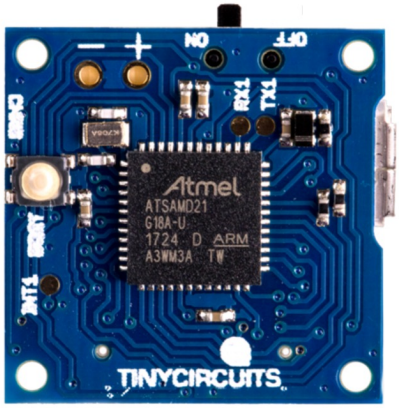
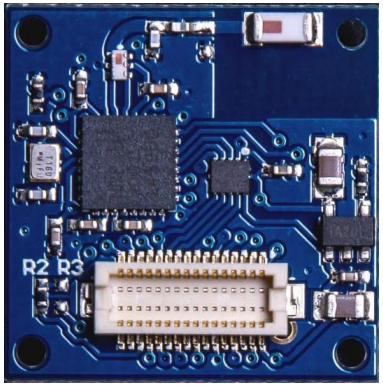
CSE190 Fall 2022



Lecture 8

Interrupts (cont.)

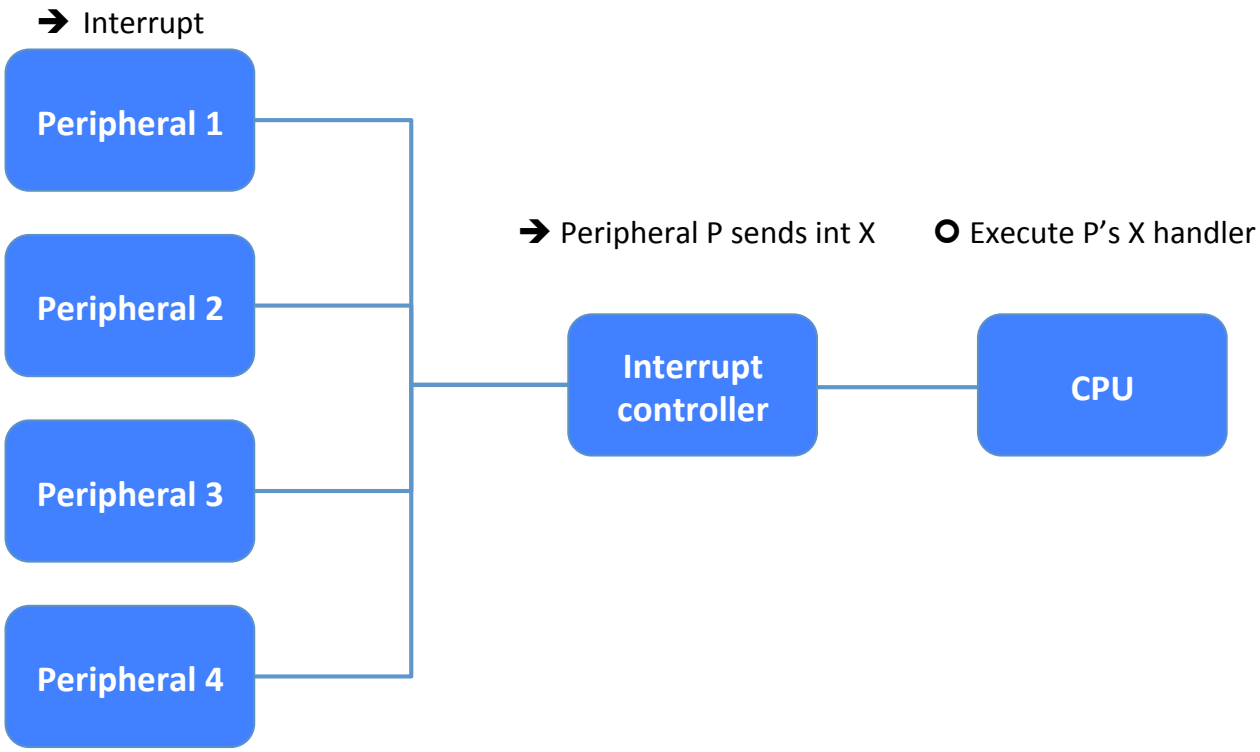


Wireless Embedded Systems

Aaron Schulman

Alternative: Interrupts

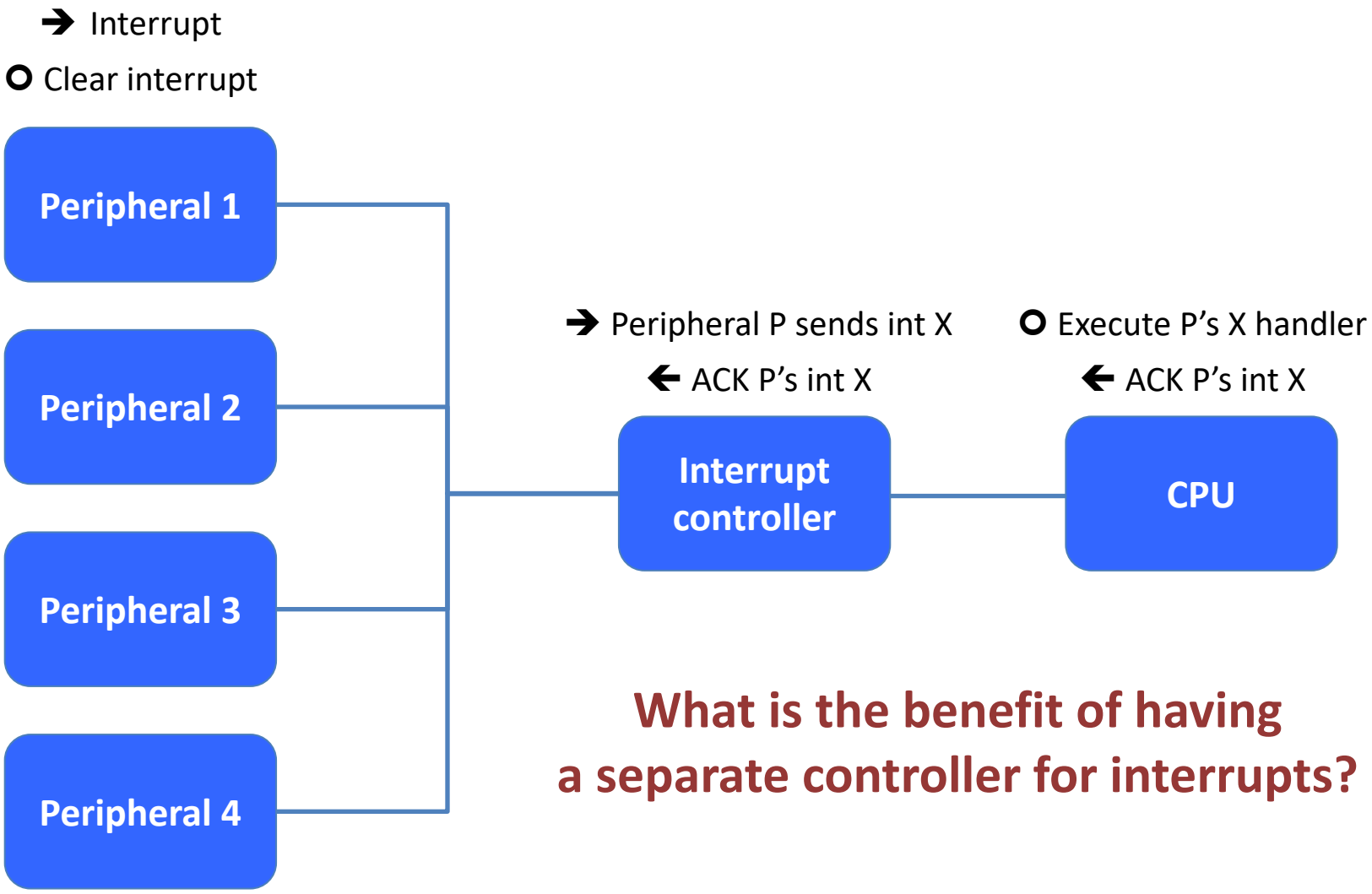
- Give each device a wire (interrupt line) that it can use to signal the processor



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?

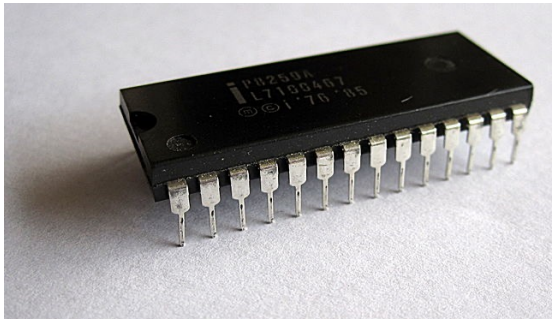


What is the benefit of having a separate controller for interrupts?

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.

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

1. Setup the Timer to fire interrupt 3 for TC match

2. Setup NVIC to fire TC3 interrupt

```
// Set TC3 Interrupt Priority to Level 0
NVIC_SetPriority(TC3_IRQn, 0);
// Enable TC3's NVIC Interrupt Line
NVIC_EnableIRQ(TC3_IRQn);
```

3. Write TC3 Interrupt Handler function

```
void TC3_Handler() {
}
```

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

```
typedef struct _DeviceVectors
{
  /* Stack pointer */
  void* pvStack;

  /* Cortex-M handlers */
  void* pfnReset_Handler;
  void* pfnNMI_Handler;
  void* pfnHardFault_Handler;
  void* pfnReservedM12;
  void* pfnReservedM11;
  void* pfnReservedM10;
  void* pfnReservedM9;
  void* pfnReservedM8;
  void* pfnReservedM7;
  void* pfnReservedM6;
  void* pfnSVC_Handler;
  void* pfnReservedM4;
  void* pfnReservedM3;
  void* pfnPendSV_Handler;
  void* pfnSysTick_Handler;

  /* Peripheral handlers */
  void* pfnPM_Handler;              /*  0 Power Manager */
....
  void* pfnSERCOM5_Handler;         /* 14 Serial Communication Interface 5 */
  void* pfnTCC0_Handler;            /* 15 Timer Counter Control 0 */
  void* pfnTCC1_Handler;            /* 16 Timer Counter Control 1 */
  void* pfnTCC2_Handler;            /* 17 Timer Counter Control 2 */
  void* pfnTC3_Handler;             /* 18 Basic Timer Counter 3 */
```

Interrupt vector is loaded into memory in startup.c (included)

```
/* Exception Table */
__attribute__ ((section(".vectors")))
const DeviceVectors exception_table = {

    /* Configure Initial Stack Pointer, using linker-generated symbols */
    (void*) (&_estack),

    (void*) Reset_Handler,
    (void*) NMI_Handler,
    (void*) SVC_Handler,
    (void*) (0UL), /* Reserved */
    (void*) (0UL), /* Reserved */
    (void*) PendSV_Handler,
    (void*) SysTick_Handler,

    /* Configurable interrupts */
    (void*) PM_Handler,           /*  0 Power Manager */
    (void*) SYSCTRL_Handler,      /*  1 System Control */
    (void*) WDT_Handler,          /*  2 Watchdog Timer */
    (void*) RTC_Handler,          /*  3 Real-Time Counter */
    (void*) EIC_Handler,          /*  4 External Interrupt Controller */
    (void*) NVMCTRL_Handler,      /*  5 Non-Volatile Memory Controller */
    (void*) DMAC_Handler,         /*  6 Direct Memory Access Controller */
#ifdef ID_USB
    (void*) USB_Handler,          /*  7 Universal Serial Bus */
#else
    (void*) TCC0_Handler,         /* 15 Timer Counter Control 0 */
    (void*) TCC1_Handler,         /* 16 Timer Counter Control 1 */
    (void*) TCC2_Handler,         /* 17 Timer Counter Control 2 */
    (void*) TC3_Handler,          /* 18 Basic Timer Counter 0 */
```