Undantagshantering och interna avbrott

Ur innehållet:

- Faults
- Software traps
- Undantagsprioriteter
- Avbrott från interna enheter, ”Systick”

Läsanvisningar:

- Arbetsbok kap 6.1-6.4

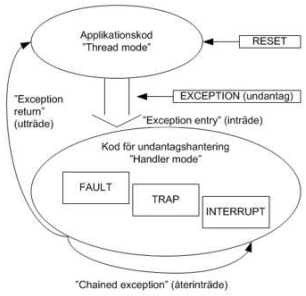
Målsättningar:

- Förklara hur undantagshantering går till
- Implementera en icke-blockerande fördröjning med SysTick

Undantagshantering - "exception"

Med "undantag" menar vi olika typer av händelser:

- RESET (*återstart*):
  - *power on* (kallstart)
  - *warm reset* (varmstart)
- FAULT:
  - Exekveringsfel – kan inte fortsätta
- TRAP:
  - Programmerat avbrott, initierat av maskininstruktion
- INTERRUPT:
  - Hårdvarusignalerat avbrott



Undantagstyper

De olika undantagstyperna har en naturlig, fallande prioritet

RESET:
- power on (kallstart)
- warm reset (varmstart)

NMI:
Non Maskable Interrupt

FAULT:
- Hard fault *generellt fel*
- Memory management fault *fel användning av minne*
- Bus fault *exvis obefintligt minne*
- Usage fault *exvis "unaligned"*

TRAP:
- SVC- instruktion
- BPKT-instruktion
- ...

INTERRUPT:

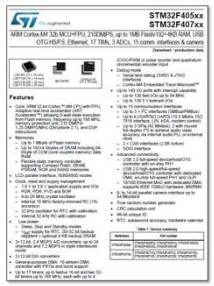
INTERNA:
- SysTick
- Watchdog
- Periferikretsar
- .....

EXTERNA:
- IO-pinnar kan konfigureras som avbrottsingångar
- EXTI (External Interrupt) modul

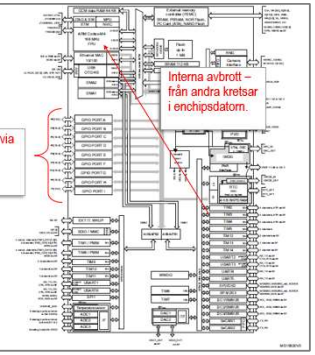
Högsta prioritet

Lägre prioritet

Interna avbrott



Externa avbrott – via portpinnar från kretsar utanför enchipsdatorn.



Vektortabellen

Anger position för de olika undantagsvektorerna.

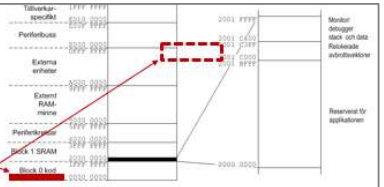
De 16 första positionerna är samma för alla Cortex-M4.

Resten av tabellen är specifik för den aktuella microcontrollern.

IRQ num	priority	name	description	vector offset
-	-	-	Reserved for initial stack pointer	0x0000 0000
-	fixed, -3	Reset	Reset	0x0000 0004
-14	fixed, -2	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-13	fixed, -1	HardFault	All class of fault	0x0000 000C
-12	settable	MemManage	Memory management	0x0000 0010
-11	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-10	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	Reserved	0x0000 001C - 0x0000 002B
-5	settable	SVCall	System service call via SWI instruction	0x0000 002C
-4	settable	Debug Monitor	Debug Monitor	0x0000 0030
-	-	-	Reserved	0x0000 0034
-2	settable	PendSV	Pendable request for system service	0x0000 0038
-1	settable	SysTick	System tick timer	0x0000 003C
0	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	settable	PVD	PVD through EXTI line detection interrupt	0x0000 0044
2	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	0x0000 004C
4	settable	FLASH	Flash global interrupt	0x0000 0050
5	settable	RCC	RCC global interrupt	0x0000 0054
6	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068
11	settable	DMA1_Stream0	DMA1 Stream0 global interrupt	0x0000 006C
79	settable	CRYP	CRYP crypto global interrupt	0x0000 0170
80	settable	HASH_RNG	Hash and Rng global interrupt	0x0000 0180
81	settable	FPU	FPU global interrupt	0x0000 0184

Relokering av vektortabellen

Vektortabellen startar på address 0 i minnet (Alltid *Read Only*-minne) ....



... men kan relokeras så att vektorerna hamnar i RWM, exempelvis:

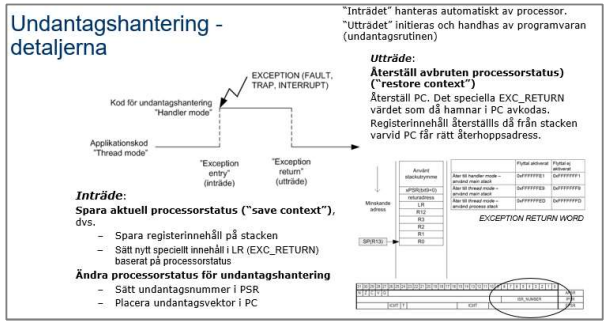
```
#define SCB_VTOR  ((volatile unsigned long *)0xE000ED08)
*SCB_VTOR = 0x2001C000;
```

Exempel:
Initiera avbrottsvektor "usage fault"

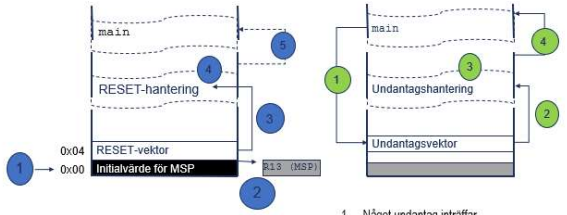
1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
2	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
.	.	.	Reserved	0x0000 001C
				0x0000 002B
3	settable	SVCall	System service call via SWI	0x0000 002C

```
void   usage_fault_handler( void );
...
*((void (**)(void) ) 0x2001C018 ) = usage_fault_handler;
```

Undantagshantering - detaljerna



Exekvering vid undantag

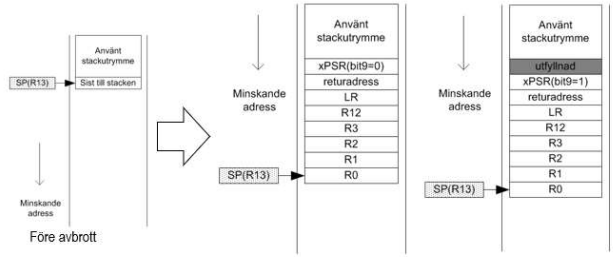


1. Återstart (Reset aktiverad)
2. Ladda MSP (Main Stack Pointer) från adress 0x00
3. Ladda RESET-vector från adress 0x04
4. RESET-hantering exekveras i "Thread mode"
5. Systemets huvudprogram startas

1. Något undantag inträffar
   - Pågående instruktion utförs klart
   - Vektortabellen adresseras
2. Den specifika undantagsvektorn hämtas
3. Undantagshantering i "Handler Mode"
4. Återgång efter undantagshantering "Thread Mode"

Stacken i avbrottsfunktionen

Ej aktiverad flyttalsenhet



Processorn fyller automatiskt ut med 4 bytes om detta skulle krävas för att SP ska innehålla adress på adress jämnt delbar med 8. Detta indikeras med att bit9 i PSR sätts till 1.

Återgång från avbrott

Värdet som laddas i PC anger detaljerna för återställning efter avbrott

EXEC RETURN WORD

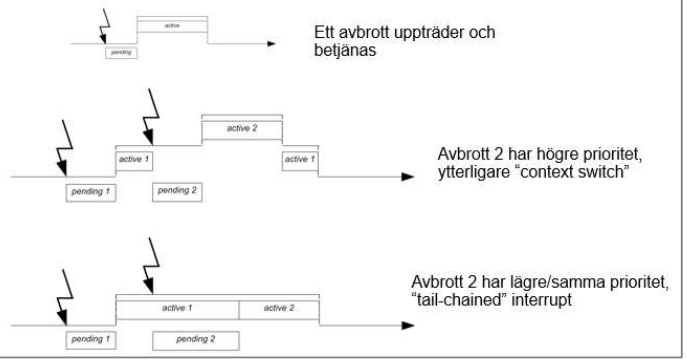
	Flyttal aktiverat	Flyttal ej aktiverat
Åter till *handler mode* – använd *main stack*	0xFFFFFFE1	0xFFFFFFF1
Åter till *thread mode* – använd *main stack*	0xFFFFFFE9	0xFFFFFFF9
Åter till *thread mode* – använd *process stack*	0xFFFFFFED	0xFFFFFFFD

- ❑ Kan återvända från avbrott med följande instruktioner då PC laddas med "det magiska värdet" 0xFFFF_FFFX
  - LDR PC, ...
  - LDM/POP då PC ingår i registerlistan
  - BX LR mest vanligt, typiskt retur från en subrutin
- ❑ Om inga ytterligare avbrott avvaktar:
  - återställs stack och tillstånd baserat på EXC_RETURN
- ❑ Om avbrott avvaktar:
  - återställning av stack/tillstånd fördröjs ("tail-chaining")
  - avvaktande avbrott betjänas

(EXC_RETURN) avgör hur stack och processortillstånd återställs efter avbrott

Avbrottsprioritet

Flera samtidiga avbrott: *pending*, *active* och *prioriteter*



Processorns olika tillstånd

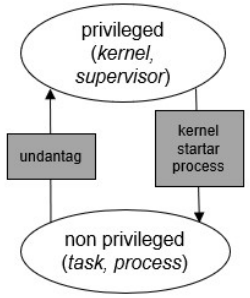
- ***Handler*** *(undantagshantering)*/***Thread mode*** *(normal exekvering)*, sköts automatiskt av processorn.

För att underlätta konstruktion av operativsystem finns också:

- ***Privileged/Non-privileged state***, kan programmeras. I *Non-priveleged* state fungerar *priviligierade* instruktioner som NOP.
- ***Main stack/Process stack*** – kan programmeras, processorn använder olika fysiska register (MSP eller PSP) som stackpekare.

Operativsystemets programvara (*kernel*) exekveras med alla resurser tillgängliga (priviligierat).

Applikationsprogrammet exekveras i en begänsad miljö (icke priviligierat), övervakad av *kernel*.



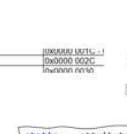
SVCall (SuperVisor Call)

En software trap instruktion för att utföra inbyggda funktioner.

- Ger en kontrollerad växling mellan *non-privileged* och *privileged* mode.
- Kompakt sätt att utföra funktion

```
@ SVC, SuperVisorCall
operationskoden är 0xDFxx, där xx utgör konstanten

__attribute__((naked))
void graphic_initalize(void)
{
  __asm volatile (" .HWORD 0xDFF0\n");
  __asm volatile (" BX LR\n");
}
```



```
static __attribute__((naked))
void svcHandler( void )
{
__asm volatile(" MOV R0,SP\n");
__asm volatile(" B   os_call\n");
__asm volatile(" .ALIGN 2\n");
}

void _graphic_initalize( void  );
void _graphic_clear_screen( int x, int y);
void _graphic_pixel_set( int x, int y);
void _graphic_pixel_clear( int x, int y);

void os_call( unsigned int * call_args)
{
  unsigned int oscall = ((char *) call_args[6]) [-2];
  switch( oscall )
  {
    case 0xF0: _graphic_initalize(); break;
    case 0xF1: _graphic_clear_screen(); break;
    case 0xF2: _graphic_pixel_set(call_args[0],call_args[1] )break;
    case 0xF3: _graphic_pixel_clear(call_args[0],call_args[1] )break;
  }
}
```

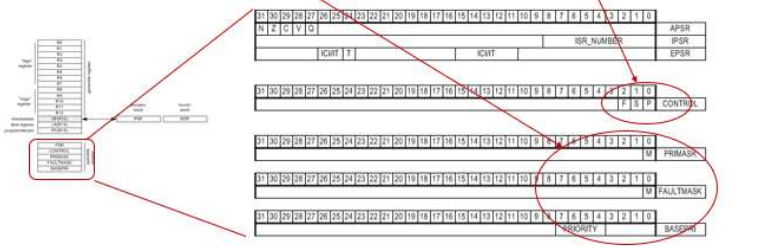
Speciella register

används för att ange:

- Om flyttalsprocessor ska aktiveras
- Vilken stack som används
- Vilket exekveringstillstånd som används
- Om undantag/avbrott ska betjänas

Exempel:

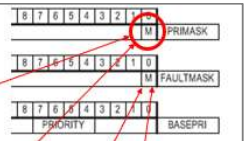
```
void __setCONTROL( unsigned int val)
{
  __asm(" MSR CONTROL,R0\n");
}
```



Maskera undantag

Man kan undvika problematiken med avbrottsprioriteter:

I undantagsrutinen kan man tillfälligt höja den egna prioriteten till högsta nivå genom att sätta prioritetsmasken i PRIMASK till 1.




```
void disable_interrupt( void )
{
  __asm(" CPSID I\n");
}
```

```
void enable_interrupt( void )
{
  __asm(" CPSIE I\n");
}
```

Undantagsrutinen måste då också återställa prioritetsmasken till 0 innan återgång.
I annat fall kommer alla framtida avbrott att blockeras.

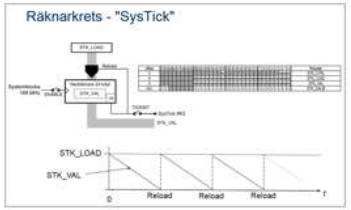
```
void irq_handler( void )
{
    disable_interrupt();
    ....
    enable_interrupt();
}
```

```
void disable_fault( void )
{
  __asm(" CPSID F\n");
}
```

```
void enable_fault( void )
{
  __asm(" CPSIE F\n");
}
```

”SysTick” med avbrott...

Räknarkrets - "SysTick"

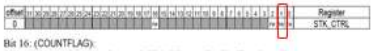


Algoritm:

STK_CTRL = 0	Återställ SysTick
STK_LOAD =	CountValue
STK_VAL = 0	Nollställ räknarregistret
STK_CTRL = 5	Starta om räknaren
Vänta till COUNTFLAG=1	
STK_CTRL = 0	Återställ SysTick

”Blockerande” – ty programmet kan inte göra något annat än vänta tills fördröjningen, dvs. COUNTFLAG blir 1, är klar....

STK_CTRL *Status och styrregister*



Bit 16: (COUNTFLAG):
Biten är 1 om räknaren räknat ned till 0. Biten nollställs då registret läses.
Bit 2: (CLKSOURCE) biten är 1 efter RESET:
0: Systemklocka/8
1: Systemklocka
Bit 1: (TICKINT): Aktivera avbrott
0: Inget avbrott genereras.
1: Då räknaren slagit om till 0 genererar SysTick avbrott.
Anm: Man kan programvarumässigt undersöka om räknaren räknat ned till 0 genom att kontrollera biten COUNTFLAG, det är alltså inte nödvändigt att använda avbrottsmekanismen.
Bit 0: (ENABLE) Aktivera räknare
Då ENABLE ettställs laddas värdet från STK_LOAD till STK_VAL och räknaren börjar räkna ned. Då räknaren nått 0, ettställs COUNTFLAG och proceduren upprepas.
0: Räknare deaktiverad
1: Räknare aktiverad

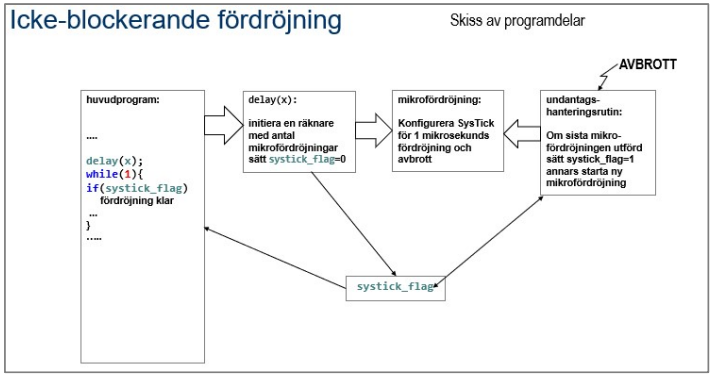
Algoritm:

STK_CTRL = 0	Återställ SysTick
STK_LOAD =	CountValue
STK_VAL = 0	Nollställ räknarregistret
STK_CTRL = 7	Starta om räknaren
Avbrott genereras då COUNTFLAG=1	

”Icke-blockerande” – ty programmet kan fortsätta med att exekvera, då fördröjningen är klar, dvs. COUNTFLAG är 1, genereras ett avbrott.

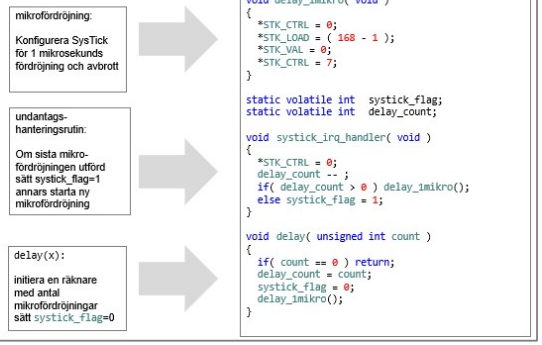
Icke-blockerande fördröjning

Skiss av programdelar



Icke-blockerande fördröjning

Implementering




```
void delay_1mikro( void )
{
  *STK_CTRL = 0;
  *STK_LOAD = ( 168 - 1 );
  *STK_VAL = 0;
  *STK_CTRL = 7;
}

static volatile int  systick_flag;
static volatile int  delay_count;

void systick_irq_handler( void )
{
  *STK_CTRL = 0;
  delay_count -- ;
  if( delay_count > 0 ) delay_1mikro();
  else systick_flag = 1;
}

void delay( unsigned int count )
{
  if( count == 0 ) return;
  delay_count = count;
  systick_flag = 0;
  delay_1mikro();
}
```