TI-RSLK

Texas Instruments Robotics System Learning Kit

Texas Instruments

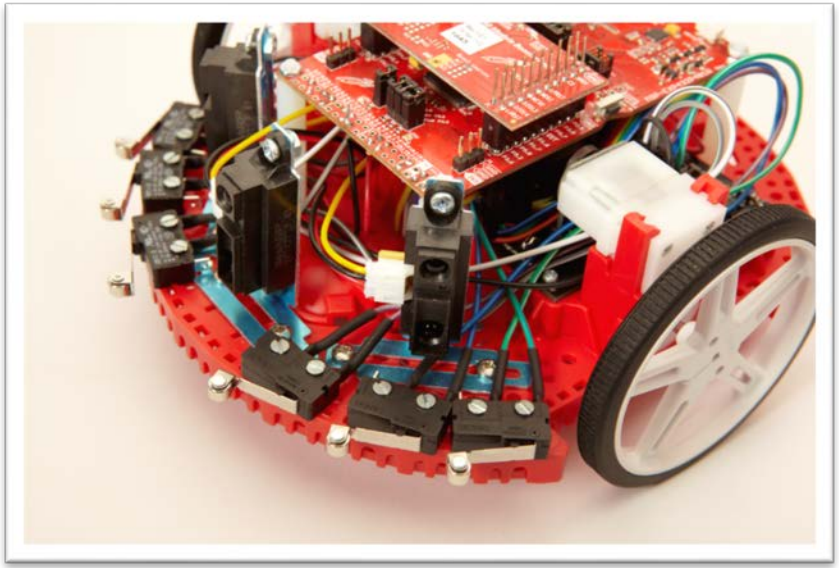
Module 10

Lecture: Debugging Real-time Systems - Theory

Debugging Real-time Systems

You will learn in this module

- How to implement minimally intrusive debugging tools
  - Dump into programming array
  - Toggle pins
- Execute profiling
  - Scope or logic analyzer
  - Observing assembly language
- Use flash ROM to record
  - Erase ROM
  - Write block



Dump Instrument

Intrusiveness is the measure to which the debugging itself affects the parameter being measured

- Short execution
  - Let t be the time to execute dump instrument
  - Let Δt be the time between executions
- Small percentage
  - Minimally intrusive if t/Δt is small

Dump

- Similar usage as **printf**
- Save into array (or into flash ROM)
- Observe later with debugger

```
#define SIZE 100
uint8_t P1Buf[SIZE];
uint8_t P2Buf[SIZE];
uint32_t I;
void Dump(void){
  if(I < SIZE){
    P1Buf[I] = P1->IN;
    P2Buf[I] = P2->OUT;
    I++;
  }
}
```

Once and stop

```
start = SysTick->VAL;
Dump(); // from lecture slide
stop = SysTick->VAL;
dT = 0x00FFFFFF&(start-stop)-11;
```

```
Dump:
00000b08:  48A2    ldr    r0, [pc, #0x288]
00000b0a:  6800    ldr    r0, [r0]
00000b0c:  2864    cmp    r0, #0x64
00000b0e:  D20F    bhs    $C$L1
00000b10:  49A0    ldr    r1, [pc, #0x280]
00000b12:  48C6    ldr    r0, [pc, #0x318]
00000b14:  4AC4    ldr    r2, [pc, #0x310]
00000b16:  6809    ldr    r1, [r1]
00000b18:  7800    ldrb   r0, [r0]
00000b1a:  5450    strb   r0, [r2, r1]
00000b1c:  499D    ldr    r1, [pc, #0x274]
00000b1e:  48C5    ldr    r0, [pc, #0x314]
00000b20:  4AC3    ldr    r2, [pc, #0x30c]
00000b22:  6809    ldr    r1, [r1]
00000b24:  7800    ldrb   r0, [r0]
00000b26:  5450    strb   r0, [r2, r1]
00000b28:  499A    ldr    r1, [pc, #0x268]
00000b2a:  6808    ldr    r0, [r1]
00000b2c:  1C40    adds   r0, r0, #1
00000b2e:  6008    str    r0, [r1]
$C$L1:
00000b30:  4770    bx     lr
```

22 instructions

73 cycles, 1.5 us

Dump Instrument

Continuous

- Saves the last 32 values
- Wrap index

```
uint16_t Buf[32];
uint32_t I=0;
void Record(uint16_t x){
  Buf[I] = x;
  I = (I+1)&0x1F;
}
```

Continuous

Filter

- Save only on certain conditions
- Reduces the volume of data to observe

```
void Record2(uint16_t x){
  if(P1->IN&0x01){
    Buf[I] = x;
    I = (I+1)&0x1F;
  }
}
```

Filtered

Execution profile

Performance debugging

- Where is it executing?
- When is it executing?
- How long does it take?

```
void main(void){
  LaunchPad_Init();
  Debug_Init();
  while(1){
    P2->OUT |= 0x01;
    Debug_Dump();
    P2->OUT &= ~0x01;
  }
}
```

```
void Happy(void){
  P2->OUT |= 0x04;
// body
  P2->OUT &= ~0x04;
}
void Sad(void){
  P2->OUT ^= 0x08;
// body
  P2->OUT &= ~0x08;
}
```

Execution profile

Eliminate the critical section

- Read-modify-write to shared global
- Bit-banding

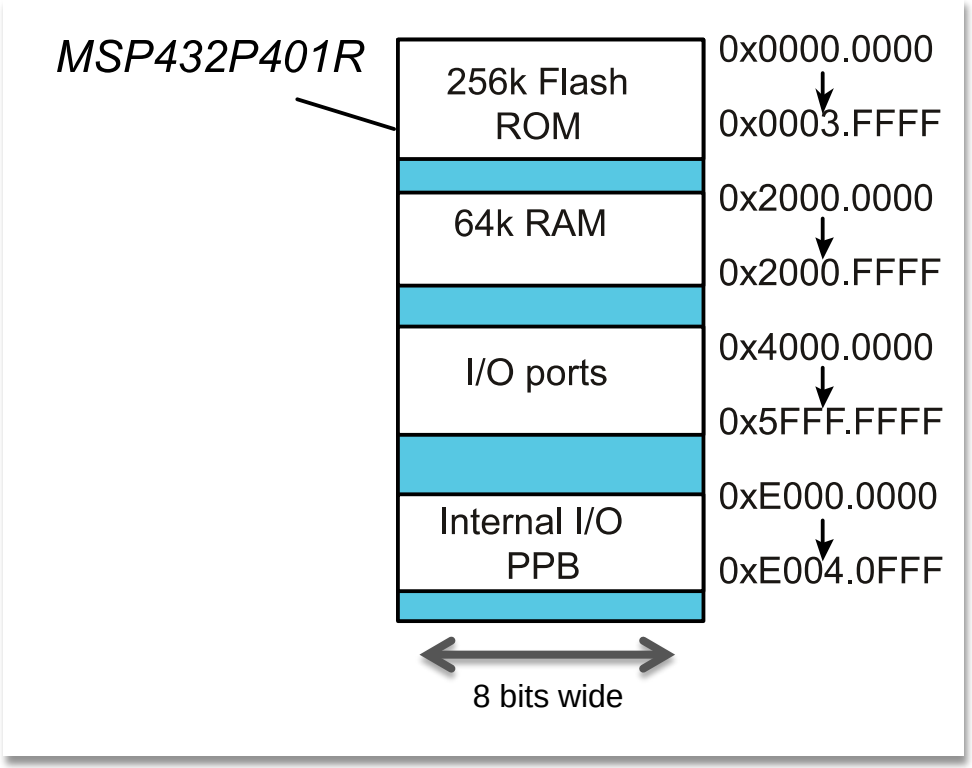
Profiling

- Toggle an output port
  - Placed at strategic places
- Use scope or logic analyzer

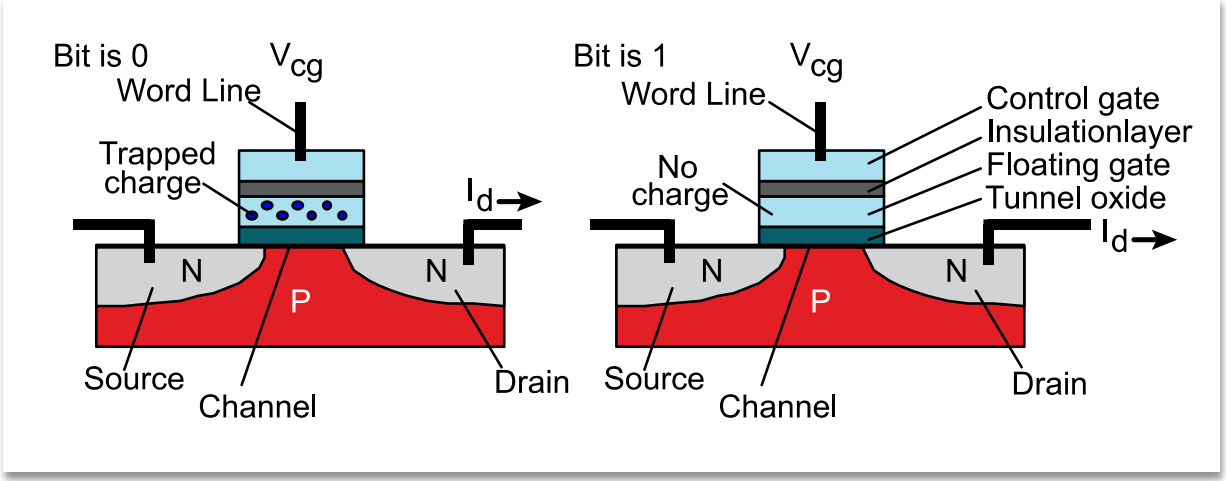
P2->OUT is 0x40004C03, *n*=0x4C03 and *b*=2.
0x42000000 + 32*0x4C03 + 4*2 = 0x42098068

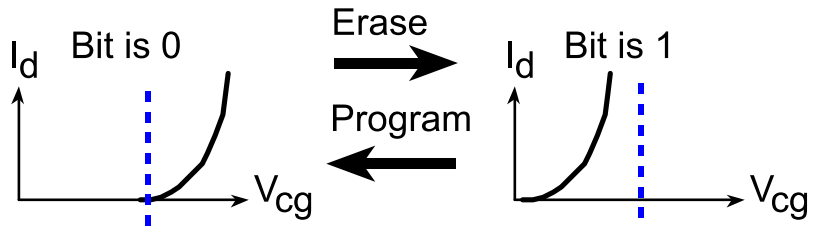
```
#define LED (*((volatile uint8_t *)(0x42098068)))
void ISR(void){
 LED ^= 1;
 LED ^= 1;
 // body
 LED ^= 1;
}
```

Memory Map


MSP432P401R
256k Flash ROM
64k RAM
I/O ports
Internal I/O PPB
0x0000.0000
0x0003.FFFF
0x2000.0000
0x2000.FFFF
0x4000.0000
0x5FFF.FFFF
0xE000.0000
0xE004.0FFF
8 bits wide

Flash ROM





Flash ROM

- High density
- Erase to 0xFF
- Program zeros
- Slow to erase
- Slow to program
- Fast to read

Flash ROM driver

0x00020000-0x0003F000

```
// Erase 4K block of flash
// Parameter 'addr' must be in flash Bank 1
// Input: addr 4K aligned flash address to erase
// Output: 0 if successful, 1 if fail
int Flash_Erase(uint32_t addr);
```

```
// Parameter 'addr' must be in flash Bank 1
// Input: source pointer to array of 32-bit data
//        addr   flash address to start writing
//        count  number of 32-bit writes
// Output: number of successful writes
int Flash_WriteArray(uint32_t *source,
       uint32_t addr, uint16_t count);
```

Summary

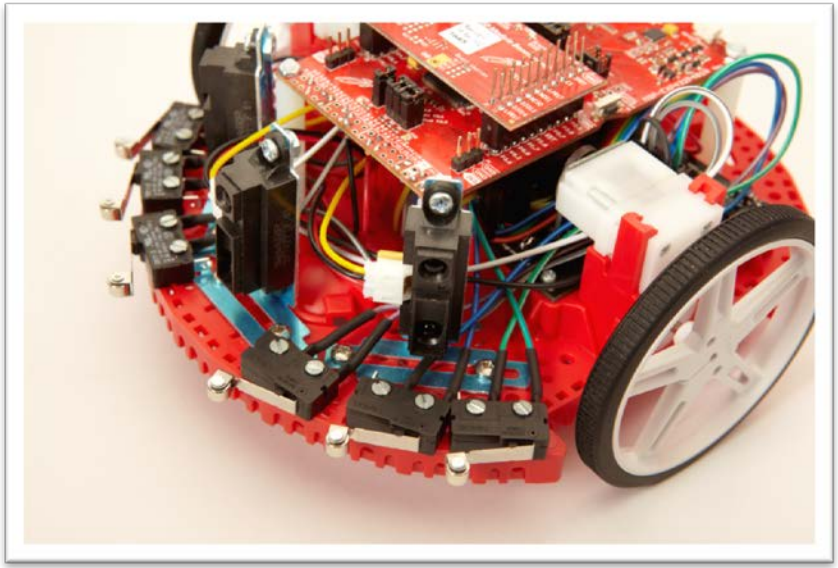
Minimally Intrusive Debugging

- Dump
- Thread profile

```
start = SysTick->VAL;
AnySoftware();
stop = SysTick->VAL;
dT = 0x00FFFFFF&(start-stop)-11;
```

Flash

- Erase
- Program
- Read



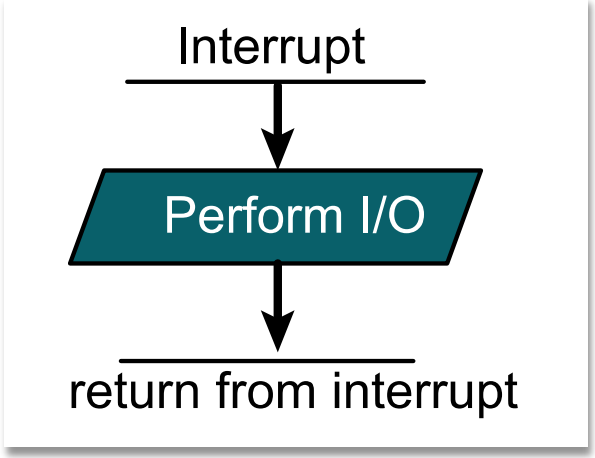
Module 10

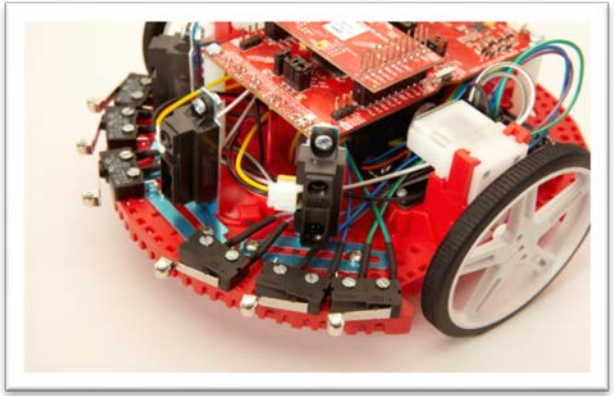
Lecture: Debugging Real-time Systems - Interrupts

Debugging Real-time Systems

You will learn in this module

- Interrupts
  - What
  - Why
  - How
- Vectors
- Priority
- Thread synchronization





Interrupts to implement concurrent execution (multi-threading)

What is an interrupt?

- Automatic transfer of software execution
- In response to a hardware event, hardware trigger → software response
- Asynchronous with current software execution

Example uses of interrupts

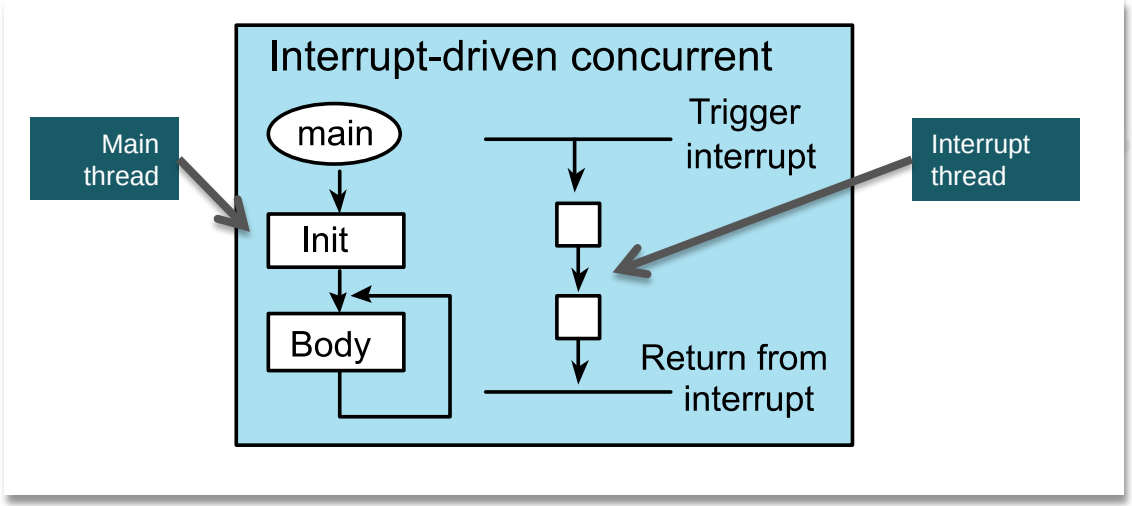
- External I/O device (like a bump sensor or motor overload)
- Internal event (like a memory fault, software trap)
- Periodic event (using a timer)

When to interrupt?

- Hardware needs service
- New input data
- Output idle
- Periodically (SysTick)

Why to use interrupts?

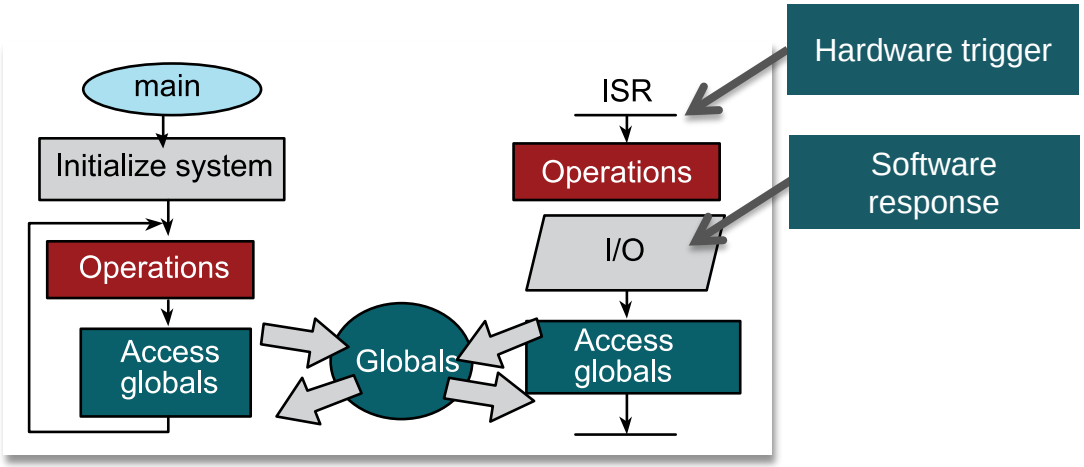
- Complex system
- Responsiveness to events
- Infrequent but important tasks



Multi-threading using Interrupts

- Running the main program
- Interrupt on external or internal event
  - Save state (on stack)
  - Change PC (vector)
- Run the interrupt service routine
  - Input/Output as needed
  - Communicate with globals
  - Return from interrupt
- Resume the main program

Thread is the action caused by executing software



Interrupt processing

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)
4. IPSR set to interrupt number
5. Sets PC to ISR address

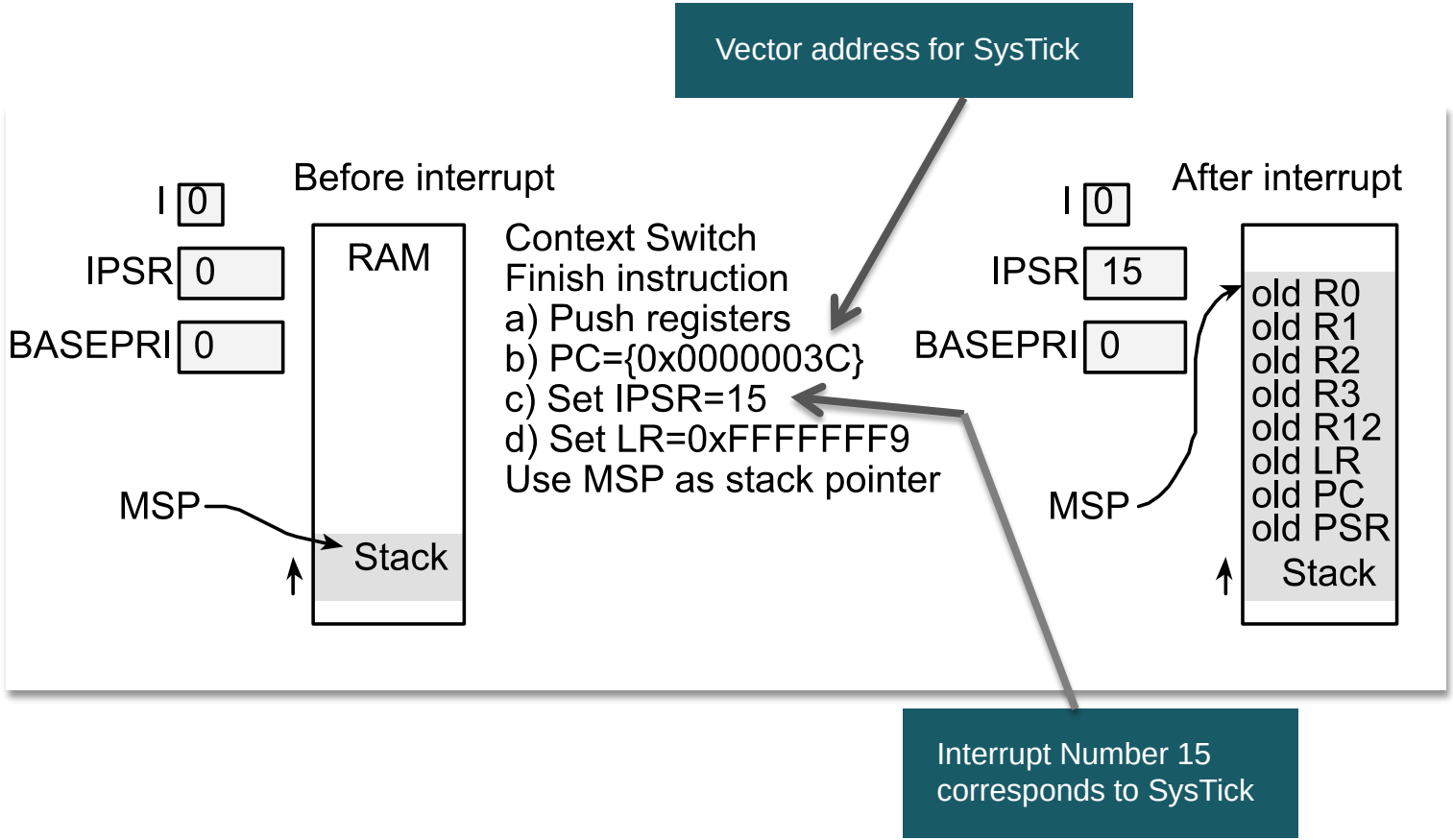
The interrupt service routine (ISR) is executed

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

The main program is resumed when ISR returns (BX LR)

1. Pulls the 8 registers from the stack

Interrupt processing



Interrupt Vectors, numbers, names, and 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

High order three bits of each byte define priority

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]

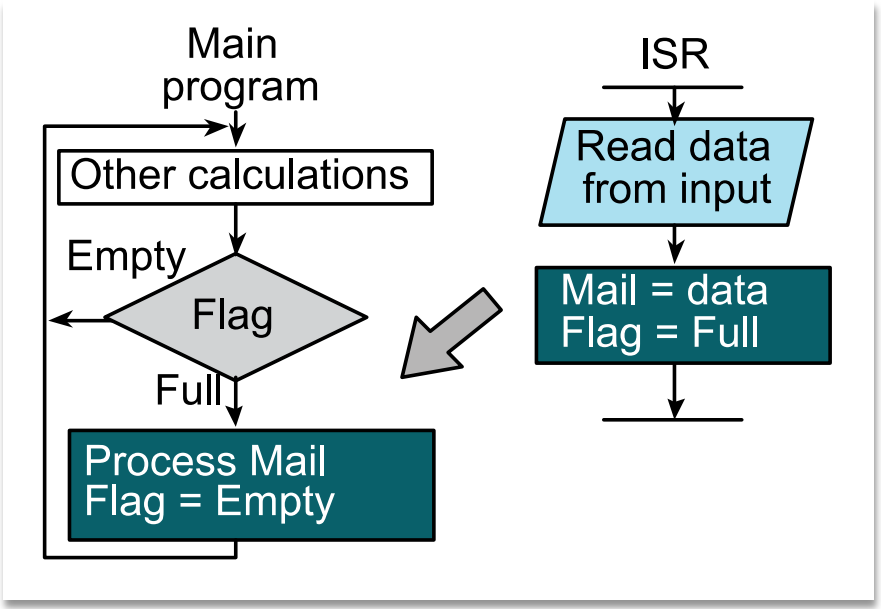
```
SCB->SHP[11] = (2)<<5;  // priority=2
```

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

Thread Synchronization

- Semaphore
  - One thread sets the flag
  - The other thread waits for, and clears

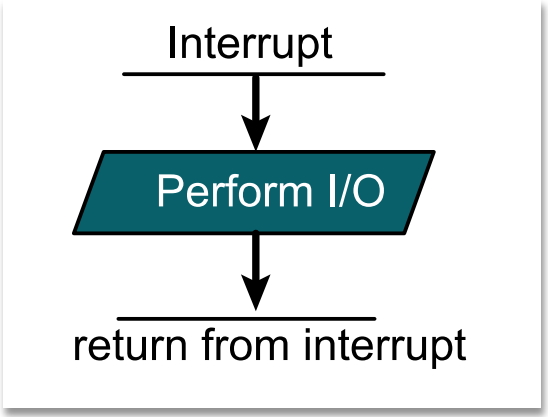
- Mailbox (semaphore plus data)
- FIFO queue (data streaming)

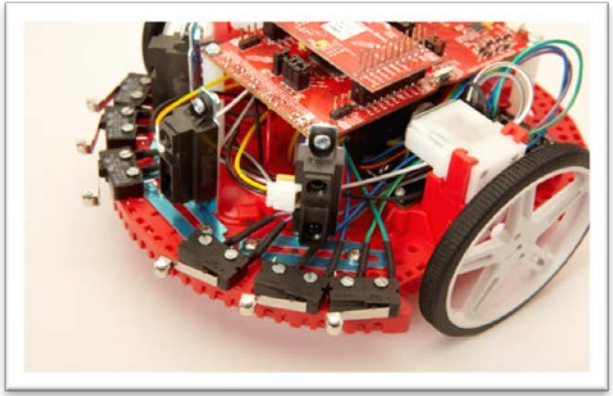


Summary

Interrupts

- Context switch (stack)
- Vector
- Initialization
  - Arm (device specific)
  - Priority
  - Enable (I bit)
- Synchronization
  - Global variables
  - Static variable





Module 10

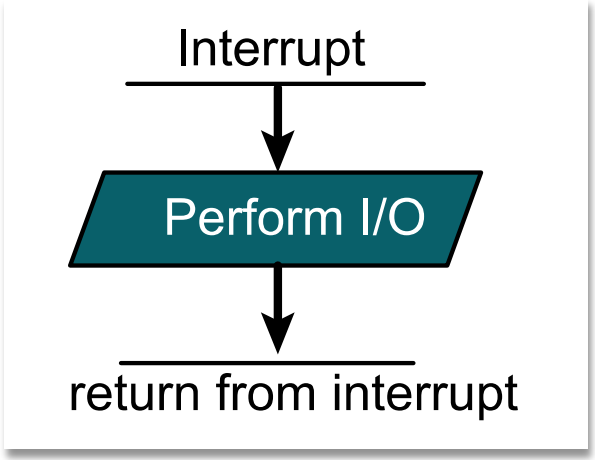
Lecture: Debugging Real-time Systems – SysTick Interrupt

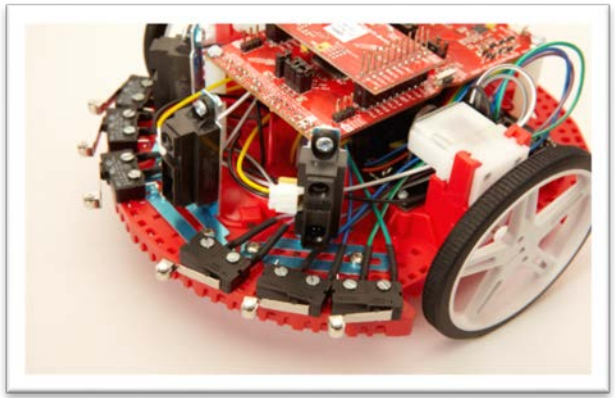
Debugging Real-time Systems

You will learn in this module

- Use SysTick to execute periodic tasks
  - Fundamentals
  - Initialization
  - Interrupt service routine

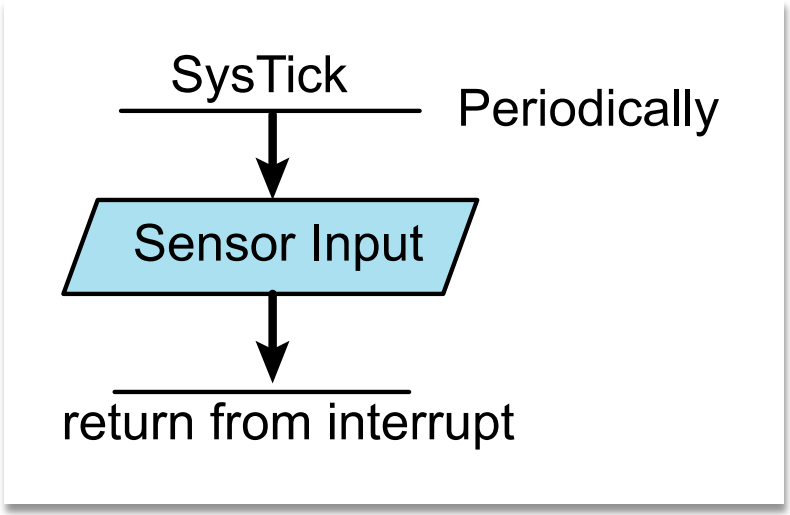
- Applications
  - Sample sensors at 100 Hz
  - Signal generation
  - Interface line sensor without wasting time
  - Digital controller





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

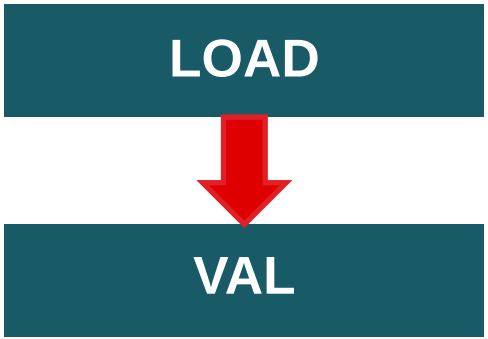
SysTick Timer (review)

SysTick performs Timer/Counter operation in all ARM

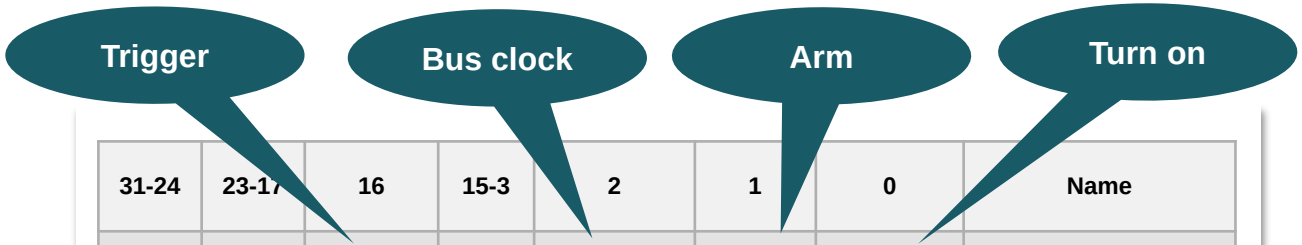
- Create time delays
- **Generate periodic interrupts**

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



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

Table 9.0 SysTick Registers

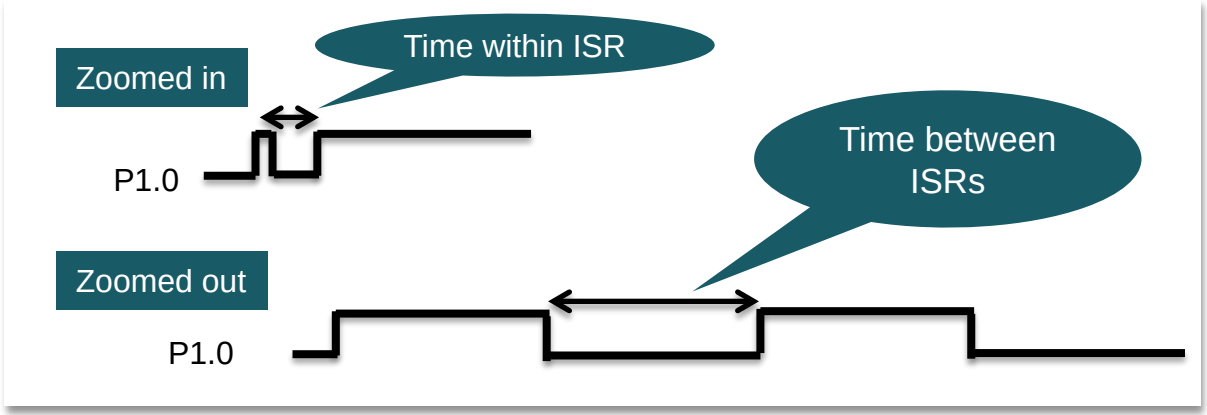
```
void SysTick_Init(uint32_t period, uint32_t
priority){
  SysTick->LOAD = period-1;
  SysTick->CTRL = 0x00000007;
  SCB->SHP[11] = priority<<5;
}
```

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

```
EnableInterrupts();
```

SysTick Interrupt Service Routine (ISR)

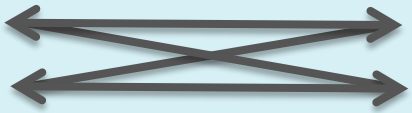
```
volatile uint32_t Time;
#define LED (*((volatile uint8_t *)(0x42098040)))
void SysTick_Handler(void){
  LED ^= 0x01;       // toggle P1.0
  LED ^= 0x01;       // toggle P1.0
  Time = Time + 1;   // body of ISR
  LED ^= 0x01;       // toggle P1.0
}
```



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

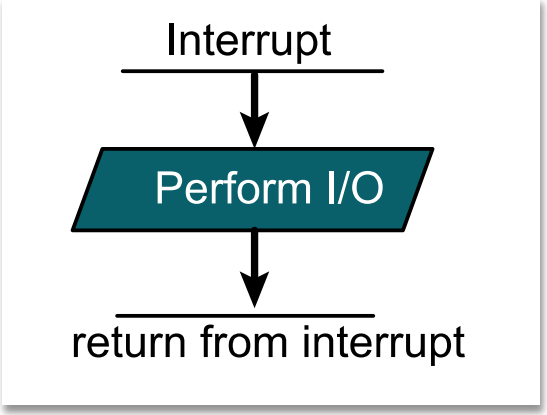
Solutions

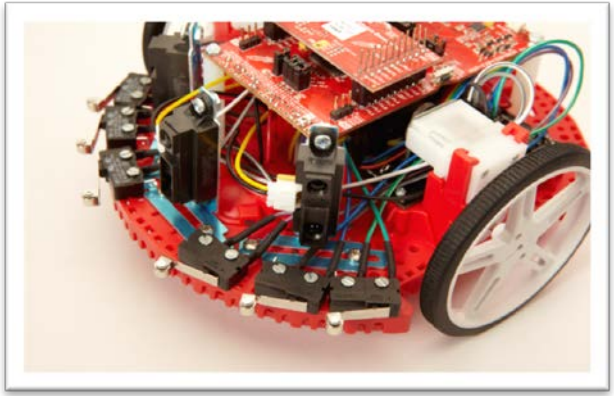
- Move to different port
- Bit-banding
- Disable, access, reenable

Summary

Interrupts

- Initialization (arm, priority, enable)
- Synchronization (globals)
- SysTick periodic interrupts
- Profiling





IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265