ENGR-304-L

Software Lab 06

Agenda

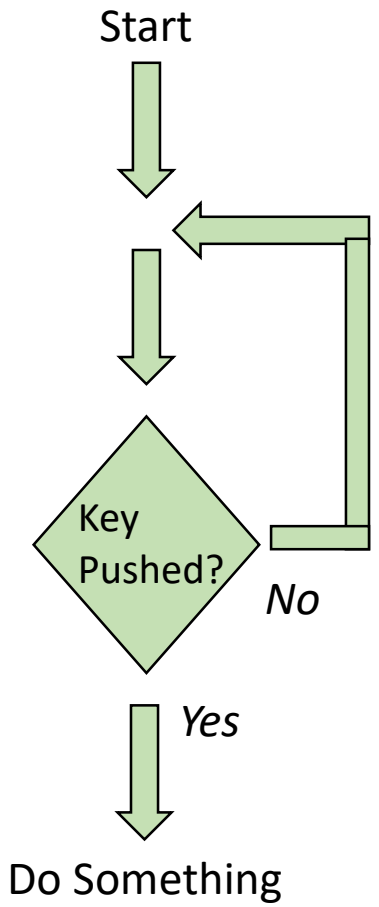
1. Attendance
2. Recap
3. Polling vs Interrupts
4. The ISR
5. Control Registers
6. Getting Started

Recap

- Bit-Masks
- IO Devices
- PIO Core & Timer - Altera DE2 IO Devices

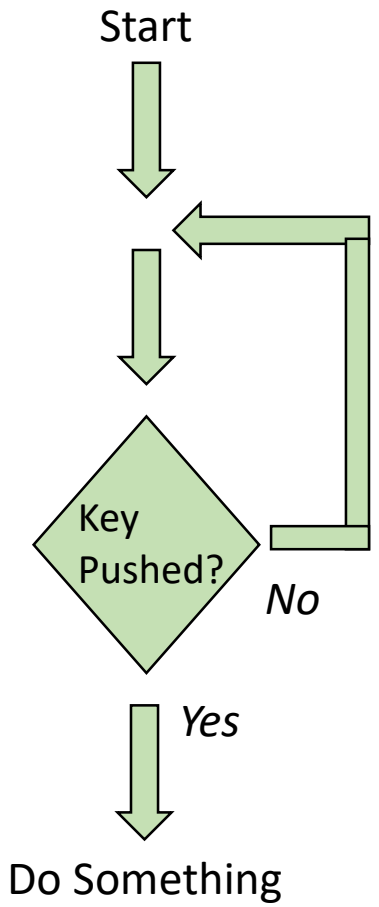
Interrupts

- Input devices sometimes use polling
- Main routine repeatedly reads/checks input



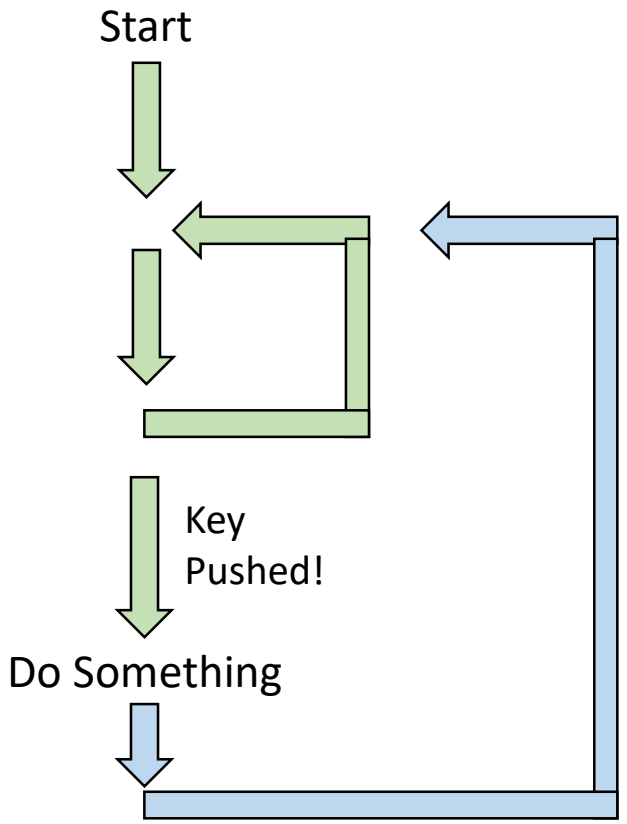
Interrupts

- Input devices sometimes use polling
- Main routine repeatedly reads/checks input
- Polling can have poor performance
  - *Poll period fast, wasted checks*
  - *Poll period slow, delayed reaction*
- Polling multiple inputs can add complexity
- Polling can also sometimes be the best solution



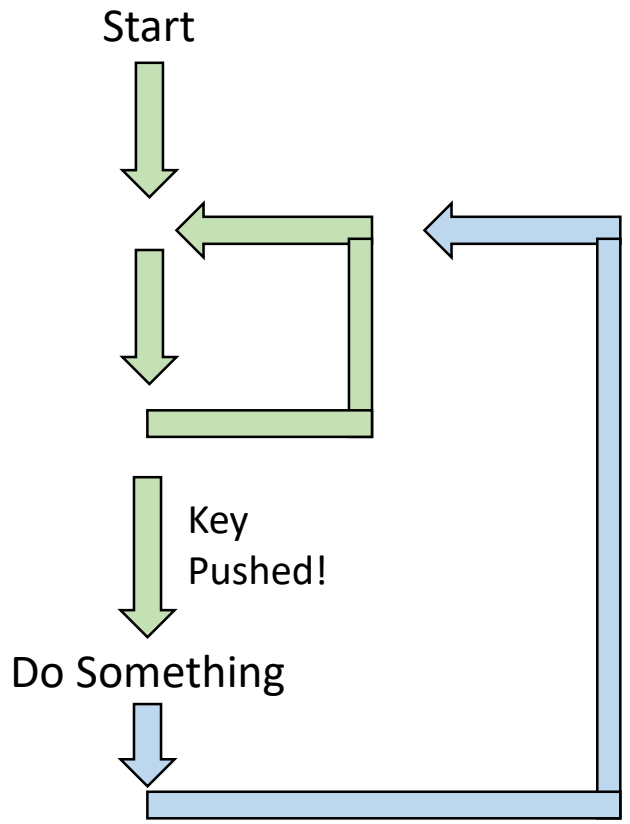
Interrupts

- Interrupts are an alternative to polling
- When input changes, take action immediately
- After taking action, resume where interrupted



Interrupts

- Interrupts are an alternative to polling
- When input changes, take action immediately
- After taking action, resume where interrupted
- Interrupts require hardware support
- Interrupt may alter behavior upon resume



Interrupts

Communication Example

- Polling ~ mailbox
- Interrupts ~ phone call

Carpooling Example

- Polling ~ waiting by the door
- Interrupts ~ hearing the doorbell

- Useful comparison & justification for each technique:

https://techdifferences.com/difference-between-interrupt-and-polling-in-os.html

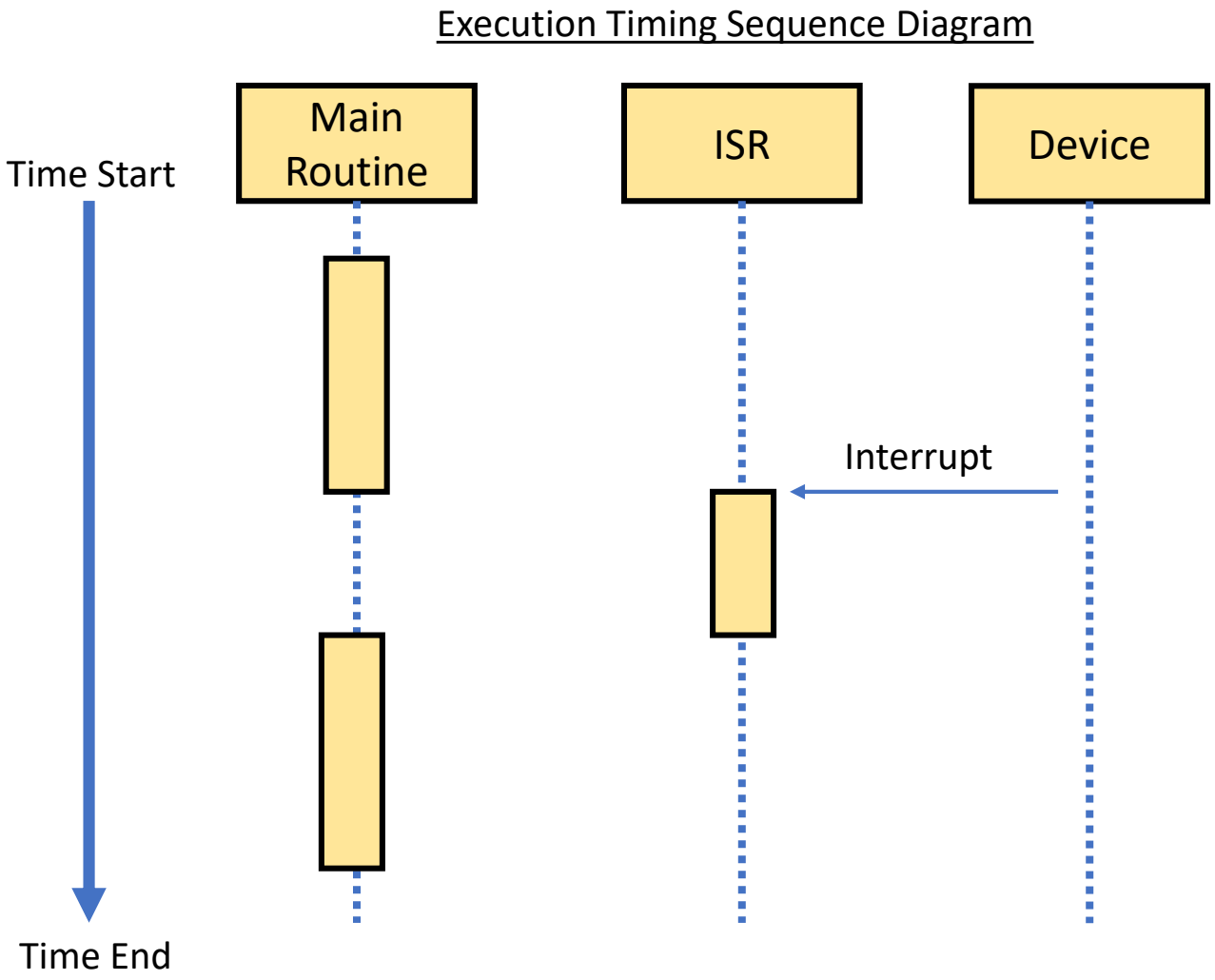
Interrupt Service Routine (ISR)

- CPU executing main routine, until interrupt occurs
- Main routine is paused, ISR is executed (like a function call)
- Main resumes after ISR “returns”

- ISR may need to receive data from main routine
- ISR may alter data to change behavior in the main routine

Interrupt Service Routine (ISR)

1. Main is executing
2. Device generates interrupt
3. Main pauses
4. ISR executes
5. Main resumes



Control Registers

- So far… 32 registers

Table 7–2. Nios II ABI Register Usage (Part 1 of 2)

Register	Name	Used by Compiler	Callee Saved (1)	Normal Usage
r0	zero	x		0x00000000
r1	at			Assembler Temporary
r2		x		Return Value (Least-significant 32 bits)
r3		x		Return Value (Most-significant 32 bits)
r4		x		Register Arguments (First 32 bits)
r5		x		Register Arguments (Second 32 bits)
r6		x		Register Arguments (Third 32 bits)
r7		x		Register Arguments (Fourth 32 bits)
r8		x		Caller-Saved General-Purpose Registers
r9		x		Caller-Saved General-Purpose Registers
r10		x		Caller-Saved General-Purpose Registers
r11		x		Caller-Saved General-Purpose Registers
r12		x		Caller-Saved General-Purpose Registers
r13		x		Caller-Saved General-Purpose Registers
r14		x		Caller-Saved General-Purpose Registers
r15		x		Caller-Saved General-Purpose Registers
r16		x	x	Callee-Saved General-Purpose Registers
r17		x	x	Callee-Saved General-Purpose Registers
r18		x	x	Callee-Saved General-Purpose Registers
r19		x	x	Callee-Saved General-Purpose Registers
r20		x	x	Callee-Saved General-Purpose Registers
r21		x	x	Callee-Saved General-Purpose Registers
r22		x	x	Callee-Saved General-Purpose Registers
r23		x	x	Callee-Saved General-Purpose Registers
r24	et			Exception Temporary
r25	bt			Break Temporary
r26	gp	x		Global Pointer
r27	sp	x		Stack Pointer
r28	fp	x		Frame Pointer (2)
r29	ea			Exception Return Address
r30	ba			Break Return Address
r31	ra	x		Return Address

Control Registers

- So far… 32 registers
- Now… add 16 more

Table 7–2. Nios II ABI Register Usage (Part 1 of 2)

Register	Name	Used by Compiler	Callee Saved (1)	Normal Usage
r0	zero	x		0x00000000
r1	at			Assembler Temporary
r2		x		Return Value (Least-significant 32 bits)
r3		x		Return Value (Most-significant 32 bits)
r4		x		Register Arguments (First 32 bits)
r5		x		Register Arguments (Second 32 bits)
r6		x		Register Arguments (Third 32 bits)
r7		x		Register Arguments (Fourth 32 bits)
r8		x		Caller-Saved General-Purpose Registers
r9		x		Caller-Saved General-Purpose Registers
r10		x		Caller-Saved General-Purpose Registers
r11		x		Caller-Saved General-Purpose Registers
r12		x		Caller-Saved General-Purpose Registers
r13		x		Caller-Saved General-Purpose Registers
r14		x		Caller-Saved General-Purpose Registers
r15		x		Caller-Saved General-Purpose Registers
r16		x	x	Callee-Saved General-Purpose Registers
r17		x	x	Callee-Saved General-Purpose Registers
r18		x	x	Callee-Saved General-Purpose Registers
r19		x	x	Callee-Saved General-Purpose Registers
r20		x	x	Callee-Saved General-Purpose Registers
r21		x	x	Callee-Saved General-Purpose Registers
r22		x	x	Callee-Saved General-Purpose Registers
r23		x	x	Callee-Saved General-Purpose Registers
r24	et			Exception Temporary
r25	bt			Break Temporary
r26	gp	x		Global Pointer
r27	sp	x		Stack Pointer
r28	fp	x		Frame Pointer (2)
r29	ea			Exception Return Address
r30	ba			Break Return Address
r31	ra	x		Return Address

Table 3–6. Control Register Names and Bits

Register	Name	Register Contents
0	`status`	Refer to Table 3–7 on page 3–12
1	`estatus`	Refer to Table 3–9 on page 3–14
2	`bstatus`	Refer to Table 3–10 on page 3–15
3	`ienable`	Internal interrupt-enable bits *(3)*
4	`ipending`	Pending internal interrupt bits *(3)*
5	`cpuid`	Unique processor identifier
6	Reserved	Reserved
7	`exception`	Refer to Table 3–11 on page 3–16
8	`pteaddr` *(1)*	Refer to Table 3–13 on page 3–16
9	`tlbacc` *(1)*	Refer to Table 3–15 on page 3–17
10	`tlbmisc` *(1)*	Refer to Table 3–17 on page 3–18
11	Reserved	Reserved
12	`badaddr`	Refer to Table 3–19 on page 3–21
13	`config` *(2)*	Refer to Table 3–21 on page 3–21
14	`mpubase` *(2)*	Refer to Table 3–23 on page 3–22
15	`mpuacc` *(2)*	Refer to Table 3–25 on page 3–23
16–31	Reserved	Reserved

NIOS Instructions & Registers

- Instructions:
  - rdctl
  - wrctl
  - eret

NIOS Instructions & Registers

- Instructions:
  - rdctl = read the value from a control register into a register, similar to mov
  - wrctl = write the value from a register into a control register, similar to mov
  - eret = return from the ISR

NIOS Instructions & Registers

- Instructions:
  - rdctl
  - wrctl
  - eret
- Control Registers:
  - ienable
  - status
  - ipending
  - estatus

NIOS Instructions & Registers

- Instructions:
  - rdctl
  - wrctl
  - eret
- Control Registers:
  - ienable = Allows enabling interrupts by #, each bit gets an interrupt #
  - status = Setting the PIE bit enables interrupts for the whole CPU
  - ipending = Holds flags for which interrupt(s) are asserted
  - estatus = Location to preserve status control register values during ISR

NIOS Instructions & Registers

- Instructions:
  - rdctl
  - wrctl
  - eret
- Control Registers:
  - ienable
  - status
  - ipending
  - estatus
- Registers:
  - sp
  - et
  - ea

NIOS Instructions & Registers

- Instructions:
  - rdctl
  - wrctl
  - eret
- Control Registers:
  - ienable
  - status
  - ipending
  - estatus
- Registers:
  - sp = shared by main & ISR, does not need to be protected, used in the same way
  - et = temporary register for use during ISR without protection
  - ea = like ra but for eret instead of ret

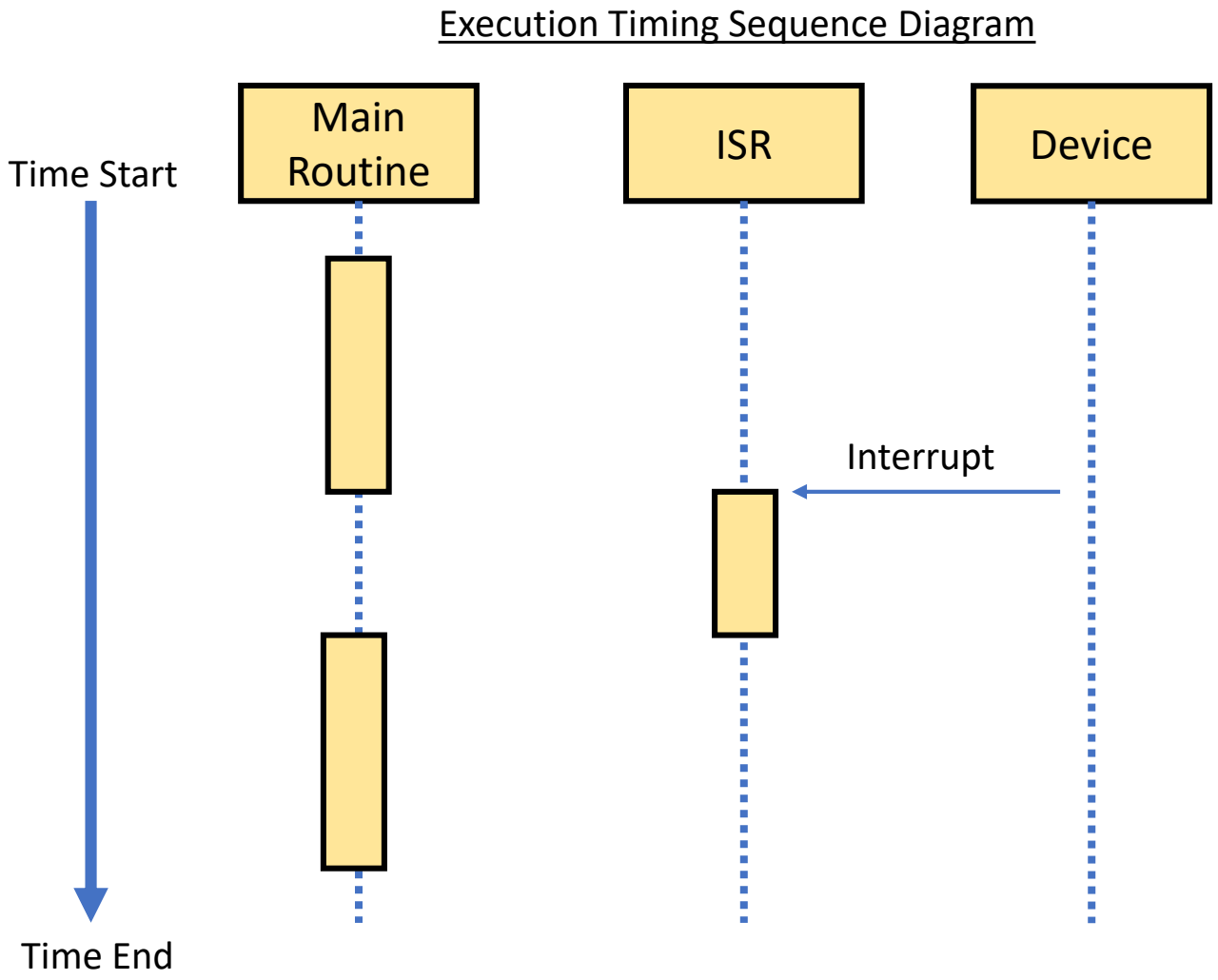
NIOS Instructions & Registers

- Instructions:
  - rdctl
  - wrctl
  - eret
- Control Registers:
  - ienable
  - status
  - ipending
  - estatus
- Registers:
  - sp
  - et
  - ea

Refer to the reference docs!

NIOS Interrupt Execution Sequence

1. Main is executing
2. Device generates interrupt
3. Main pauses
   - CPU HW copies PC to EA
   - CPU HW copies status to estatus
   - CPU HW prevents further interrupts
   - CPU HW sets PC to ISR's location
4. ISR executes
   - Checks interrupt assertions
   - Handles interrupts
   - Clears interrupt assertions
5. Main resumes
   - Restores estatus to status
   - Restores EA to PC
   - Re-allows interrupts



NIOS Interrupt Setup

- Interrupt Enabling
  - Enable the device's interrupts (ex: set mask register)
  - Enable the CPU's interrupts by #
  - Clear any left-over interrupt flags
  - Enable the CPU's interrupts globally (status PIE to 1)

NIOS Interrupt Setup

- Interrupt Enabling
  - Enable the device's interrupts (ex: set mask register)
  - Enable the CPU's interrupts by #
  - Clear any left-over interrupt flags
  - Enable the CPU's interrupts globally (status PIE to 1)
- ISR Creation
  - Correct EA if external interrupt (code given)
  - Push registers to the stack if needed
  - Check each enabled CPU interrupt by #
  - Handle asserted interrupts by taking action
  - Clear interrupt flags (ex: Timeout bit)
  - Pop registers from the stack if needed
  - Use eret to resume main routine

Lab SW06 Tips

- Lab SW06 builds on SW05, but use the new template files
- Read the template files and comments closely, becomes familiar with them before inserting your own code

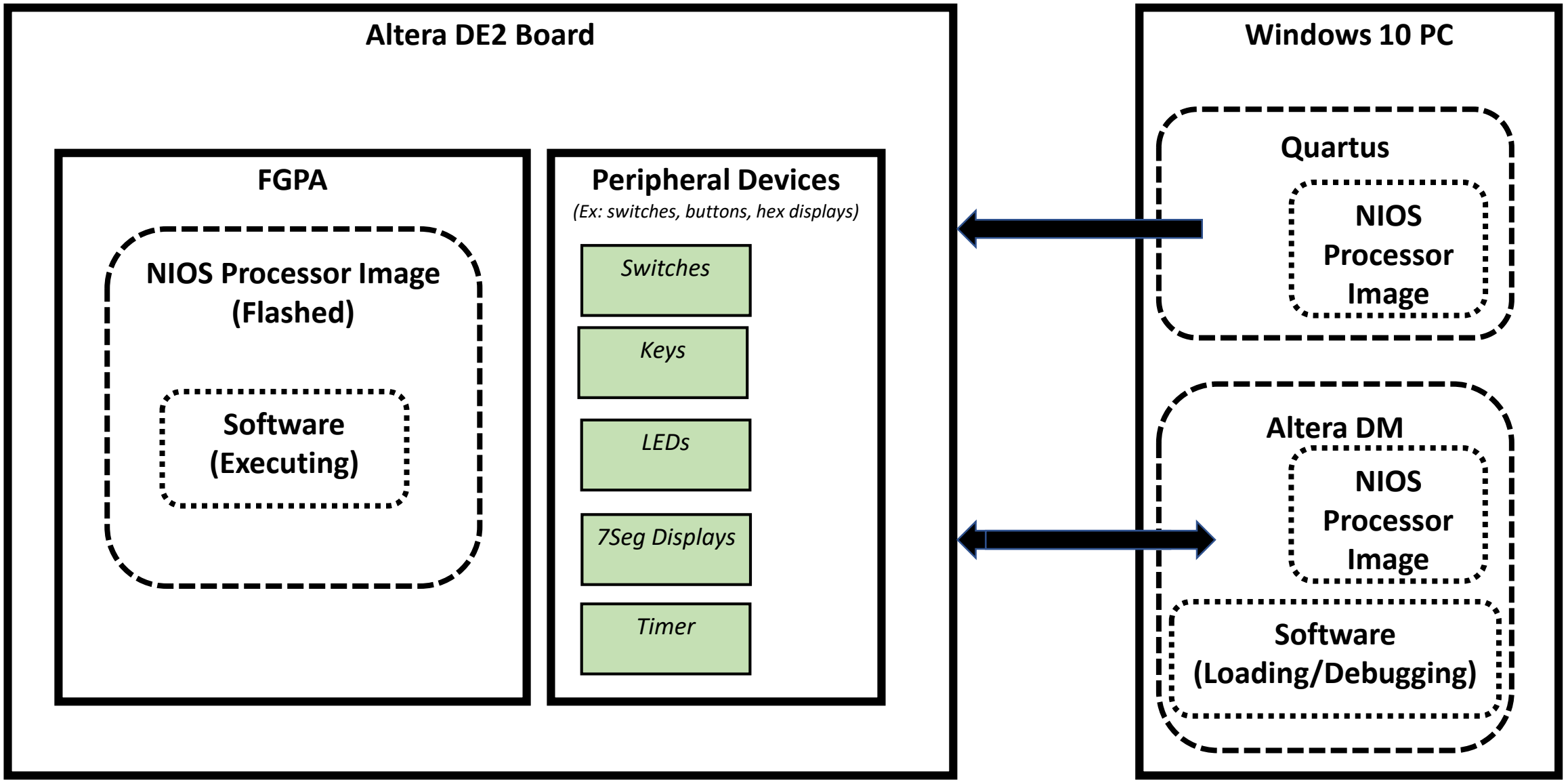
- Don’t forget to initialize IO devices before use (if needed)
- Don’t forget to protect registers on the stack during the ISR *(except r0, sp, et, ea)*

- Refer to “NIOS IO” and “NIOS Interrupts” slide decks for details & examples

Getting Started

1. Setup project directory for SW06 on the **H: drive**
2. Download SW06 files from Moodle or S: drive
3. Start following directions in the lab assignment document

Lab Components



PIO Core (Peripheral I/O device/port)

Table 9–2. Register Map for the PIO Core

Offset	Register Name		R/W	(n-1)	...	2	1	0
0	`data`	read access	R	Data value currently on PIO inputs				
		write access	W	New value to drive on PIO outputs				
1	`direction` *(1)*		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	`interruptmask` *(1)*		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	`edgecapture` *(1), (2)*		R/W	Edge detection for each input port.				

Interval Timer Core

Table 24–3. Register Map—32-bit Timer

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	*(1)*					RUN	TO
1	control	RW	*(1)*			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits [15:0])						
3	periodh	RW	Timeout Period – 1 (bits [31:16])						
4	snapl	RW	Counter Snapshot (bits [15:0])						
5	snaph	RW	Counter Snapshot (bits [31:16])						

Control Registers

Table 3–6. Control Register Names and Bits

Register	Name	Register Contents
0	`status`	Refer to Table 3–7 on page 3–12
1	`estatus`	Refer to Table 3–9 on page 3–14
2	`bstatus`	Refer to Table 3–10 on page 3–15
3	`ienable`	Internal interrupt-enable bits *(3)*
4	`ipending`	Pending internal interrupt bits *(3)*
5	`cpuid`	Unique processor identifier
6	Reserved	Reserved
7	`exception`	Refer to Table 3–11 on page 3–16
8	`pteaddr` *(1)*	Refer to Table 3–13 on page 3–16
9	`tlbacc` *(1)*	Refer to Table 3–15 on page 3–17
10	`tlbmisc` *(1)*	Refer to Table 3–17 on page 3–18
11	Reserved	Reserved
12	`badaddr`	Refer to Table 3–19 on page 3–21
13	`config` *(2)*	Refer to Table 3–21 on page 3–21
14	`mpubase` *(2)*	Refer to Table 3–23 on page 3–22
15	`mpuacc` *(2)*	Refer to Table 3–25 on page 3–23
16–31	Reserved	Reserved