

Quiz 1 Monday/Tuesday (9/21,22)

- Topics: Activities 1-5
- Starts promptly at 10:05 – ½ hour length
- On Paper. Only allowed:
 - Writing tools
 - Calculator, and
 - **Crib Sheet** [provided]
- How to study?
 - Make sure you understand Activities 1-5!
 - Know the “why” behind what is done, not just the “how”.
 - Review lecture materials
- Show up late or miss quiz?
 - Take remaining time in quiz (or ½ time, whichever is larger) for no penalty
 - Take quiz with full time at 25% penalty

ECSE-2350
Embedded Control
Lecture 3

Introduction to Interrupts

Polling of Events

- **A method for getting the state of something.**

- For example: Waiting for pushbutton to be released:

```
while(GPIO_readPins(GPIOA,GPIO_PIN5) == 0);
```

- Or checking for a pushbutton to be pressed:

```
if(GPIO_readPins(GPIOA,GPIO_PIN2)) {...}
```

- Or waiting for a keyboard key to be entered:

```
uint8_t input = getchar();
```

Polling of Events

- **Requires quick repetitive checks of a value**
 - This limits the ability to check two things “simultaneously” ...
 - Combining the previous examples:

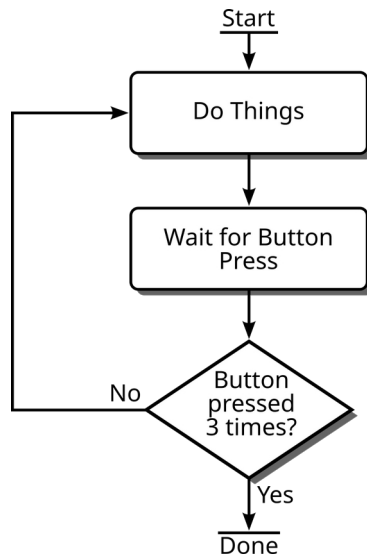
```
while(1){  
    uint8_t input = getchar();  
    if(GPIO_readPins(GPIOA,GPIO_PIN2))  
        printf("PA2 pressed");  
    while(GPIO_readPins(GPIOA,GPIO_PIN5)==0);  
}
```

- When waiting for keypress, program will miss PA2 press, etc.

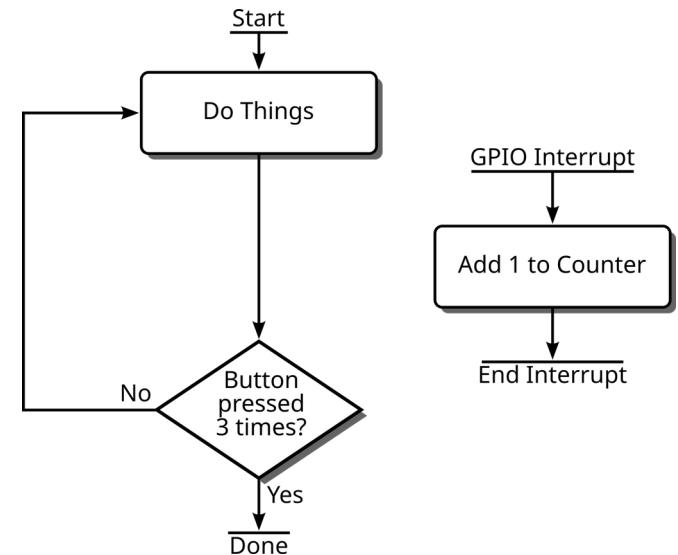
Interrupts

- Enable program to automatically detect signals *without requiring polling*.
 - **Example:** 3 Pushbutton presses to end program

Polling

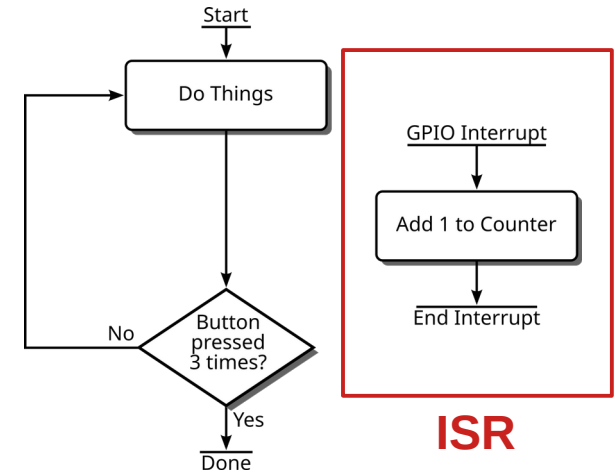


Interrupt



Interrupts – How to use

- Set up the module like you normally.
- Identify the desired interrupt.
 - Multiple interrupt sources per module are typical
- Write function to run when interrupt happens
 - This function is the **Interrupt Service Routine (ISR)**, **IRQ Handler**, etc.
 - Different mechanism for doing this, dependent on μC
 - This function is **NOT** called by your code. It is called by the hardware
- Enable the Interrupt



Interrupts – ISR or IRQ Handler

- The ISR is just a function that runs when an interrupt triggers.
- The ISR **MUST** acknowledge the interrupt.
 - Aka: “Clear the interrupt flag”
- Can do direct actions in ISR or...
- Set a variable to mark the event happened
 - Used back in main program loop
- Generally want ISRs to be SHORT and SIMPLE
 - **DO NOT USE printf in an interrupt!**
 - **Slow, can break system timing and printf**
 - **Exception: Temporary debugging statements to verify IRQ functionality**

Interrupts

- **Example:** Using GPIO interrupt to end program after 3 button presses.

```
uint8_t presses = 0;
int main() {
    sysInit();
    gpioInit();

    uint8_t running = 1;
    while(running){
        // Other code here...
        if(presses == 3){
            running = 0;
        }
    }
}
void GPIO_IRQHandler(){
    GPIO_clearInterrupt(GPIOA, GPIO_PIN2);
    presses++;
}
```

This is an example of code that is

Non-Blocking

Checking of the amount of presses occurred does not prevent “Other code” from running

Interrupts

- **Example:** Using GPIO interrupt to end program after 3 button presses.

```
uint8_t presses = 0, pressed = 0;
int main() {
    sysInit();
    gpioInit();

    while(presses < 3){
        // Other code here...
        while(!pressed);
        pressed = 0;
        presses++;
    }
}

void GPIO_IRQHandler(){
    GPIO_clearInterrupt(GPIOA, GPIO_PIN2);
    pressed = 1;
}
```

This is an example of code that is

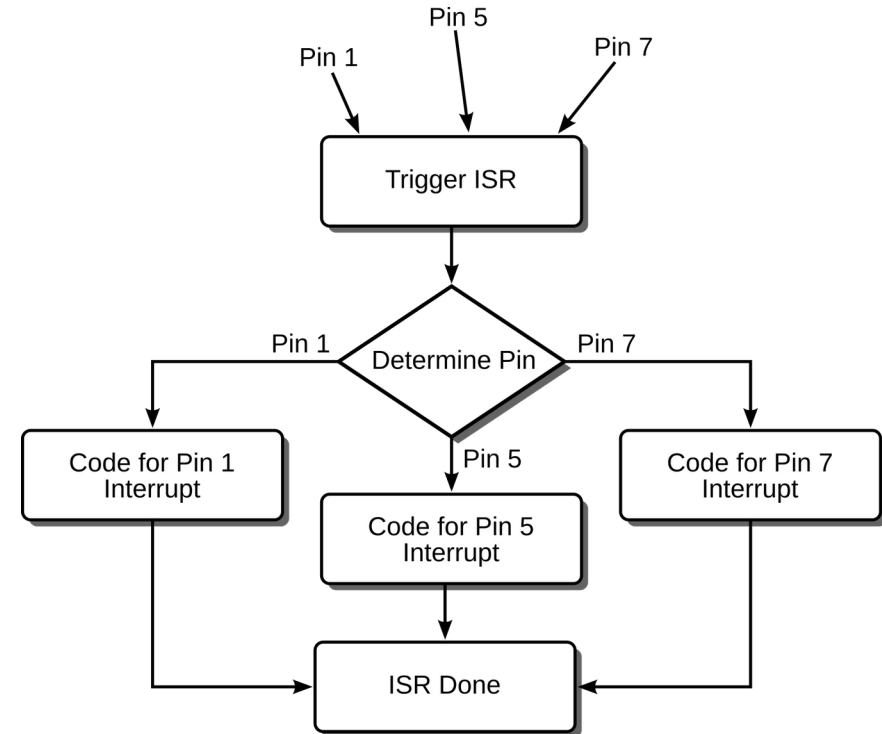
Blocking

Even though interrupts are used, “Other code” cannot run again until a press occurs.

Variable `pressed` is a “*flag*” – a variable used to indicate an event has occurred

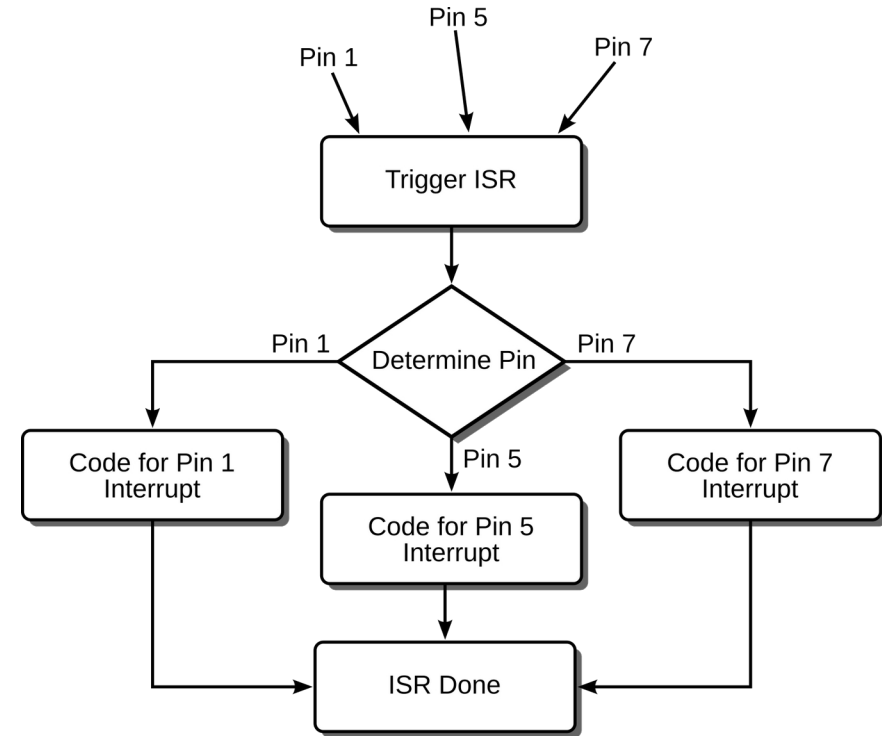
Interrupts – Multiple Sources

- Multiple triggers can cause single interrupt/IRQ Handler.
 - Example: GPIO: One IRQ Handler per all GPIO pins.
 - In ISR: Check to see what triggered the interrupt
 - Address each trigger individually
 - Not necessary if only 1 trigger active!



Interrupts – Multiple Sources (flags)

```
uint8_t pin1=0,pin5=0,pin7=0; // flags!  
int main() {  
    ...  
    while(1){  
        if(pin1){  
            pin1 = 0;  
            // Code for Pin 1 Interrupt  
        }  
        if(pin5){  
            ...  
        } // Repeat for pin7  
    }  
}  
  
void GPIOA_IRQHandler(){  
    uint32_t pend = GPIO_getPendingInterrupts(GPIOA);  
    if(pend & GPIO_PIN1){ // *** Do not use && ***  
        GPIO_clearInterrupt(GPIOA,GPIO_PIN1);  
        pin1 = 1;  
        // Repeat for Pins 5&7  
    }  
}
```



Interrupts – Multiple Sources

```
int main() {
    sysInit();
    gpioInit();
    while(1){
    }
}

void GPIO_IRQHandler(){
    uint32_t pend = GPIO_getPendingInterrupts(GPIOA);
    if(pend & GPIO_PIN1){ // *** Do not use && ***
        GPIO_clearInterrupt(GPIOA,GPIO_PIN1);
        // Code for Pin 1 Interrupt
    }
    else if(pend & GPIO_PIN5){
        GPIO_clearInterrupt(GPIOA,GPIO_PIN5);
        // Code for Pin 5 Interrupt
    } else ...
    // Repeat for Pin 7
}
```

