Edge Interrupts

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #16

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Introduction:

Interrupts are a strange beast

- but they're really useful.

An interrupt is

- A subroutine
- Called by hardware

With interrupts, someone else is calling subroutines in your program.

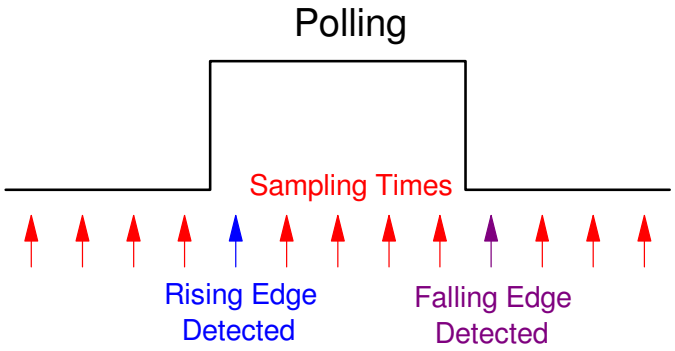
This lecture looks at

- Edge triggered interrupts on a Pi-Pico,
- Why you might want to use them, and
- Some things you can do with them

Polling vs. Interrupts

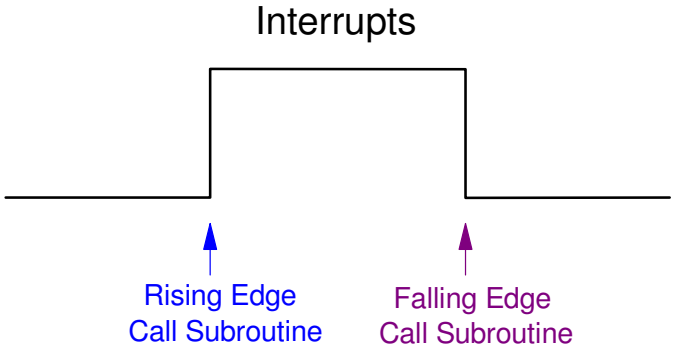
Polling:

- So far, we've been using polling
- The main routine checks the input pins over and over
- Edges are detected when the value of the pin is different from its last reading



Interrupts:

- In this lecture, we introduce interrupts
- With interrupts, hardware detects the rising & falling edge
- When detected, a subroutine is called
  - The interrupt service routine



Interrupts and Global Variables

Global variables are variables that *everyone* can see

In Computer Science I, you're told

- *Never never use global variables*
- *Global variables mess up your program*
- *Global variables make debugging a pain*

In this course,

- *Go ahead and use global variables*
- *Global variables can be useful*
- *Sometimes they're necessary*

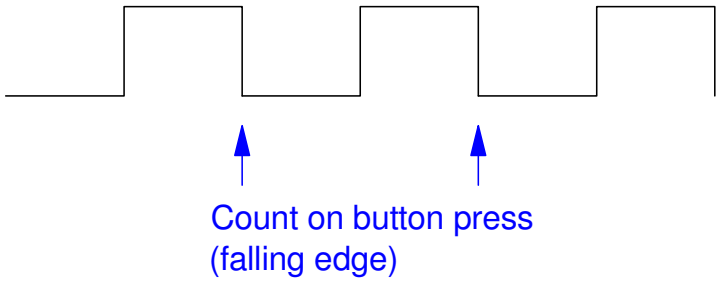
```
from machine import Pin

interrupt_flag=0
N = 0

pin = Pin(15,Pin.IN,Pin.PULL_UP)
def IntServe(pin):
    global interrupt_flag
    global N
    interrupt_flag=1
    N = N + 1

pin.irq(trigger=Pin.IRQ_FALLING,
handler=IntServe)

while(1):
    if(interrupt_flag):
        print("N = ", N)
        interrupt_flag=0
```

About the only way to pass data to a subroutine you didn't call is through global variables.

Edge Interrupts: Up Counter

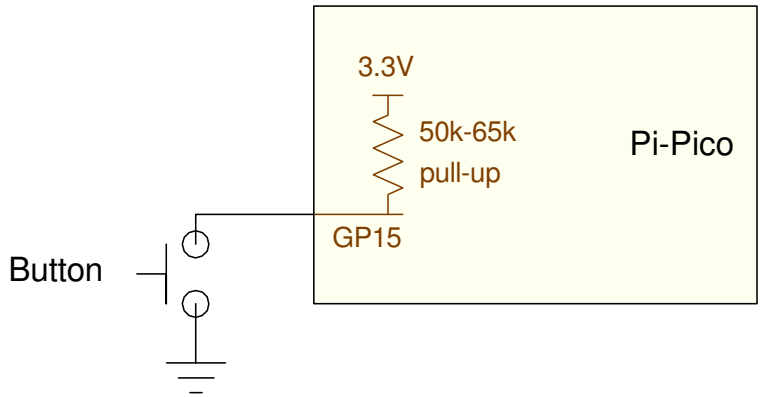
Interrupt Example: Up Counter

- Increase a count by one each time you press button GP15.



Similar to before, assume

- GP15 is connected to an internal pull-up resistor to +3.3V, and
- When you press the button, pin GP15 is grounded.



Net Result:

- Button Press = Falling Edge
- Count every falling edge on GP15

Up Counter: Python Code

IntServe(pin):

- The interrupt service routine
- Increment N every time called
- Set a flag every time N increments
- N & flag are global variables

Set up the interrupt for falling edges

- Tell Python the name of the interrupt service routine

Main Routine

- Each time the flag is set
- Print the count and
- Clear the flag

```
# Up Counter using interrupts
from machine import Pin

flag=0
N = 0

pin = Pin(15,Pin.IN,Pin.PULL_UP)
def IntServe(pin):
    global flag
    global N
    flag = 1
    N = N + 1

pin.irq(trigger=Pin.IRQ_FALLING,
handler=IntServe)

while(1):
    if(flag):
        print("N = ", N)
        flag=0
```

Shell

```
N = 1
N = 2
N = 3
```

Note the following:

- Each time you press the button, an interrupt is called (falling edge)
- Data is passed using global variables
  - About the only way to do it
- To detect rising edges, use
  `trigger=Pin.IRQ_RISING`
- To detect falling edges, use
  `trigger=Pin.IRQ_FALLING`
- To detect *both* edges

```
trigger = Pin.IRQ_RISING |
          Pin.IRQ_FALLING
```

```
# Up Counter using interrupts
from machine import Pin

flag=0
N = 0

pin = Pin(15,Pin.IN,Pin.PULL_UP)
def IntServe(pin):
    global flag
    global N
    flag = 1
    N = N + 1

pin.irq(trigger=Pin.IRQ_FALLING,
handler=IntServe)

while(1):
    if(flag):
        print("N = ", N)
        flag=0
```

Shell

```
N = 1
N = 2
N = 3
```

Also also...

Keep the interrupt routine simple

- Get in then get out
- Don't use loops inside interrupts: they take too long to execute.
- Never use a while-loop inside an interrupt: if you're stuck inside the interrupt the main routine doesn't execute.
- If-statements are OK - you only evaluate these once.
- Any variables you want to save for later or pass to the main routine need to be global variables.

```
# Up Counter using interrupts
from machine import Pin

flag=0
N = 0

pin = Pin(15,Pin.IN,Pin.PULL_UP)
def IntServe(pin):
    global flag
    global N
    flag = 1
    N = N + 1

pin.irq(trigger=Pin.IRQ_FALLING,
handler=IntServe)

while(1):
    if(flag):
        print("N = ", N)
        interrupt_flag=0
```

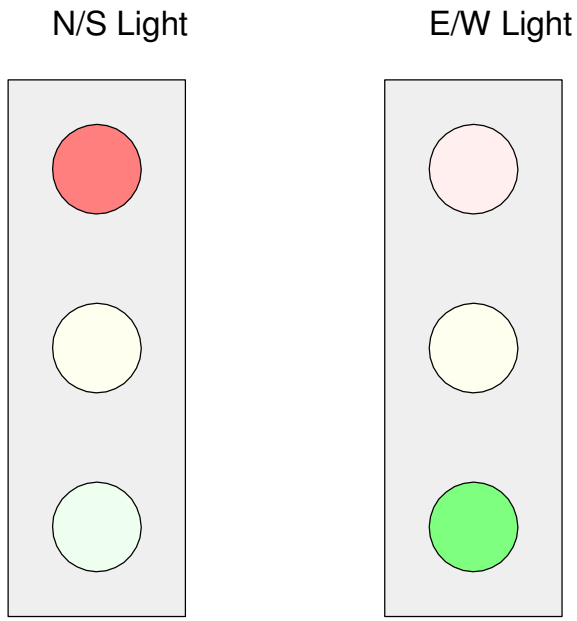
Shell

```
N = 1
N = 2
N = 3
```

Example: StopLight & Bottom-Up Programming:

Write a program for a stoplight

- N/S light: Green / Yellow / Red
- E/W light: Green / Yellow / Red



Use interrupts to change the colors

- Ring counter: 0 / 1 / 2 / 3 / 4 / 5
- N/S = G / Y / R / R / R / R
- E/W = R / R / R / G / Y / R

Use bottom-up programming

- Write the program step-by-step
- Check & validate code as you write it

Step 1: Display Routine

Subroutine which is passed:

- Light:
  - 0 = Red
  - 1 = Yellow
  - 2 = Green
- (x,y) location of the light
  - N/S and E/W light

Start with the display routine with bottom-up programming

Being able to see your results will help with debugging the rest of the program

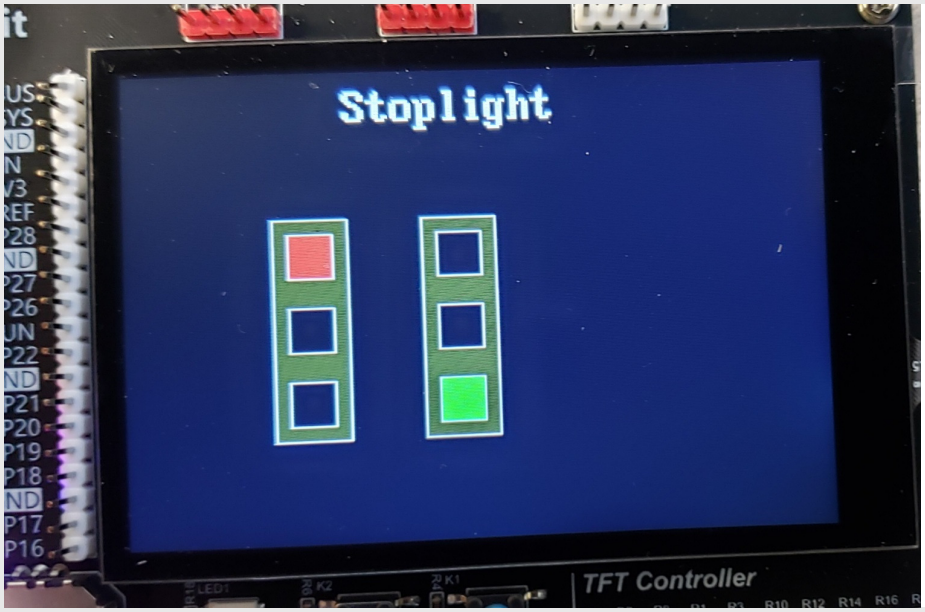
```
def Display(Light, x, y):
    Red = LCD.RGB(200,0,0)
    Yellow = LCD.RGB(200,200,0)
    Green = LCD.RGB(0,200,0)
    Brown = LCD.RGB(30,30,0)
    Black = 0
    White = LCD.RGB(200,200,200)

    LCD.Solid_Box(x, y, x+50, y+150, Brown)
    LCD.Box(x, y, x+50, y+150, White)

    y += 10
    x += 10
    if(Light == 0):
        LCD.Solid_Box(x,y,x+30,y+30,Red)
    else:
        LCD.Solid_Box(x,y,x+30,y+30,Black)
    LCD.Box(x,y,x+30,y+30,White)

    y += 50
    if(Light == 1):
        LCD.Solid_Box(x,y,x+30,y+30,Yellow)
    else:
        LCD.Solid_Box(x,y,x+30,y+30,Black)
    LCD.Box(x,y,x+30,y+30,White)

    y += 50
    if(Light == 2):
        LCD.Solid_Box(x,y,x+30,y+30,Green)
    else:
        LCD.Solid_Box(x,y,x+30,y+30,Black)
    LCD.Box(x,y,x+30,y+30,White)
```

Test Code

Any time you write a routine, test it

- Write a short test program
- Check that the program is working correctly

```
# Test code for stoplight
import time
import LCD

def Display(Light, x, y)
    :

Navy = LCD.RGB(0,0,5)
White = LCD.RGB(200,200,200)
LCD.Init()
LCD.Clear(Navy)
LCD.Text2('Stoplight',200,20,White,Navy)
Display(0, 100, 100)
Display(2, 200, 100)
```



Step 2: Light Sequence

Set up a ring counter

- Count mod 6
- Count once per second

Go through the light sequence:

- 0: G - R
- 1: Y - R
- 2: R - R
- 3: R - G
- 4: R - Y
- 5: R - R

Verify the code works

```
# Main Routine: Cycle lights every second
Red = 0
Yellow = 1
Green = 2
while (1):
    N = (N + 1) % 6
    if(N == 0):
        Display(Green, 100, 100)
        Display(Red, 200, 100)
    if(N==1):
        Display(Yellow, 100, 100)
        Display(Red, 200, 100)
    if(N==2):
        Display(Red, 100, 100)
        Display(Red, 200, 100)
    if(N==3):
        Display(Red, 100, 100)
        Display(Green, 200, 100)
    if(N==4):
        Display(Red, 100, 100)
        Display(Yellow, 200, 100)
    if(N==5):
        Display(Red, 100, 100)
        Display(Red, 200, 100)
    print(N)
    time.sleep(1)
```

Note: While this routine works, it's really inefficient. >99% of the time is spent in the sleep() statement.

Step 3: Add interrupts

Instead of counting every second,

- Count each button press

Note:

- Counting is mod 6
  - same as before
- Global variables are used
  - only way to pass data
- A flag is used
  - tells the main routine it's time to update the LCD display

```
from machine import Pin
import time

N = 0
flag = 0

pin1 = Pin(15,Pin.IN,Pin.PULL_UP)

def CLK(pin1):
    global N, flag
    flag = 1
    N = (N+1)%6

pin.irq(trigger=Pin.IRQ_FALLING, handler=CLK)
```

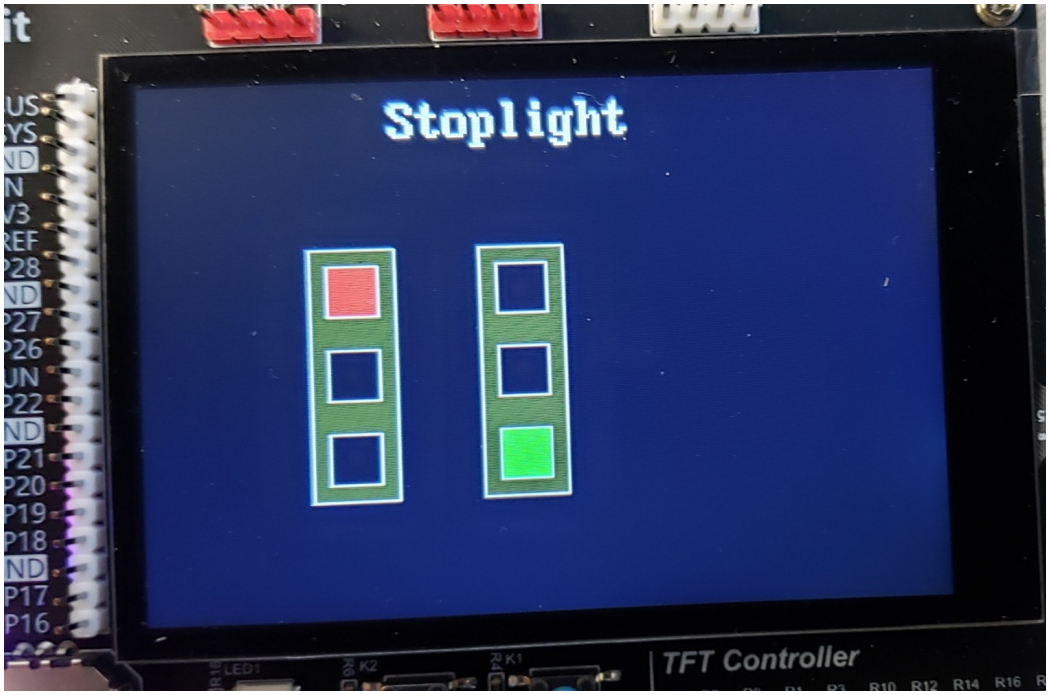
Main Routine with Interrupts:

- Main routine checks if the light changed
  - flag == 1
- The lights are updated based upon the ring-counter
  - N = 0: green / red
  - N = 1: yellow / red
  - etc.

```
# Main Routine
while (1):
    if(flag == 1):
        flag = 0
        if(N == 0):
            Display(Green, 100, 100)
            Display(Red, 200, 100)
        if(N==1):
            Display(Yellow, 100, 100)
            Display(Red, 200, 100)
        if(N==2):
            Display(Red, 100, 100)
            Display(Red, 200, 100)
        if(N==3):
            Display(Red, 100, 100)
            Display(Green, 200, 100)
        if(N==4):
            Display(Red, 100, 100)
            Display(Yellow, 200, 100)
        if(N==5):
            Display(Red, 100, 100)
            Display(Red, 200, 100)
        print(N)
```

Some problems with this routine is

- You have to manually press for each change in the light.
  - We'll fix this when we cover timer interrupts in the next lecture.
- Sometimes you get multiple counts due to bouncing in the button.
  - We'll fix this too



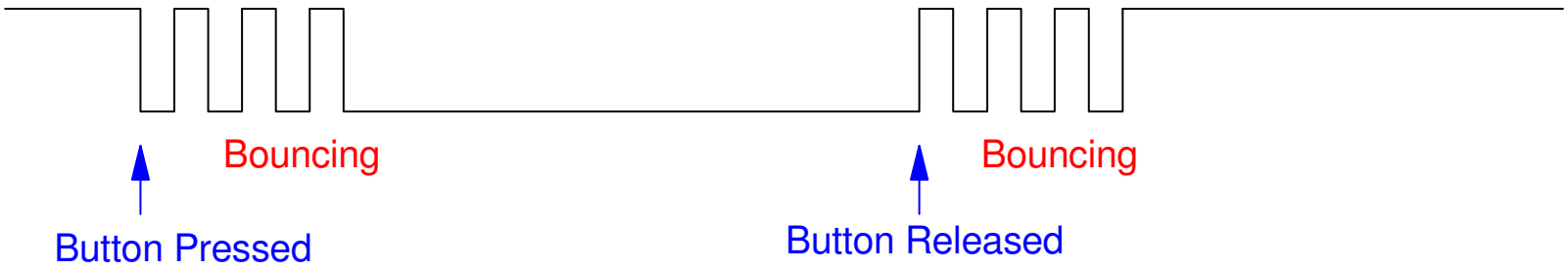
Debouncing:

Bouncing is a feature of mechanical switches

- When they open and close, they bounce
- This results in multiple edges each button press and each release

Debouncing eliminates this

- Results in a single edge when you press and release the button
- Both hardware and software can be used



Bouncing is when you get multiple edges each time you press and release a button

Hardware Debouncing:

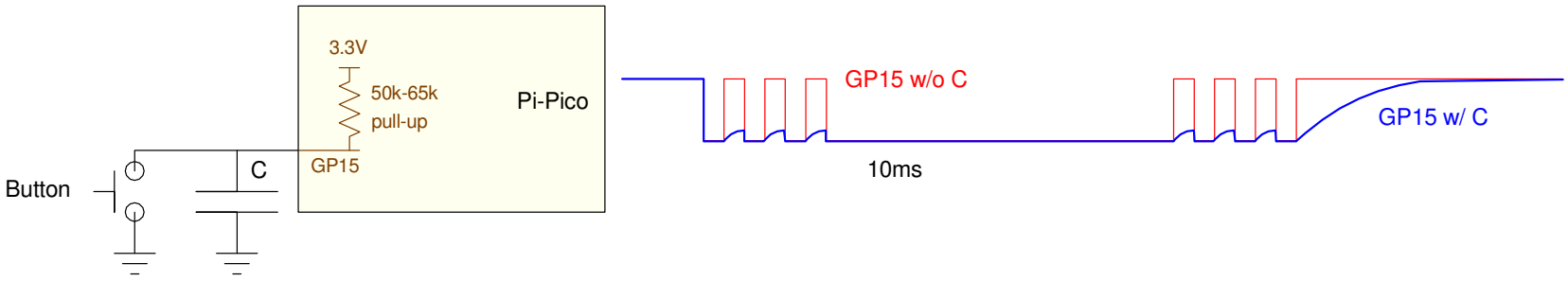
Add a capacitor

- Filters out the bouncing
- Picking C so that the RC time constant is equal to the bounce time
  - usually works
- Assuming this is 10ms:

$$RC = 10ms$$

R is the internal pull-up resistor (50k to 65k), resulting in

$$C = 153nF..200nF$$



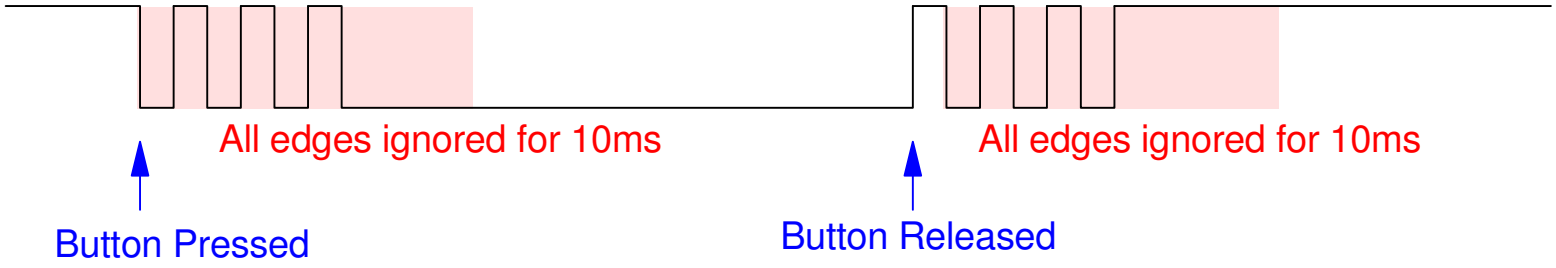
Adding a capacitor filters is one way to remove bouncing

Software Debouncing:

- Anything you can do in hardware you can do in software.

Add a dead-time

- Once you detect an edge, ignore all subsequent edges for X ms
- Check the resulting logic level to see it it was a rising or falling edge



Software Debouncing: All edges are ignored for 10ms once a falling edge is detected

Software Debouncing with a sleep() statement

When a falling edge is detected,

- Wait 10ms for the bouncing to stop
- Check if this was a falling edge
  - pin value is zero

```
def CLK(pin1):
    global N, flag
    time.sleep(0.01)
    if(pin1.value() == 0):
       flag = 1
       N = (N+1)%6
```

This is a little inefficient:

- A 10ms wait is a *lot* of clocks
- A timer interrupt would be more efficient
  - next lecture

Multiple Interrupts: Hungry-Hungry Hippo (Take 2):

https://youtu.be/9Owv0h8wz-I?feature=shared

Next, let's look at multiple interrupts.

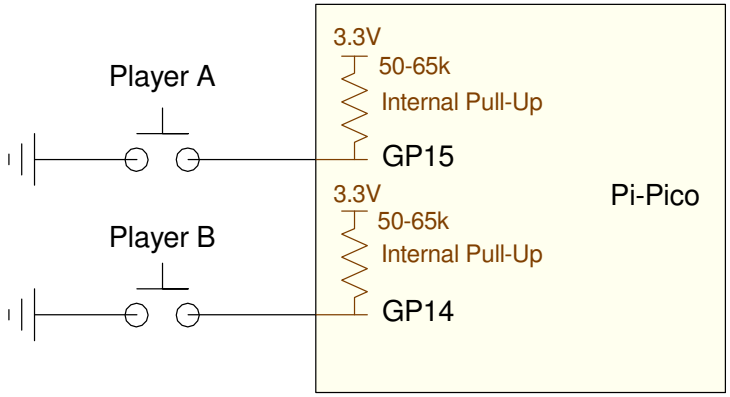
Specifically,

- Play *Hungry-Hungry Hippo*
- With two players
  - More could be added
- With a 10 second time limit
  - After 10 seconds the game is over
  - Button presses after 10 seconds are ignored
- Whoever has the high-score after 10 seconds wins

Hardware:

Input Buttons:

- Player A: Button GP15
- Player B: Button GP14
- More could be added



Use an internal pull-up resistor

Falling edges = Button presses

- Count falling edges

Software

- Add a counter for each player
  - Global variables N1 and N2
- Define the input pins
  - GP15 and GP14
- Define the interrupt service routines
  - player1
  - player2
- Set up the interrupts
  - falling edge interupts

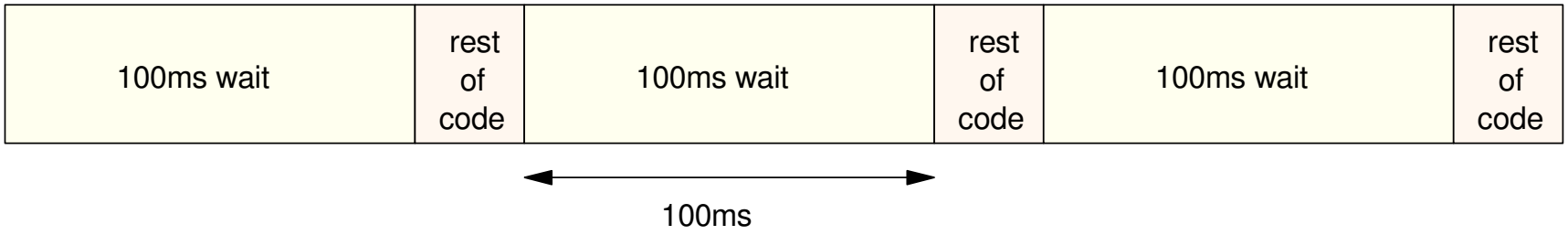
The main routine just displays

- Display the score
- Wait 100ms
- Repeat for 10 seconds

```
# Hungry-Hungry Hippo (Take 2)

from machine import Pin
from time import sleep

N1 = N2 = 0

pin1 = Pin(15, Pin.IN, Pin.PULL_UP)
pin2 = Pin(14, Pin.IN, Pin.PULL_UP)

def player1(pin1):
    global N1
    N1 = N1 + 1

def player2(pin2):
    global N2
    N2 = N2 + 1

pin1.irq(trigger=Pin.IRQ_FALLING,
handler=player1)
pin2.irq(trigger=Pin.IRQ_FALLING,
handler=player2)

Time = 0.0

while (Time < 10):
    print(Time, N1, N2)
    Time += 0.1
    sleep(0.1)
```

Note that the timing in this program is a little off

- Each game is slightly longer than 10.00 seconds.
- Due to the loop time is more than 100ms
  - sleep(0.1) = 100ms
  - rest of code also takes time



This problem can be overcome by using *timer* interrupts

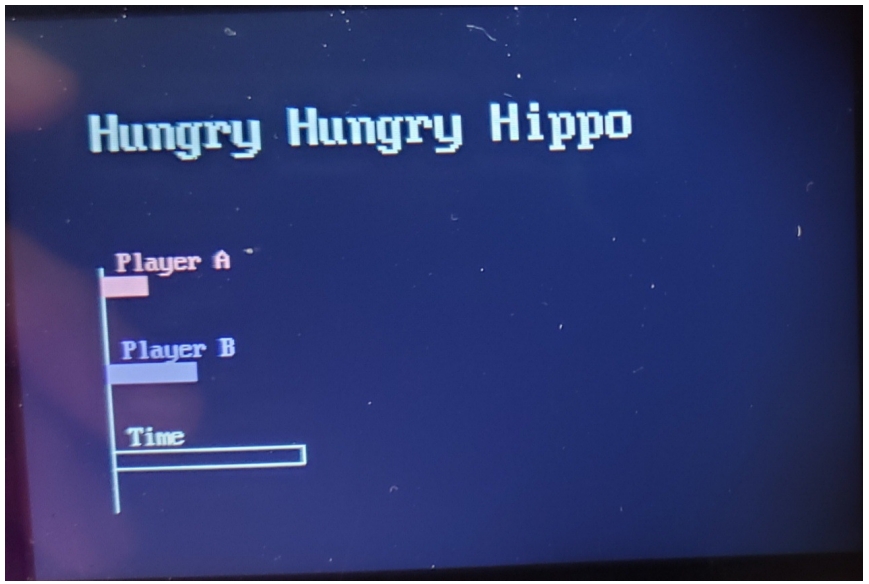
- next lecture

Once you are able to record the number of button presses for two (or more) players, you can vary the game with slight modifications along with the LCD display:

Hungry-Hungry-Hippo Variations:

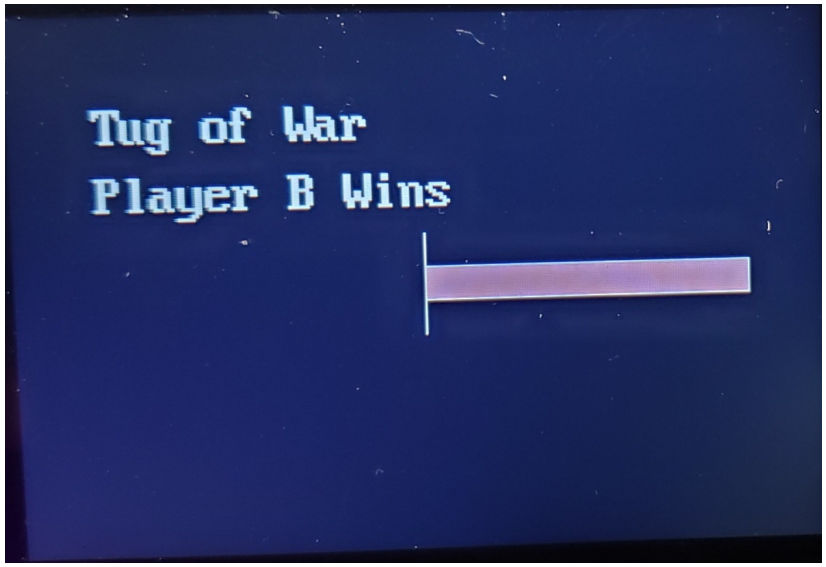
Keep running score on the LCD:

- Update the LCD display every 100ms
- Show the score for player A an dB
- Show the time remaining in the game



Tug of War:

- Display the difference in score
- When the difference exceeds a threshold, the game is over



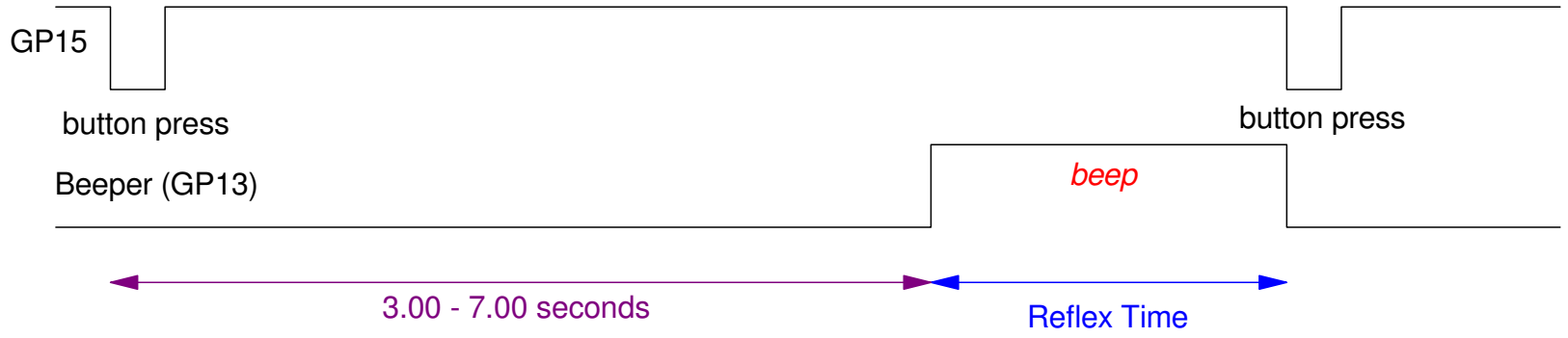
Reflex Game:

With edge interrupts you can record the time of an event

- within a few clocks - the time it takes to trigger the interrupt

With this, you can measure your reflex time.

- The game starts by pressing a button connected to pin GP15.
- Once pressed, the Pico waits between three and seven seconds.
- Once done waiting, the beeper turns on and that time is saved
- The program then waits for you to press GP15 again
- Once pressed, the time between the beeper going on and pressing GP15 is recorded and displayed



Reflex Game: Press GP15 to start the game. Press GP15 as soon as you hear the beeper

Code:

By using interrupts

- It's harder to cheat
  - can't just hold down the button
- Timing is more accurate
  - time is measured to 1us
  - time.ticks_us()

Global variables are used

- tells the main routine the button was pressed
- tells the main routine when the button was pressed

Results:

```
Reflex Time(us) =  274368
Reflex Time(us) =  162795
Reflex Time(us) =  141069
```

```
from machine import Pin
from time import sleep, ticks_us
from random import random

beeper = Pin(13, Pin.OUT)
pin1 = Pin(15,Pin.IN,Pin.PULL_UP)

T0 = T1 = flag = 0

def B15(pin1):
    global T1, flag
    T1 = ticks_us()
    flag = 1

pin1.irq(trigger=Pin.IRQ_FALLING, handler=B15)

while(1):
    beeper.value(0)
    flag = 0
    while(flag == 0):
        pass
    print('Starting:  3..7 seconds later')
    dT = random() * 4 + 3
    time.sleep(dT)
    flag = 0
    beeper.value(1)
    T0 = ticks_us()
    while(flag == 0):
        pass
    beeper.value(0)
    Reflex = T1 - T0
    print('Reflex Time(us) = ',T1-T0)
```

Optical Encoders & Pong Game

Finally, one of the more useful things you can do with edge interrupts is read an optical encoder - also known as a digital potentiometer.

Analog Potentiometers:

- A long strand of resistance wire
- With a center tap

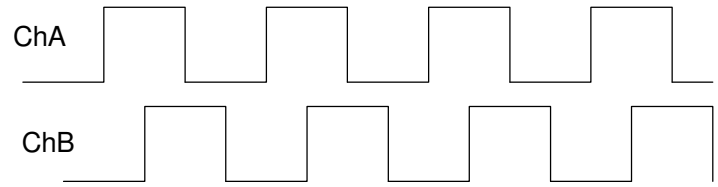
Angle can be read

- Connect ends to 0V and 3.3V
- Wiper voltage is proportional to angle
  - 0V to 3.3V analog

Digital Potentiometers:

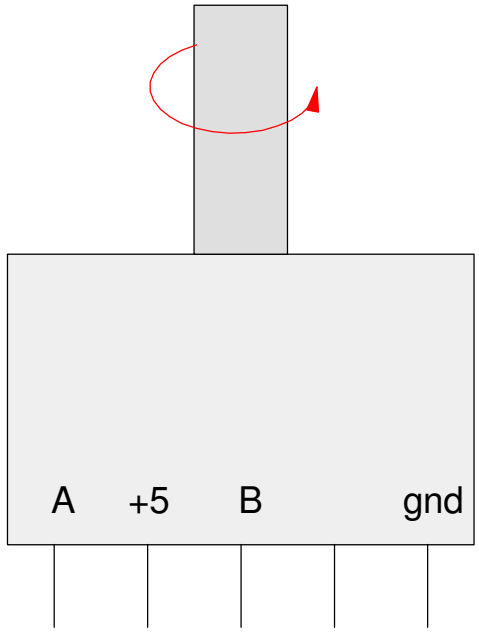
Output is a square wave

- Two channels (A and B)
- 90 degrees apart
  - phase quadrature



What this does...

- You can measure speed
  - count pulses per second
- You can measure angle
  - count pulses
  - usually using interrupts



Counting Edges on Channel A

- 200 pulses per rotation
- 400 edges per rotation

Note:

- The interrupt counts both rising and falling edges

```
from machine import Pin
import time

N = 0

pin1 = Pin(15,Pin.IN,Pin.PULL_UP)
pin2 = Pin(14,Pin.IN,Pin.PULL_UP)

def ChA(pin1):
    global N
    if(pin1.value() == pin2.value()):
        N -= 1
    else:
        N += 1

pin1.irq(trigger=Pin.IRQ_FALLING |
Pin.IRQ_RISING, handler=ChA)

while (1):
    print(N)
    time.sleep_ms(100)
```

Counting edges on A and B

- Count
  - Rising and falling edges on A
  - Rising and fallign edges on B
- Result is 800 counts per rotation

Note:

- Counting is done in the background
  - using interrupts
- The main loop just sees N 'magically' changing

With this, you can use an optical encoder as the input to a game

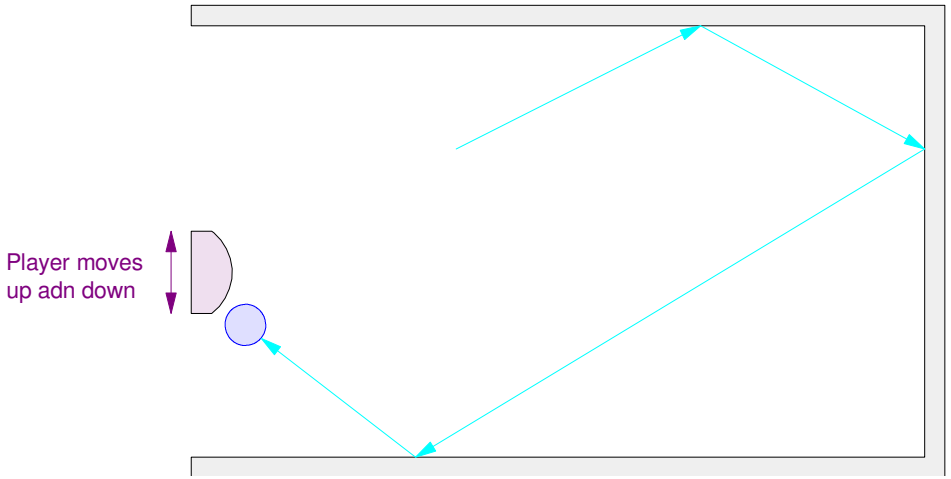
```
from machine import Pin
import time

N = 0

pin1 = Pin(15,Pin.IN,Pin.PULL_UP)
pin2 = Pin(14,Pin.IN,Pin.PULL_UP)

def ChA(pin1):
    global N
    if(pin1.value() == pin2.value()):
        N -= 1
    else:
        N += 1

def ChB(pin2):
    global N
    if(pin1.value() == pin2.value()):
        N += 1
    else:
        N -= 1

pin1.irq(trigger=Pin.IRQ_FALLING |
Pin.IRQ_RISING, handler=ChA)
pin2.irq(trigger=Pin.IRQ_FALLING |
Pin.IRQ_RISING, handler=ChB)

while (1):
    print(N)
    time.sleep_ms(100)
```

Python routine for counting edges on both channel A and B

Pong Game

Example of using an optical encoder

- Encoder moves a paddle up and down
- A ball is bouncing around the display
- If the ball hits an edge, it reflects
- If the ball hits the paddle, it bounces right and stays in play
- The game is over if the ball hits the left edge



Sort of like playing one-person ping-pong.

Program Description:

Optical Encoder Input:

- Same code as before

Paddle Display

- Draw a 320x2 rectangle on the left side of the display
- Paddle is white
  - other pixels are black
  - location y-20 to y+20
- Don't allow the paddle to go off screen
  - limit y to the range of (11, 309)

```
def Paddle(x, y):
    x = int(x)
    y = int(y)
    if(y < 11):
        y = 11
    if(y > 309):
        y = 309
    LCD.Address_Set(x, 0, x+1, 319)
    X = bytearray([])
    for i in range(0,638):
        X.append(0)
    for i in range(2*y-20,2*y+20):
        X[i] = 0xFF;
    LCD.StartWrite()
    LCD.Write16xN(X)
    LCD.EndWrite()
```

Bouncing Ball

Update the (x,y) position of the ball

- integrate velocity
- uses Euler integration

If you hit an edge, reflect

- Px > 200 (right edge)
- Py < 0 (top edge)
- Py > 319 (bottom edge)
- Px < 3 (paddle)
- Px < 0
  - game over
  - you missed the paddle

Score goes up every time you hit the paddle

```
while(1):
    a1 = a2d1.read_u16()
    y = 320-k*a1
    Paddle(3,y)
    LCD.Pixel2(Px, Py, Navy)
    Px += Vx*dt
    Py += Vy*dt
    if(Px > 200):
        Vx = -abs(Vx)
        Beep()
    if(Py < 0):
        Vy = abs(Vy)
        Beep()
    if(Py > 319):
        Vy = -abs(Vy)
        Beep()
    if(Px < 3):
        if(abs(y - Py) < 10):
            Vx = abs(Vx)
            Vy += y - Py
            Score += 1
            LCD.Number2(Score, 4, 0,
300, 100, White, Navy)
        Beep()
```

Summary

Edge interrupts are kind of confusing but useful

With edge interrupts, you can

- Call a subroutine only when needed
  - when an edge was detected
- Keep track of the number of edges more accurately
- Count edges much faster (and more efficiently) than with polling

Using edge interrupts also simplifies code

- You don't have to check for the current and previous state of an input

But...

- You almost *have* to use global variables to pass data to and from the interrupt
- Hardware calls the interrupt, not you