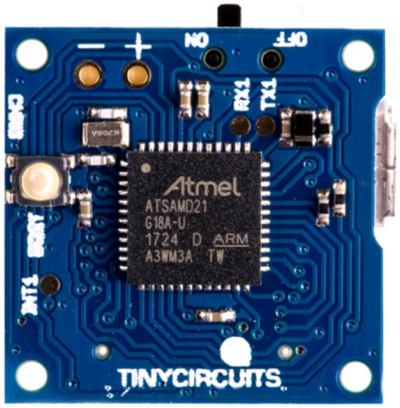
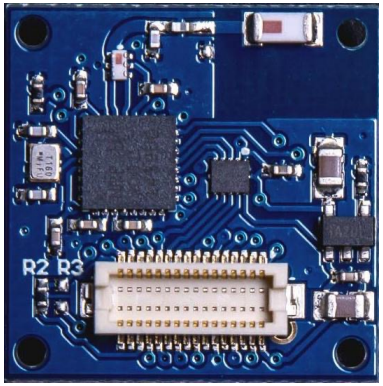
CSE190 Fall 2023



Lecture 7
Interrupts



Wireless Embedded Systems

Aaron Schulman

Interrupts

How peripherals notify the CPU that an event has occurred.

Example: GPIO detected that a button was just pressed.

Interrupts

Definition

- An event external to the currently executing process that causes a change in the normal flow of software execution; usually generated by hardware peripherals, but can also be generated by the CPU.
- Key point is that interrupts are asynchronous w.r.t. current software procedure
- Typically indicate that some device needs service immediately

Why interrupts?

- MCUs have many external peripherals
  - Keyboard, mouse, screen, disk drives, scanner, printer, sound card, camera, etc.
- These devices occasionally need the CPU to act
  - But we can't predict when
- We want to keep the CPU busy (or idle for power) between these events
  - Need a way for CPU to find out when a particular peripheral needs attention

Possible Solution: Polling

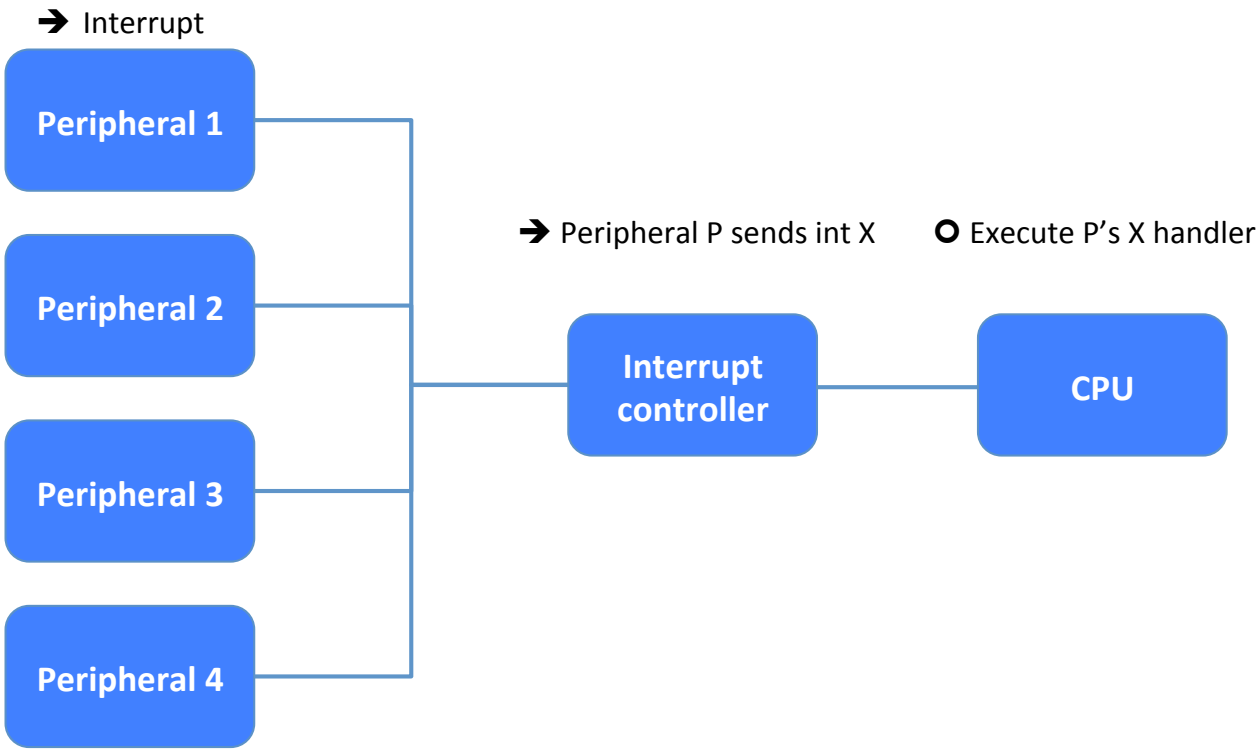
- CPU periodically checks each device to see if it needs service
  - "Polling is like picking up your phone every few seconds to see if you have a call. …"

Possible Solution: Polling

- CPU periodically checks each device to see if it needs service
  - "Polling is like picking up your phone every few seconds to see if you have a call. …"
  - Cons: takes CPU time even when no requests pending
  - Pros: can be efficient if events arrive rapidly

Alternative: Interrupts

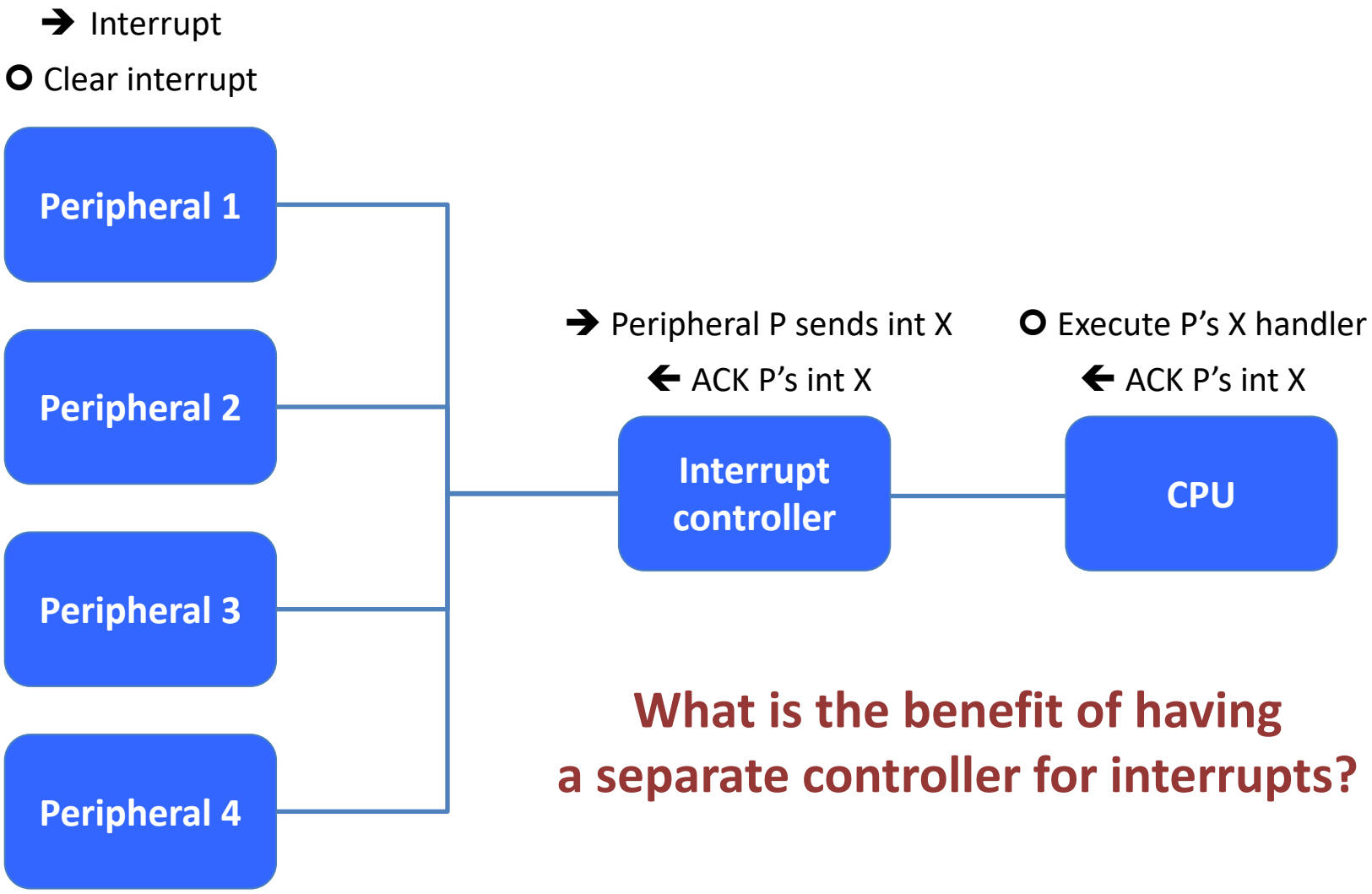
- Give each device a wire (interrupt line) that it can use to signal the processor



Alternative: Interrupts

- Give each device a wire (interrupt line) that it can use to signal the processor
  - When interrupt signaled, processor executes a routine called an interrupt handler to deal with the interrupt
  - No overhead when no requests pending

How do interrupts work?

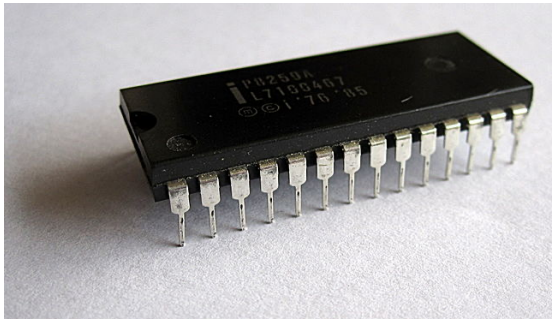


What is the benefit of having a separate controller for interrupts?

The Interrupt controller

- **Handles simultaneous interrupts**
  - Receives and sequences interrupts while the CPU handles them
- **Maintains interrupt flags**
  - CPU can poll/clear interrupt flags instead of jumping to a handler
- **Multiplexes many wires to few wires**
  - CPU doesn't need a interrupt wire to each peripheral
- This is the **NVIC (Nested Vectored Interrupt Controller)** on ARM Cortex M CPUs.

Fun fact: Interrupt controllers used to be separate chips!



Intel 8259A IRQ chip

Image by Nixdorf - Own work

CPU execution of interrupt handlers

INTERRUPT OCCURS

1. Wait for instruction to end
2. Push the program counter to the stack
3. Push all active registers to the stack
4. Jump to the interrupt handler in the interrupt vector
5. Run the interrupt handler code
6. Pop the registers off of the stack
7. Pop the program counter off of the stack

How a firmware programmer uses interrupts

1. Tell the peripheral which interrupts you want it to enable.
2. Tell the interrupt controller to enable that interrupt.
3. Tell the interrupt handler what that interrupt's priority is.
4. Tell the processor where the interrupt handler is (function address).
5. When the interrupt handler fires, do the work then clear the int flag.

How do you use your first interrupt handler in the project?

1. Setup the Timer to fire interrupt (IRQ) for TIM2 match

2. Setup NVIC to fire TIM2 interrupt

```
// Set TIM2 Interrupt Priority to Level 0
NVIC_SetPriority(TIM2_IRQn, 0);
// Enable TC3's NVIC Interrupt Line
NVIC_EnableIRQ(TIM2_IRQn);
```

3. Write TIM2 Interrupt Handler function

```
void TIM2_IRQHandler() {
}
```

How does the CPU know to call that handler? Interrupt Vectors

```
******************************************************************************
* @file      startup_stm32l475xx.s
* @author    MCD Application Team
* @brief     STM32L475xx devices vector table for GCC toolchain.
*            This module performs:
*                - Set the initial SP
*                - Set the initial PC == Reset_Handler,
*                - Set the vector table entries with the exceptions ISR address,
*                - Configure the clock system
*                - Branches to main in the C library (which eventually
*                  calls main()).
*            After Reset the Cortex-M4 processor is in Thread mode,
*            priority is Privileged, and the Stack is set to Main.
******************************************************************************
* @attention
*
* Copyright (c) 2017 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
******************************************************************************
*/
```