ECCS-3351
Embedded Realtime Applications (ERA)

Interrupts Concept

JONATHAN W. VALVANO

Modified by Drs. Kropp, Oun, and Youssfi

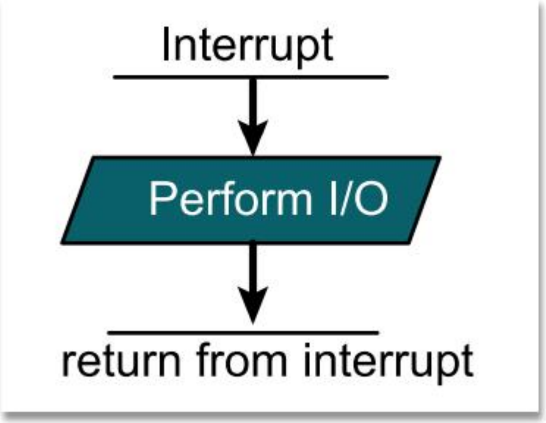
Interrupts

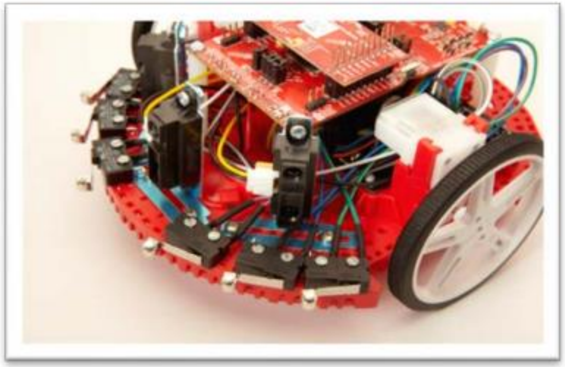
Concept

- What are interrupts?
- Why have interrupts?
- How do they work?

Implementation

- Vectors
- Priority
- Synchronization





What is an interrupt?

- Mechanism by which other modules may interrupt the normal sequencing of the processor

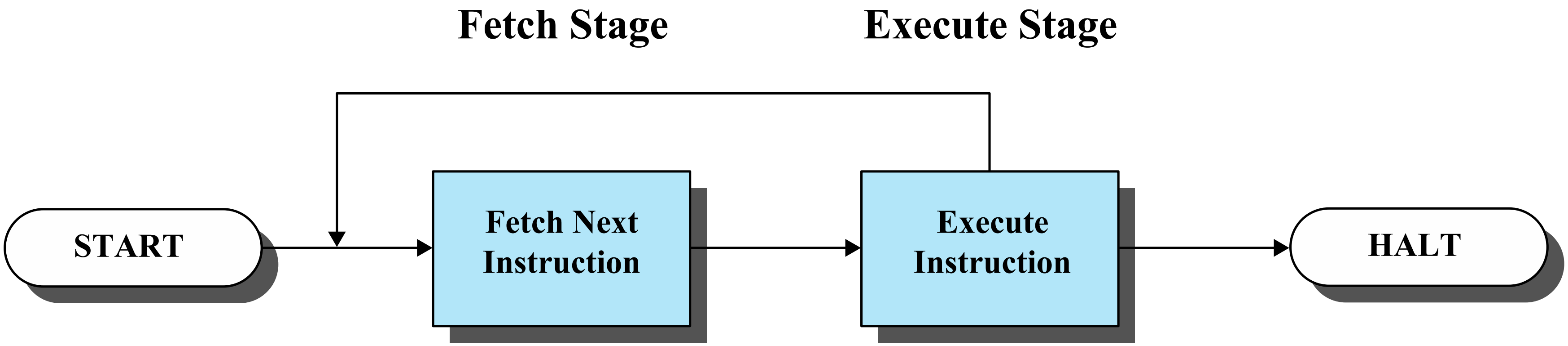
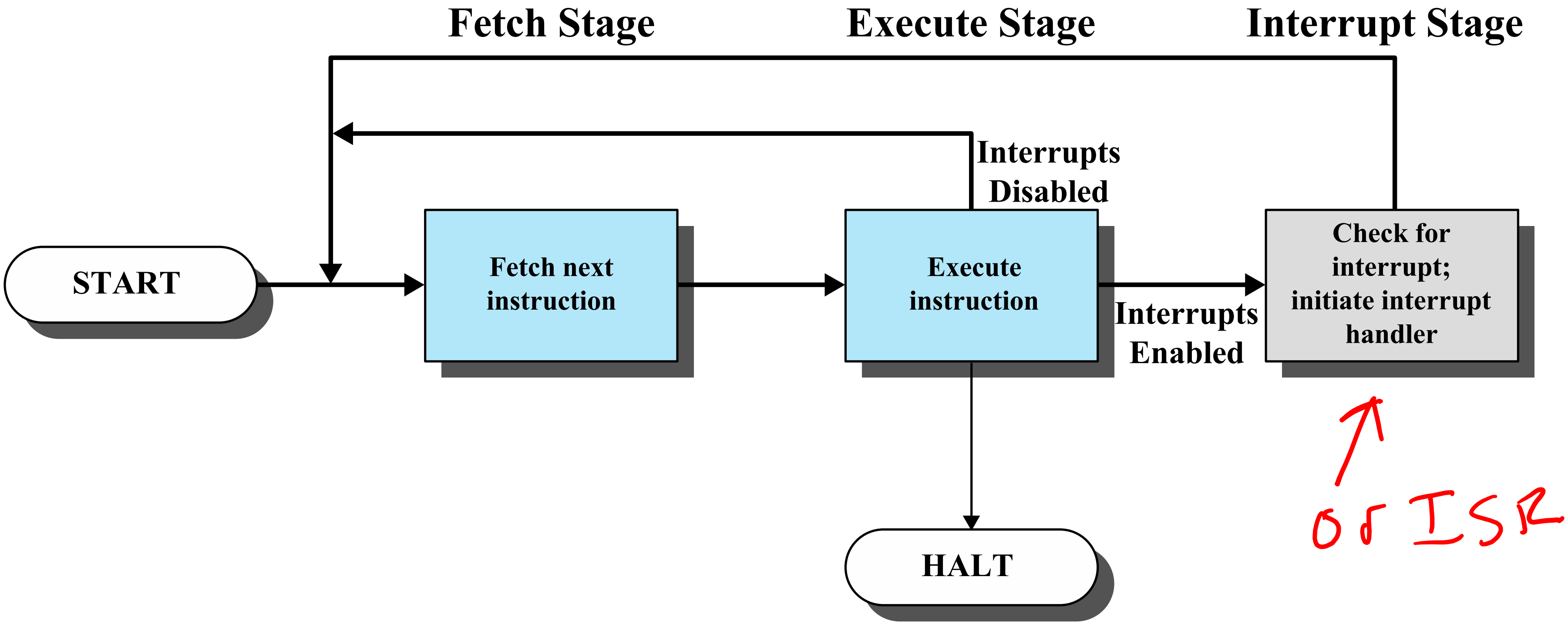


Image by William Stallings

What is an interrupt?

- Mechanism by which other modules may interrupt the normal sequencing of the processor



or ISR

Interrupts Concept

Image by William Stallings

Example uses of interrupts

- External I/O device
    - E.g., when you bump into a obstacle, turn left
    - E.g., process a key-press
- Internal event
    - E.g., Memory fault, divide by zero
- Periodic event
    - E.g., PWM,
    - E.g., switching between two concurrent tasks

Why are interrupts important

- Embedded systems need to be as low latency as possible
- Imagine if there was a delay to:
  - Hitting the breaks on a car
  - Pressing a key on an electric piano
  - A safety switch on an industrial controller
- These are *infrequent* but *important* tasks
- Interrupts allows an embedded system to respond quickly to external events

Why are interrupts important

- Embedded systems need to have:
    - Low latency
    - Fast response times
- Latency: time between request and service initiation
- Response time: time between request and service completion
- Embedded systems must guarantee a worst-cases latency and response time to certain actions

How are interrupts created

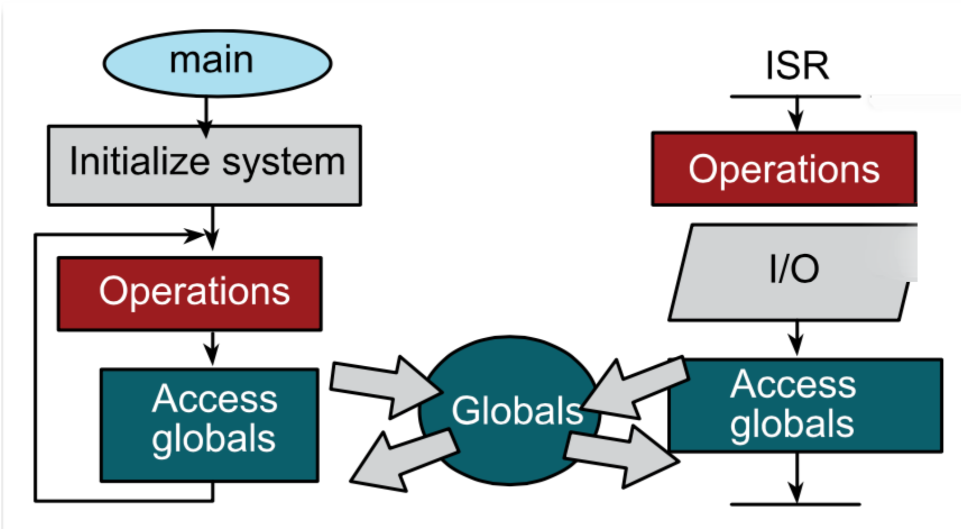
- Interrupts leverage hardware to automatically transfer software execution from one thread to another
  - Hardware event → software response
- This puts interrupt management out of the hands of software writers
- When an interrupt occurs, a new *thread* is created to handle the event (ISR)

What is a thread?

- A process is a program that is running on a CPU
- A thread is a independent sub-task of a process
- Threads independently of each other until they are joined
- Threads might run
  - physically in parallel (running on separate cores)
    - More common on desktop CPUs
  - Logically in parallel (time sharing with another thread or process on a single core)
    - More common in embedded systems
- Threads have access to the resources from the parent process

Multi-threading Using Interrupts

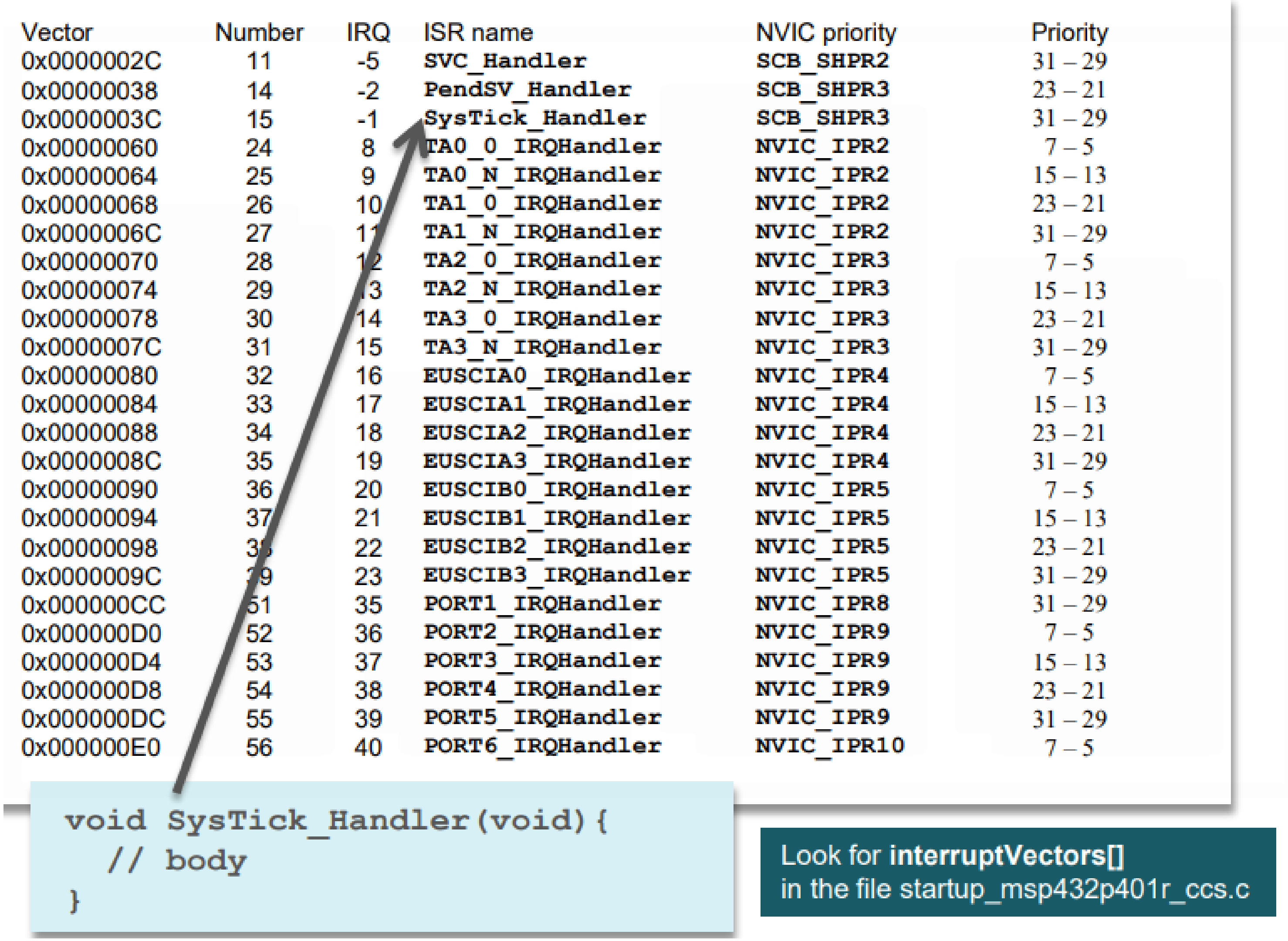
1. CPU is running the current thread
2. Interrupt occurs!
   a) Save state (on stack)
   b) Change PC (vector)
3. Run the interrupt service routine
   a) Input / Output as needed
   b) Communicate with global
   c) Return from interrupt
4. Jump back to step 1



Interrupt Vectors, numbers, priority

- Each hardware interrupt source has a unique interrupt number
- The interrupt number is used as index into interrupt vector table to find the ISR for that hardware
  - Vector table maps interrupt numbers to ISRs
- Compiler sets up the table
- E.g: SysTick has interrupt number 15
  - See startup_msp432p401r_ccs.c

Interrupt Vectors, numbers, priority

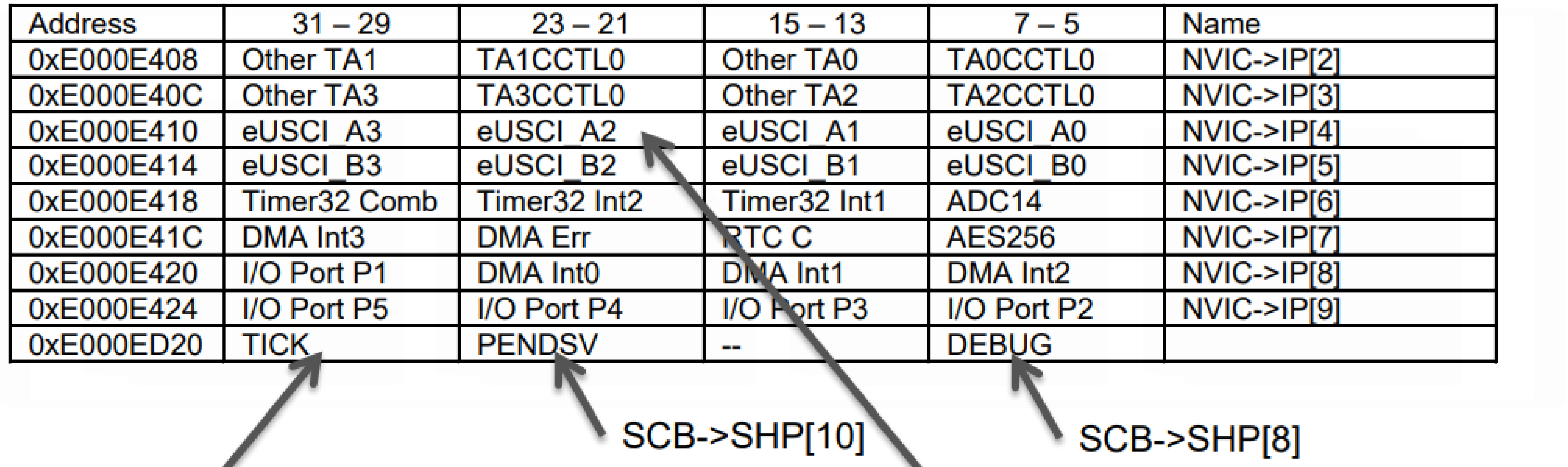


Vector	Number	IRQ	ISR name	NVIC priority	Priority
0x0000002C	11	-5	SVC_Handler	SCB_SHPR2	31 – 29
0x00000038	14	-2	PendSV_Handler	SCB_SHPR3	23 – 21
0x0000003C	15	-1	SysTick_Handler	SCB_SHPR3	31 – 29
0x00000060	24	8	TA0_0_IRQHandler	NVIC_IPR2	7 – 5
0x00000064	25	9	TA0_N_IRQHandler	NVIC_IPR2	15 – 13
0x00000068	26	10	TA1_0_IRQHandler	NVIC_IPR2	23 – 21
0x0000006C	27	11	TA1_N_IRQHandler	NVIC_IPR2	31 – 29
0x00000070	28	12	TA2_0_IRQHandler	NVIC_IPR3	7 – 5
0x00000074	29	13	TA2_N_IRQHandler	NVIC_IPR3	15 – 13
0x00000078	30	14	TA3_0_IRQHandler	NVIC_IPR3	23 – 21
0x0000007C	31	15	TA3_N_IRQHandler	NVIC_IPR3	31 – 29
0x00000080	32	16	EUSCIA0_IRQHandler	NVIC_IPR4	7 – 5
0x00000084	33	17	EUSCIA1_IRQHandler	NVIC_IPR4	15 – 13
0x00000088	34	18	EUSCIA2_IRQHandler	NVIC_IPR4	23 – 21
0x0000008C	35	19	EUSCIA3_IRQHandler	NVIC_IPR4	31 – 29
0x00000090	36	20	EUSCIB0_IRQHandler	NVIC_IPR5	7 – 5
0x00000094	37	21	EUSCIB1_IRQHandler	NVIC_IPR5	15 – 13
0x00000098	38	22	EUSCIB2_IRQHandler	NVIC_IPR5	23 – 21
0x0000009C	39	23	EUSCIB3_IRQHandler	NVIC_IPR5	31 – 29
0x000000CC	51	35	PORT1_IRQHandler	NVIC_IPR8	31 – 29
0x000000D0	52	36	PORT2_IRQHandler	NVIC_IPR9	7 – 5
0x000000D4	53	37	PORT3_IRQHandler	NVIC_IPR9	15 – 13
0x000000D8	54	38	PORT4_IRQHandler	NVIC_IPR9	23 – 21
0x000000DC	55	39	PORT5_IRQHandler	NVIC_IPR9	31 – 29
0x000000E0	56	40	PORT6_IRQHandler	NVIC_IPR10	7 – 5

```
void SysTick_Handler(void){
  // body
}
```

Look for **interruptVectors[]** in the file startup_msp432p401r_ccs.c

Interrupt Priority Registers



Address	31 – 29	23 – 21	15 – 13	7 – 5	Name
0xE000E408	Other TA1	TA1CCTL0	Other TA0	TA0CCTL0	NVIC->IP[2]
0xE000E40C	Other TA3	TA3CCTL0	Other TA2	TA2CCTL0	NVIC->IP[3]
0xE000E410	eUSCI_A3	eUSCI_A2	eUSCI_A1	eUSCI_A0	NVIC->IP[4]
0xE000E414	eUSCI_B3	eUSCI_B2	eUSCI_B1	eUSCI_B0	NVIC->IP[5]
0xE000E418	Timer32 Comb	Timer32 Int2	Timer32 Int1	ADC14	NVIC->IP[6]
0xE000E41C	DMA Int3	DMA Err	RTC C	AES256	NVIC->IP[7]
0xE000E420	I/O Port P1	DMA Int0	DMA Int1	DMA Int2	NVIC->IP[8]
0xE000E424	I/O Port P5	I/O Port P4	I/O Port P3	I/O Port P2	NVIC->IP[9]
0xE000ED20	TICK	PENDSV	--	DEBUG	

SCB->SHP[10]

SCB->SHP[8]

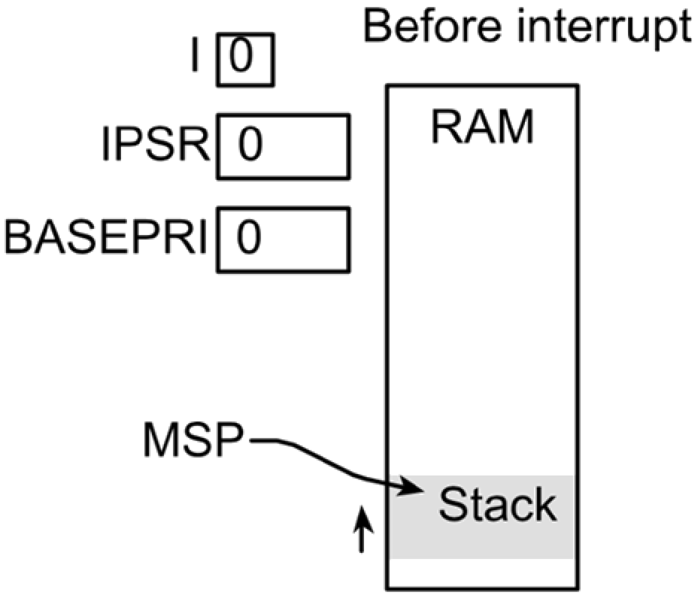
```
SCB->SHP[11] = (2)<<5;   // priority=2
```

```
NVIC->IP[4] = (NVIC->IP[4]&0xFF00FFFF)|0x00400000; // priority 2
```

Interrupts Implementation on MSP432

Hardware action: The execution of the main program is suspended

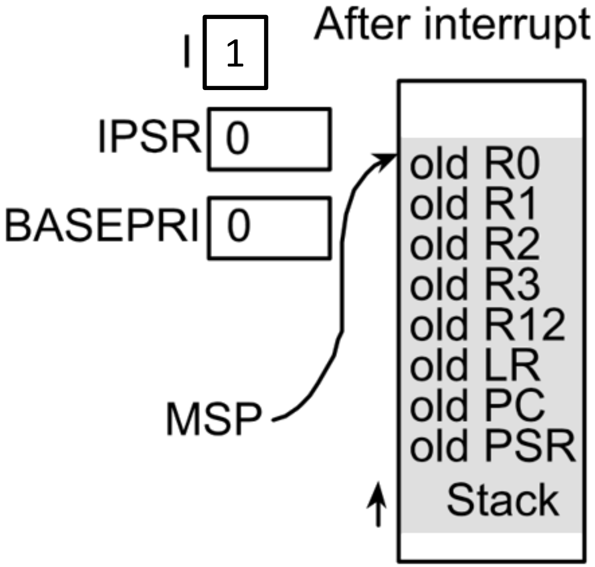
1. The current instruction is finished,



Interrupts Implementation on MSP432

Hardware action: The execution of the main program is suspended

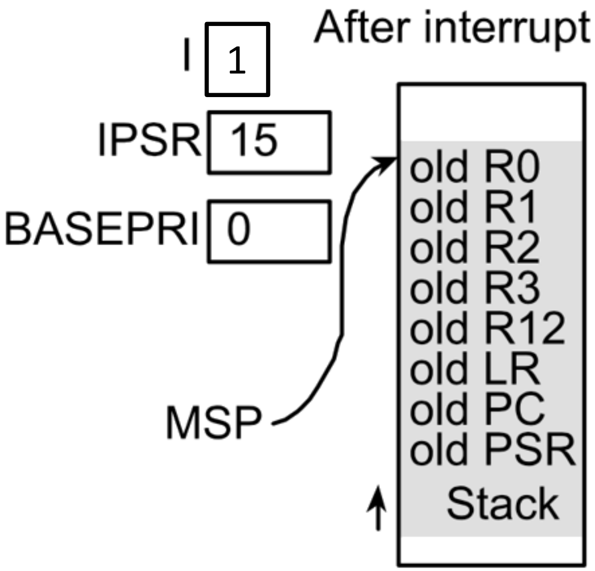
1. The current instruction is finished,
2. Suspend execution and push 8 registers on the stack
3. LR set to 0xFFFFFFF9 (indicates interrupt return) & set interrupt flag to true



Interrupts Implementation on MSP432

Hardware action: The execution of the main program is suspended

1. The current instruction is finished,
2. Suspend execution and push 8 registers on the stack
3. LR set to 0xFFFFFFF9 (indicates interrupt return) & set interrupt flag to true
4. IPSR set to interrupt number
5. Sets PC to ISR address



Interrupts Implementation on MSP432

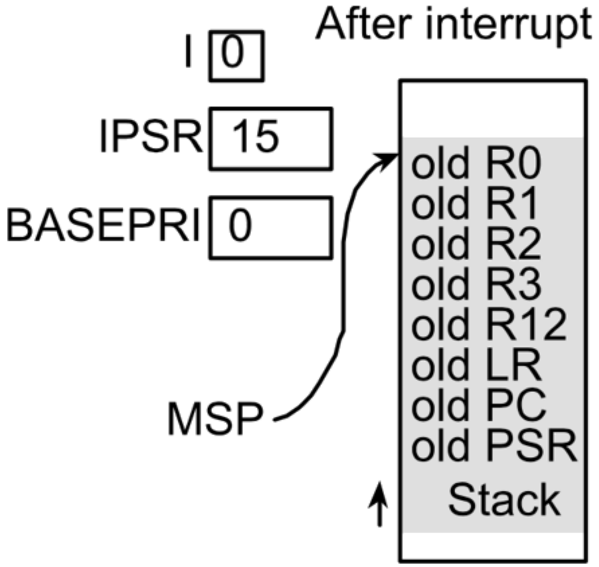
Software response: The interrupt service routine (ISR) is executed

1. Clears the flag that requested the interrupt
2. Performs necessary operations
3. Communicates using global variables

Interrupts Implementation on MSP432

Return from Interrupt: The main program is resumed when ISR returns (BX LR)

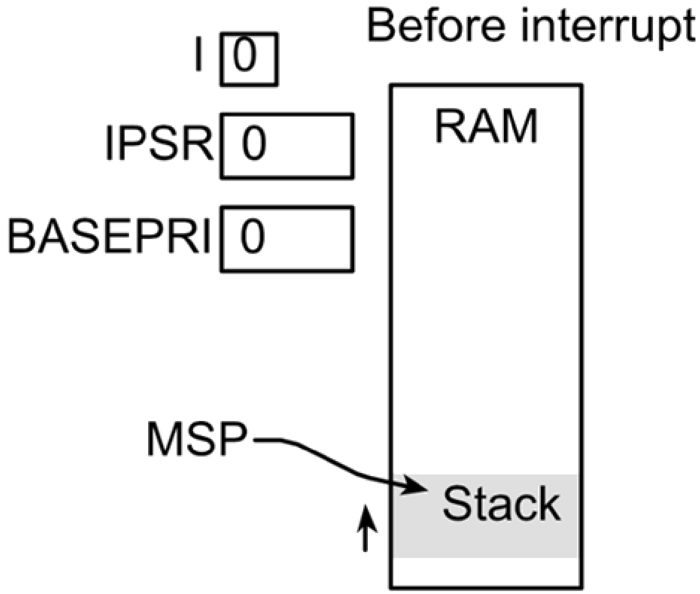
1. Pops the 8 registers from the stack



Interrupts Implementation on MSP432

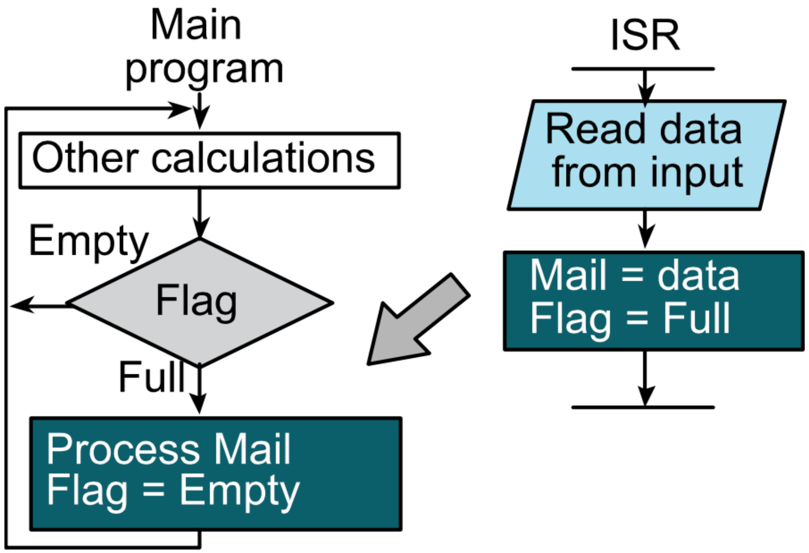
Return from Interrupt: The main program is resumed when ISR returns (BX LR)

1. Pops the 8 registers from the stack
2. Original PC is restored
3. IPSR is set to 0



Thread Synchronization

- Semaphore
  - One thread sets the flag
  - The other thread waits for, and clears
- Mailbox (semaphore plus data)
- FIFO queue (data streaming)



Periodic Interrupts

Data acquisition

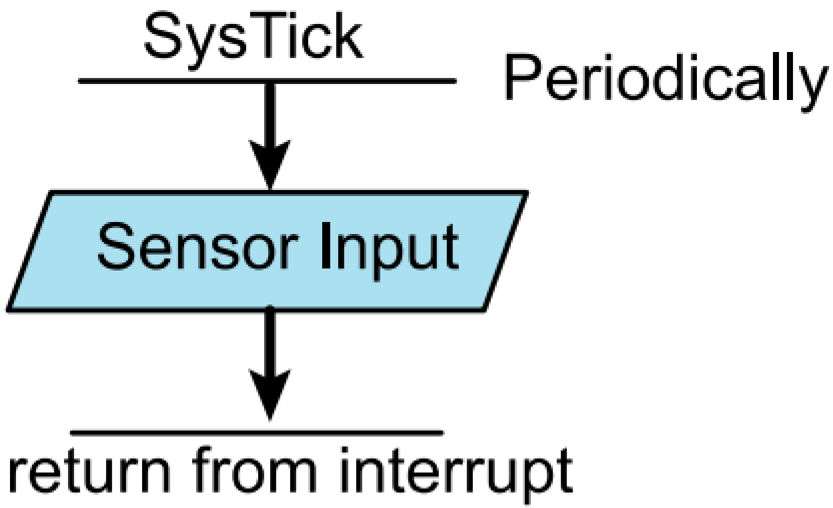
- Sample sensor data at a fixed rate
- Sample ADC at a fixed rate

Signal generation output

- Send to DAC at a fixed rate (audio)
- Transmit messages at a fixed rate

Digital controller

- FSM
- Linear control system (motor controllers)



Where to put the data?

- Global/static variable
- Array
- Mailbox (variable, flag)
- Put FIFO

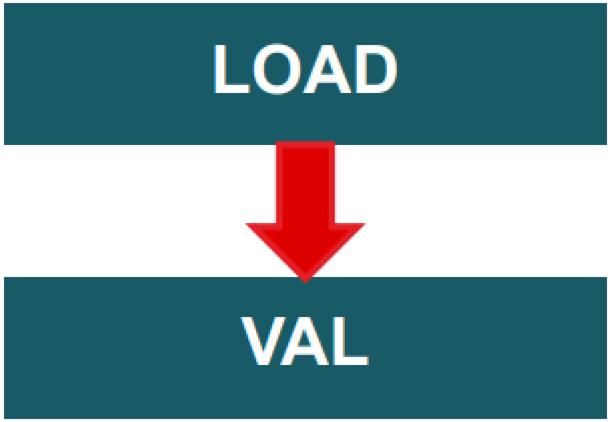
SysTick Timer

SysTick performs Timer/Counter operation in all ARM

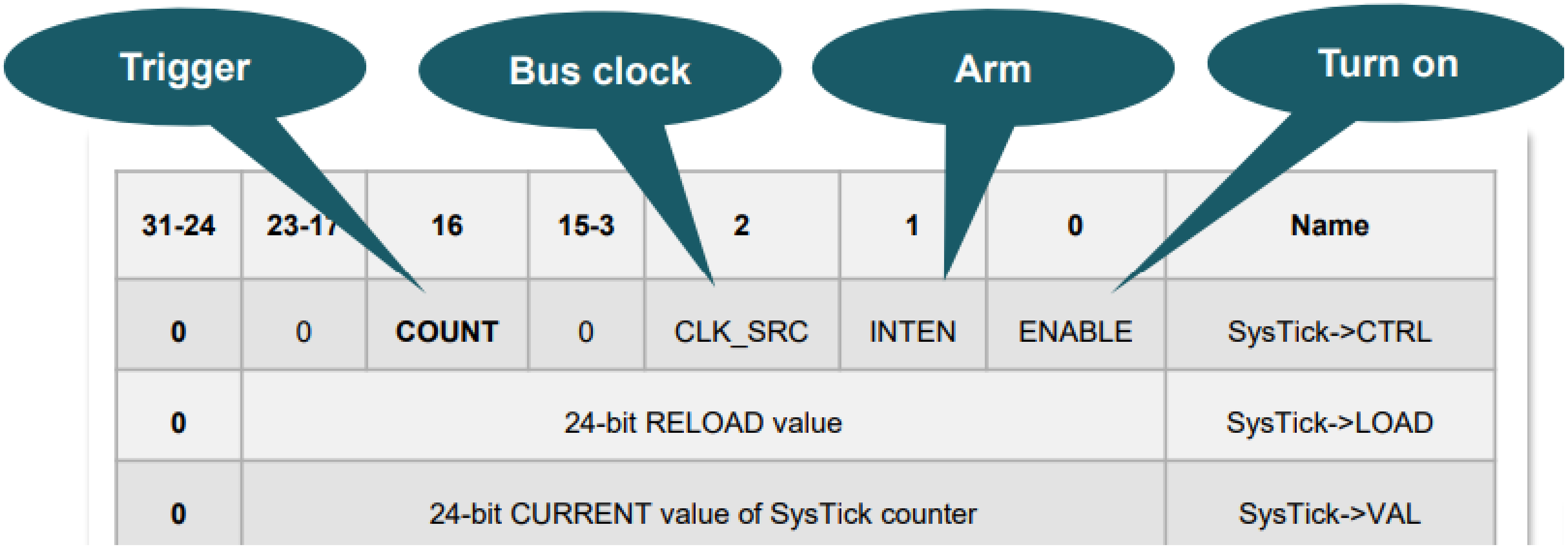
- Create time delays
- Generate periodic interrupts

How it works

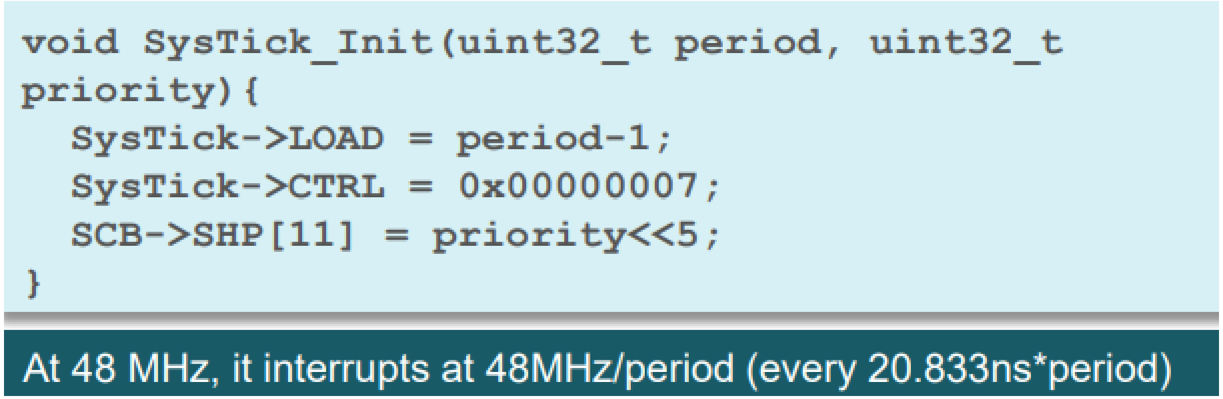
- 24-bit down counter decrements at bus clock frequency
- With a 48 MHz bus clock, decrements every 20.833 ns
- Software sets a 24-bit LOAD value of n
- The counter, VAL, goes from n → 0
    - Sequence: n, n-1, n-2, n-3… 2, 1, 0, n, n-1…
- SysTick is a modulo n+1 counter:
- VAL = (VAL - 1) mod (n+1)



SysTick Timer Initialization

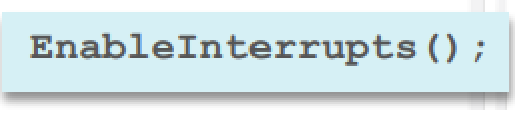

Trigger
Bus clock
Arm
Turn on

31-24	23-17	16	15-3	2	1	0	Name
0	0	**COUNT**	0	CLK_SRC	INTEN	ENABLE	SysTick->CTRL
0	24-bit RELOAD value						SysTick->LOAD
0	24-bit CURRENT value of SysTick counter						SysTick->VAL



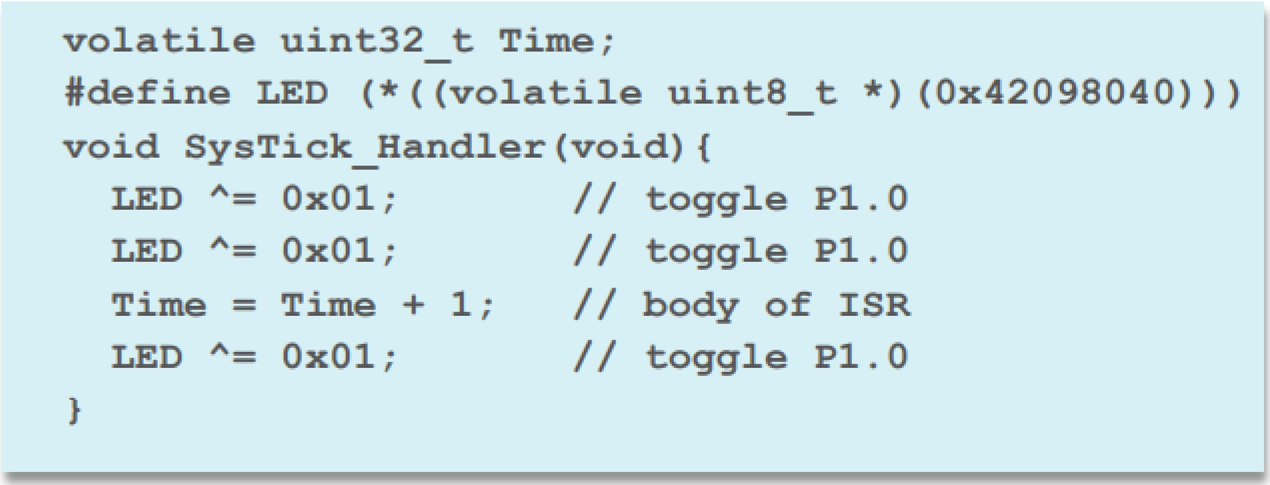
```
void SysTick_Init(uint32_t period, uint32_t
priority){
  SysTick->LOAD = period-1;
  SysTick->CTRL = 0x00000007;
  SCB->SHP[11] = priority<<5;
}
```

At 48 MHz, it interrupts at 48MHz/period (every 20.833ns*period)

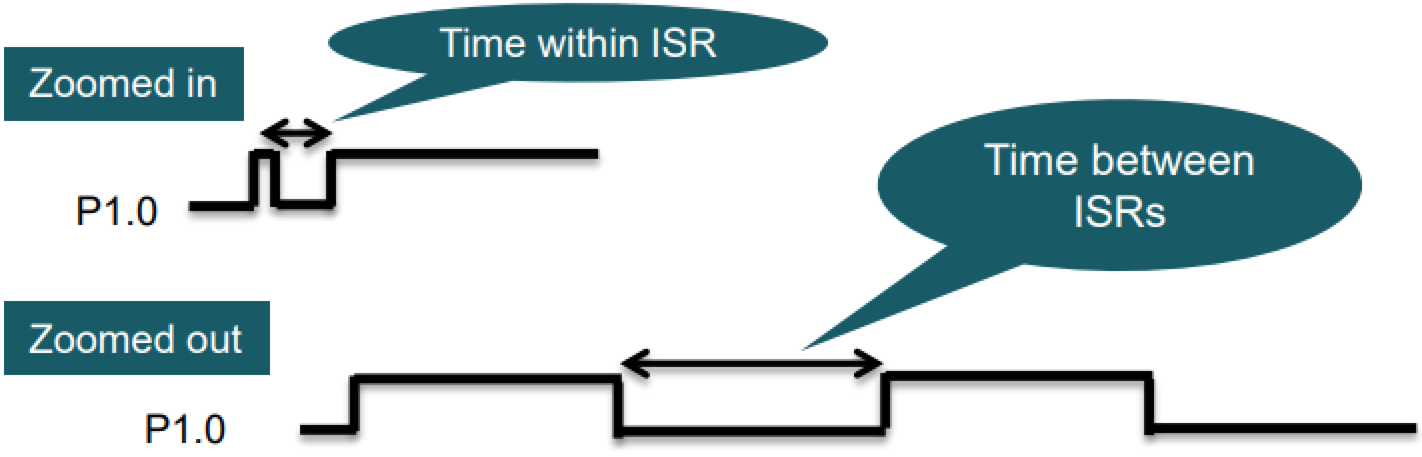


```
EnableInterrupts();
```

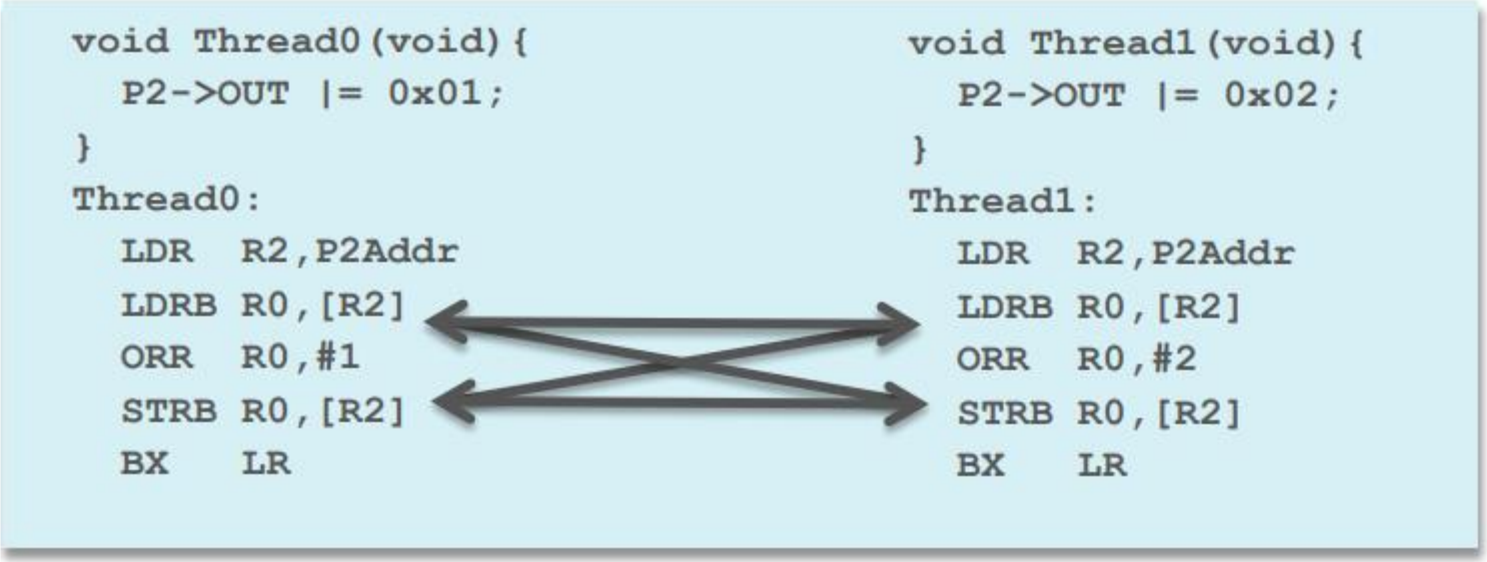
SysTick Interrupt Service Routine (ISR)



```
volatile uint32_t Time;
#define LED (*((volatile uint8_t *)(0x42098040)))
void SysTick_Handler(void){
  LED ^= 0x01;         // toggle P1.0
  LED ^= 0x01;         // toggle P1.0
  Time = Time + 1;     // body of ISR
  LED ^= 0x01;         // toggle P1.0
}
```



Critical Section



```
void Thread0(void){
  P2->OUT |= 0x01;
}
Thread0:
  LDR   R2,P2Addr
  LDRB  R0,[R2]
  ORR   R0,#1
  STRB  R0,[R2]
  BX    LR
```

```
void Thread1(void){
  P2->OUT |= 0x02;
}
Thread1:
  LDR   R2,P2Addr
  LDRB  R0,[R2]
  ORR   R0,#2
  STRB  R0,[R2]
  BX    LR
```

Nonatomic sequence

Shared global

Read-modify-write, write-write, write-read

Solutions

- Move to different port
- Bit-banding
- Disable, access, reenable