Real Time Operating Systems

Terminology

uC/OS-III, The Real-Time Kernel, or a High Performance, Scalable, ROMable, Preemptive, Multitasking Kernel for Microprocessors, Microcontrollers & DSPs, Book & Board Included, Hardcover, by Jean J Labrosse, $199.95

MicroC OS II: The Real Time Kernel, by Jean J. Labrosse , 2002, ISBN 1-5782-0103-9, $72.76

The Definitive Guide to the ARM Cortex-M3 TI, Second Edition, Paperback, Joseph Yiu, $53.95

Chapters 5 8 13, **Embedded Microcomputer Systems: Real Time Interfacing**, Third Edition, Jonathan W. Valvano, ISBN 1111426252, $83

Reference McDermott EE382N-4

Thread or Task

```
void Producer(void){
unsigned short data;
  data = ADC_In(1);
  if(OS_Fifo_Put(data) == 0){
    DataLost++;
  }
}
```

```
void Display(void){
unsigned long data,voltage;
  for(;;){
    data = OS_MailBox_Recv();
    voltage = 31*data/64;
    LCD_Message(0,"v(mV) =",voltage);
  }
}
```

```
void Consumer(void){
unsigned short data,average;
unsigned long sum;
unsigned short n;
  for(;;){
    sum = 0;
    for(n = 0; n < LENGTH; n++){
      data = OS_Fifo_Get();
      sum = sum + data;
    }
    average = sum/LENGTH;
    OS_MailBox_Send(average);
  }
}
```

Show main, threads in Robot RTOS

Real-time tasks

- Hard real-time
  - Bounded latency
- Soft real-time
  - Execute ASAP
- Not real-time

Thread Classification

- Periodic, execution at regular intervals
  - E.g., ADC, DAC, motor control
  - E.g., Check CO levels
- Aperiodic, execution can not be anticipated
  - Execution is frequent
  - E.g., New position detected as wheel turns
- Sporadic, execution can not be anticipated
  - Execution is infrequent
  - E.g., Faults, errors, catastrophes

Thread Scheduler

- List possible thread states
- List possible scheduling algorithms
  - What?
  - How?
  - Why?
- Performance measures
  - Utilization
  - Latency
  - Bandwidth

When to run scheduler??

Round robin
Weighted round robin
Priority

Static
Dynamic
Deterministic

Priority

- Execute highest priority first
  - Can you have two tasks at same priority?
- Minimize latency on real-time tasks
- Assign a dollar cost for delays
  - Minimize cost

Priority Schedulers

- Earliest deadline first, dynamic
- Earliest slack-time first , dynamic
  - Slack = (time to deadline)-(work left to do)
- Rate monotonic scheduling, static
  - Assign priority based on how often Ti is runs
  - Lower Ti (more frequent) are higher priority

Rate Monotonic Scheduling Theorem

- All n tasks are periodic
  - Priority based on period of T_i
  - Maximum execution time E_i
- No synchronization between tasks
- Execute highest priority task first

$$\sum \frac{E_i}{T_i} \leq n\left(2^{1/\mathrm{n}} - 1\right) \leq \ln\left(2\right)$$

Time Management

- System time
- Time stamps
  - When did it occur?
  - Performance measures
- Thread sleeping
- Measurements
  - Input capture period -> wheel RPM
  - Input capture PW -> ultrasonic distance

Communication

- Types
  - Data sharing
  - Pipes=FIFO (one to one, buffered, ordered)
  - Mailbox (one to one, unbuffered)
  - Messages (many to many)
- Deadlock
  - prevention, avoidance, detection, recovery
- Performance measures
  - Latency
  - Bandwidth
  - Error rate

Critical Sections

- Permanently allocated object
  - Shared variables
  - I/O ports
- Write access changes official copy
- Read access creates two copies
  - Original copy in memory
  - Temporary copy in register
- Nonatomic access, load/store architecture

Reentrant

- Variables in registers, stack
- No nonatomic write sequence
  - Permanently allocated object
  - WR, RMW, WW sequence

Look at programming manual
LDREX   STREX
CortexM3Programmer.pdf p.33, p.71
CortexM3InstructionSet.pdf p.39

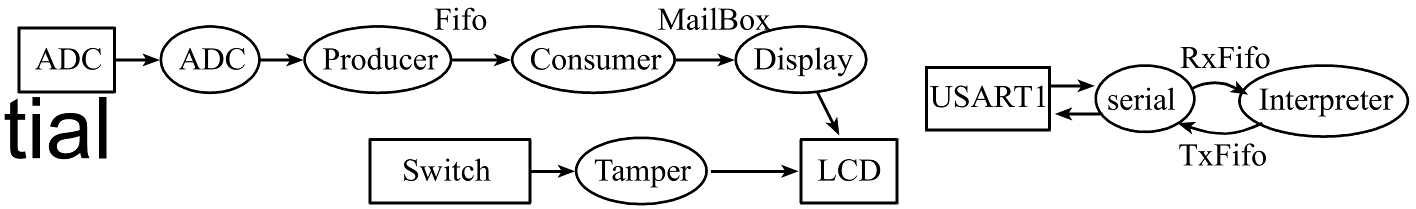
Making the access atomic

- Disable all interrupts
- Lock the scheduler
  - No other foreground threads can run
  - Background ISR will occur
- Mutex semaphore
  - Blocks other threads trying to access info
  - All nonrelated operations not delayed

Measure time with I=1
- Maximum time
- Total time

Show code with NestCnt++
If NestCnt-- == 0 then run or
don’t run scheduler??

Synchronization

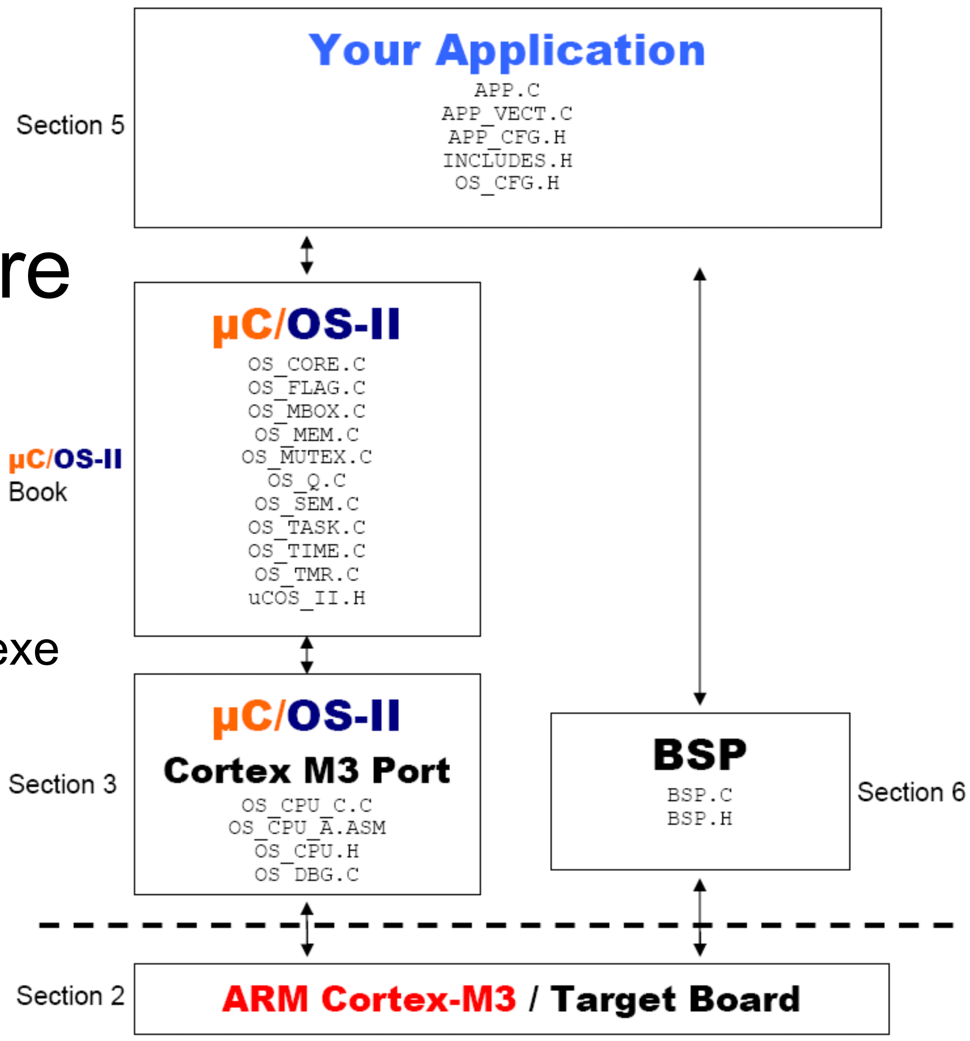


- Sequential
- Fork, spawn, join
- Rendezvous
- Trigger, event flags
  - or, and
  - I/O event (e.g., I/O edge, RX, TX)
  - periodic time (e.g., TATOMIS)
- Sleep

Portability

- Small kernel
- Common structure
- Hardware abstraction layer

Micrium-ARM-uCOS-II-Cortex-M3.exe
Show Micrium directory



Hooks

- Run user supplied code at strategic places
- Allows you to
  - Extend the OS
  - Implement debugging
  - Implement performance testing
  - Implement black box recording
- Collect run-time performance data

Additional OS terms

- Run-time configurable
  - Priority, stack size, fifo size, time slice
- Certification
  - Medical, transportation, nuclear, military
- Scalable
  - 10 threads versus 200 threads
- ROMable

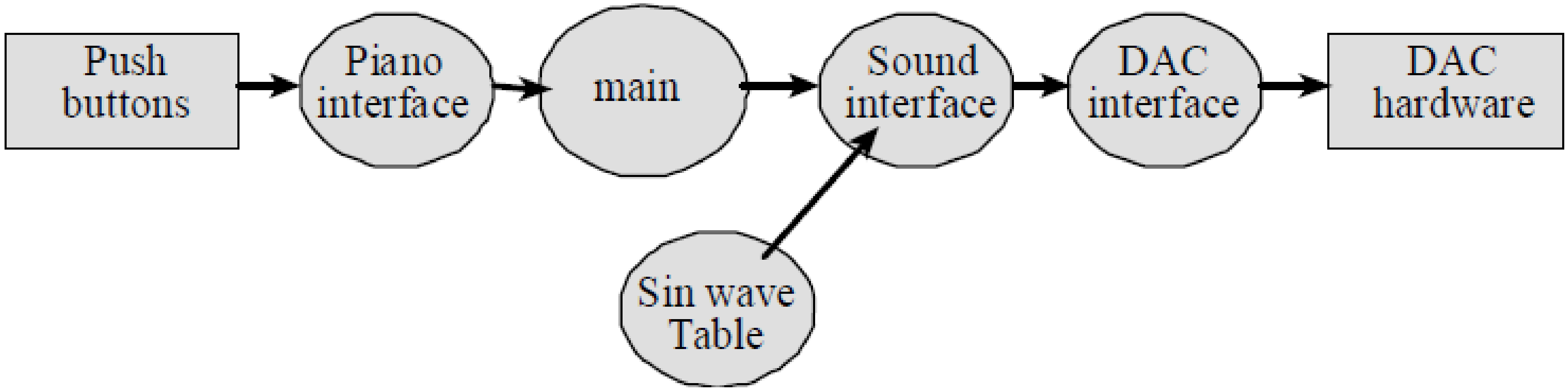
Performance measures

- Breakdown Utilization (BU)
  - The percentage of resource utilization below which the RTOS can guarantee that all deadlines will be met.
- Normalized Mean Response Time (NMRT)
  - The ratio of the “best case” time interval a task becomes ready to execute and then terminates, and the actual CPU time consumed.
- Guaranteed ratio (GR)
  - For dynamic scheduling, the number of tasks whose deadlines can be guaranteed to be met versus the total number of tasks requesting execution.

Case Study: Tone Generation http://www.toneconnect.com/

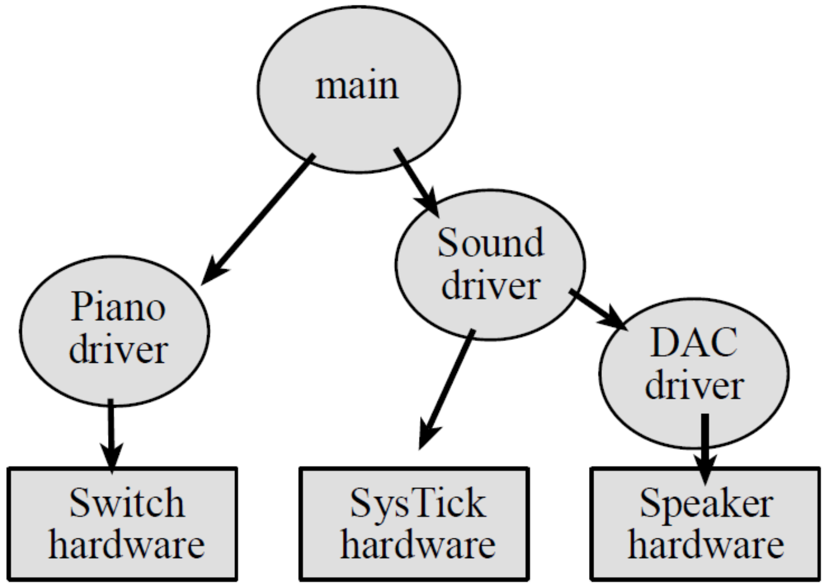
- Components
  - the software modules that are required to generate the tones as a function of switch inputs
- Operational Organization
  - the structure of the interactions between the software modules

Data Flow

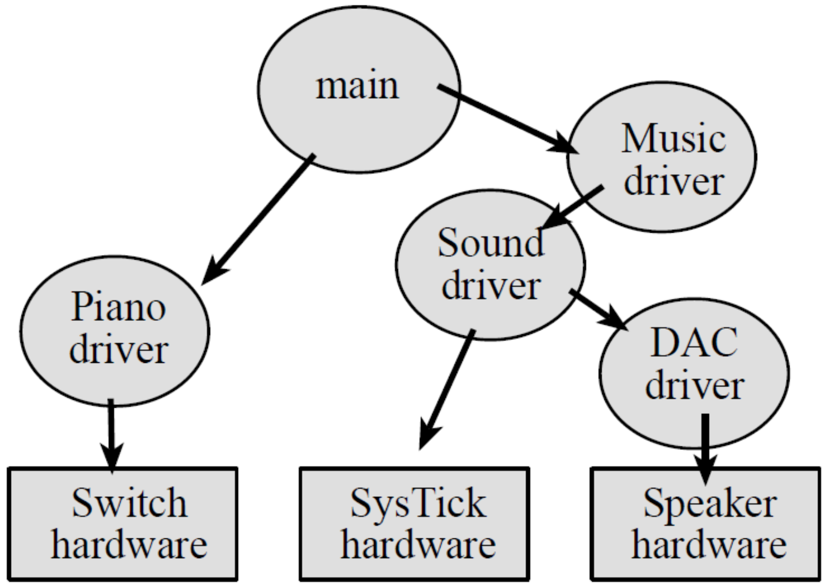


Push buttons
Piano interface
main
Sound interface
DAC interface
DAC hardware
Sin wave Table

Call Graph



main
Piano driver
Sound driver
DAC driver
Switch hardware
SysTick hardware
Speaker hardware

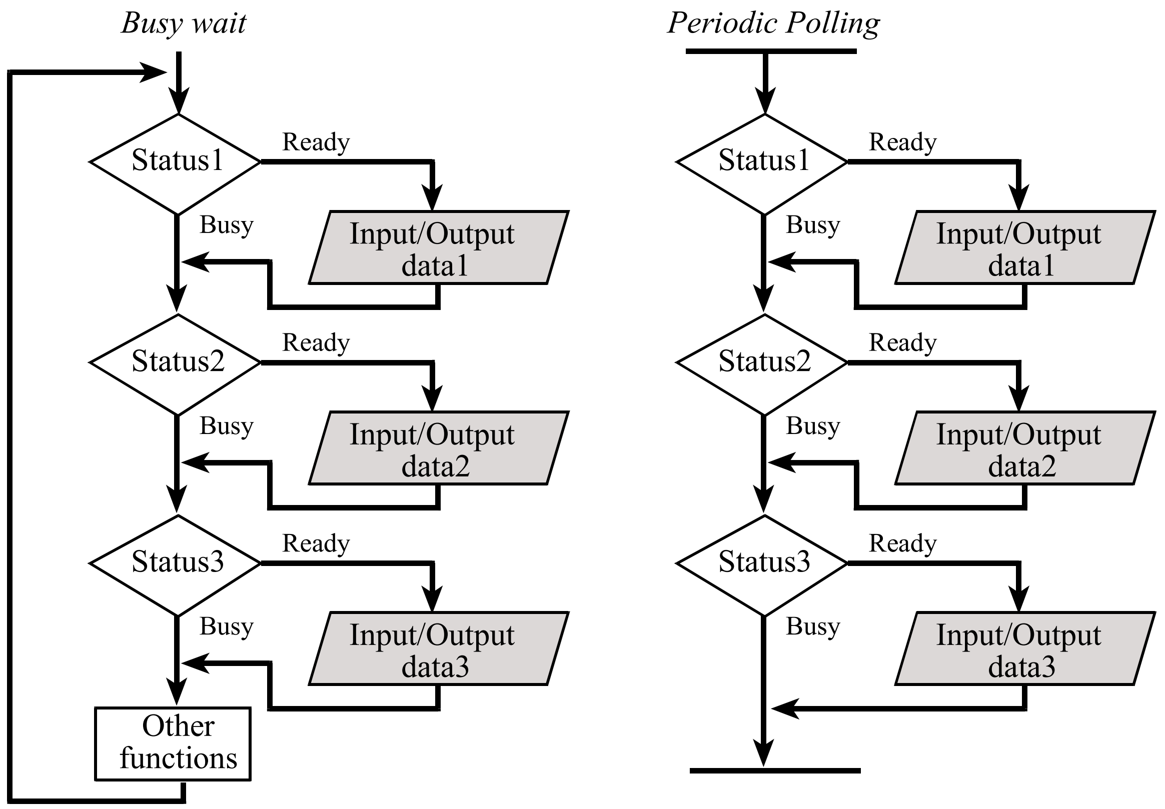


main
Music driver
Piano driver
Sound driver
DAC driver
Switch hardware
SysTick hardware
Speaker hardware

SysTick Periodic Interrupts

- SysTick Timer
  - source of interrupts with fixed period
  - useful for data acquisition and control systems
    - ADC and DAC applications
  - alternative to busy/wait polling
    - periodic polling
    - free foreground from continuous polling
    - useful for low bandwidth devices

SysTick Periodic Interrupts



SysTick Periodic Interrupts

- Operation
  - SysTick employs a 24-bit counter
  - Counter *counts down* from specified value
  - Decrements are made at the bus clock rate, f_{BUS}
    - ex: if f_{BUS} is 50 MHz, counter will decrement every 20 ns

SysTick Periodic Interrupts

Address	31-24	23-17	16	15-3	2	1	0	Name
$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

Address	31-29	28-24	23-21	20-8	7-5	4-0	Name
$E000ED20	TICK	0	PENDSV	0	DEBUG	0	NVIC_SYS_PRI3_R

SysTick Registers

SysTick Periodic Interrupts

- Operation
  - value contained in **RELOAD** register is loaded into the down counter
  - interrupt generated after down counter = 0
    - interrupt rate = f_{BUS} /(n+1)
    - ex: fBUS = 50 MHz, n = 113,636,
      ➔ interrupt rate = 440 Hz

SysTick Periodic Interrupts

- Initialization ritual
  - Clear **ENABLE** it to prevent operation during initialization
    - do this when intializing any interrupt source
  - Load selected value into **RELOAD** register
  - Clear **CURRENT** value of the counter
  - Set operation mode in **CONTROL** register
    - **CLK_SRC** = 1 for internal clock
    - **INTEN** = 1 to enable interrupts
  - Set interrupt priority in **TICK** field of **NVIC_SYS_PRI3_R**

SysTick Periodic Interrupts

```
volatile unsigned long Counts;
#define GPIO_PORTD0 (*((volatile unsigned long *)0x40007004))
void SysTick_Init(unsigned long period){
  SYSCTL_RCGC2_R |= 0x00000008; // activate port D
  Counts = 0;
  GPIO_PORTD_DIR_R |= 0x01; // make PD0 out
  GPIO_PORTD_DEN_R |= 0x01; // enable digital I/O on PD0
  NVIC_ST_CTRL_R = 0; // disable SysTick during setup
  NVIC_ST_RELOAD_R = period - 1;// reload value
  NVIC_ST_CURRENT_R = 0; // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; //priority 2
  NVIC_ST_CTRL_R = 0x00000007;// enable with core clock and interrupts
  EnableInterrupts();
}
void SysTick_Handler(void){
  GPIO_PORTD0 ^= 0x01; // toggle PD0
  Counts = Counts + 1; }
```

note `volatile` type qualifier

Timer Periodic Interrupts

- Periodic Timer
  - Timers provided in pairs
    - can be combined as 32-bit counter or operate independently as two 16-bit counters
  - similar to SysTick with 16-bit down-counter
    - 8-bit pre-scaler divides clock source to effectively produce a 24-bit counter
  - timer compared to pre-loaded value and sets a trigger on equality
    - an associated output pin level may also be inverted
  - May run in continuous or one-shot mode

Timer Periodic Interrupts

- Applications
  - delays
  - periodic interrupts
  - control ADC sampling rates
  - output compare
    - generating variable length pulses on a GPIO pin
  - input capture
    - measuring pulse width or frequency

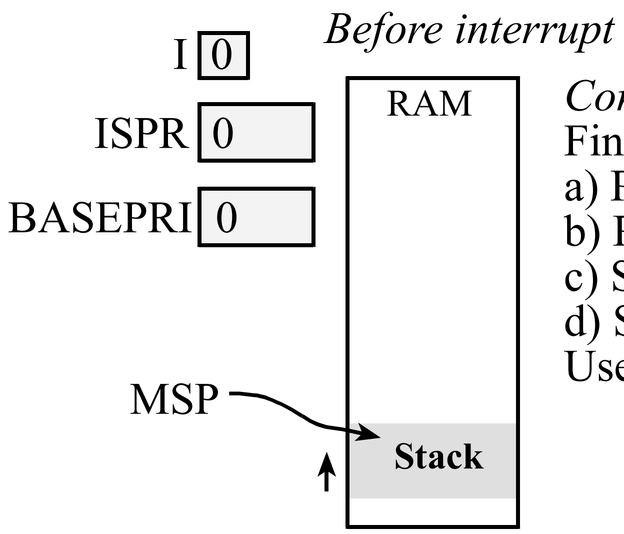
Timer Periodic Interrupts

- Timer components
  - A flag bit, e.g.,TATORIS
  - A control bit to connect the output to the ADC as a trigger, e.g., TAOTE,
  - An interrupt arm bit, e.g., TATOIM
  - A 16-bit output compare register, e.g., **TIMER0_TAILR_R**
  - A 8-bit prescale register, e.g., **TIMER0_TAPR_R**
  - A 8-bit prescale match register, e.g., **TIMER0_TAPMR_R**
  - An *optional* external output pin, e.g., CCP0,

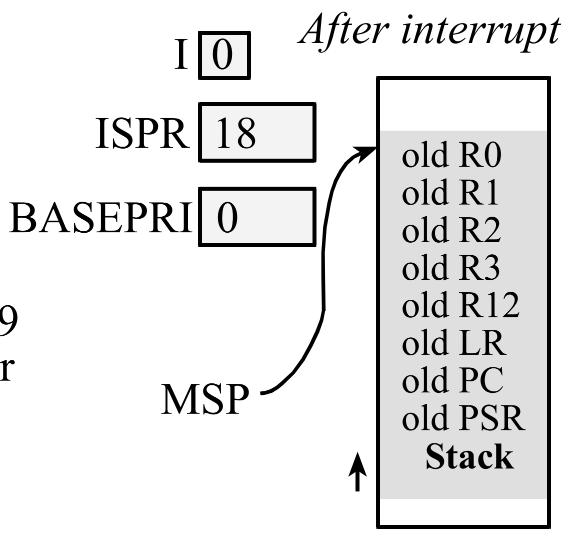
Interrupt Processing

- The execution of the main program is suspended
  - 1. the current instruction is finished,
  - 2. suspend execution and push registers (R0-R3, R12, LR, PC, PSR) on the stack
  - 3. LR set to 0xFFFFFFF9 (indicates interrupt return)
  - 4. IPSR set to interrupt number
  - 5. sets PC to ISR address
- 2) the interrupt service routine (ISR), or background thread is executed,
  - clears the flag that requested the interrupt
  - performs necessary operations
  - communicates using global variables
- 3) the main program is resumed when ISR executes **BX LR**.
  - pulls the registers from the stack

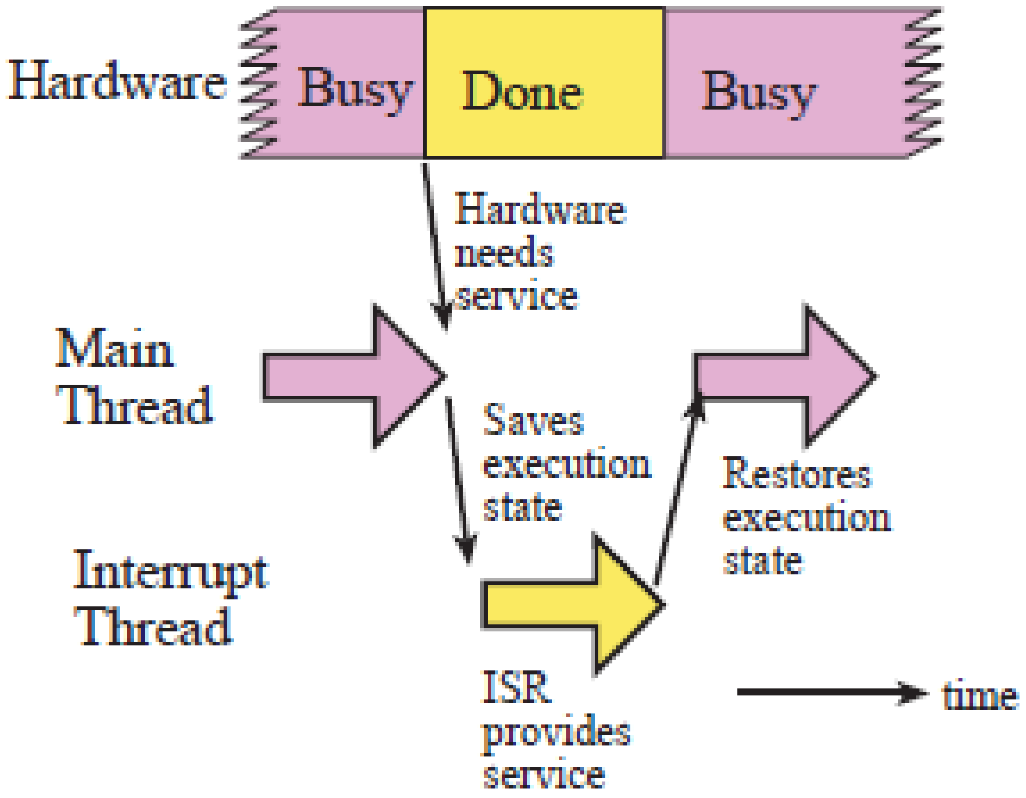
Example: Port C interrupt



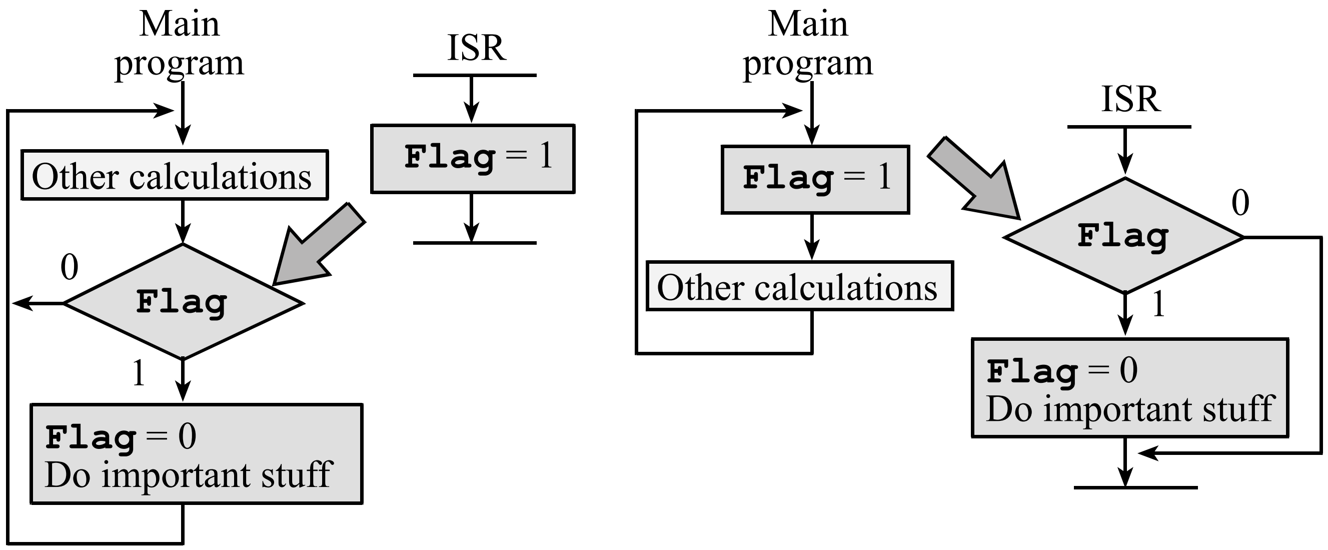
Context Switch
Finish instruction
a) Push registers
b) PC = {0x00000048}
c) Set ISPR = 18
d) Set LR = 0xFFFFFFF9
Use MSP as stack pointer



Interrupt Processing



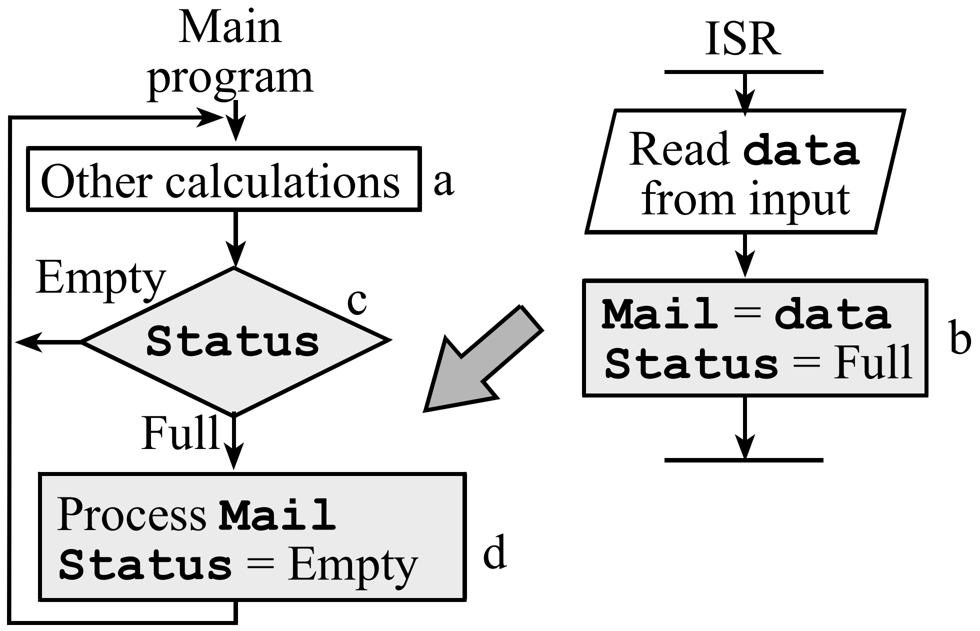
Thread Synchronization

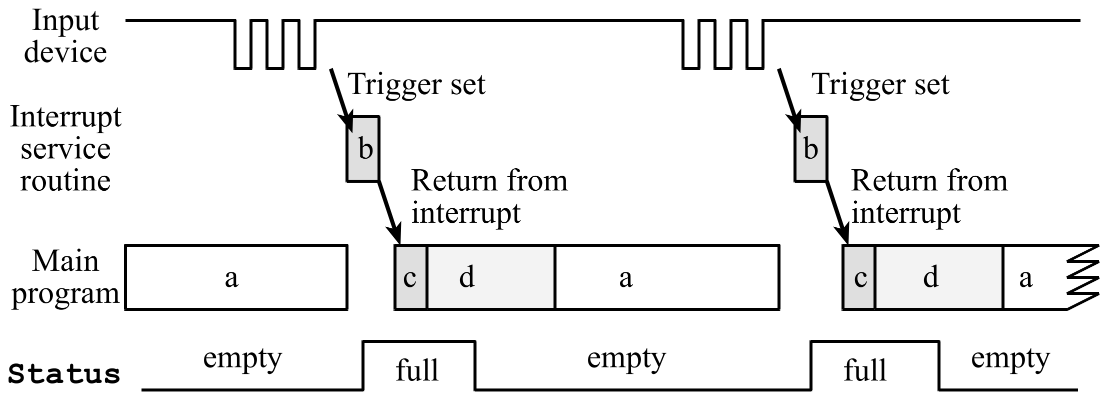


***Semaphore* synchronizes foreground/background threads**
***signal* sets flag, *wait* checks flag**

Thread Synchronization

MAILBOX USED TO PASS DATA BETWEEN THREADS





Interrupt Rituals

- **Things you must do in every ritual**
  - **Arm (specify a flag may interrupt)**
  - **Configure NVIC**
  - **Enable Interrupts**

Interrupt Service Routine (ISR)

- **Things you must do in every interrupt service routine**
  - **Acknowledge (clear flag that requested the interrupt)**
  - **Maintain contents of LR**

INTERRUPT VECTORS

Vectoraddress	Number	IRQ	ISR name in Startup.s	NVIC	Priority bits
0x00000038	14	-2	PendSV_Handler	NVIC_SYS_PRI3_R	23 – 21
0x0000003C	15	-1	SysTick_Handler	NVIC_SYS_PRI3_R	31 – 29
0x00000040	16	0	GPIOPortA_Handler	NVIC_PRI0_R	7 – 5
0x00000044	17	1	GPIOPortB_Handler	NVIC_PRI0_R	15 – 13
0x00000048	18	2	GPIOPortC_Handler	NVIC_PRI0_R	23 – 21
0x0000004C	19	3	GPIOPortD_Handler	NVIC_PRI0_R	31 – 29
0x00000050	20	4	GPIOPortE_Handler	NVIC_PRI1_R	7 – 5
0x00000054	21	5	UART0_Handler	NVIC_PRI1_R	15 – 13
0x00000058	22	6	UART1_Handler	NVIC_PRI1_R	23 – 21
0x0000005C	23	7	SSI0_Handler	NVIC_PRI1_R	31 – 29
0x00000060	24	8	I2C0_Handler	NVIC_PRI2_R	7 – 5
0x00000064	25	9	PWMFault_Handler	NVIC_PRI2_R	15 – 13
0x00000068	26	10	PWM0_Handler	NVIC_PRI2_R	23 – 21
0x0000006C	27	11	PWM1_Handler	NVIC_PRI2_R	31 – 29
0x00000070	28	12	PWM2_Handler	NVIC_PRI3_R	7 – 5
0x00000074	29	13	Quadrature0_Handler	NVIC_PRI3_R	15 – 13
0x00000078	30	14	ADC0_Handler	NVIC_PRI3_R	23 – 21
0x0000007C	31	15	ADC1_Handler	NVIC_PRI3_R	31 – 29
0x00000080	32	16	ADC2_Handler	NVIC_PRI4_R	7 – 5
0x00000084	33	17	ADC3_Handler	NVIC_PRI4_R	15 – 13
0x00000088	34	18	WDT_Handler	NVIC_PRI4_R	23 – 21
0x0000008C	35	19	Timer0A_Handler	NVIC_PRI4_R	31 – 29
0x00000090	36	20	Timer0B_Handler	NVIC_PRI5_R	7 – 5
0x00000094	37	21	Timer1A_Handler	NVIC_PRI5_R	15 – 13
0x00000098	38	22	Timer1B_Handler	NVIC_PRI5_R	23 – 21
0x0000009C	39	23	Timer2A_Handler	NVIC_PRI5_R	31 – 29
0x000000A0	40	24	Timer2B_Handler	NVIC_PRI6_R	7 – 5
0x000000A4	41	25	Comp0_Handler	NVIC_PRI6_R	15 – 13
0x000000A8	42	26	Comp1_Handler	NVIC_PRI6_R	23 – 21
0x000000AC	43	27	Comp2_Handler	NVIC_PRI6_R	31 – 29
0x000000B0	44	28	SysCtl_Handler	NVIC_PRI7_R	7 – 5
0x000000B4	45	29	FlashCtl_Handler	NVIC_PRI7_R	15 – 13
0x000000B8	46	30	GPIOPortF_Handler	NVIC_PRI7_R	23 – 21
0x000000BC	47	31	GPIOPortG_Handler	NVIC_PRI7_R	31 – 29
0x000000C0	48	32	GPIOPortH_Handler	NVIC_PRI8_R	7 – 5
0x000000C4	49	33	UART2_Handler	NVIC_PRI8_R	15 – 13
0x000000C8	50	34	SSI1_Handler	NVIC_PRI8_R	23 – 21
0x000000CC	51	35	Timer3A_Handler	NVIC_PRI8_R	31 – 29
0x000000D0	52	36	Timer3B_Handler	NVIC_PRI9_R	7 – 5
0x000000D4	53	37	I2C1_Handler	NVIC_PRI9_R	15 – 13
0x000000D8	54	38	Quadrature1_Handler	NVIC_PRI9_R	23 – 21
0x000000DC	55	39	CAN0_Handler	NVIC_PRI9_R	31 – 29
0x000000E0	56	40	CAN1_Handler	NVIC_PRI10_R	7 – 5
0x000000E4	57	41	CAN2_Handler	NVIC_PRI10_R	15 – 13
0x000000E8	58	42	Ethernet_Handler	NVIC_PRI10_R	23 – 21
0x000000EC	59	43	Hibernate_Handler	NVIC_PRI10_R	31 – 29
0x000000F0	60	44	USB0_Handler	NVIC_PRI11_R	7 – 5
0x000000F4	61	45	PWM3_Handler	NVIC_PRI11_R	15 – 13
0x000000F8	62	46	uDMA_Handler	NVIC_PRI11_R	23 – 21
0x000000FC	63	47	uDMA_Error	NVIC_PRI11_R	31 – 29

NVIC Registers – high order three bits of each byte define priority

Interrupt *does not* set I bit ➔ higher priority interrupts can interrupt lower priority

Address	31 – 29	23 – 21	15 – 13	7 – 5	Name
0xE000E400	GPIO Port D	GPIO Port C	GPIO Port B	GPIO Port A	NVIC_PRI0_R
0xE000E404	SSI0, Rx Tx	UART1, Rx Tx	UART0, Rx Tx	GPIO Port E	NVIC_PRI1_R
0xE000E408	PWM Gen 1	PWM Gen 0	PWM Fault	I2C0	NVIC_PRI2_R
0xE000E40C	ADC Seq 1	ADC Seq 0	Quad Encoder	PWM Gen 2	NVIC_PRI3_R
0xE000E410	Timer 0A	Watchdog	ADC Seq 3	ADC Seq 2	NVIC_PRI4_R
0xE000E414	Timer 2A	Timer 1B	Timer 1A	Timer 0B	NVIC_PRI5_R
0xE000E418	Comp 2	Comp 1	Comp 0	Timer 2B	NVIC_PRI6_R
0xE000E41C	GPIO Port G	GPIO Port F	Flash Control	System Control	NVIC_PRI7_R
0xE000E420	Timer 3A	SSI1, Rx Tx	UART2, Rx Tx	GPIO Port H	NVIC_PRI8_R
0xE000E424	CAN0	Quad Encoder 1	I2C1	Timer 3B	NVIC_PRI9_R
0xE000E428	Hibernate	Ethernet	CAN2	CAN1	NVIC_PRI10_R
0xE000E42C	uDMA Error	uDMA Soft Tfr	PWM Gen 3	USB0	NVIC_PRI11_R
0xE000ED20	SysTick	PendSV	--	Debug	NVIC_SYS_PRI3_R

NVIC Interrupt Enable Registers

- Two enable registers - **`NVIC_EN0_R`** and **`NVIC_EN1_R`**
  - each 32-bit register has a single enable bit for a particular device
  - **`NVIC_EN0_R`** control the IRQ numbers 0 to 31 (interrupt numbers 16 – 47)
  - **`NVIC_EN1_R`** control the IRQ numbers 32 to 47 (interrupt numbers 48 – 63)

Latency

- Software latency or interface latency
  - Time from when new input is ready until time software reads data.
  - Time from when output is idle until time software writes new data.
  - Execute tasks at periodic intervals
- Interrupts guarantee an upper bound on the software response time
  - Count maximum time running with **I=1**, plus
  - Time to process the interrupt.

Latency

- Real-time system
  - a system that can *guarantee* a worst case latency
- Throughput/bandwidth
  - maximum data flow (bytes/s) that can be processed by the system
- Priority
  - determines the order of service among two or more requests

Interrupt Events

- Respond to infrequent but important events.
  - Alarm conditions like low battery power and
  - Error conditions can be handled with interrupts.
- Periodic interrupts, generated by the timer at a regular rate
  - Clocks and timers
  - Computer-based data input/output
  - DAC used play music
  - ADC used to acquire data
  - Digital control systems.
- I/O synchronization

Periodic Interrupts

- **Data acquisition samples ADC**
- **Signal generation output to DAC**
  - **Audio player**
  - **Communications**
- **Digital controller**
  - **FSM**
  - **Linear control system (EE362K)**

Periodic Interrupts

- Moore FSM – Foreground Solution
  - **1. Perform output for the current state**
  - **2. Wait for specified amount of time**
  - **3. Input from the switches**
  - **4. Go to the next state depending on the input**

What occupies most of the time?

Periodic Interrupts

- Background solution
  - **Ritual**
  - **1. Perform output for the current state**
  - **2. Set timer to wait for specified amount of time**
    - **Timer interrupt service routine**
  - **3. Input from the switches**
  - **4. Go to the next state depending on the input**

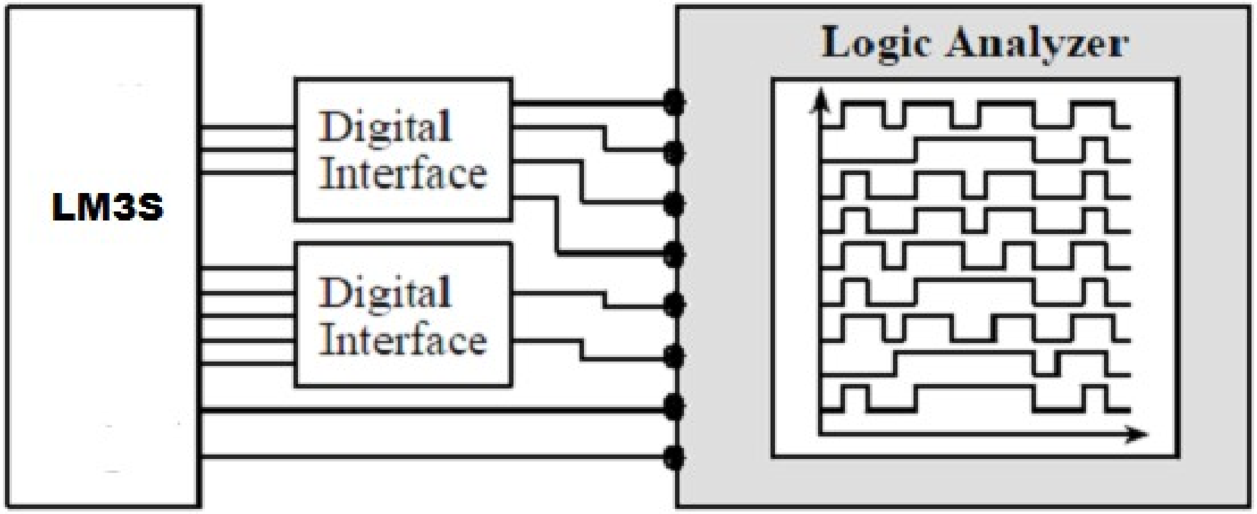
Periodic Interrupts

- Background solution advantages
  - More accurate
  - Frees up cycles to perform other tasks

Interrupt Debugging

- **Profiling**
  - 1) Is the interrupt occurring? Is a function being called?
  - Add global counters, initialize to 0
  - Add **counter++** inside ISR, inside function
  - 2) Is the interrupt occurring? Is a function being called?
  - Find unused I/O pins, initialize to outputs
  - Set GPIO pin at beginning of ISR, function
  - Toggle GPIO pin at end of ISR, function
  - View bits with a logic analyzer or scope

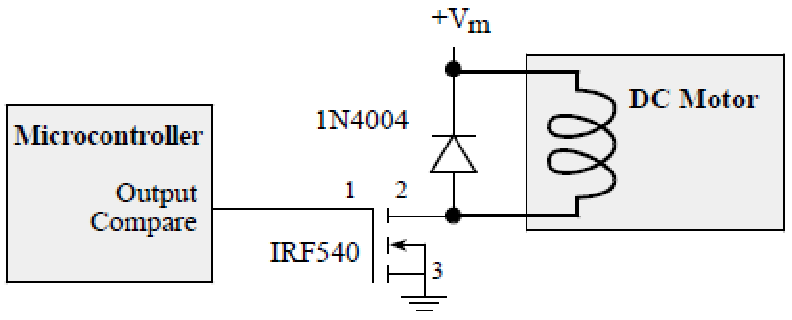
Logic Analyzer

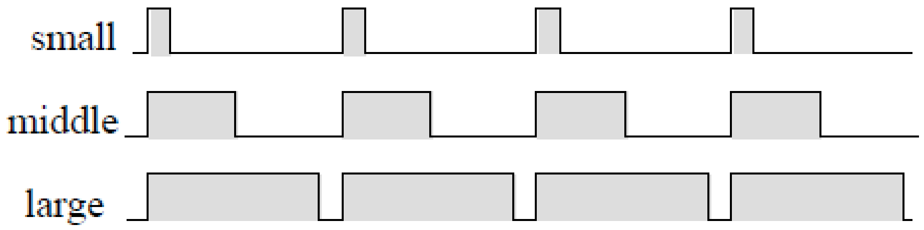


Display values of digital signals as f(t)

Periodic Interrupts

- Example – DC Motor Controller
  - **Output compare every 1ms**
  - **Length of pulse is a variable from 0 to 10**
  - **Every 10 interrupts make GPIO pin high**
  - **Every Length interrupts make GPIO pin low**
  - Duty cycle is **Length/10**
  - Maximum power is V_m^2/R
  - Delivered power is V_m^2/R * **Length/10**


+Vm
DC Motor
1N4004
Microcontroller
Output Compare
1
2
IRF540
3


small
middle
large