Timer

- Document, calculatrice et Internet interdits
- Documentation nécessaire en annexe
- Durée : 1h30
- **Conseil : Ecrivez vos programmes au brouillon avant de les écrire au propre !**

I. Questions cours (4 points)

1. Supposons un registre REG de 8 bits dont la valeur est inconnue :
   - REG =XXXX XXXX

   En utilisant les masques, **donnez les lignes de code** permettant d'obtenir :
   - REG = XX10 XXXX

2. Quel est l'intérêt des masques ?

II. Programme clignotement (6 points) :

1. La librairie à utiliser est `LED.h` (**Annexe 1**). Ecrivez un programme complet (déclaration librairie, fonction et programme principal) qui fait clignoter les quatre LEDs. On utilisera une variable globale `unsigned int led`. Ecrivez la fonction `void inverse(void)` suivante :
- Si LEDs éteintes, on allume les LEDs ;
- Sinon on éteint les LEDs ;

Pour que le clignotement soit visible, on utilisera une boucle `for` dans le programme principal entre deux inversions : `for (i=0; i<10000000; i++);`

Le programme principal (`main`) fera appel à la fonction `inverse()` qui effectue l'inversion des LEDs.

2. Commentez votre programme.

III. Temporisation par timer et interruptions (10 points)

1. La fréquence de fonctionnement du timer est de 84MHz et `prescaler` = 10000. Quelle est la valeur de `period` pour générer une temporisation de 0,1s ?

2. Les librairies à utiliser sont : `LED.h` et `stm32f4XX.h`. A l'aide de la documentation en **Annexe 2**, écrivez un deuxième programme complet (déclaration librairies, fonctions et programme principal) qui effectue le clignotement des LEDs, cette fois avec une temporisation de 0,1s. Pour faire cela, écrivez les 3 fonctions suivantes :

- `void TIM4_Initialize(void)` qui permet d'initialiser et de démarrer le timer `TIM4`.
  Les registres à configurer sont :
  - `RCC_APB1ENR` pour activer l'horloge périphérique `RCC APB1` sur lequel est connecté le timer `TIM4`.
  - `TIM4_CR1` pour configurer tous les bits à 0 (comptage croissant, division d'horloge = 1, etc).
  - `TIM4_PSC` pour définir le prescaler du timer `TIM4`.
  - `TIM4_ARR` pour spécifier la valeur de la période.
  - `TIM4_DIER` pour activer les interruptions sur dépassement (configuration : *update interrupt*)
  - `TIM4_CR1` pour démarrer le compteur.

- `void NVIC_Initialize(void)` qui permet d'initialiser les interruptions pour le timer `TIM4`. Configurer pour cela le registre :
  - `NVIC_ISER[0]` pour que le contrôleur d'interruption autorise les interruptions du timer `TIM4`. Cette étape consiste à mettre à 1 le bit n° `X` du registre `NVIC_ISER[0]` (où `X` est le numéro d'interruption attribué au timer `TIM4` à trouver en **Annexe 3**).
    Dans le code il faut utiliser la notation `NVIC->ISER[0]` car c'est un tableau.
- `void TIM4_IRQHandler(void)` est la routine d'interruption du timer `TIM4`.
  L'interruption est indiquée par le bit `UIF` du registre `TIM4_SR` (**Annexe 2**), qui doit être testé au début (`update interrupt pending == 1`) et réinitialisé à la fin (`update interrupt pending = 0`) pour que les interruptions fonctionnent. Cette routine d'interruption doit inverser l'état de led à chaque interruption du timer (appelle fonction `inverse()`).

3. Ecrivez le programme principal `int main ()`.

4. Commentez votre programme.

I-1

```
REG &= ~(0x3 << 4);
REG |=  (0x1 << 5);
```

I-2

C'est pour mettre les bits 5 et 4 à 10.

II

```
#include "LED.h"
unsigned int led = 0;
void inverse(void){
    uint8_t num;
    if(led)
        for(num = 0; num < 4; num++)
            LED_Off(num);
    else
        for(num = 0; num < 4; num++)
            LED_On(num);
    led = !led;
}
int main(void){
    uint32_t i;
    LED_Initialize();
    while(1){
        inverse();
        for(i=0;i<10000000;i++);
    }
}
```

III-1

Period = 840

III-2,3

```
#include "LED.h"
#include "stm32f4xx.h"
void TIM4_Initialize(void){
    RCC->APB1ENR |= (1 << 2);
    TIM4->CR1 = 0;
    TIM4->PSC = 10000 - 1;
    TIM4->ARR = 840 - 1;
    TIM4->DIER |= (1 << 0);
    TIM4->CR1 = 1;
}
void NVIC_Initialize(void){
    NVIC->ISER[0] |= (1 << 30);
}
void TIM4_IRQHandler(void){
    if((TIM4->SR & (1 << 0)) == 1)
        inverse();
    TIM4->SR &= ~(1 << 0);
}
int main(void){
    LED_Initialize();
    TIM4_Initialize();
    NVIC_Initialize();
    while(1);
}
```

Annexe 1

```
/*-----------------------------------------------------------------------
 * Name:    LED.h
 *-----------------------------------------------------------------------*/

#ifndef __LED_H
#define __LED_H

#include <stdint.h>

extern void     LED_Initialize   (void);
extern void     LED_Uninitialize (void);
extern void     LED_On           (uint32_t num);
extern void     LED_Off          (uint32_t num);
extern void     LED_Out          (uint32_t val);
extern uint32_t LED_Num          (void);

#endif /* __LED_H */
```

```
/*----------------------------------------------------------------------
 * Name:    LED.c
 *----------------------------------------------------------------------*/

#include "LED.h"
#include "GPIO_STM32F4xx.h"


const GPIO_PIN_ID Pin_LED[] = {
  { GPIOD, 12 },
  { GPIOD, 13 },
  { GPIOD, 14 },
  { GPIOD, 15 }
};

#define NUM_LEDS (sizeof(Pin_LED)/sizeof(GPIO_PIN_ID))

/*----------------------------------------------------------------------
 *      LED_Initialize:  Initialize LEDs
 *
 * Parameters: (none)
 * Return:     (none)
 *----------------------------------------------------------------------*/
void LED_Initialize (void) {
  uint32_t n;

  /* Configure pins: Push-pull Output Mode (50 MHz) with Pull-down
resistors */
  for (n = 0; n < NUM_LEDS; n++) {
    GPIO_PortClock   (Pin_LED[n].port, true);
    GPIO_PinWrite    (Pin_LED[n].port, Pin_LED[n].num, 0);
    GPIO_PinConfigure(Pin_LED[n].port, Pin_LED[n].num,
                      GPIO_MODE_OUTPUT,
                      GPIO_OUTPUT_PUSH_PULL,
                      GPIO_OUTPUT_SPEED_50MHz,
                      GPIO_PULL_DOWN);
  }
}


/*----------------------------------------------------------------------
 *      LED_On: Turns on requested LED
```

```
 *
 * Parameters:  num - LED number
 * Return:      (none)
 *----------------------------------------------------------------------*/
void LED_On (uint32_t num) {
  GPIO_PinWrite(Pin_LED[num].port, Pin_LED[num].num, 1);
}

/*----------------------------------------------------------------------
 *      LED_Off: Turns off requested LED
 *
 * Parameters:  num - LED number
 * Return:      (none)
 *----------------------------------------------------------------------*/
void LED_Off (uint32_t num) {
  GPIO_PinWrite(Pin_LED[num].port, Pin_LED[num].num, 0);
}

/*----------------------------------------------------------------------
 *      LED_Num: Get number of available LEDs
 *
 * Parameters: (none)
 * Return:      number of available LEDs
 *----------------------------------------------------------------------*/
uint32_t LED_Num (void) {
  return (NUM_LEDS);
}
```

Annexe 2

6.3.15 RCC APB1 peripheral clock enable register for STM32F405xx/07xx and STM32F415xx/17xx(RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	Reser-ved	CAN2 EN	CAN1 EN	Reser-ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Reser-ved
		rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 3 **TIM5EN:** TIM5 clock enable
Set and cleared by software.
0: TIM5 clock disabled
1: TIM5 clock enabled

Bit 2 **TIM4EN:** TIM4 clock enable
Set and cleared by software.
0: TIM4 clock disabled
1: TIM4 clock enabled

Bit 1 **TIM3EN:** TIM3 clock enable
Set and cleared by software.
0: TIM3 clock disabled
1: TIM3 clock enabled

Bit 0 **TIM2EN:** TIM2 clock enable
Set and cleared by software.
0: TIM2 clock disabled
1: TIM2 clock enabled

15.4 TIM2 to TIM5 registers

Refer to *Section 1.1 on page 47* for a list of abbreviations used in register descriptions.

The 32-bit peripheral registers have to be written by words (32 bits). All other peripheral registers have to be written by half-words (16 bits) or words (32 bits). Read accesses can be done by bytes (8 bits), half-words (16 bits) or words (32 bits).

15.4.1 TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

00: $t_{DTS} = t_{CK\_INT}$
01: $t_{DTS} = 2 \times t_{CK\_INT}$
10: $t_{DTS} = 4 \times t_{CK\_INT}$
11: Reserved

Bit 7 **ARPE**: Auto-reload preload enable

0: TIMx_ARR register is not buffered
1: TIMx_ARR register is buffered

Bits 6:5 **CMS**: Center-aligned mode selection

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.
10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.
11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

Bit 4 **DIR**: Direction

0: Counter used as upcounter
1: Counter used as downcounter

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM**: One-pulse mode

0: Counter is not stopped at update event
1: Counter stops counting at the next update event (clearing the bit CEN)

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the UEV event sources.
0: Any of the following events generate an update interrupt or DMA request if enabled. These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 **CEN**: Counter enable

0: Counter disabled

1: Counter enabled

Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one-pulse mode, when an update event occurs.

15.4.11 TIMx prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to f_{CK_PSC} / (PSC[15:0] + 1).
PSC contains the value to be loaded in the active prescaler register at each update event.

15.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **ARR[15:0]**: Auto-reloadvalue

ARR is the value to be loaded in the actual auto-reload register.
Refer to the *Section 15.3.1: Time-base unit on page 423* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

15.4.4 TIMx DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TDE**: Trigger DMA request enable
0: Trigger DMA request disabled.
1: Trigger DMA request enabled.

Bit 13 Reserved, always read as 0

Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable
0: CC4 DMA request disabled.
1: CC4 DMA request enabled.

Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable
0: CC3 DMA request disabled.
1: CC3 DMA request enabled.

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable
0: CC2 DMA request disabled.
1: CC2 DMA request enabled.

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable
0: CC1 DMA request disabled.
1: CC1 DMA request enabled.

Bit 8 **UDE**: Update DMA request enable
0: Update DMA request disabled.
1: Update DMA request enabled.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **TIE**: Trigger interrupt enable
0: Trigger interrupt disabled.
1: Trigger interrupt enabled.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4IE**: Capture/Compare 4 interrupt enable
0: CC4 interrupt disabled.
1: CC4 interrupt enabled.

Bit 3 **CC3IE**: Capture/Compare 3 interrupt enable
0: CC3 interrupt disabled
1: CC3 interrupt enabled

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable
0: CC2 interrupt disabled
1: CC2 interrupt enabled

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable
0: CC1 interrupt disabled
1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable
0: Update interrupt disabled
1: Update interrupt enabled

15.4.5 TIMx status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bit 15:13 Reserved, must be kept at reset value.

Bit 12 **CC4OF**: Capture/Compare 4 overcapture flag
refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 overcapture flag
refer to CC1OF description

Bit 10 **CC2OF**: Capture/compare 2 overcapture flag
refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag
This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to ‘0.
0: No overcapture has been detected
1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **TIF**: Trigger interrupt flag
This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode. It is set when the counter starts or stops when gated mode is selected. It is cleared by software.
0: No trigger event occurred
1: Trigger interrupt pending

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4IF**: Capture/Compare 4 interrupt flag
refer to CC1IF description

Bit 3 **CC3IF**: Capture/Compare 3 interrupt flag
refer to CC1IF description

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag
refer to CC1IF description

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag
If channel CC1 is configured as output:
This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx_CR1 register description). It is cleared by software.
0: No match
1: The content of the counter TIMx_CNT matches the content of the TIMx_CCR1 register. When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)
If channel CC1 is configured as input:
This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1 register.
0: No input capture occurred
1: The counter value has been captured in TIMx_CCR1 register (An edge has been detected on IC1 which matches the selected polarity)

Bit 0 **UIF**: Update interrupt flag

- This bit is set by hardware on an update event. It is cleared by software.
  0: No update occurred.
  1: Update interrupt pending. This bit is set by hardware when the registers are updated:
- At overflow or underflow (for TIM2 to TIM5) and if UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.

When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

Annexe 3

Table 45. Vector table for STM32F405xx/07xx and STM32F415xx/17xx (continued)

Position	Priority	Type of priority	Acronym	Description	Address
16	23	settable	DMA1_Stream5	DMA1 Stream5 global interrupt	0x0000 0080
17	24	settable	DMA1_Stream6	DMA1 Stream6 global interrupt	0x0000 0084
18	25	settable	ADC	ADC1, ADC2 and ADC3 global interrupts	0x0000 0088
19	26	settable	CAN1_TX	CAN1 TX interrupts	0x0000 008C
20	27	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000 0090
21	28	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000 0094
22	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000 0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C
24	31	settable	TIM1_BRK_TIM9	TIM1 Break interrupt and TIM9 global interrupt	0x0000 00A0
25	32	settable	TIM1_UP_TIM10	TIM1 Update interrupt and TIM10 global interrupt	0x0000 00A4
26	33	settable	TIM1_TRG_COM_TIM11	TIM1 Trigger and Commutation interrupts and TIM11 global interrupt	0x0000 00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000 00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000 00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000 00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000 00B8
31	38	settable	I2C1_EV	I^2C1 event interrupt	0x0000 00BC
32	39	settable	I2C1_ER	I^2C1 error interrupt	0x0000 00C0
33	40	settable	I2C2_EV	I^2C2 event interrupt	0x0000 00C4
34	41	settable	I2C2_ER	I^2C2 error interrupt	0x0000 00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000 00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000 00D0
37	44	settable	USART1	USART1 global interrupt	0x0000 00D4
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

Table B3-9 Implemented NVIC registers, except NVIC_IPRs

ICTR.INTLINESNUM	Maximum number of interrupts	Last implemented NVIC_ISER	Corresponding interrupts
0b0000	32	NVIC_ISER0	0-31
0b0001	64	NVIC_ISER1	32-63
0b0010	96	NVIC_ISER2	64-95
0b0011	128	NVIC_ISER3	96-127
0b0100	160	NVIC_ISER4	128-159
0b0101	192	NVIC_ISER5	160-191
0b0110	224	NVIC_ISER6	192-223
0b0111	256	NVIC_ISER7	224-255
0b1000	288	NVIC_ISER8	256-287
0b1001	320	NVIC_ISER9	288-319
0b1010	352	NVIC_ISER10	320-351
0b1011	384	NVIC_ISER11	352-383
0b1100	416	NVIC_ISER12	384-415
0b1101	448	NVIC_ISER13	416-447
0b1110	480	NVIC_ISER14	448-479
0b1111	496	NVIC_ISER15	480-495

B3.4.4 Interrupt Set-Enable Registers, NVIC_ISER0-NVIC_ISER15

The NVIC_ISER*n* characteristics are:

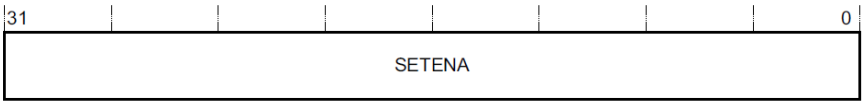
Purpose Enables, or reads the enable state of a group of interrupts.

Usage constraints NVIC_ISER*n*[31:0] are the set-enable bits for interrupts (31+(32*n*)) - (32*n*).
When *n*=15, bits[31:16] are reserved.

Configurations At least one register is always implemented, see *Implemented NVIC registers* on page B3-682.

Attributes See Table B3-8 on page B3-682.

The NVIC_ISER*n* bit assignments are:



SETENA, bits[*m*] For register NVIC_ISER*n*, enables or shows the current enabled state of interrupt (*m*+(32*n*)):

0 On reads, interrupt disabled.
On writes, no effect.

1 On reads, interrupt enabled.
On writes, enable interrupt.

m takes the values from 31 to 0, except for NVIC_ISER15, where:

- *m* takes the values from 15 to 0.
- Register bits[31:16] are reserved, RAZ/WI.

Software can enable multiple interrupts in a single write to NVIC_ISER*n*.