Quiz 1 Tomorrow (2/5)

- **Topics: Activities 1-5**
- **Starts promptly at 10:05 (S1) or 2:05 (S2) – ½ hour length**
- **On Gradescope: open computers, notes, website, CCS, calculator**
  - **NO Chats, human or otherwise. Search engine AI results included.**
  - **NO Copying from or Pasting to the Quiz**
  - **Sit facing away from the center of the room such that screen is visible to staff (if we can't see your screen, we will ask you to adjust)**
- **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 4

Part 1: Timers

Timers: General Applications

- **Keeping track of time (surprise!)**
- Counting non-periodic events
- Producing periodic signals
  - Pulse Width Modulation
  - Square Wave Generator (e.g., Clocks)
- Producing One-Shot Events
- Measuring external frequencies
  - Encoders

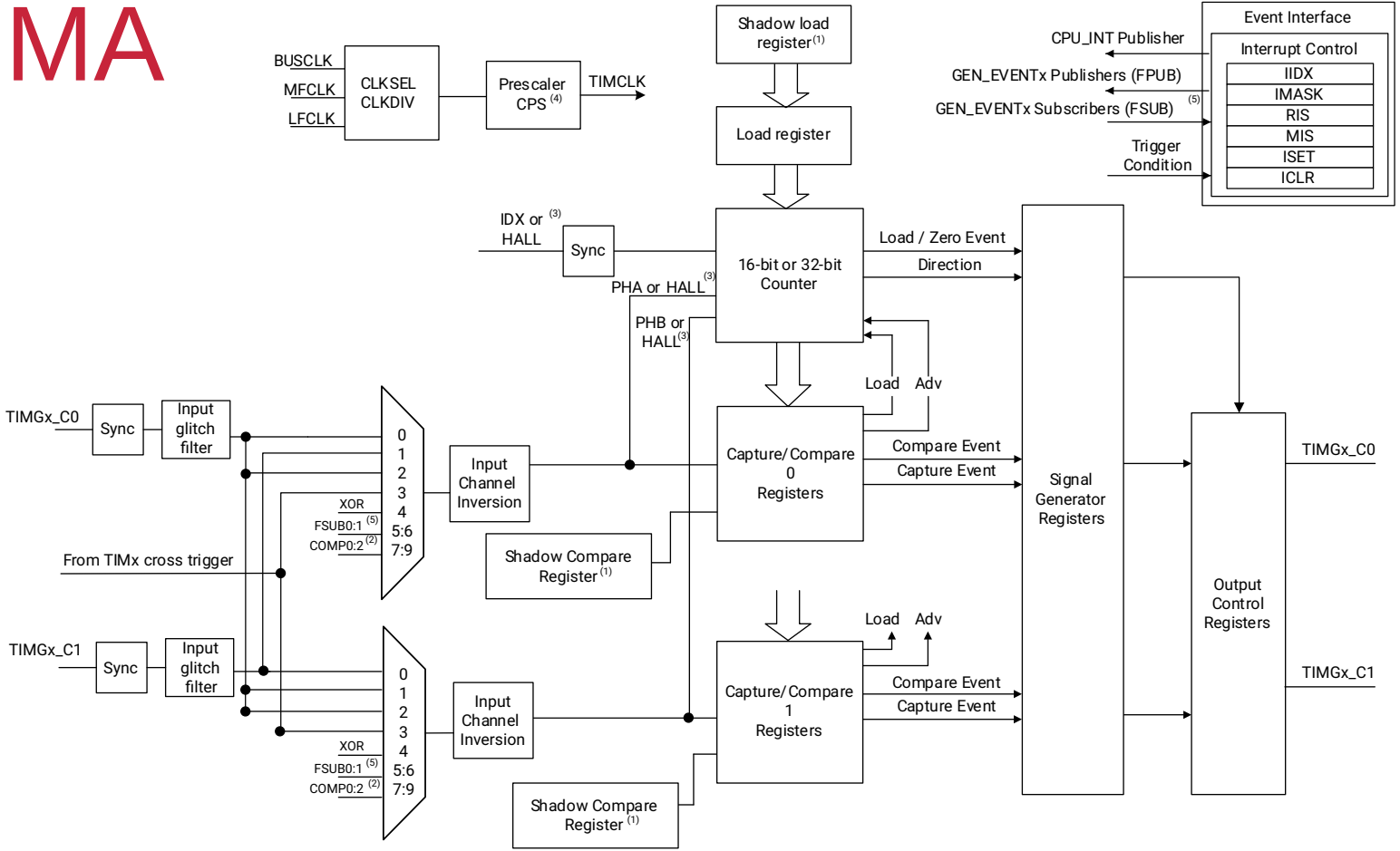
Introduction to Timers

- **On paper...**

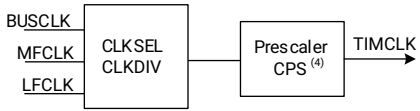
MSPM0G3507 Implementations

- Specific to the MSPM0G3507:
  - **SysTick**: Basic ARM Counter (24-bit)
    - Counts DOWN from an initial value
  - **TIMG**: TI General Purpose Timers (16-bit)
    - Counts DOWN, UP or UP/DOWN to period/max value
    - Contains 2 “Capture-Compare Registers (CCRs)”
    - 5 independent instances of TIMG on MSPM0G3507:
      - TIMG0, TIMG6, TIMG7, TIMG8, TIMG12 (32-bit)
    - Way more flexibility as compared to SysTick
  - **TIMA**: TI Advanced Timers (16-bit)
    - Similar to TIMG in base functionality. We have 2: TIMA0, TIMA1
  - Others: **Watchdog**, **Real Time Clock** ...

TIMG/TIMA


BUSCLK
MFCLK
LFCLK
CLKSEL
CLKDIV
Prescaler
CPS (4)
TIMCLK
Shadow load register (1)
Load register
Event Interface
CPU_INT Publisher
Interrupt Control
IIDX
IMASK
RIS
MIS
ISET
ICLR
GEN_EVENTx Publishers (FPUB)
GEN_EVENTx Subscribers (FSUB) (5)
Trigger Condition
IDX or HALL (3)
Sync
PHA or HALL (3)
PHB or HALL (3)
16-bit or 32-bit Counter
Load / Zero Event
Direction
Load
Adv
TIMGx_C0
Sync
Input glitch filter
XOR
FSUB0:1 (5)
COMP0:2 (2)
0
1
2
3
4
5:6
7:9
Input Channel Inversion
Capture/Compare 0 Registers
Compare Event
Capture Event
Shadow Compare Register (1)
From TIMx cross trigger
Signal Generator Registers
Output Control Registers
TIMGx_C0
TIMGx_C1
TIMGx_C1
Sync
Input glitch filter
Input Channel Inversion
Capture/Compare 1 Registers
Shadow Compare Register (1)

TIMG/TIMA



Input Clock Configuration

BUSCLK – 32 MHz peripheral bus clock
MFCLK – 4 MHz mid-frequency clock
LFCLK – 32.768 kHz low-frequency clock

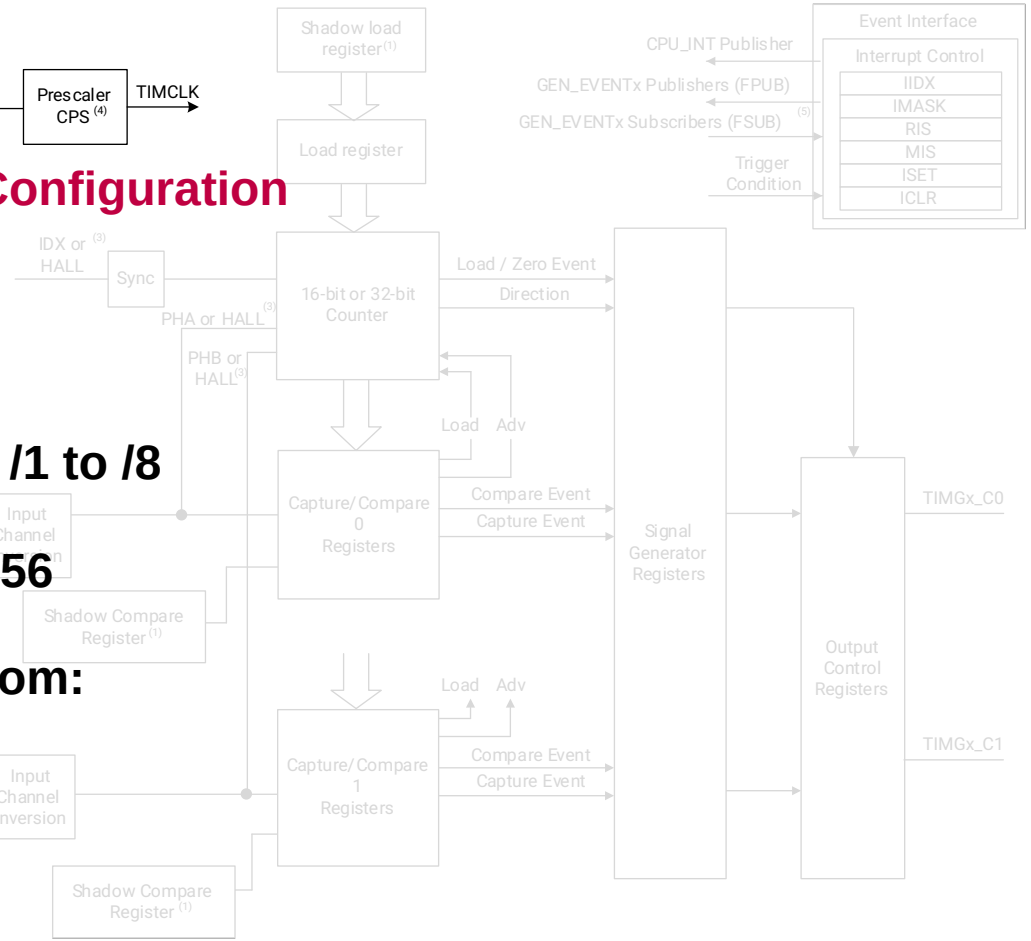
CLKDIV : Clock Divider, ranging between /1 to /8

Prescalar : Another Clock Divider, /1 to /256

TIMCLK : Actual counting rate. Ranges from:

Max: (32 MHz / 1) / 1 = 32 MHz

Min: (32.768 kHz / 8) / 256 = 16 Hz



TIMG/TIMA

Load register sets count period

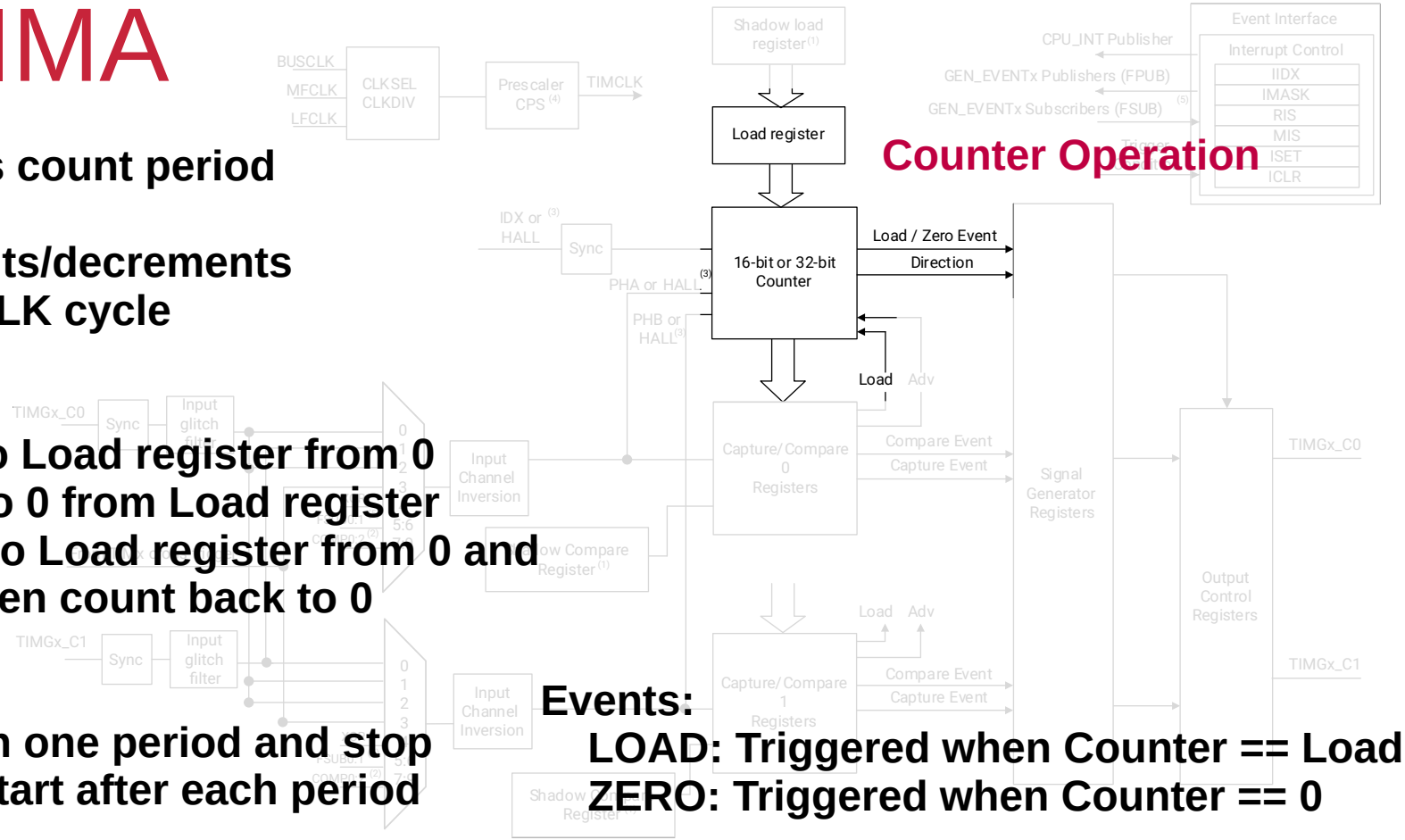
Counter increments/decrements with each TIMCLK cycle

Counter modes:

- **UP: Count up to Load register from 0**
- **DOWN: Down to 0 from Load register**
- **UP/DOWN: Up to Load register from 0 and then count back to 0**

Repetition:

- **ONE SHOT: Run one period and stop**
- **PERIODIC: Restart after each period**



Events:

- **LOAD: Triggered when Counter == Load**
- **ZERO: Triggered when Counter == 0**

engr2350_timers.h

```c
typedef struct {
    TIMER_MODE mode;
    TIMER_CLOCK_SRC clksrc;
    TIMER_CLOCK_DIV clkdivratio;
    uint8_t clkprescale;
    uint32_t period;
} Timers_TimerConfig;
```

```c
void Timers_initTimer(GPTIMER_Regs *timer,Timers_TimerConfig *config);
void Timers_startTimer(GPTIMER_Regs *timer);
```

Coming Up...

- Activity 7 Countdown Timer:
  - Application: A Countdown Timer
  - Wait for timer to overflow
  - Increment time
  - A “poor” implementation...why?
- Activity 8 Stopwatch: A ***better*** implementation...

ECSE-2350
Embedded Control
Lecture 4 - Cont.

Part 2: Interrupts / Event Based Prog.

Polling

- **A method for getting the state of something.**
  - For example: Waiting for pushbutton to be released:

```
while(GPIO_readPins(GPIOA,GPIO_PIN2) == 0);
```

  - Or waiting for 0.1 second and updating time:

```
while(1){
    if(Timers_getCounter(TIMG0)==######){
        Update_Time();
    }
}
```

BAD! Slightly better way in Activity 7 (still bad).

```
while(1){
    while(Timers_getCounter(TIMG0) != #####);
    Update_Time();
}
```

Polling

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

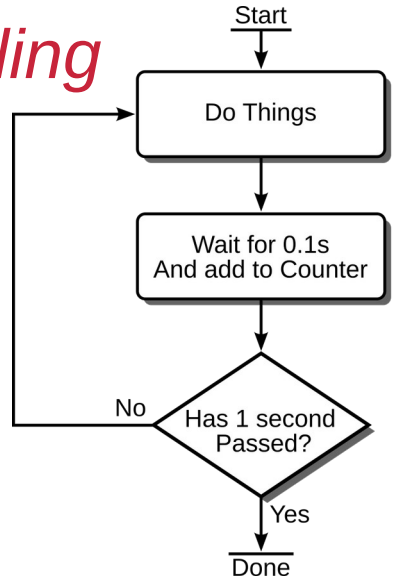
```
while(1){
    while(GPIO_readPins(GPIOA,GPIO_PIN2) == 0);
    if(Timers_getCounter(TIMG0)==#####){
        Update_Time();
    }
}
```

  - When looking for pushbutton release, program will miss timer reset...

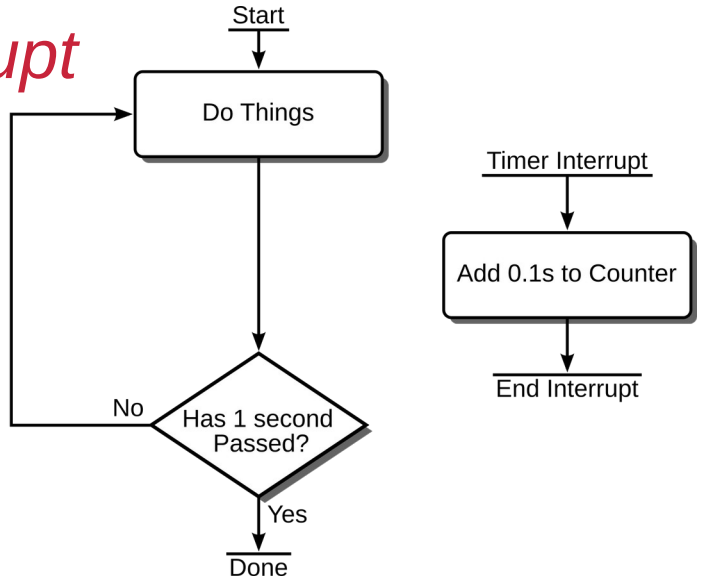
Interrupts

- Enable program to automatically detect signals *without requiring **polling***.

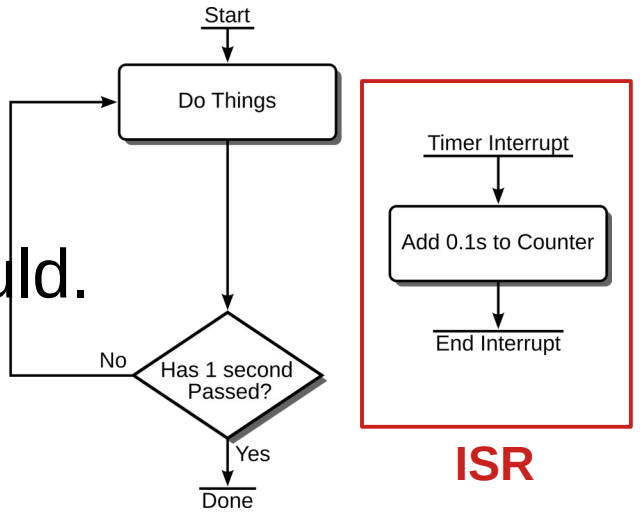
Polling



Interrupt



Interrupts – How to use



- Set up the module like you normally would.
- Identify the desired interrupt.
  - Typically multiple interrupt sources per module
- 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 Timer interrupt to count seconds

Blocking

```c
uint8_t timer_count;
int main() {
    sysInit();
    timerInit();

    uint32_t seconds;
    while(1){
        // Other code here...
        timer_count = 0;
        while(timer_count < 10);
        seconds++;
        printf("Seconds: %u\n",seconds);
    }
}

void TIMG0_IRQHandler(){
    Timers_clearInterrupt(TIMG0,TIMER_INTSRC_ZERO);
    timer_count++;
}
```

Non-Blocking

```c
uint8_t timer_count;
int main() {
    sysInit();
    timerInit();

    uint32_t seconds;
    while(1){
        // Other code here...
        if(timer_count >= 10){
            timer_count -= 10;
            seconds++;
            printf("Seconds: %u\n",seconds);
        }
    }
}

void TIMG0_IRQHandler(){
    Timers_clearInterrupt(TIMG0,TIMER_INTSRC_ZERO);
    timer_count++;
}
```

Event-Based Programming

- (1) µC programs can sometimes be written entirely or mostly in interrupts:
- (2) Other implementations would address interrupt actions as needed within the main program loop:

```
int main(){
    // In main...
    while(1);
}
void push1_IRQHandler(){
    // Do stuff for push1
}
void slide_IRQHandler(){
    // Do stuff for slide switch
}
void timer_IRQHandler(){
    // count time
    time++;
    if(time >= #){
        // Do stuff for timer elapse
    }
}
```

```
int main(){
    // In main...
    if(push1_press){ // set in interru
        push1_press = 0; // clear flag
        // Do stuff for push1
    }
    if(slide_toggle){
        slide_toggle = 0;
        // Do stuff for slide switch
    }
    if(timer_elapsed){
        timer_elapsed = 0;
        // Do stuff for timer elapse

    }
}
```

Event-Based Programming

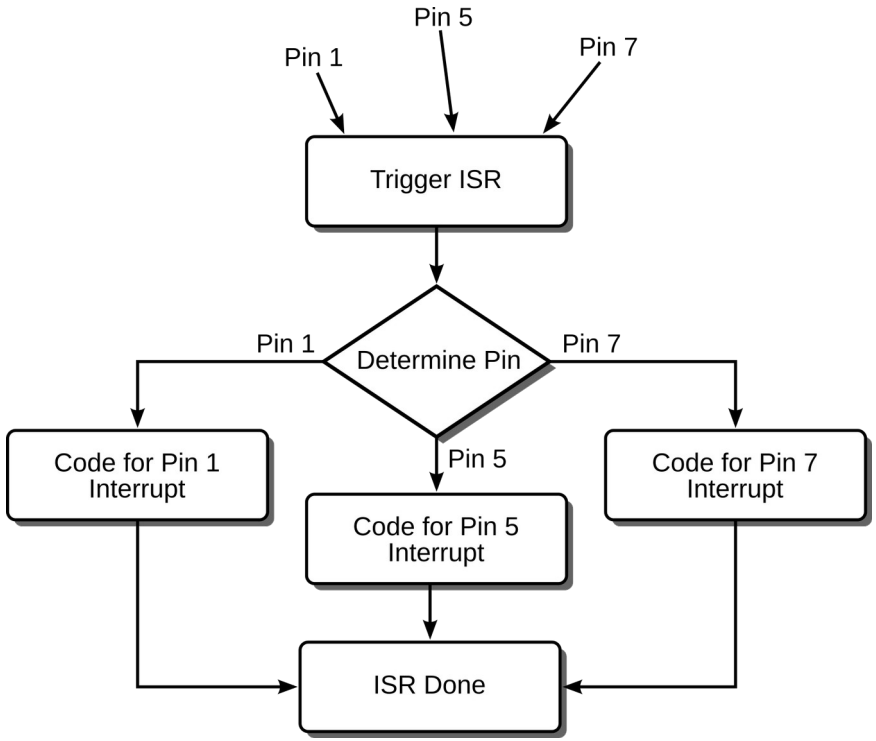
- Example: Detecting a pushbutton press and waiting for 1 second afterwards.

```c
uint8_t one_second_wait = 0;
void main(){
    sysInit();
    GPIOInit();
    timerInit();

    uint32_t var;
    while(1){
        while(one_second_wait);
        var++;
        printf("Current count: %u\n",var);
    }
}
```

```c
void GPIO_IRQHandler(){
    GPIO_clearInterrupt(GPIOA,GPIO_PIN2);
    __delay_cycles(320e3); //debounce
    Timers_startCounter(TIMG0); // in periodic mode
    one_second_wait = 1;
}

void TIMG0_IRQHandler(){
    Timers_clearInterrupt(TIMG0,TIMER_INTSRC_LOAD);
    timer_count++;
    if(timer_count == 10){
        timer_count = 0;
        one_second_wait = 0;
        Timers_stopTimer(TIMG0);
    }
}
```

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!



Activity 8

- Implement a stopwatch with interrupts
- Add GPIO interrupts to manage “Lap” times, pause, and reset.

Lab 2

- Simple game using Timers/Interrupts and an RGB LED
- Part A: Produce development plan/support
  - State diagrams, flow charts, pseudocode (etc.)
- Part B: Implement, test, and demonstrate portions of game
- Part C: Finish the game and demonstrate

Exam 1 – February 25th

- 1 hour, 50 minutes
- On paper – 2-sided crib sheet provided
  - Cannot use any other crib sheet
  - Crib sheet to-be-posted on Webex
- See Webex for last semester's exam
- Topics: Everything up-to and including this lecture
  - Activities 1-8, Lab 1 (partially lab 2)
- Bring calculator