Externa avbrott

Anslutning av extern avbrottsvippa, programmering med konfigurering och hantering av externa avbrott. Introduktion till ”time-sharing”, enkel ”task-switch”.

Ur innehållet:

- NVIC och EXTI (SYSCFG)
- Avbrottsvippa
- Pseudoparallell exekvering.

Läsanvisningar:

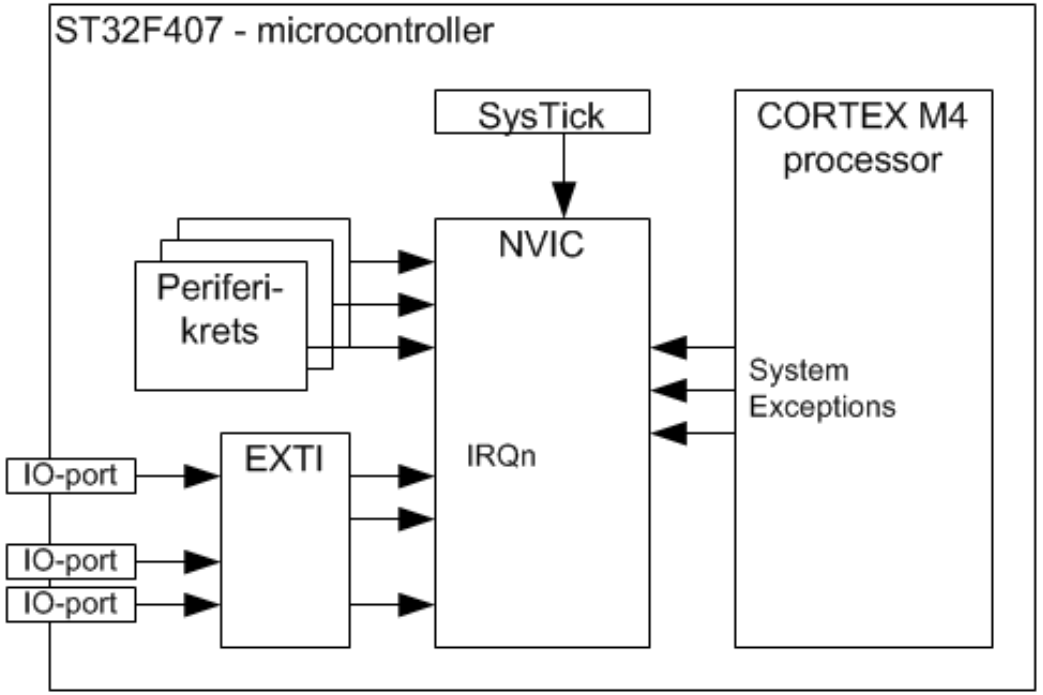
- Arbetsbok kapitel 6

Externa avbrott

Externa avbrott är en form av undantag.

Registerinnehåll (xPSR, PC, LR, R12, R3-R0) sparas och återställs automatiskt av processorn

Avbrottshanterare kan skrivas uteslutande med ‘C’-kod.

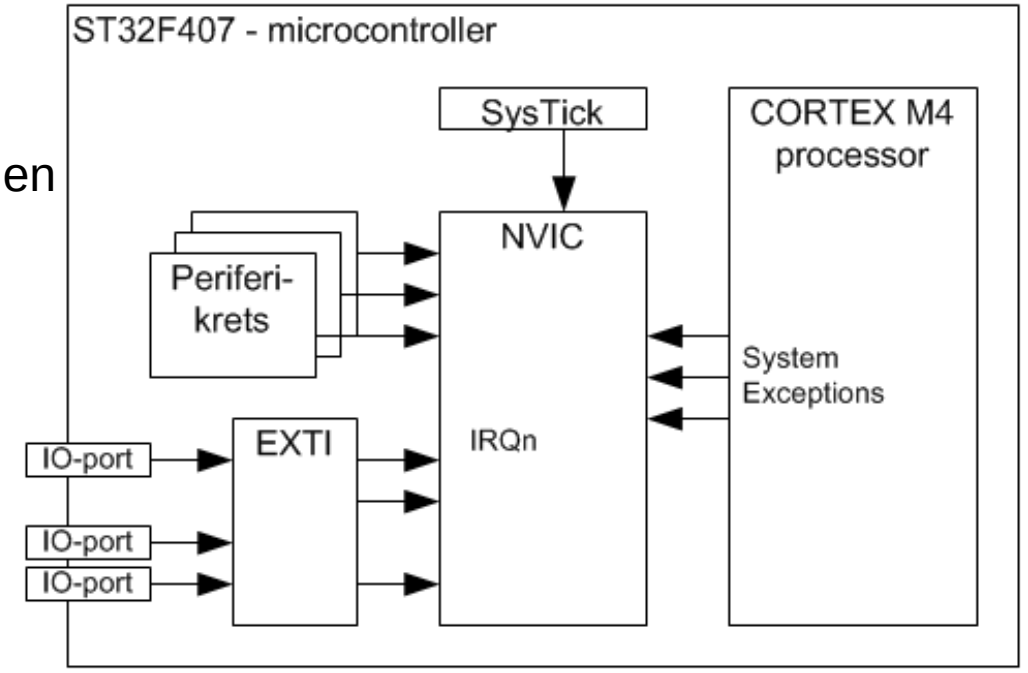


	6	settable	SysTick	System tick timer
0	7	settable	WWDG	Window Watchdog interrupt
1	8	settable	PVD	PVD through EXTI line detection interrupt
2	9	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line
3	10	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line
4	11	settable	FLASH	Flash global interrupt
5	12	settable	RCC	RCC global interrupt
6	13	settable	EXTI0	EXTI Line0 interrupt
7	14	settable	EXTI1	EXTI Line1 interrupt
8	15	settable	EXTI2	EXTI Line2 interrupt
9	16	settable	EXTI3	EXTI Line3 interrupt
10	17	settable	EXTI4	EXTI Line4 interrupt
11	18	settable	DMA1_Stream0	DMA1 Stream0 global interrupt
12	19	settable	DMA1_Stream1	DMA1 Stream1 global interrupt

Externa avbrottskällor

Avbrottsingångar konfigureras i tre steg

- SYSCFG-modulen används för att koppla en GPIO-pinne från någon port till en specifik avbrottslina (1 av 16) .
- EXTI-modulen (External interrupt/event controller) programmeras.
- NVIC-modulen (Nested vectored interrupt controller) programmeras för att hantera avbrott från denna avbrottsvektor (NVIC_ISERx).



Irq	Hanteringsrutin	Beskrivning
EXTI0_IRQn	EXTI0_IRQHandler	för IO-pinnar anslutna till avbrottslina 0
EXTI1_IRQn	EXTI1_IRQHandler	för IO-pinnar anslutna till avbrottslina 1
EXTI2_IRQn	EXTI2_IRQHandler	för IO-pinnar anslutna till avbrottslina 2
EXTI3_IRQn	EXTI3_IRQHandler	för IO-pinnar anslutna till avbrottslina 3
EXTI4_IRQn	EXTI4_IRQHandler	för IO-pinnar anslutna till avbrottslina 4
EXTI9_5_IRQn	EXTI9_5_IRQHandler	för IO-pinnar anslutna till avbrottslinor 5 till 9
EXTI15_10_IRQn	EXTI15_10_IRQHandler	för IO-pinnar anslutna till avbrottslinor 10 till 15

SYSCFG

EXTICRx används för att koppla IO-pinnar till processorns avbrottssystem

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0																																	SYSCFG_MEMRMP
4																																	SYSCFG_PMC
8																																	SYSCFG_EXTICR1
0xC																																	SYSCFG_ EXTICR2
0x10																																	SYSCFG_ EXTICR3
0x14																																	SYSCFG_ EXTICR4
0x20																																	SYSCFG_ CMPCR

EXTICRx är fyra kontrollregister med samma struktur:

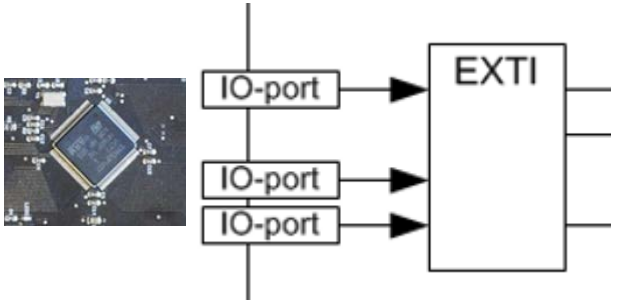
offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
8	EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]				SYSCFG_EXTICR1
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bitarna i EXTICRx bestämmer hur en IO-pinne dirigeras till någon av de 16 avbrottslinorna EXTI0..EXTI15 genom att motsvarande fält skrivs med fyra bitar.
Undantag: PK[15:8] används ej.

0000: PA[x]	0001: PB[x]	0010: PC[x]	0011: PD[x]	0100: PE[x]	0101: PF[x]
0110: PG[x]	0111: PH[x]	1000: PI[x]	1001: PJ[x]	1010: PK[x]	

EXEMPEL

Koppla pinne PC0 till “Externavbrott 0”



5	12	settable	RCC	RCC global interrupt	0x0000 0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
8	EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]				SYSCFG_EXTICR1
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

0000: PA[x]	0001: PB[x]	0010: PC[x]	0011: PD[x]	0100: PE[x]	0101: PF[x]
0110: PG[x]	0111: PH[x]	1000: PI[x]	1001: PJ[x]	1010: PK[x]	

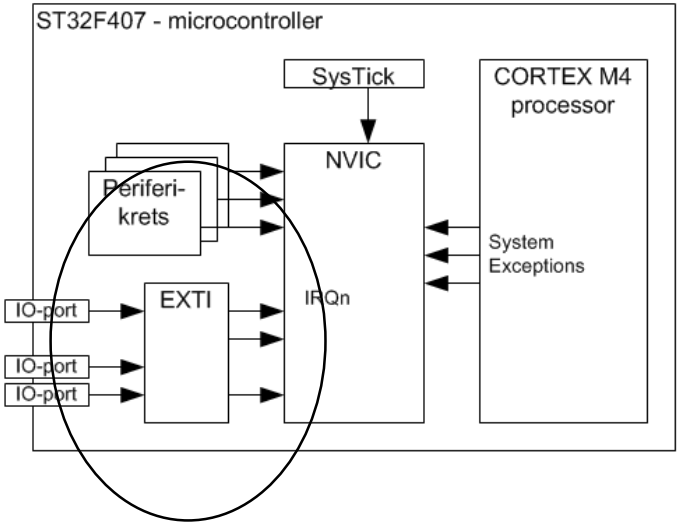
```
*SYSCFG_EXTICR1 &= 0xFFF0;      /* Nollställ bitfältet */
*SYSCFG_EXTICR1 |= 0x0002;      /* 0010 -> EXTI0[3:0] */
```

EXTI-modul

Totalt finns det 23 olika möjliga avbrottslinor i EXTI-modulen, även om inte alla används i STM32F407. Några kan ha gemensam avbrottsvektor.

Irq	Hanteringsrutin	Beskrivning
EXTI0_IRQn	EXTI0_IRQHandler	för IO-pinnar anslutna till avbrottslina 0
EXTI1_IRQn	EXTI1_IRQHandler	för IO-pinnar anslutna till avbrottslina 1
EXTI2_IRQn	EXTI2_IRQHandler	för IO-pinnar anslutna till avbrottslina 2
EXTI3_IRQn	EXTI3_IRQHandler	för IO-pinnar anslutna till avbrottslina 3
EXTI4_IRQn	EXTI4_IRQHandler	för IO-pinnar anslutna till avbrottslina 4
EXTI9_5_IRQn	EXTI9_5_IRQHandler	för IO-pinnar anslutna till avbrottslinor 5 till 9
EXTI15_10_IRQn	EXTI15_10_IRQHandler	för IO-pinnar anslutna till avbrottslinor 10 till 15

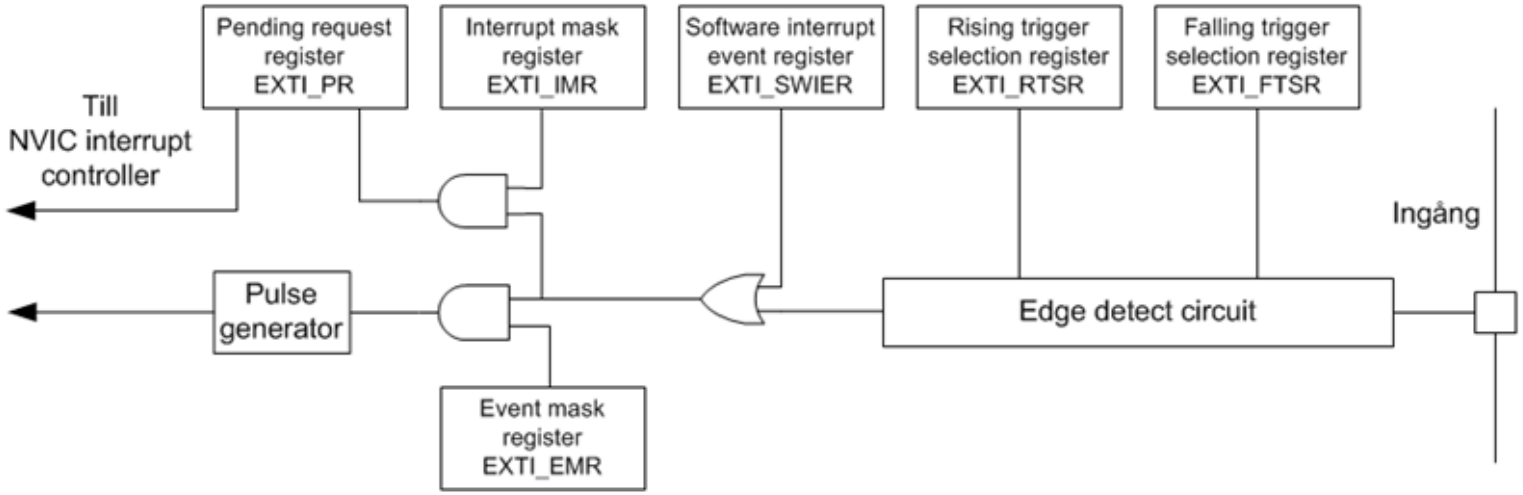
Irq	Beskrivning
EXTI16	PVD utgång fast ansluten till EXTIO16
EXTI17	RTC "Alarm event" fast ansluten till EXTIO17
EXTI18	USB OTG FS "Wakeup event" fast ansluten till EXTIO18
EXTI19	Ethernet "Wakeup event" fast ansluten till EXTIO19
EXTI20	USB OTG HS "Wakeup event" fast ansluten till EXTIO20
EXTI21	RTC "Tamper and TimeStamp" fast ansluten till EXTIO21
EXTI22	RTC "Wakeup event" fast ansluten till EXTIO22



offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0																																	EXTI_IMR
4																																	EXTI_EMR
8																																	EXTI_RTSR
0xC																																	EXTI_FTSR
0x10																																	EXTI_SWIER
0x14																																	EXTI_PR

Port pinnar Px 15 .. Px 0 (x=A,B,C,D,E,F)

EXTI– registren har samma struktur



EXTI_IMR (0x40013C00) Interrupt Mask Register

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0										rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_IMR

Kontrollerar EXTI 22

Kontrollerar EXTI 0

EXEMPEL: Aktivera avbrott EXTI0

```
*EXTI_IMR |= 1;
```

EXTI– triggfunktioner

EXTI_RTSR (0x40013C08) Rising Trigger Selection Register

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
8										rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_RTSR

EXEMPEL: Generera avbrott då insignalen från den pinne som “routats” till EXTI0 går från 0 till1, dvs “positiv flank”:

```
*EXTI_RTSR |= 1;
```

EXTI_FTSR (0x40013C0C) Falling Trigger Selection Register

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0xC										rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_FTSR

EXEMPEL: Generera avbrott då insignalen från den pinne som “routats” till EXTI2 går från 1 till 0, dvs “negativ flank”:

```
*EXTI_FTSR |= 4;
```

EXTI– ”Software interrupt”

Avbrott kan också genereras av programvara.
Om avbrott är aktiverat för lina *x*, kan programvara aktivera detta genom att skriva '1' till motsvarande bit i EXTI_SWIER.
Denna bit återställs genom skrivning till EXTI_PR.

EXTI_SWIER (0x40013C10) Software Interrupt Event Register

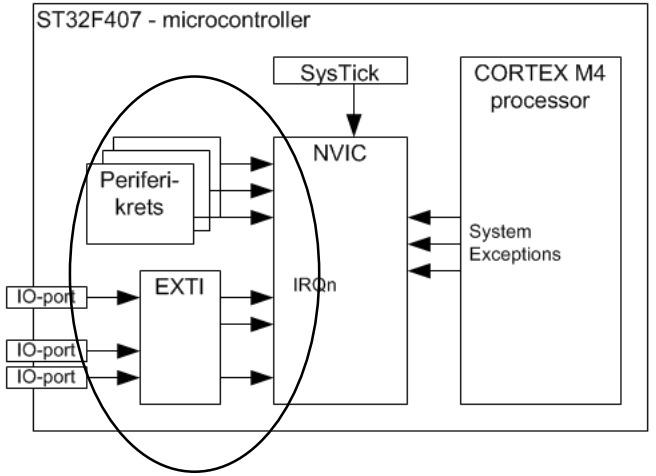
offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x10										rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_SWIER

EXTI_PR (0x40013C14) Pending Register

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x14										rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_PR

NVIC – Kontrollerar alla avbrott

offset	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	mnemonic
0x000	SETENA[31:0] rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw	NVIC_ISER0
0x004	SETENA[63:32] rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw	NVIC_ISER1
0x008	Reserverat / SETENA[80:64] rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw	NVIC_ISER2
	CLRENA[31:0]	



Fem uppsättningar register för olika funktioner, men med samma struktur. Arkitekturen tillåter maximalt 80 avbrottsgenererande enheter, alltså har varje registeruppsättning 80 bitar.

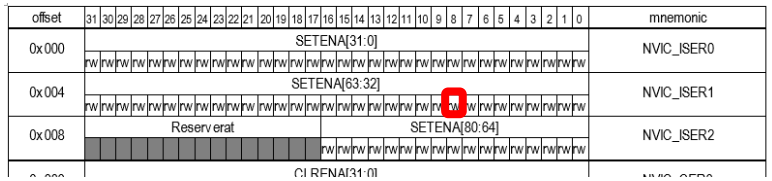
- Interrupt Set Enable Registers, NVIC_ISERx
- Interrupt Clear Enable Registers, NVIC_ICERx
- Interrupt Set Pendig Registers, NVIC_ISEPRx
- Interrupt Clear Pending Registers, NVIC_ICPRx
- Interrupt Active Bit Registers, NVIC_IABRx

Bestäm avbrottsvektor och bitnummer i NVIC för EXTI15_10

Vi letar upp EXTI15_10 i vektortabellen. Den första kolumnen anger avbrottets nummer i NVIC, dvs. 40. Kolumnen längst till höger anger offset för avbrottsvektorns placering i vektortabellen:

38	45	settable	USART2	USART2 global interrupt	0x0000 00D8
39	46	settable	USART3	USART3 global interrupt	0x0000 00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000 00E0
41	48	settable	RTC_Alarm	RTC Alarms (A and B) through EXTI line interrupt	0x0000 00E4
42	49	settable	OTG_FS	WKUP USB On-The-Go FS Wakeup through EXTI line interrupt	0x0000 00E8
43	50	settable	TIM8_BRK_TIM12	TIM8 Break interrupt and TIM12 global interrupt	0x0000 00EC

I detta fall identifierar avbrottsnummer 40. Vi har bara 32 bitar i varje NVIC-register så vi hamnar på (40-32) bit 8 i NVIC_ xxxx register **2 (dvs. NVIC_xxxx1).** Bitmasken för denna bit är 0x00000100, vilket också kan skrivas (1<<8).



EXEMPEL: Aktivera avbrott EXTI15_10

```
*NVIC_ISER1 = (1<<8);
```

Externt avbrott från avbrottsvippa

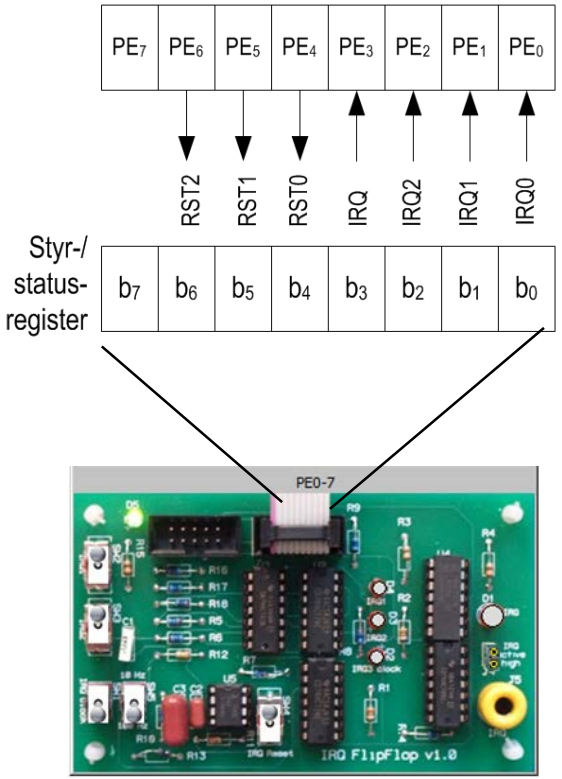
EXEMPEL:
Skriv en enkel applikation som använder PE3 hos MD407 som avbrottsingång och som registrerar varje avbrott, med en enkel räknare, och i huvudprogrammet skriver ut antalet avbrott till en visningsenhet enligt:

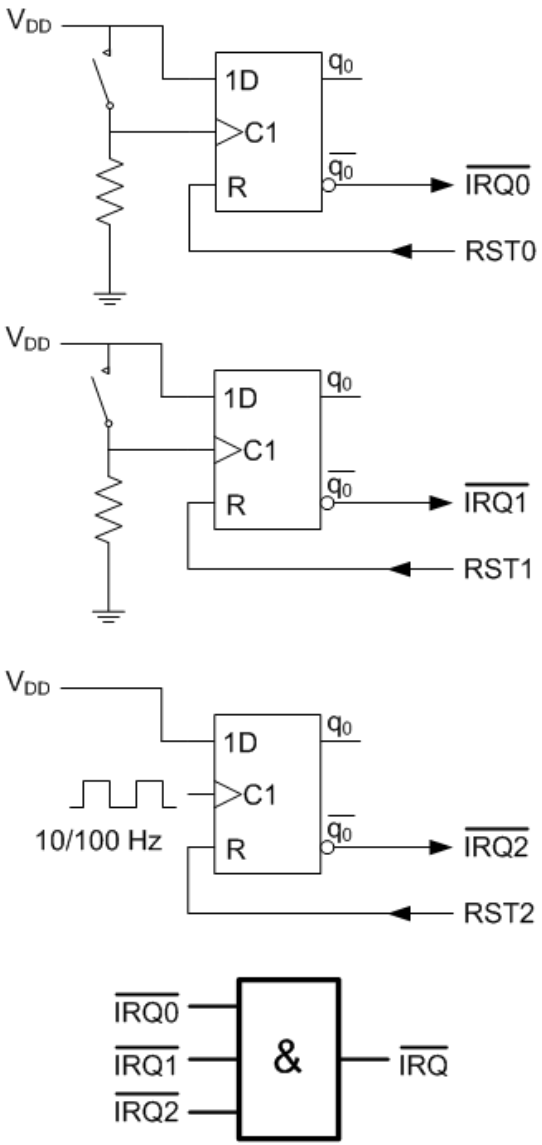
```
void main(void)
{
    app_init();
    while(1){
        * portBargraphOdrLow = count;
    }

}
```

Vi demonstrerar med simulator…

IRQ-FlipFlop - avbrottsvippor





Algoritmer...

EXTI_PR *Pending Register*

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x14																																	EXTI_PR
											rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit PR[22..0]:
Motsvarande bit sätts i detta register då ett triggvillkor är uppfyllt. Biten återställs genom att skrivas med '1'.
0: Ingen Trigg.
1: Trigg har uppträtt

Algoritm: interrupt handler
Om avbrott från EXTI3:
  inkrementera count;
  kvittera avbrott från EXTI3

```
if( (*EXTI_PR & EXTI3_IRQ_BPOS) != 0 )
```

```
*EXTI_PR |= EXTI3_IRQ_BPOS;
```

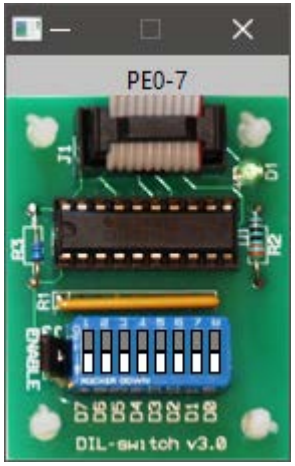
Algoritm: application init
Sätt upp PD0-7 som utport för visningsenhet
Koppla PE3 till avbrottslina EXTI3;

Konfigurera EXTI3 för att generera avbrott;
Konfigurera för avbrott enbart på negativ flank

Sätt upp avbrottsvektor
Konfigurera de bitar i NVIC som kontrollerar den avbrottslina som EXTI3 kopplats till

EXEMPEL - En enkel ”TASK-switch”

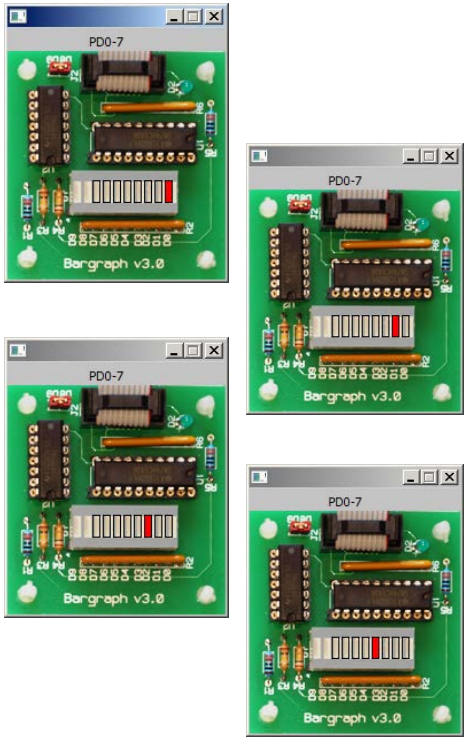
En enkel applikation med fyra olika program, (*processerna*)
p0, p1, p2 och p3,
där växling mellan programmen utförs av ett överordnat program SV (*Supervisor*) vid ett avbrott från EXTI0 (bit 0).



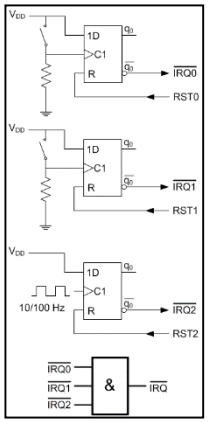
”Processerna”...

```
void p0( void )
{
  while(1)
    *GPIO_ODR_LOW = 1;
}
void p1( void )
{
  while(1)
    *GPIO_ODR_LOW = 2;
}
void p2( void )
{
  while(1)
    *GPIO_ODR_LOW = 4;
}
void p3( void )
{
  while(1)
    *GPIO_ODR_LOW = 8;
}
```

Processerna tänder var sin diod i diodrampen, vi kan då tydligt se vilken process som körs...



Processtillstånd



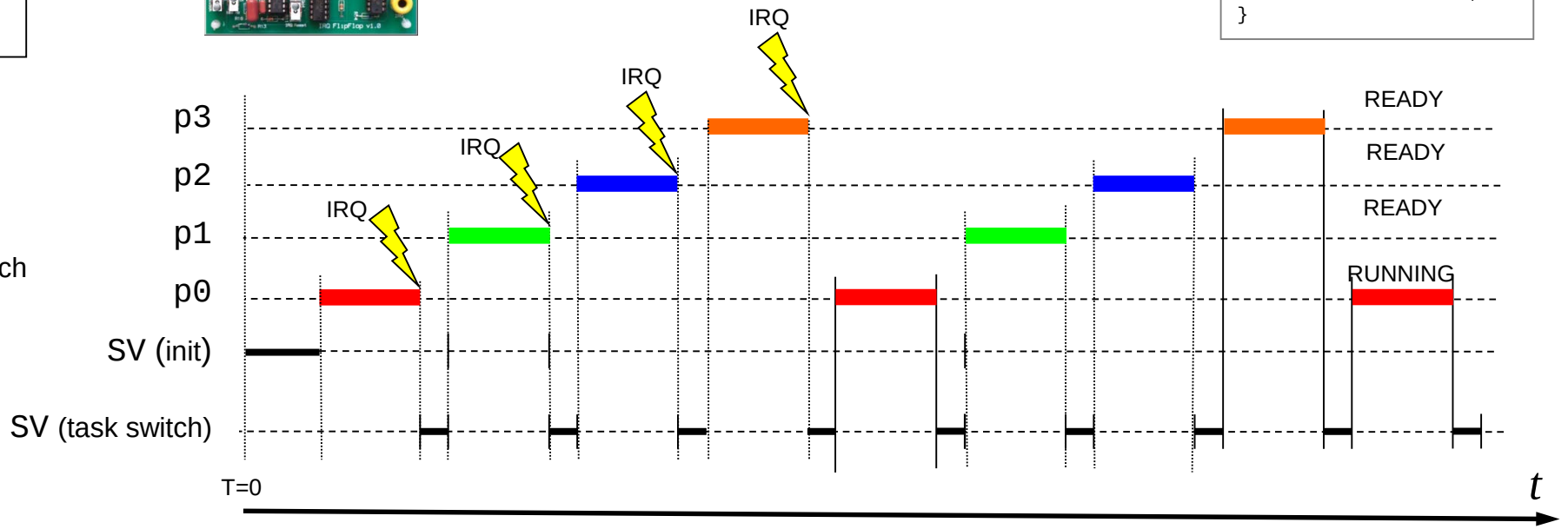
Avbrotten kan också triggas manuellt eller periodiskt från FLIP-FLOP-kortet...

Vi har bara en CPU, som vi växlar mellan processerna. Processen kan därför befinna sig i olika *tillstånd.*

"Processerna"...

```
void    p0( void )
{
  while(1)
    *GPIO_ODR_LOW = 1;
}
void    p1( void )
{
  while(1)
    *GPIO_ODR_LOW = 2;
}
void    p2( void )
{
  while(1)
    *GPIO_ODR_LOW = 4;
}
void    p3( void )
{
  while(1)
    *GPIO_ODR_LOW = 8;
}
```

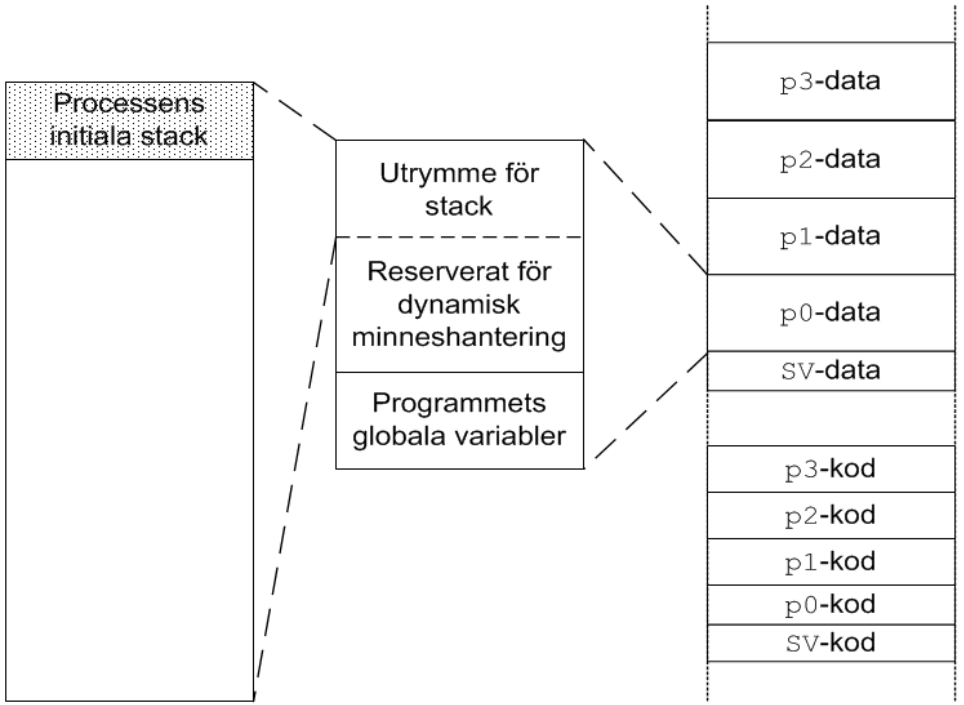
SV sköter initiering och växling



Processbyte - stackbyte

Huvudprogrammet (SV)...

```
void main(void)
{
  flipflopInit();
  taskswitchInit();
  *GPIO_MODER       = 0x00005555;
  *GPIO_OTYPER      = 0x00000000;
  *((void (**)(void) ) EXTI3_IRQVEC ) =
          irq_handler;
  p0();
  while(1){}
 }
```



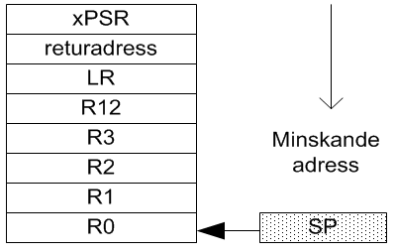
Vid avbrott: Stackbyte (SV)...

// Spara flyktig omgivning
// Spara RUNNING's stackpekare (RUNNING blir READY)
// Sätt in nästa process från READY som RUNNING
// Återställ ny process flyktiga omgivning
// Återvänd från undantagshantering

De olika processernas data-areor är separerade i minnet. Det gäller dock inte data som finns i processorns register.

Innehållet i processorns register kallas därför processens ***flyktiga omgivning***.

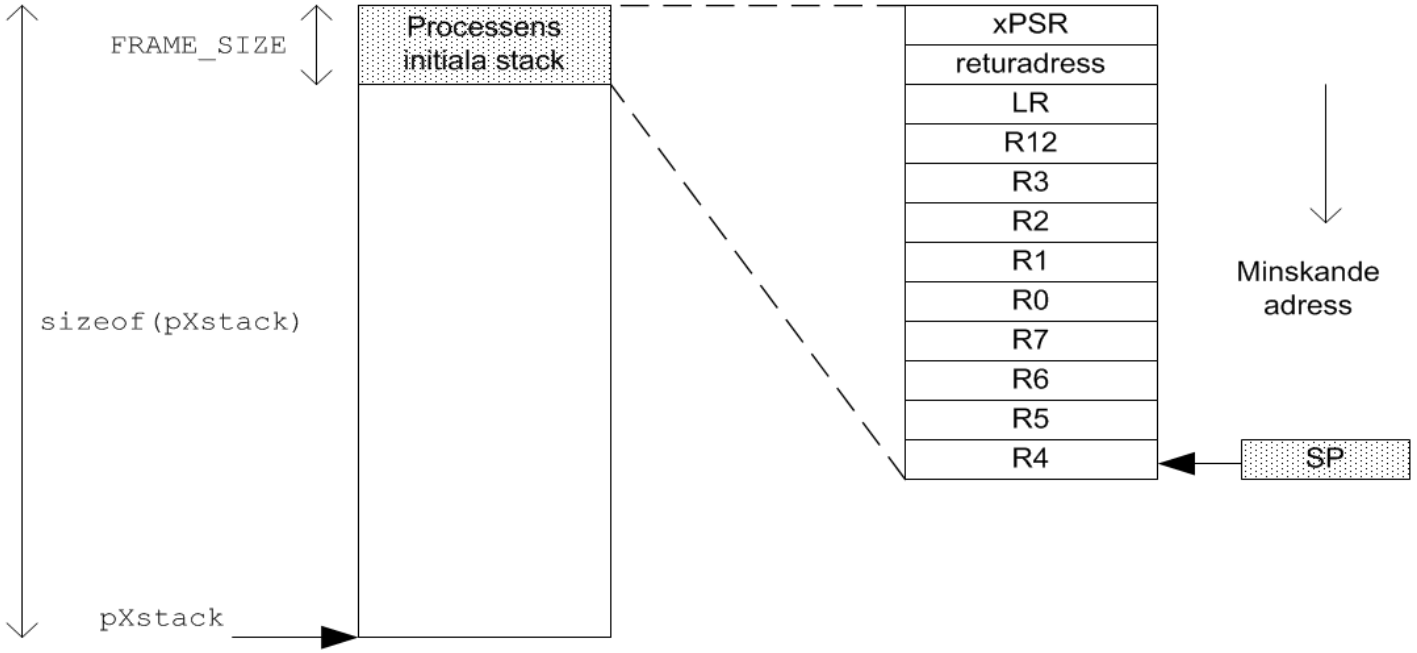
Observera att endast en del av den flyktiga omgivningen sparas automatiskt som undantagsstack, övriga register, i detta fall R4-R7, måste vi ta hand om själva...



Processens stack - med flyktig omgivning

SV datstrukturer

```
#define         STACK_SIZE     64   /* 64*4 bytes */
unsigned  int   p0stack[STACK_SIZE];
unsigned  int   p1stack[STACK_SIZE];
unsigned  int   p2stack[STACK_SIZE];
unsigned  int   p3stack[STACK_SIZE];
unsigned  int   process_stacks[4];
int             running;
```



För att kunna växla in en process första gången måste vi skapa en initial flyktig omgivning...

SV - initiering

Huvudprogrammet (SV)...

```
void main(void)
{
  flipflopInit();
  taskswitchInit();
  *GPIO_MODER          = 0x00005555;
  *GPIO_OTYPER         = 0x00000000;
  *((void (**)(void) ) EXTI3_IRQVEC ) =
              irq_handler;
  p0();
  while(1){}
 }
```

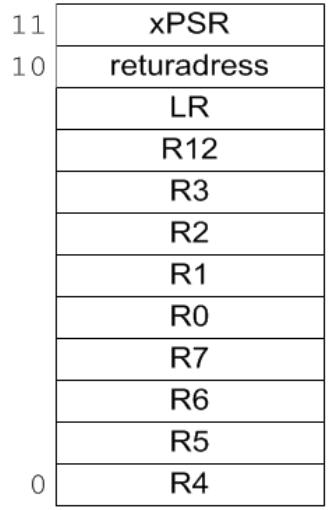
Datstrukturer

```
unsigned int process_stacks[4];
int       running;
```

Eftersom p0 startas direct av SV räcker det med att sätt upp initiala stackar för p1, p2 och p3.

```
void taskswitchInit( void )
{
     #define   FRAME_SIZE     48   // Flyktiga omgivningen...
     int       *p;
     process_stacks[1] = (int) p1stack + ( sizeof( p1stack) ) - FRAME_SIZE;
     p = (int *) process_stacks[1];
     p[10] = (unsigned int) p1;
     p[11] = 0x01000000;
     // osv. analogt för övriga processer...

     running = 0;   // Eftersom p0 startas direkt...
}
```



SV - stackbyte

Vid avbrott: Stackbyte (SV)...
// Spara flyktig omgivning
// Spara RUNNING's stackpekare (RUNNING blir READY)
// *Mellanakt*.
// Sätt in nästa process från READY som RUNNING
// Kvittera avbrott från avbrottsvippa
// Ladda stackpekare för RUNNING
// Återställ ny process flyktiga omgivning
// Återvänd från undantagshantering

Spara flyktig omgivning...

```
PUSH {R4-R7}
LDR  R1,=process_stacks
LDR  R2,=running
LDR  R2,[R2]
LSL  R2,#2
MOV  R0,SP
STR  R0,[R1,R2]
```

Mellanakt...

```
PUSH {LR}
BL   ack_irq
POP  {R0}
MOV  LR,R0
```

```
void  schedule( void )
{
  int next = running;
  switch( running )
  {
    case    0: next = 1; break;
    case    1: next = 2; break;
    case    2: next = 3; break;
    case    3: next = 0; break;
    default:      break;
  }
  running = next;
}
```

Återställ flyktig omgivning...

```
LDR  R1,=process_stacks
LDR  R2,=running
LDR  R2,[R2]
LSL  R2,#2
LDR  R0,[R1,R2]
MOV  SP,R0
POP  {R4-R7}
```

Återvänd från undantagshantering...

```
BX   LR
```

Vi demonstrerar med simulator…