in pairs. For example, the pair P0IS[1:0] is responsible for the edge on which the EXTI0 interrupt is triggered. There are four possible options here:

00 — Voltage drop and low level
01 — Transition from low level to high (__/)
10— High to low transition (__)
11 — Any level change

Library fans can write something like this to set up an interrupt:

```
EXTI_SetPinSensitivity(EXTI_Pin_1,EXTI_Trigger_Falling);
```

For example, here's how you can make the button on the STM8L-Discovery (pin C1) generate an EXTI1 interrupt when the voltage drops (i.e. when the log level changes from 1 to 0):

```
GPIOC->DDR |= GPIO_Pin _1; //На вход
GPIOC->CR1 |= GPIO_Pin _1; //Подтягивающий резистор (хотя он тут и не ну
```

```
GPIOC->CR2 |= GPIO_Pin _1; //Прерывание разрешено
EXTI->CR2 |= 1<<3; //Устанавливаем 1 бит в PBIS
```

And this way, the button hanging on B3 will be configured to generate an EXTIB/G interrupt for any change in level:

```
GPIOB->DDR |= GPIO_Pin _3; //На вход
GPIOB->CR2 |= GPIO_Pin _3; //Прерывание разрешено
EXTI->CR3 |= 3; //Устанавливаем 1 и 2 бит в PBIS
EXTI->CONF1 |= EXTI_CONF1_PBLIS; //Переключаем младшую половину порта B
```

Now let's see **how to correctly format an interrupt handler** .
In **IAR,** the handler template looks like this:

```
INTERRUPT_HANDLER(Interrupt_Name, Interrupt_Number)
{
 //Код-код-код-код
 //Не забудь сбросить флаг прерывания перед выходом
}
```

Interrupt_Name is the interrupt name. It can be anything.
Interrupt_Number is the interrupt number in the table. The table in the datasheet is "Interrupt vector mapping"

For example, the EXTI1 interrupt handler, which simply switches the LED to E7:

```
INTERRUPT_HANDLER(EXTI1_IRQHandler, 9)
{
 GPIOE->ODR ^= GPIO_Pin_7;
 EXTI->SR1 |= 1<<1;
}
```

Please note that **before exiting the interrupt, you need to reset its flag** .
Otherwise, it's Groundhog Day - the MC will spin forever in this handler.

The flags are scattered in the **EXTI_SR1** and **EXTI_SR2** registers . **To reset the flag, you need to write 1 to it** . Library lovers can simply write:

```
EXTI_ClearITPendingBit(EXTI_IT_Pin1);
```

In this case, this is what you need to do, because there are no constants for resetting interrupt flags anyway, which leads to the appearance of magic numbers (1<<1) in the code.

These same **flags allow you to determine what caused the interrupt** . In STM8, several interrupt sources per vector are quite common, so you can get lost without flags.

Interrupt priorities

Priorities are a useful feature that was sorely lacking in the AVR — **you don't have to worry about a time-critical interrupt being delayed because another, less important one is being processed.**

Zilmas already raised this topic in one of his articles. So that you don't have to look for hundreds of pieces of information in different places, I'll tell you about priorities.

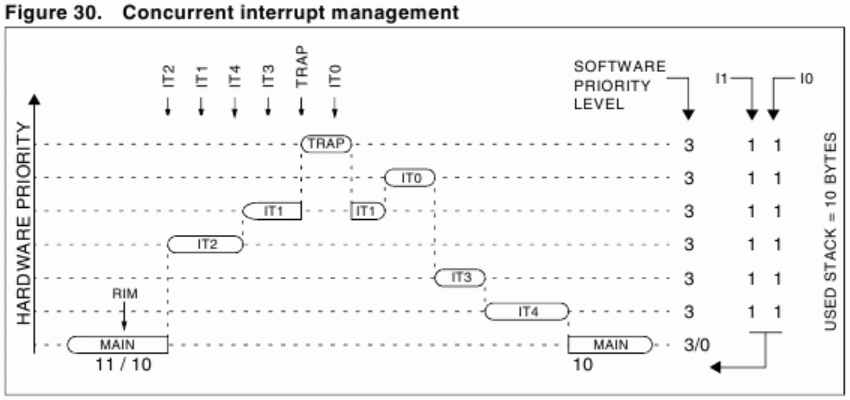
All interrupts have two types of priorities — **software and hardware** . RESET has the highest hardware priority, then TRAP, and so on down the interrupt table. The default software priority is the same for all interrupts. It can be configured via the **ITC_SPRx** registers , of which there are as many as 8 in the STM8L15x. **Two bits are allocated for each interrupt vector** , and priority 00 cannot be set.

Again, one line will be enough for library fans:

```
ITC_SetSoftwarePriority(EXTI2_IRQn, ITC_PriorityLevel_1);
```

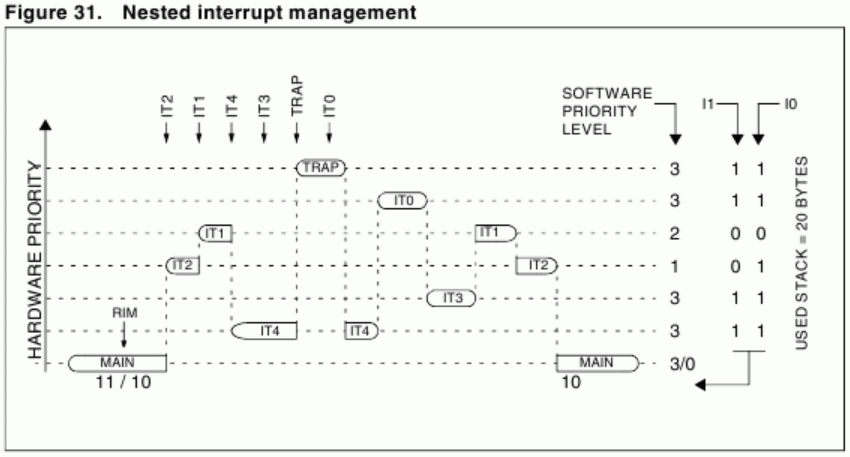
While the software priorities of all interrupts are the same, **collisions between interrupts are resolved on a first come, first served basis** . If other interrupts occur while one is being processed, they will be processed after the first one is finished. The one with the higher hardware priority

Figure 30. Concurrent interrupt management



will be the first to go. As soon as we change the software priority of at least one interrupt, the controller switches to nested interrupts mode. Now **the execution of the handler can be interrupted by another interrupt** if it has a higher software priority. If the software priorities are the same, then the hardware priority comes into play: if the new interrupt has a higher priority, it takes over control.

Figure 31. Nested interrupt management



Software priority 3 is the highest (it is set by default), and **1 is the lowest** (0 cannot be set).

As an example , we will write a program that, when you press the button on pin C3 (you will have to attach it to the discoverer yourself) generates interrupts. In this interrupt, **the blue** LED will blink in an infinite loop (sic!) . In parallel with this chaos, pressing the User button (C1) will switch **the green** LED - this interrupt has a higher priority, so the infinite loop will not interfere with it.

Here's how it looks:

The source code of the program (project for IAR) is attached as usual.

STM8 , interruptions , competition2

+2 August 19, 2011, 7:46 PM **dcoder** 1

Files in topic: EXTI.zip

Comments (9)

RSS Collapse / Expand

Excellent description. Keep it up!!! 0

bdpcvit
August 19, 2011, 11:39 PM

Thank you!
Next up is ADC 0

dcoder
August 19, 2011, 11:47 PM

Come on, write more, I'll soon become an STM specialist too. 0

Oxygen
August 20, 2011, 2:05 PM

Yeah, and for S it doesn't work as expected. And I didn't find a way to reset the interrupt flag. 0

sergeyb2009
August 26, 2011, 08:46

Cool description!
There is a small inaccuracy in the example for setting up the button, probably not EXTI->CR2 |= 1<<3,
but EXTI->CR1 |= 1<<3 0

ingor-ru
July 11, 2012, 4:34 PM

I can't figure out what's going on. Help if anyone knows.
I configure the button on the 8L-Discovery like this:
bres PC_DDR,#1; to the input
bset PC_CR1,#1; enable pull-up
bset PC_CR2,#1; enable interrupts on the button
bset EXTI_CR1,#3; interrupt on falling edge

in interrupt processing I switch the LED like this:

EXTI1_Interrupt.l; External interrupt on the front — button
blink.l
clrw X
loop1.l
incw X
cpw X, #6000
jrult loop1
bcpl PC_ODR,#7; change the state of the LED on the PC7
bset EXTI_SR1,#1; reset the interrupt flag
iret
Assembler from ST. In the end, everything works, but if you press and hold the button, the interrupt
is triggered constantly!!? Why????? Moreover, it doesn't matter whether the interrupt is set up on the front or on the fall, and the input is with a pull-up or floating... The result is the same.
Thanks in advance.

0

ingor-ru
July 11, 2012, 4:46 PM

For me this happens because of the chatter (I looked at it with an oscilloscope)

0

Azzoo
August 29, 2012, 01:13

It's strange, STM also behaves this way if a low-level interrupt is configured.

0

Azzoo
September 21, 2012, 05:21

Can you give more details about resetting the interrupt flag from the external pin?

```
EXTI->SR1 |= 1<<1;
```

— as I understand it, this is in STM8L???

because in STM8S there are only two registers in the EXTI space:
EXTI->CR1
EXTI->CR2
so nothing needs to be reset?

0

Doka
May 20, 2015, 21:50

Only registered and authorized users can leave comments.