

EE445M/EE360L.12 Embedded and Real-Time Systems/ Real-Time Operating Systems

Lecture 4: Threads, Scheduler, Thread Communication & Synchronization

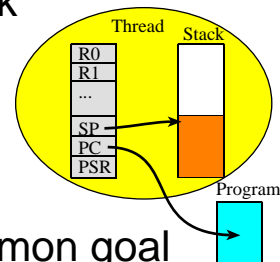
Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

1

Thread or Light-Weight Process

- Execution of a software task
- Has its own registers
- Has its own stack
- Local variables are private
- Threads cooperate for common goal
- Private global variables
 - Managed by the OS
 - Allocated in the TCB (e.g., `td`)



Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

2

Thread Communication/Sharing

- Shared Globals
- Mailbox (Lab 2)
- FIFO queues (Lab 2)
- Message (Lab 6)

Lecture 4 J. Valvano, A. Gerstlauer 3
EE445M/EE380L.12

Thread Control Block (TCB)

- Id
- Stack pointer
- Sleep counter
- Blocked pt (Lab 3)
- Priority (Lab 3)
- Next or Next/Previous links

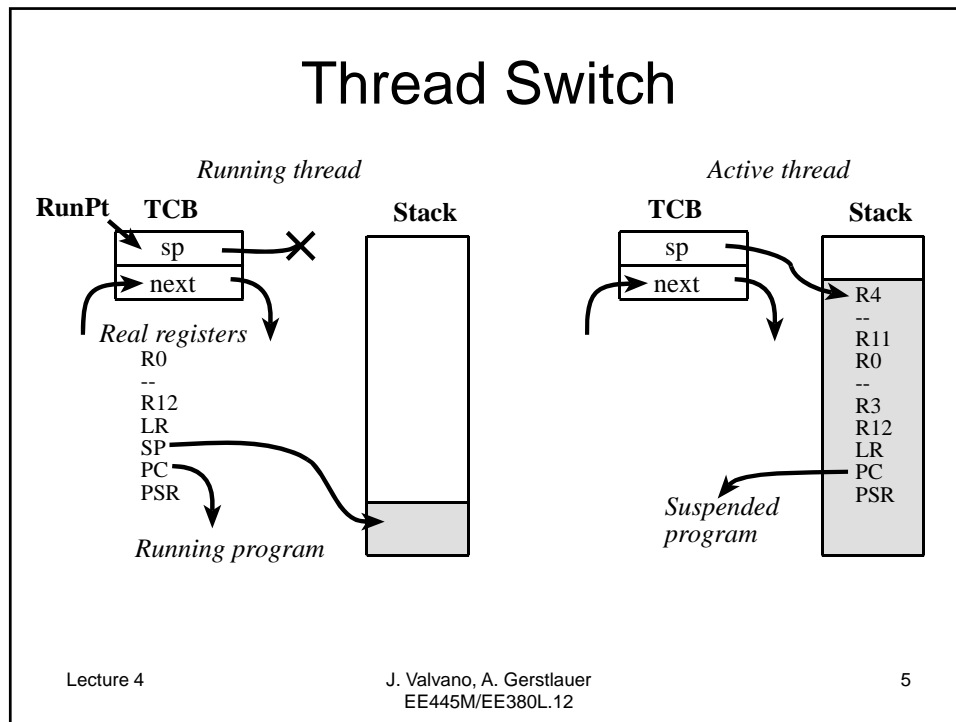
Where are the registers saved?

```

struct TCB {
    // order?!, types??
};
typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
    
```

Look at TCB of uC/OS-II, struct os_tcb in
 Micrium\Software\uCOS-II\Source\ucos_ii.h

Lecture 4 J. Valvano, A. Gerstlauer
EE445M/EE380L.12



PendSV Thread Switch (1)

- PendSV handler
 - Give PendSV handler lowest priority
 - Prevent switching out background tasks
 - Use C code to find next thread

TCB of a running thread
TCB of a thread not running

CortexM
stack pointer
stack pointer

R0-R14,
TCB link
TCB link

PC, PSR
Id
Id

SP
stack area
stack area

local variables
R0-R14,
R0-R14,

return pointers
PC, PSR
PC, PSR

return pointers
local variables
local variables

return pointers
return pointers

- Trigger PendSV


```

NVIC_INT_CTRL_R = 0x10000000;
                
```

```

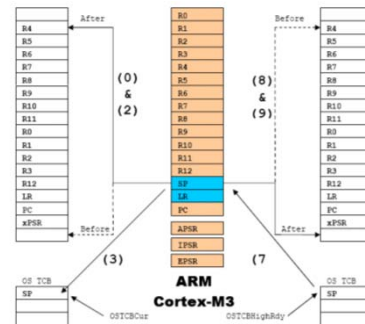
NVIC_INT_CTRL EQU 0xE00ED04
NVIC_PENDSVSET EQU 0x1000000
ContextSwitch
LDR R0, =NVIC_INT_CTRL
LDR R1, =NVIC_PENDSVSET
STR R1, [R0]
BX LR
                
```

Page 160 of tm4c123gh6pm.pdf

PendSV Thread Switch (2)

- 1) Disable interrupts
- 2) Save registers R4 to R11 on the user stack
- 3) Save stack pointer into TCB
- 4) Choose next thread
- 5) Retrieve new stack pointer
- 6) Restore registers R4 to R11
- 7) Reenable interrupts
- 8) Return from interrupt

Run *Testmain1*
 -Show TCB chain
 -Show stacks
 -Explain switch



Micrium\Software\uCOS-III\Ports\ARM-Cortex-M3\Generic\RealView\os_cpu_a.asm

Assembly Thread Switch

```

PendSV_Handler          ; 1) Saves R0-R3,R12,LR,PC,PSR
    CPSID I              ; 2) Make atomic
    PUSH {R4-R11}        ; 3) Save remaining regs r4-11
    LDR R0, =RunPt        ; 4) R0=pointer to RunPt, old
    LDR R1, [R0]          ; R1 = RunPt
    STR SP, [R1]          ; 5) Save SP into TCB
    LDR R1, [R1,#4]      ; 6) R1 = RunPt->next
    STR R1, [R0]          ; RunPt = R1
    LDR SP, [R1]          ; 7) new thread SP; SP=RunPt->sp;
    POP {R4-R11}         ; 8) restore regs r4-11
    CPSIE I              ; 9) tasks run enabled
    BX LR                ; 10) restore R0-R3,R12,LR,PC,PSR
    
```

Program 4.9

[RTOS_4C123.zip](#)

Thread Management

- TCB
- Stacks
- Scheduler

See [Testmain1](#)

See [Testmain2](#)

Reference book, chapter 4

Lecture 4
J. Valvano, A. Gerstlauer
EE445M/EE380L.12
9

Thread States

```

    graph TD
      dead((dead)) -- OS_AddThread --> active((active))
      run((run)) -- calls OS_suspend --> active
      run -- "time slice is over, OS takes control away" --> active
      active -- "OS grants control" --> run
      run -- calls OS_sleep --> sleep((sleep))
      sleep -- time over --> active
      run -- calls OS_kill --> dead
  
```

Lab 3 will add Blocked

Lecture 4
J. Valvano, A. Gerstlauer
EE445M/EE380L.12
10

Thread Scheduler

- When to invoke
 - Cooperative: `os_suspend()`
 - Preemptive: `SysTick`

- What **Active** task to **Run**
 - Round robin (Lab 2)
 - Weighted round robin
 - Priority (Lab 3)

Lecture 4

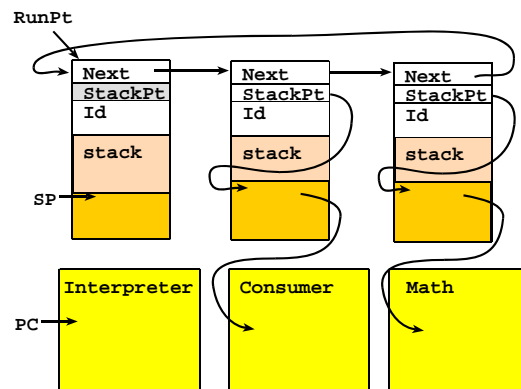
J. Valvano, A. Gerstlauer
EE445M/EE380L.12

11

Round Robin Scheduler

```
OS_AddThread(&Interpreter);
OS_AddThread(&Consumer);
OS_AddThread(&Math);
OS_Launch(TIMESLICE); // doesn't return
```

RunPt



Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

12

Decisions

- PendSV/SysTick or SysTick only?
 - Everything in one handler?
 - How to handle sleep?
- Privileged/Unprivileged?
 - Trap or regular function call?
 - How do you link OS to user code?
- MSP/PSP or MSP?
 - Protection versus speed?
 - Check for stack overflow
 - Check for valid parameters

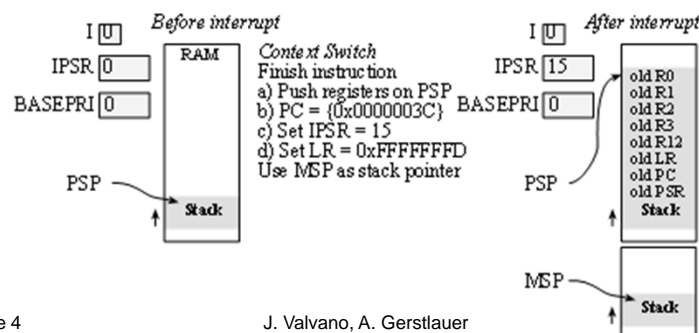
Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

13

Thread Switch with PSP (1)

- Bottom 8 bits of LR
 - 0xE1 11110001 Return to Handler mode MSP (using floating point state)
 - 0xE9 11101001 Return to Thread mode MSP (using floating point state)
 - 0xED 11101101 Return to Thread mode PSP (using floating point state)
 - 0xF1 11110001 Return to Handler mode MSP
 - 0xF9 11111001 Return to Thread mode MSP
 - **0xFD 11111101 Return to Thread mode PSP**



Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

14

Thread Switch with PSP (2)

```

; This code uses MSP for user and OS (Program 4.9 from book)
SysTick_Handler          ; 1) Saves R0-R3,R12,LR,PC,PSR
    CPSID    I           ; 2) Prevent interrupt during switch
    PUSH     {R4-R11}    ; 3) Save remaining regs r4-11
    LDR      R0, =RunPt  ; 4) R0=pointer to RunPt, old thread
    LDR      R1, [R0]    ;      R1 = RunPt
    STR      SP, [R1]    ; 5) Save SP into TCB
    LDR      R1, [R1,#4] ; 6) R1 = RunPt->next
    STR      R1, [R0]    ;      RunPt = R1
    LDR      SP, [R1]    ; 7) new thread SP; SP = RunPt->sp;
    POP      {R4-R11}    ; 8) restore regs r4-11
    CPSIE    I           ; 9) run with interrupts enabled
    BX      LR           ; 10) restore R0-R3,R12,LR,PC,PSR

```

Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

15

Thread Switch with PSP (3)

```

; tasks use PSP, OS/ISR use MSP, Micrium OS-II
SysTick_Handler          ; 1) R0-R3,R12,LR,PC,PSR on PSP
    CPSID    I           ; 2) Prevent interrupt during switch
    MRS      R2, PSP     ; R2=PSP, the process stack pointer
    SUBS     R2, R2, #0x20
    STM      R2, {R4-R11} ; 3) Save remaining regs r4-11
    LDR      R0, =RunPt  ; 4) R0=pointer to RunPt, old thread
    LDR      R1, [R0]    ;      R1 = RunPt
    STR      R2, [R1]    ; 5) Save PSP into TCB
    LDR      R1, [R1,#4] ; 6) R1 = RunPt->next
    STR      R1, [R0]    ;      RunPt = R1
    LDR      R2, [R1]    ; 7) new thread PSP in R2
    LDM      R2, {R4-R11} ; 8) restore regs r4-11
    ADDS     R2, R2, #0x20
    MSR      PSP, R2     ; Load PSP with new process SP
    ORR      LR, LR, #0x04 ; 0xFFFFFDF (return to thread PSP)
    CPSIE    I           ; 9) run with interrupts enabled
    BX      LR           ; 10) restore R0-R3,R12,LR,PC,PSR

```

Lecture 4

OS calls implemented with trap (SVC)

16

NVIC

- Set priorities
 - PendSV low
 - Timer1 high
- Trigger PendSV
 - `NVIC_INT_CTRL_R`
 - Page 160 of tm4c123gh6pm.pdf

Launch

- Set SysTick period
- Set PendSV priority
- Using RunPt
 - Pop initialize Reg
- Enable interrupts
- Branch to user

Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

17

To do first (1)

- Debugging
- Interrupts
- OS_AddThread
- Assembly
- NVIC
- PendSV
- OS_Suspend
- OS_Launch

To do last (2)

- Stack size
- FIFO size
- Timer1 period
- SysTick period
- PSP
 - Just use MSP
- Semaphores

Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

18

Lab 2 Part 1 (1)

- Debugging
 - How to breakpoint, run to, dump, heartbeat
- Interrupts
 - How to arm, acknowledge, set vectors
 - What does the stack look like? What is in LR?
- OS_AddThread
 - Static allocation of TCBs and Stack
 - Execute 1,2,3 times and look at TCBs and Stack
- Assembly
 - PendSV, push/pull registers, load and store SP
 - Enable, disable interrupts
 - Access global variables like RunPt

Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

19

Lab 2 Part 1 (2)

- NVIC
 - Arm/disarm, priority
- PendSV
 - How to trigger
 - Write a PendSV handler to switch tasks
- OS_Suspend (scheduler and PendSV)
- OS_Launch (*this is hard*)
 - Run to a line at the beginning of the thread
 - Make sure TCB and stack are correct

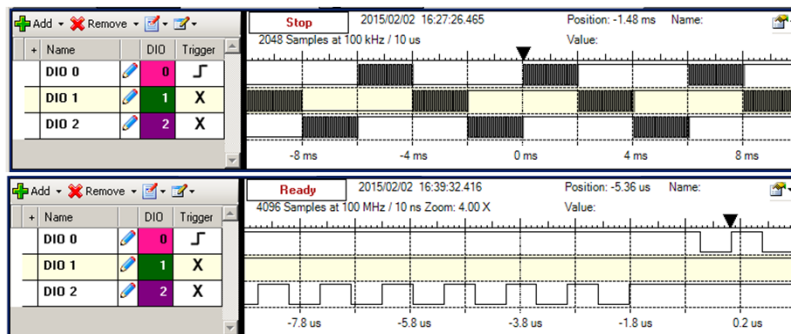
Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

20

Debugging tips

- Visualize the stacks
- Dumps and logs
- Logic analyzer

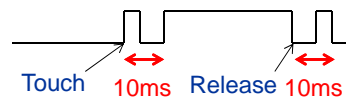


Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

21

Aperiodic Tasks (1)



- Switch debouncing
 - Assume a minimum touch time 500ms
 - Assume a maximum bounce time 10ms
- On touch
 - Signal user, call user function (no latency)
 - Disarm. **AddThread(&BounceWait)**
- BounceWait
 - Sleep for more than 10, less than 500 ms
 - Rearm. **OS_Kill()**

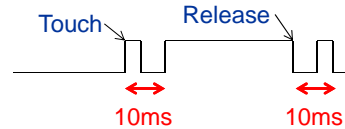
Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

22

Aperiodic Tasks (2)

- **Switch debouncing**
 - Assume a maximum bounce time 10ms
- **Interrupt on both rise and fall**
 - If it is a rise, signal touch event
 - If it is a fall, signal release event
 - Disarm. **AddThread(&DebounceTask)**
- **DebounceTask**
 - Sleep for 10 ms. **OS_Sleep(10)**
 - Rearm, Set a global with the input pin value
 - **OS_Kill()**



Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

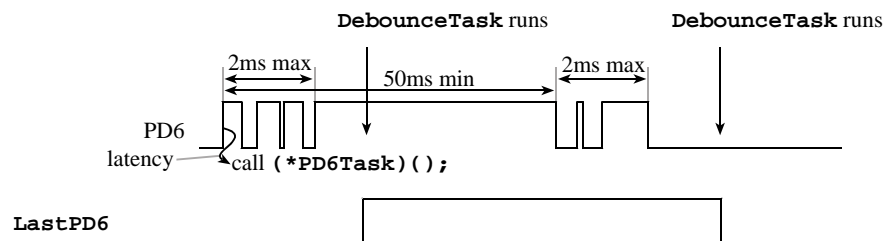
23

Switch Debounce

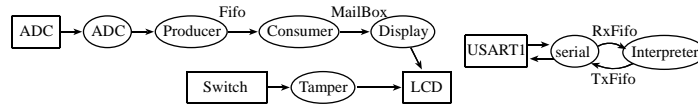
```
void static DebounceTask(void) {
    OS_Sleep(10); // foreground sleeping, must run within 50ms
    LastPD6 = PD6; // read while it is not bouncing
    GPIO_PORTD_ICR_R = 0x40; // clear flag6
    GPIO_PORTD_IM_R |= 0x40; // enable interrupt on PD6
    OS_Kill();
}

void GPIOPortD_Handler(void){
    if(LastPD6 == 0) // if previous was low, this is rising edge
        (*PD6Task()); // execute user task
    GPIO_PORTD_IM_R &= ~0x40; // disarm interrupt on PD6
    OS_AddThread(&DebounceTask);
}
```

Quiz 1, Question 9,
Spring 2012



Thread Communication



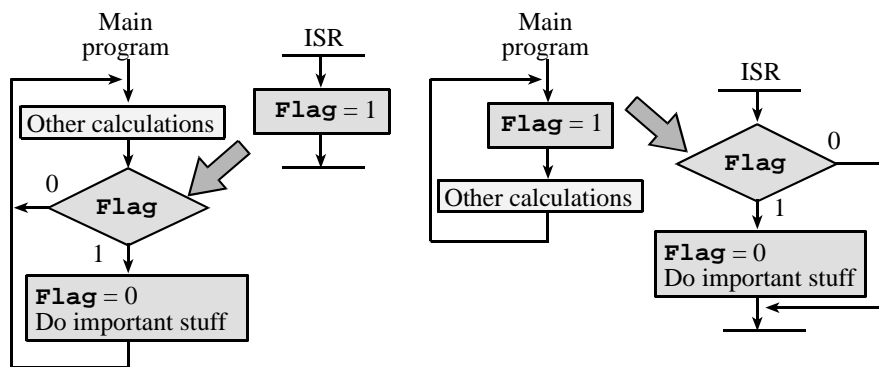
- Types
 - Data sharing (global variable)
 - Flag, Mailbox (one to one, unbuffered)
 - Pipes=FIFO (one to one, buffered, ordered)
 - Messages (many to many)
- Performance measures
 - Latency
 - Bandwidth
 - Error rate

Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

25

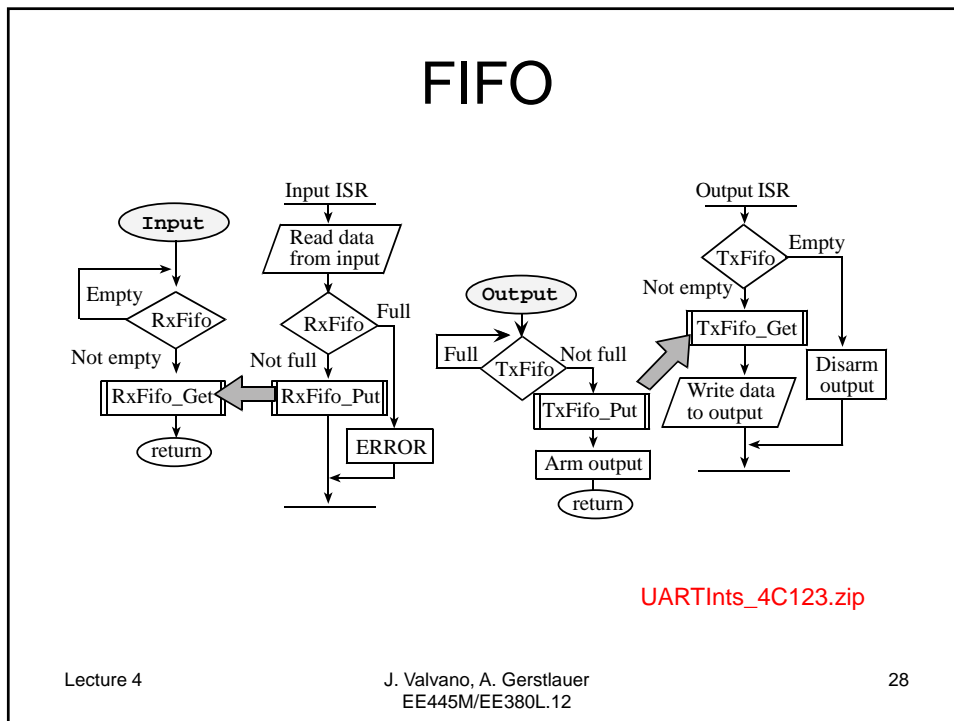
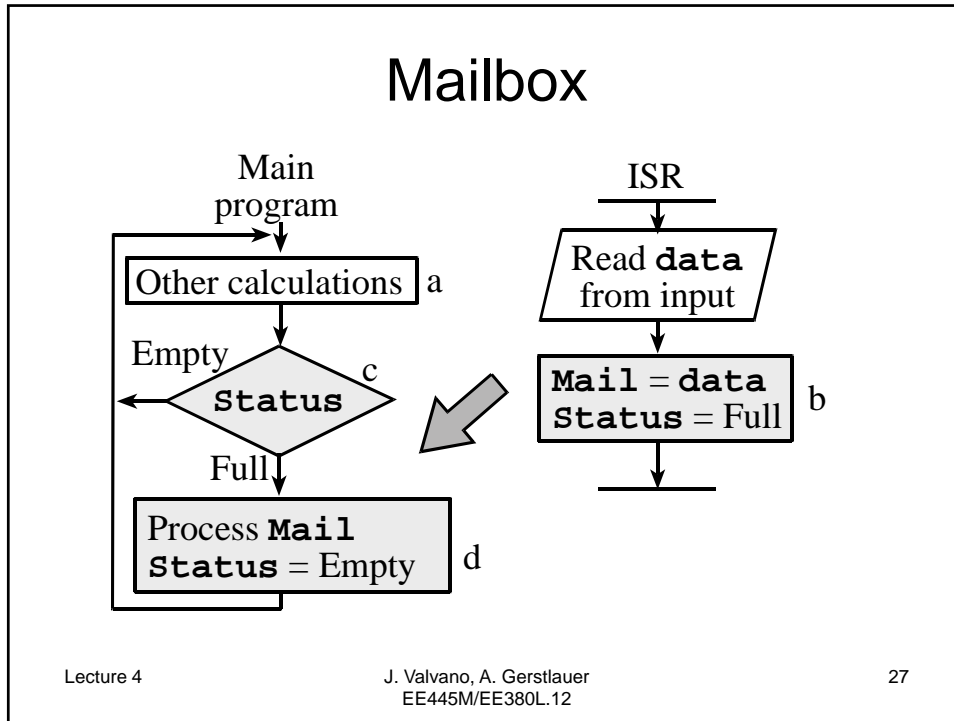
Flag



Lecture 4

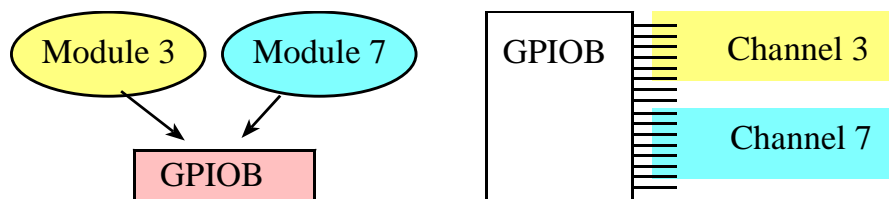
J. Valvano, A. Gerstlauer
EE445M/EE380L.12

26



Race, Critical Section

- Two or more threads access the same global
 - Permanently allocated shared resource (memory, I/O port, ...)
- At least one access is a write



Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

29

Race Condition

- Timing bug
 - Result depends on the sequence of threads
 - E.g. two threads writing to the same global
- Hard to debug
 - Depends on specific order/interleaving
 - Non-deterministic (external events)
 - Hard to reproduce/stabilize (“Heisenbug”)
- Critical or non-critical
 - Final program output affected?

Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

30

Critical Section

- Load/store architecture
 - Write access changes official copy
 - Read access creates two copies
 - Original copy in memory
 - Temporary copy in register
- Non-atomic access sequence
 - Begins/ends with access to permanent resource
 - Involves at least one write
 - RMW(+W), WW(+R/W), WR(+W), RR(+W)

Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

31

Thread-Safe, Reentrant

- Thread-safe code
 - No global resources
 - Variables in registers, stack
 - No critical section
 - No write access sequence
 - Mutual exclusion
 - Make accesses atomic (no preemption)
 - Prevent other threads from entering critical section
- Reentrant code
 - Multiple threads can (re-)enter same section
 - No non-atomic RMW, WW, WR sequence

Lecture 4

J. Valvano, A. Gerstlauer
EE445M/EE380L.12

32

Mutual Exclusion

- Disable all interrupts
 - Make atomic
- 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
 - Thread-safe, but not reentrant

Measure time with interrupts disabled
 - Maximum time
 - Total time

Lecture 4

LDREX
STREX Cortex-M3/M4F Instruction Set, pg. 50

Thread Synchronization

- Sequential
- Rendezvous, Barrier
 - Fork/spawn & join
- Trigger, event flags
 - OR, AND
 - I/O event (e.g., I/O edge, RX, TX)
- Time
 - Periodic time triggered (e.g., TATOMIS)
 - Sleep

Lecture 4

J. Valvano, A. Gerstlauer
 EE445M/EE380L.12

34