EECS 373
Design of Microprocessor-Based Systems

Prabal Dutta
University of Michigan

Lecture 7: Interrupts (2)
September 23, 2014

Some slides prepared by Mark Brehob

Announcements

- Homework 2 due now.
- Homework 3 will be posted later this week.
- Start thinking about projects
- Start planning for "special topics"

High-level review of interrupts

- Why do we need them? Why are the alternatives unacceptable?
  - Convince me!
- What sources of interrupts are there?
  - Hardware and software!
- What makes them difficult to deal with?
  - Interrupt controllers are complex: there is a lot to do!
    - Enable/disable, prioritize, allow premption (nested interrupts), etc.
  - Software issues are non-trivial
    - Can't trash work of task you interrupted
    - Need to be able to restore state
    - Shared data issues are a *real* pain

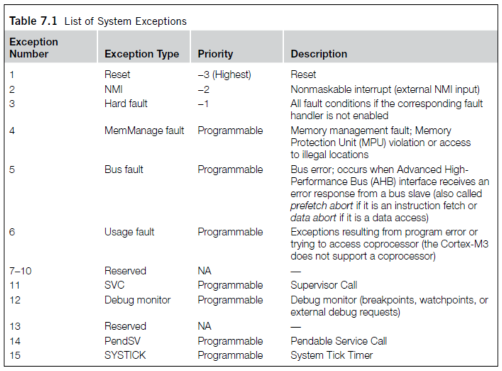


Table 7.1 List of System Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	–3 (Highest)	Reset
2	NMI	–2	Nonmaskable interrupt (external NMI input)
3	Hard fault	–1	All fault conditions if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; Memory Protection Unit (MPU) violation or access to illegal locations
5	Bus fault	Programmable	Bus error; occurs when Advanced High-Performance Bus (AHB) interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage fault	Programmable	Exceptions resulting from program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	—
11	SVC	Programmable	Supervisor Call
12	Debug monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	—
14	PendSV	Programmable	Pendable Service Call
15	SYSTICK	Programmable	System Tick Timer

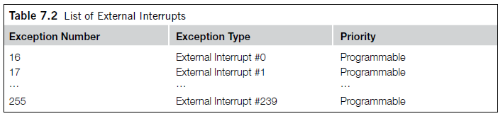


Table 7.2 List of External Interrupts

Exception Number	Exception Type	Priority
16	External Interrupt #0	Programmable
17	External Interrupt #1	Programmable
…	…	…
255	External Interrupt #239	Programmable

SmartFusion interrupt sources

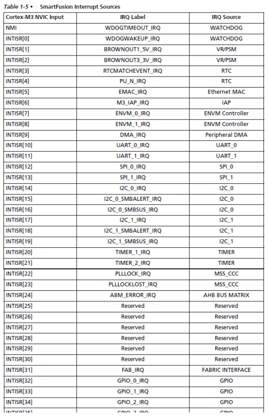
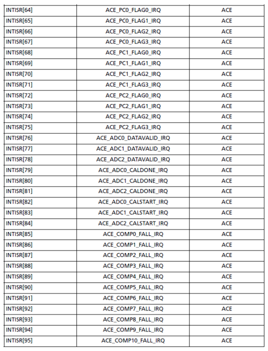


Table 1-5 • SmartFusion Interrupt Sources

Cortex-M3 NVIC Input	IRQ Label	IRQ Source
NMI	WDOGTIMEOUT_IRQ	WATCHDOG
INTISR[0]	WDOGWAKEUP_IRQ	WATCHDOG
INTISR[1]	BROWNOUT1_5V_IRQ	VR/PSM
INTISR[2]	BROWNOUT3_3V_IRQ	VR/PSM
INTISR[3]	RTCMATCHEVENT_IRQ	RTC
INTISR[4]	PU_N_IRQ	RTC
INTISR[5]	EMAC_IRQ	Ethernet MAC
INTISR[6]	M3_IAP_IRQ	IAP
INTISR[7]	ENVM_0_IRQ	ENVM Controller
INTISR[8]	ENVM_1_IRQ	ENVM Controller
INTISR[9]	DMA_IRQ	Peripheral DMA
INTISR[10]	UART_0_IRQ	UART_0
INTISR[11]	UART_1_IRQ	UART_1
INTISR[12]	SPI_0_IRQ	SPI_0
INTISR[13]	SPI_1_IRQ	SPI_1
INTISR[14]	I2C_0_IRQ	I2C_0
INTISR[15]	I2C_0_SMBALERT_IRQ	I2C_0
INTISR[16]	I2C_0_SMBSUS_IRQ	I2C_0
INTISR[17]	I2C_1_IRQ	I2C_1
INTISR[18]	I2C_1_SMBALERT_IRQ	I2C_1
INTISR[19]	I2C_1_SMBSUS_IRQ	I2C_1
INTISR[20]	TIMER_1_IRQ	TIMER
INTISR[21]	TIMER_2_IRQ	TIMER
INTISR[22]	PLLLOCK_IRQ	MSS_CCC
INTISR[23]	PLLLOCKLOST_IRQ	MSS_CCC
INTISR[24]	ABM_ERROR_IRQ	AHB BUS MATRIX
INTISR[25]	Reserved	Reserved
INTISR[26]	Reserved	Reserved
INTISR[27]	Reserved	Reserved
INTISR[28]	Reserved	Reserved
INTISR[29]	Reserved	Reserved
INTISR[30]	Reserved	Reserved
INTISR[31]	FAB_IRQ	FABRIC INTERFACE
INTISR[32]	GPIO_0_IRQ	GPIO
INTISR[33]	GPIO_1_IRQ	GPIO
INTISR[34]	GPIO_2_IRQ	GPIO
INTISR[35]	GPIO_3_IRQ	GPIO



INTISR[64]	ACE_PC0_FLAG0_IRQ	ACE
INTISR[65]	ACE_PC0_FLAG1_IRQ	ACE
INTISR[66]	ACE_PC0_FLAG2_IRQ	ACE
INTISR[67]	ACE_PC0_FLAG3_IRQ	ACE
INTISR[68]	ACE_PC1_FLAG0_IRQ	ACE
INTISR[69]	ACE_PC1_FLAG1_IRQ	ACE
INTISR[70]	ACE_PC1_FLAG2_IRQ	ACE
INTISR[71]	ACE_PC1_FLAG3_IRQ	ACE
INTISR[72]	ACE_PC2_FLAG0_IRQ	ACE
INTISR[73]	ACE_PC2_FLAG1_IRQ	ACE
INTISR[74]	ACE_PC2_FLAG2_IRQ	ACE
INTISR[75]	ACE_PC2_FLAG3_IRQ	ACE
INTISR[76]	ACE_ADC0_DATAVALID_IRQ	ACE
INTISR[77]	ACE_ADC1_DATAVALID_IRQ	ACE
INTISR[78]	ACE_ADC2_DATAVALID_IRQ	ACE
INTISR[79]	ACE_ADC0_CALDONE_IRQ	ACE
INTISR[80]	ACE_ADC1_CALDONE_IRQ	ACE
INTISR[81]	ACE_ADC2_CALDONE_IRQ	ACE
INTISR[82]	ACE_ADC0_CALSTART_IRQ	ACE
INTISR[83]	ACE_ADC1_CALSTART_IRQ	ACE
INTISR[84]	ACE_ADC2_CALSTART_IRQ	ACE
INTISR[85]	ACE_COMP0_FALL_IRQ	ACE
INTISR[86]	ACE_COMP1_FALL_IRQ	ACE
INTISR[87]	ACE_COMP2_FALL_IRQ	ACE
INTISR[88]	ACE_COMP3_FALL_IRQ	ACE
INTISR[89]	ACE_COMP4_FALL_IRQ	ACE
INTISR[90]	ACE_COMP5_FALL_IRQ	ACE
INTISR[91]	ACE_COMP6_FALL_IRQ	ACE
INTISR[92]	ACE_COMP7_FALL_IRQ	ACE
INTISR[93]	ACE_COMP8_FALL_IRQ	ACE
INTISR[94]	ACE_COMP9_FALL_IRQ	ACE
INTISR[95]	ACE_COMP10_FALL_IRQ	ACE

54 more ACE specific interrupts

GPIO_3_IRQ to GPIO_31_IRQ cut

And the interrupt vectors (in startup_a2fxxxm3.s found in CMSIS, startup_gcc)

```
g_pfnVectors:
    .word  _estack
    .word  Reset_Handler
    .word  NMI_Handler
    .word  HardFault_Handler
    .word  MemManage_Handler
    .word  BusFault_Handler
    .word  UsageFault_Handler
    .word  0
    .word  0
    .word  0
    .word  0
    .word  SVC_Handler
    .word  DebugMon_Handler
    .word  0
    .word  PendSV_Handler
    .word  SysTick_Handler
    .word  WdogWakeup_IRQHandler
    .word  BrownOut_1_5V_IRQHandler
    .word  BrownOut_3_3V_IRQHandler
..............
```
(they continue)

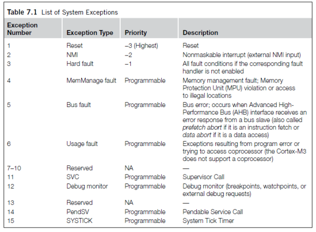


Table 7.1 List of System Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	–3 (Highest)	Reset
2	NMI	–2	Nonmaskable interrupt (external NMI input)
3	Hard fault	–1	All fault conditions if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; Memory Protection Unit (MPU) violation or access to illegal locations
5	Bus fault	Programmable	Bus error; occurs when Advanced High-Performance Bus (AHB) interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage fault	Programmable	Exceptions resulting from program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	—
11	SVC	Programmable	Supervisor Call
12	Debug monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	—
14	PendSV	Programmable	Pendable Service Call
15	SYSTICK	Programmable	System Tick Timer

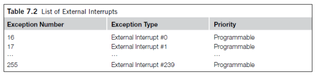
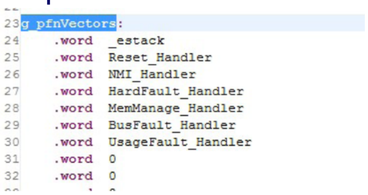


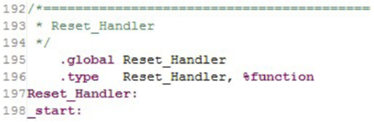
Table 7.2 List of External Interrupts

Exception Number	Exception Type	Priority
16	External Interrupt #0	Programmable
17	External Interrupt #1	Programmable
…	…	…
255	External Interrupt #239	Programmable

Interrupt handlers

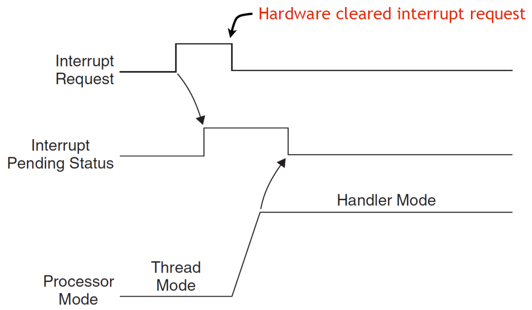


```
g_pfnVectors:
    .word   _estack
    .word   Reset_Handler
    .word   NMI_Handler
    .word   HardFault_Handler
    .word   MemManage_Handler
    .word   BusFault_Handler
    .word   UsageFault_Handler
    .word   0
    .word   0
```



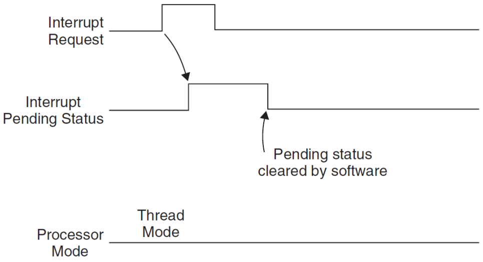
```
/*==========================================
 * Reset_Handler
 */
    .global Reset_Handler
    .type   Reset_Handler, %function
Reset_Handler:
_start:
```

Pending interrupts



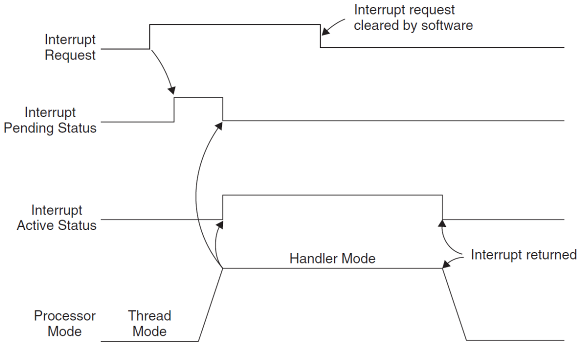
The normal case. Once Interrupt request is seen, processor puts it in "pending" state even if hardware drops the request.
IPS is cleared by the hardware once we jump to the ISR.

This figure and those following are from *The Definitive Guide to the ARM Cortex-M3, Section 7.4*

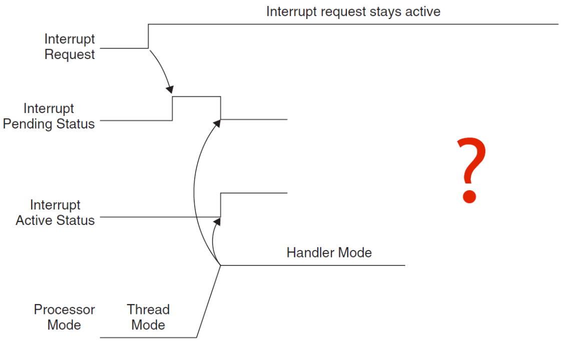


In this case, the processor never took the interrupt because we cleared the IPS by hand (via a memory-mapped I/O register)

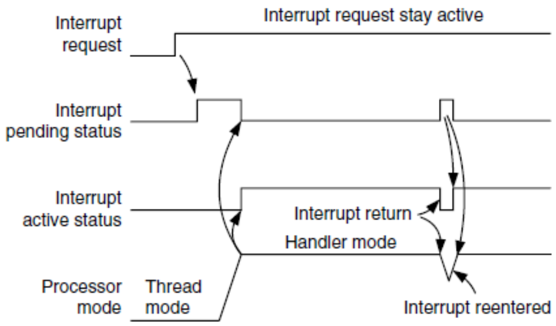
Active Status set during handler execution



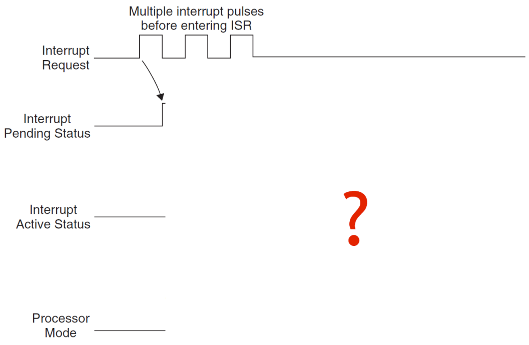
Interrupt Request not Cleared



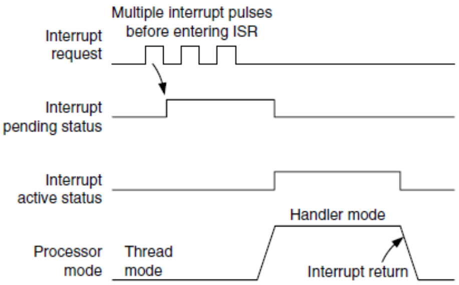
Answer



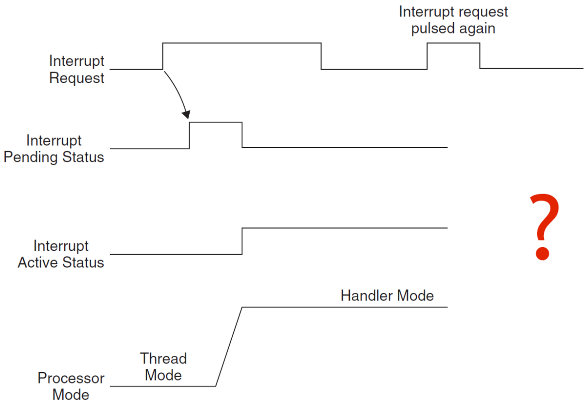
Interrupt pulses before entering ISR



Answer



New Interrupt Request after Pending Cleared



Configuring the NVIC

- Interrupt Set Enable and Clear Enable
  - 0xE000E100-0xE000E11C, 0xE000E180-0xE000E19C

0xE000E100	SETENA0	R/W	0	Enable for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E180	CLRENA0	R/W	0	Clear enable for external interrupt #0–31 bit[0] for interrupt #0 bit[1] for interrupt #1 … bit[31] for interrupt #31 Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current enable status

Configuring the NVIC (2)

- Set Pending & Clear Pending
  - 0xE000E200-0xE000E21C, 0xE000E280-0xE000E29C

0xE000E200	SETPEND0	R/W	0	Pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E280	CLRPEND0	R/W	0	Clear pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current pending status

Configuring the NVIC (3)

- Interrupt Active Status Register
  - 0xE000E300-0xE000E31C

Address	Name	Type	Reset Value	Description
0xE000E300	ACTIVE0	R	0	Active status for external interrupt #0–31 bit[0] for interrupt #0 bit[1] for interrupt #1 … bit[31] for interrupt #31
0xE000E304	ACTIVE1	R	0	Active status for external interrupt #32–63
…	–	–	–	–

Interrupt Priority

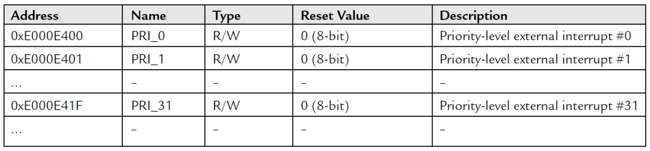
- What do we do if several interrupts arrive at the same time?
- NVIC allows to set priorities for (almost) every interrupt
- 3 fixed highest priorities, up to 256 programmable priorities
  - 128 preemption levels
  - Not all priorities have to be implemented by a vendor!

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented			Not implemented, read as zero				

  - SmartFusion has 32 priority levels, i.e., 0x00, 0x08, …, 0xF8
- Higher priority interrupts can pre-empt lower priorities
- Priority can be sub-divided into priority groups
  - splits priority register into two halves, *preempt priority* and *subpriority*
  - preempt priority: indicates if an interrupt can preempt another
  - subpriority: used if two interrupts of same group arrive concurrently

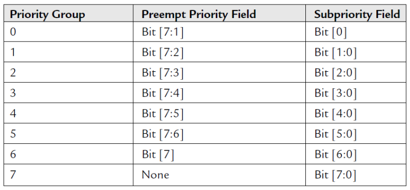
Interrupt Priority (2)

- Interrupt Priority Level Registers
  - 0xE000E400-0xE000E4EF



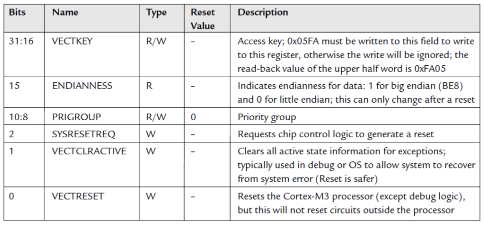
Address	Name	Type	Reset Value	Description
0xE000E400	PRI_0	R/W	0 (8-bit)	Priority-level external interrupt #0
0xE000E401	PRI_1	R/W	0 (8-bit)	Priority-level external interrupt #1
…	-	-	-	-
0xE000E41F	PRI_31	R/W	0 (8-bit)	Priority-level external interrupt #31
…	-	-	-	-

Preemption Priority and Subpriority



Priority Group	Preempt Priority Field	Subpriority Field
0	Bit [7:1]	Bit [0]
1	Bit [7:2]	Bit [1:0]
2	Bit [7:3]	Bit [2:0]
3	Bit [7:4]	Bit [3:0]
4	Bit [7:5]	Bit [4:0]
5	Bit [7:6]	Bit [5:0]
6	Bit [7]	Bit [6:0]
7	None	Bit [7:0]

Application Interrupt and Reset Control Register (Address 0xE000ED0C)



Bits	Name	Type	Reset Value	Description
31:16	VECTKEY	R/W	-	Access key; 0x05FA must be written to this field to write to this register, otherwise the write will be ignored; the read-back value of the upper half word is 0xFA05
15	ENDIANNESS	R	-	Indicates endianness for data: 1 for big endian (BE8) and 0 for little endian; this can only change after a reset
10:8	PRIGROUP	R/W	0	Priority group
2	SYSRESETREQ	W	-	Requests chip control logic to generate a reset
1	VECTCLRACTIVE	W	-	Clears all active state information for exceptions; typically used in debug or OS to allow system to recover from system error (Reset is safer)
0	VECTRESET	W	-	Resets the Cortex-M3 processor (except debug logic), but this will not reset circuits outside the processor

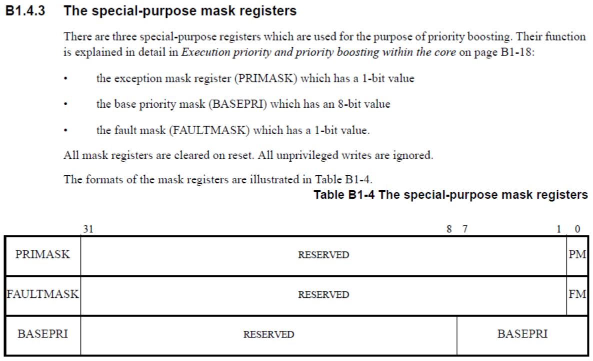
PRIMASK, FAULTMASK, and BASEPRI

- What if we quickly want to disable all interrupts?

- Write 1 into PRIMASK to disable all interrupt except NMI
  - MOV R0, #1
  - MSR PRIMASK, R0
- Write 0 into PRIMASK to enable all interrupts
- FAULTMASK is the same as PRIMASK, but also blocks hard fault (priority -1)

- What if we want to disable all interrupts below a certain priority?
- Write priority into BASEPRI
  - MOV R0, #0x60
  - MSR BASEPRI, R0

Masking



B1.4.3 The special-purpose mask registers

There are three special-purpose registers which are used for the purpose of priority boosting. Their function is explained in detail in *Execution priority and priority boosting within the core* on page B1-18:

- the exception mask register (PRIMASK) which has a 1-bit value
- the base priority mask (BASEPRI) which has an 8-bit value
- the fault mask (FAULTMASK) which has a 1-bit value.

All mask registers are cleared on reset. All unprivileged writes are ignored.

The formats of the mask registers are illustrated in Table B1-4.

Table B1-4 The special-purpose mask registers

	31 8	7 1	0
PRIMASK	RESERVED		PM
FAULTMASK	RESERVED		FM
BASEPRI	RESERVED	BASEPRI	

Interrupt Service Routines

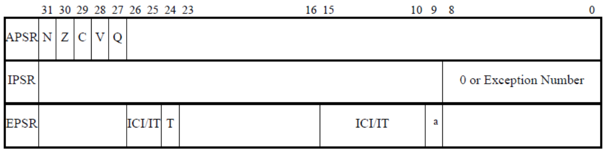
1. Automatic saving of registers upon exception
   - PC, PSR, R0-R3, R12, LR pushed on the stack
2. While bus busy, fetch exception vector
3. Update SP to new location
4. Update IPSR (low part of PSR) with new exception number
5. Set PC to vector handler
6. Update LR to special value EXC_RETURN

- Several other NVIC registers get updated
- Latency: as short as 12 cycles

The xPSR register layout

The APSR, IPSR and EPSR registers are allocated as mutually exclusive bitfields within a 32-bit register. The combination of the APSR, IPSR and EPSR registers is referred to as the xPSR register.

Table B1-2 The xPSR register layout



ARM interrupt summary

1. We've got a bunch of memory-mapped registers that control things (**NVIC**)
   - Enable/disable individual interrupts
   - Set/clear pending
   - Interrupt priority and preemption
2. We've got to understand how the hardware interrupt lines interact with the NVIC
3. And how we figure out where to set the PC to point to for a given interrupt source.

1. NVIC registers (example)

- Set Pending & Clear Pending
  - 0xE000E200-0xE000E21C, 0xE000E280-0xE000E29C

0xE000E200	SETPEND0	R/W	0	Pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E280	CLRPEND0	R/W	0	Clear pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current pending status

1. More registers (example)

- Interrupt Priority Level Registers
  - 0xE000E400-0xE000E4EF

Address	Name	Type	Reset Value	Description
0xE000E400	PRI_0	R/W	0 (8-bit)	Priority-level external interrupt #0
0xE000E401	PRI_1	R/W	0 (8-bit)	Priority-level external interrupt #1
…	–	–	–	–
0xE000E41F	PRI_31	R/W	0 (8-bit)	Priority-level external interrupt #31
…	–	–	–	–

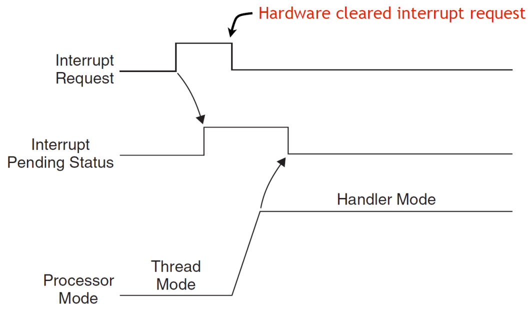
1. Yet another part of the NVIC registers!

Priority Group	Preempt Priority Field	Subpriority Field
0	Bit [7:1]	Bit [0]
1	Bit [7:2]	Bit [1:0]
2	Bit [7:3]	Bit [2:0]
3	Bit [7:4]	Bit [3:0]
4	Bit [7:5]	Bit [4:0]
5	Bit [7:6]	Bit [5:0]
6	Bit [7]	Bit [6:0]
7	None	Bit [7:0]

Application Interrupt and Reset Control Register (Address 0xE000ED0C)

Bits	Name	Type	Reset Value	Description
31:16	VECTKEY	R/W	–	Access key; 0x05FA must be written to this field to write to this register, otherwise the write will be ignored; the read-back value of the upper half word is 0xFA05
15	ENDIANNESS	R	–	Indicates endianness for data: 1 for big endian (BE8) and 0 for little endian; this can only change after a reset
10:8	PRIGROUP	R/W	0	Priority group
2	SYSRESETREQ	W	–	Requests chip control logic to generate a reset
1	VECTCLRACTIVE	W	–	Clears all active state information for exceptions; typically used in debug or OS to allow system to recover from system error (Reset is safer)
0	VECTRESET	W	–	Resets the Cortex-M3 processor (except debug logic), but this will not reset circuits outside the processor

2. How external lines interact with the NVIC



The normal case. Once Interrupt request is seen, processor puts it in "pending" state even if hardware drops the request.
IPS is cleared by the hardware once we jump to the ISR.

This figure and those following are from *The Definitive Guide to the ARM Cortex-M3, Section 7.4*

3. How the hardware figures out what to set the PC to

```
g_pfnVectors:
      .word  _estack
      .word  Reset_Handler
      .word  NMI_Handler
      .word  HardFault_Handler
      .word  MemManage_Handler
      .word  BusFault_Handler
      .word  UsageFault_Handler
      .word  0
      .word  0
      .word  0
      .word  0
      .word  SVC_Handler
      .word  DebugMon_Handler
      .word  0
      .word  PendSV_Handler
      .word  SysTick_Handler
      .word  WdogWakeup_IRQHandler
      .word  BrownOut_1_5V_IRQHandler
      .word  BrownOut_3_3V_IRQHandler
..............  (they continue)
```

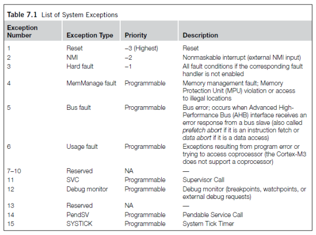


Table 7.1 List of System Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	–3 (Highest)	Reset
2	NMI	–2	Nonmaskable interrupt (external NMI input)
3	Hard fault	–1	All fault conditions if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; Memory Protection Unit (MPU) violation or access to illegal locations
5	Bus fault	Programmable	Bus error; occurs when Advanced High-Performance Bus (AHB) interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage fault	Programmable	Exceptions resulting from program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	—
11	SVC	Programmable	Supervisor Call
12	Debug monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	—
14	PendSV	Programmable	Pendable Service Call
15	SYSTICK	Programmable	System Tick Timer

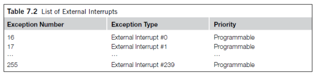


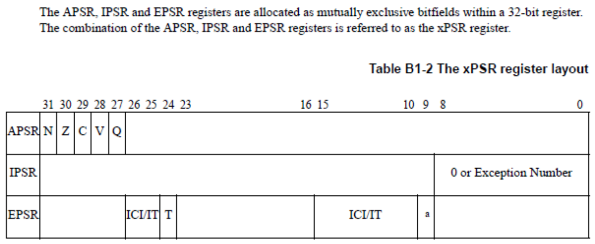
Table 7.2 List of External Interrupts

Exception Number	Exception Type	Priority
16	External Interrupt #0	Programmable
17	External Interrupt #1	Programmable
...	...	...
255	External Interrupt #239	Programmable

So let's say a GPIO pin goes high
- When will we get an interrupt?
- What happens if the interrupt is allowed to proceed?

What happens when we return from an ISR?

Other stuff: The xPSR register layout

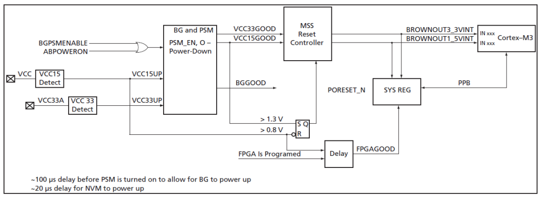


The APSR, IPSR and EPSR registers are allocated as mutually exclusive bitfields within a 32-bit register. The combination of the APSR, IPSR and EPSR registers is referred to as the xPSR register.

Table B1-2 The xPSR register layout

	31	30	29	28	27	26 25	24	23 16	15 10	9	8 0
APSR	N	Z	C	V	Q						
IPSR											0 or Exception Number
EPSR						ICI/IT	T		ICI/IT	a	

Example of Complexity: The Reset Interrupt



~100 µs delay before PSM is turned on to allow for BG to power up
~20 µs delay for NVM to power up

1) No power
2) System is held in RESET as long as VCC15 < 0.8V
    a) In reset: registers forced to default
    b) RC-Osc begins to oscillate
    c) MSS_CCC drives RC-Osc/4 into FCLK
    d) PORESET_N is held low
3) Once VCC15GOOD, PORESET_N goes high
    a) MSS reads from eNVM address 0x0 and 0x4

Interrupt types

- Two main types
  - Level-triggered
  - Edge-triggered

Level-triggered interrupts

- Signaled by asserting a line low or high
- Interrupting device drives line low or high and holds it there until it is serviced
- Device deasserts when directed to or after serviced
- Can share the line among multiple devices (w/ OD+PU)
- Active devices assert the line
- Inactive devices let the line float
- Easy to share line w/o losing interrupts
- But servicing increases CPU load → example
- And requires CPU to keep cycling through to check
- Different ISR costs suggests careful ordering of ISR checks
- Can't detect a new interrupt when one is already asserted

Edge-triggered interrupts

- Signaled by a level *transition* (e.g. rising/falling edge)
- Interrupting device drive a pulse (train) onto INT line
- What if the pulse is too short? Need a pulse extender!
- Sharing *is* possible...under some circumstances
- INT line has a pull up and all devices are OC/OD.
- Devices *pulse* lines
- Could we miss an interrupt? Maybe...if close in time
- What happens if interrupts merge? Need one more ISR pass
- Must check trailing edge of interrupt
- Easy to detect "new interrupts"
- Benefits: more immune to unserviceable interrupts
- Pitfalls: spurious edges, missed edges
- Source of "lockups" in early computers

Group talks in EECS 373

Special topics talks

- Groups of 2-3 folks
  - Not your lab partner (or your project group member)
  - This is 1% of your grade (20% of the presentation)
- 12 minutes for the talk, ~3 minutes for questions
- Four parts
  - Meet with me 2-3 weeks ahead of time to discuss topic
  - 1st practice talk 1-2 weeks before scheduled date (20%)
  - 2nd practice talk 1-2 days before scheduled date (20%)
  - Give talk in class (40%)

Special topics talk (2)

- Each talk must include
  - Explanation of how the topic relates to embedded systems
  - An understanding of high-level issues including tradeoffs
    - Must produce at least two original graphs explaining tradeoffs.
  - Some *detailed* explanation of a relevant part of the topic
  - Where others can go to learn more information
- Time permitting
  - We'll take 10 minutes at the end of class to form groups of 2-3
  - We'll discuss some topics that I'd like to see (BLE, Cortex-M3s, accelerometers, gyroscopes, microphones, etc.)