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


Reference McDermott EE382N-4
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

Round robin
Weighted round robin
Priority
Static
Dynamic
Deterministic

When to run scheduler??
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

$$\sum \frac{E_i}{T_i} \leq n\left(2^{1/n} - 1\right) \leq 1\ln(2)$$
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
    - Switch hardware
  - Sound driver
    - SysTick hardware
  - DAC driver
    - Speaker hardware

- main
  - Sound driver
    - Piano driver
    - Switch hardware
  - DAC driver
    - 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

Busy wait

- Status 1
  - Busy
  - Ready
  - Input/Output data 1
  - Busy
  - Ready
  - Input/Output data 2
  - Busy
  - Ready
  - Input/Output data 3
  - Other functions

Periodic Polling

- Status 1
  - Busy
  - Input/Output data 1
  - Busy
  - Ready
  - Input/Output data 2
  - Busy
  - Ready
  - Input/Output data 3
  - Busy
  - Ready
  - Input/Output data 3
SysTick Periodic Interrupts

• Operation
  – SysTick employs a 24-bit counter
  – Counter *counts down* from specified value
  – Decrement 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

<table>
<thead>
<tr>
<th>Address</th>
<th>31-24</th>
<th>23-17</th>
<th>16</th>
<th>15-3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E000E010</td>
<td>0</td>
<td>0</td>
<td>COUNT</td>
<td>0</td>
<td>CLK_SRC</td>
<td>INTEN</td>
<td>ENABLE</td>
<td>NVIC_ST_CTRL_R</td>
</tr>
<tr>
<td>$E000E014</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24-bit RELOAD value</td>
<td>NVIC_ST_RELOAD_R</td>
</tr>
<tr>
<td>$E000E018</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24-bit CURRENT value of SysTick counter</td>
<td>NVIC_ST_CURRENT_R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>31-29</th>
<th>28-24</th>
<th>23-21</th>
<th>20-8</th>
<th>7-5</th>
<th>4-0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E000ED20</td>
<td>TICK</td>
<td>0</td>
<td>PENDSV</td>
<td>0</td>
<td>DEBUG</td>
<td>0</td>
<td>NVIC_SYS_PRI3_R</td>
</tr>
</tbody>
</table>

## 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 = $f_{\text{BUS}} / (n+1)$
    • ex: $f_{\text{BUS}} = 50$ MHz, $n = 113,636$,
      $\Rightarrow$ interrupt rate = 440 Hz
SysTick Periodic Interrupts

• Initialization ritual
  – Clear **ENABLE** it to prevent operation during initialization
    • do this when initializing 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., TIMERO_TAILR_R
  - A 8-bit prescale register, e.g., TIMERO_TAPR_R
  - A 8-bit prescale match register, e.g., TIMERO_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

**Before interrupt**

**Context Switch**
Finish instruction
a) Push registers
b) PC = \{0x0000048\}
c) Set ISPR = 18
d) Set LR = 0xFFFFFFFF9
Use MSP as stack pointer

**After interrupt**

- old R0
- old R1
- old R2
- old R3
- old R12
- old LR
- old PC
- old PSR

Stack
Interrupt Processing

Diagram showing the process of interrupt handling in a computer system. The diagram illustrates the sequence of events from the occurrence of an interrupt to the resumption of the main thread of execution. The process includes:

1. The hardware becomes busy due to an interrupt.
2. The hardware needs service and invokes the interrupt service routine (ISR).
3. The ISR saves the execution state of the main thread.
4. The ISR provides service to the hardware.
5. The ISR restores the execution state of the main thread.
6. Time passes as the system continues to execute.

The diagram uses arrows to indicate the flow of control and the sequence of events.
Thread Synchronization

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

MAILBOX USED TO PASS DATA BETWEEN THREADS

Main program

Other calculations

Empty

Full

Status

Process Mail Status = Empty

ISR

Read data from input

Mail = data Status = Full

Input device

Interrupt service routine

Main program

Status

empty full empty full empty

Trigger set

Return from interrupt

Trigger set

Return from interrupt

d

b
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
<table>
<thead>
<tr>
<th>Vector address</th>
<th>Number</th>
<th>IRQ</th>
<th>ISR name in Startup.s</th>
<th>NVIC</th>
<th>Priority bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000038</td>
<td>14</td>
<td>-2</td>
<td>PendSV_Handler</td>
<td>NVIC_SYS_PRI3_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x00000003C</td>
<td>15</td>
<td>-1</td>
<td>SysTick_Handler</td>
<td>NVIC_SYS_PRI3_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x000000040</td>
<td>16</td>
<td>0</td>
<td>GPIOPortA_Handler</td>
<td>NVIC_PRI0_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x000000044</td>
<td>17</td>
<td>1</td>
<td>GPIOPortB_Handler</td>
<td>NVIC_PRI0_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x000000048</td>
<td>18</td>
<td>2</td>
<td>GPIOPortC_Handler</td>
<td>NVIC_PRI0_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x00000004C</td>
<td>19</td>
<td>3</td>
<td>GPIOPortD_Handler</td>
<td>NVIC_PRI0_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x000000050</td>
<td>20</td>
<td>4</td>
<td>GPIOPortE_Handler</td>
<td>NVIC_PRI1_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x000000054</td>
<td>21</td>
<td>5</td>
<td>UART1_Handler</td>
<td>NVIC_PRI1_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x000000058</td>
<td>22</td>
<td>6</td>
<td>UART2_Handler</td>
<td>NVIC_PRI1_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x000000060</td>
<td>24</td>
<td>8</td>
<td>I2C0_Handler</td>
<td>NVIC_PRI2_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x000000064</td>
<td>25</td>
<td>9</td>
<td>PWM0Fault_Handler</td>
<td>NVIC_PRI2_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x000000068</td>
<td>26</td>
<td>10</td>
<td>PWM0_Handler</td>
<td>NVIC_PRI2_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x00000006C</td>
<td>27</td>
<td>11</td>
<td>PWM1_Handler</td>
<td>NVIC_PRI2_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x000000070</td>
<td>28</td>
<td>12</td>
<td>PWM2_Handler</td>
<td>NVIC_PRI3_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x000000074</td>
<td>29</td>
<td>13</td>
<td>Quadrature0_Handler</td>
<td>NVIC_PRI3_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x000000078</td>
<td>30</td>
<td>14</td>
<td>ADC0_Handler</td>
<td>NVIC_PRI3_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x00000007C</td>
<td>31</td>
<td>15</td>
<td>ADC1_Handler</td>
<td>NVIC_PRI3_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x000000080</td>
<td>32</td>
<td>16</td>
<td>ADC2_Handler</td>
<td>NVIC_PRI4_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x000000084</td>
<td>33</td>
<td>17</td>
<td>ADC3_Handler</td>
<td>NVIC_PRI4_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x000000088</td>
<td>34</td>
<td>18</td>
<td>WDT_Handler</td>
<td>NVIC_PRI4_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x00000008C</td>
<td>35</td>
<td>19</td>
<td>Timer0A_Handler</td>
<td>NVIC_PRI4_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x000000090</td>
<td>36</td>
<td>20</td>
<td>Timer0B_Handler</td>
<td>NVIC_PRI5_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x000000094</td>
<td>37</td>
<td>21</td>
<td>Timer1A_Handler</td>
<td>NVIC_PRI5_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x000000098</td>
<td>38</td>
<td>22</td>
<td>Timer1B_Handler</td>
<td>NVIC_PRI5_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x00000009C</td>
<td>39</td>
<td>23</td>
<td>Timer2A_Handler</td>
<td>NVIC_PRI5_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x0000000A0</td>
<td>40</td>
<td>24</td>
<td>Timer2B_Handler</td>
<td>NVIC_PRI6_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x0000000A4</td>
<td>41</td>
<td>25</td>
<td>Comp0_Handler</td>
<td>NVIC_PRI6_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x0000000A8</td>
<td>42</td>
<td>26</td>
<td>Comp1_Handler</td>
<td>NVIC_PRI6_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x0000000AC</td>
<td>43</td>
<td>27</td>
<td>Comp2_Handler</td>
<td>NVIC_PRI6_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x0000000B0</td>
<td>44</td>
<td>28</td>
<td>SysCtl_Handler</td>
<td>NVIC_PRI7_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x0000000B4</td>
<td>45</td>
<td>29</td>
<td>FlashCtl_Handler</td>
<td>NVIC_PRI7_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x0000000B8</td>
<td>46</td>
<td>30</td>
<td>GPIOPortF_Handler</td>
<td>NVIC_PRI7_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x0000000BC</td>
<td>47</td>
<td>31</td>
<td>GPIOPortG_Handler</td>
<td>NVIC_PRI7_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x0000000C0</td>
<td>48</td>
<td>32</td>
<td>GPIOPortH_Handler</td>
<td>NVIC_PRI8_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x0000000C4</td>
<td>49</td>
<td>33</td>
<td>UART2_Handler</td>
<td>NVIC_PRI8_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x0000000C8</td>
<td>50</td>
<td>34</td>
<td>SS10_Handler</td>
<td>NVIC_PRI8_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x0000000CC</td>
<td>51</td>
<td>35</td>
<td>Timer3A_Handler</td>
<td>NVIC_PRI8_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x0000000D0</td>
<td>52</td>
<td>36</td>
<td>Timer3B_Handler</td>
<td>NVIC_PRI9_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x0000000D4</td>
<td>53</td>
<td>37</td>
<td>I2C1_Handler</td>
<td>NVIC_PRI9_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x0000000D8</td>
<td>54</td>
<td>38</td>
<td>Quadrature1_Handler</td>
<td>NVIC_PRI9_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x0000000DC</td>
<td>55</td>
<td>39</td>
<td>CAN0_Handler</td>
<td>NVIC_PRI9_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x0000000E0</td>
<td>56</td>
<td>40</td>
<td>CAN1_Handler</td>
<td>NVIC_PRI10_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x0000000E4</td>
<td>57</td>
<td>41</td>
<td>CAN2_Handler</td>
<td>NVIC_PRI10_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x0000000E8</td>
<td>58</td>
<td>42</td>
<td>Ethernet_Handler</td>
<td>NVIC_PRI10_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x0000000EC</td>
<td>59</td>
<td>43</td>
<td>Hibernate_Handler</td>
<td>NVIC_PRI10_R</td>
<td>31 – 29</td>
</tr>
<tr>
<td>0x0000000F0</td>
<td>60</td>
<td>44</td>
<td>USB0_Handler</td>
<td>NVIC_PRI11_R</td>
<td>7 – 5</td>
</tr>
<tr>
<td>0x0000000F4</td>
<td>61</td>
<td>45</td>
<td>PWM3_Handler</td>
<td>NVIC_PRI11_R</td>
<td>15 – 13</td>
</tr>
<tr>
<td>0x0000000F8</td>
<td>62</td>
<td>46</td>
<td>uDMA_Handler</td>
<td>NVIC_PRI11_R</td>
<td>23 – 21</td>
</tr>
<tr>
<td>0x0000000FC</td>
<td>63</td>
<td>47</td>
<td>uDMA_Error</td>
<td>NVIC_PRI11_R</td>
<td>31 – 29</td>
</tr>
</tbody>
</table>
## NVIC Registers – high order three bits of each byte define priority

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

<table>
<thead>
<tr>
<th>Address</th>
<th>31 – 29</th>
<th>23 – 21</th>
<th>15 – 13</th>
<th>7 – 5</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xE000E400</td>
<td>GPIO Port D</td>
<td>GPIO Port C</td>
<td>GPIO Port B</td>
<td>GPIO Port A</td>
<td>NVIC_PRI0_R</td>
</tr>
<tr>
<td>0xE000E404</td>
<td>SSI0, Rx Tx</td>
<td>UART1, Rx Tx</td>
<td>UART0, Rx Tx</td>
<td>GPIO Port E</td>
<td>NVIC_PRI1_R</td>
</tr>
<tr>
<td>0xE000E408</td>
<td>PWM Gen 1</td>
<td>PWM Gen 0</td>
<td>PWM Fault</td>
<td>I2C0</td>
<td>NVIC_PRI2_R</td>
</tr>
<tr>
<td>0xE000E40C</td>
<td>ADC Seq 1</td>
<td>ADC Seq 0</td>
<td>Quad Encoder</td>
<td>PWM Gen 2</td>
<td>NVIC_PRI3_R</td>
</tr>
<tr>
<td>0xE000E410</td>
<td>Timer 0A</td>
<td>Watchdog</td>
<td>ADC Seq 3</td>
<td>ADC Seq 2</td>
<td>NVIC_PRI4_R</td>
</tr>
<tr>
<td>0xE000E414</td>
<td>Timer 2A</td>
<td>Timer 1B</td>
<td>Timer 1A</td>
<td>Timer 0B</td>
<td>NVIC_PRI5_R</td>
</tr>
<tr>
<td>0xE000E418</td>
<td>Comp 2</td>
<td>Comp 1</td>
<td>Comp 0</td>
<td>Timer 2B</td>
<td>NVIC_PRI6_R</td>
</tr>
<tr>
<td>0xE000E41C</td>
<td>GPIO Port G</td>
<td>GPIO Port F</td>
<td>Flash Control</td>
<td>System Control</td>
<td>NVIC_PRI7_R</td>
</tr>
<tr>
<td>0xE000E420</td>
<td>Timer 3A</td>
<td>SS11, Rx Tx</td>
<td>UART2, Rx Tx</td>
<td>GPIO Port H</td>
<td>NVIC_PRI8_R</td>
</tr>
<tr>
<td>0xE000E424</td>
<td>CAN0</td>
<td>Quad Encoder 1</td>
<td>I2C1</td>
<td>Timer 3B</td>
<td>NVIC_PRI9_R</td>
</tr>
<tr>
<td>0xE000E428</td>
<td>Hibernate</td>
<td>Ethernet</td>
<td>CAN2</td>
<td>CAN1</td>
<td>NVIC_PRI10_R</td>
</tr>
<tr>
<td>0xE000E42C</td>
<td>uDMA Error</td>
<td>uDMA Soft Tfr</td>
<td>PWM Gen 3</td>
<td>USB0</td>
<td>NVIC_PRI11_R</td>
</tr>
<tr>
<td>0xE000ED20</td>
<td>SysTick</td>
<td>PendSV</td>
<td>--</td>
<td>Debug</td>
<td>NVIC_SYS_PRI3_R</td>
</tr>
</tbody>
</table>
NVIC Interrupt Enable Registers

- Two enable registers - \texttt{NVIC\_EN0\_R} and \texttt{NVIC\_EN1\_R}
  - each 32-bit register has a single enable bit for a particular device
  - \texttt{NVIC\_EN0\_R} control the IRQ numbers 0 to 31 (interrupt numbers 16 – 47)
  - \texttt{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 $\frac{V_m^2}{R}$
  - Delivered power is $\frac{V_m^2}{R} \times \frac{\text{Length}}{10}$