



INSTITUTO DE FÍSICA

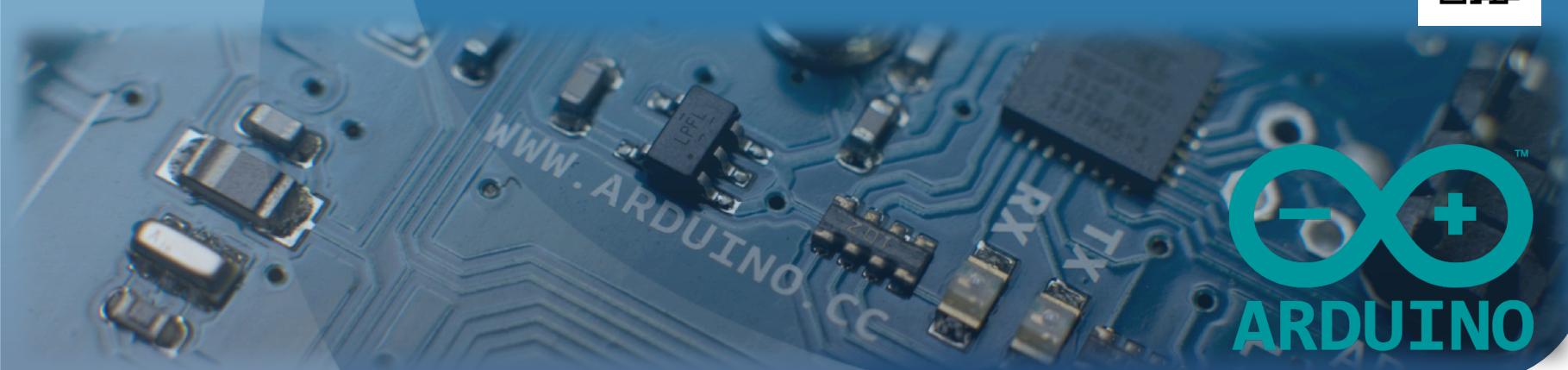
1º minicurso de Arduino no IFUSP

4 a 22 de maio de 2015
Prof. Alexandre Suaide

Arduino é uma plataforma eletrônica para prototipagem flexível,
de baixo custo, fácil de usar e aberta.

Inscrições abertas até 17 de abril
Mais informações sobre o minicurso e inscrições em

<http://e.usp.br/2ia>



Conteúdo de hoje

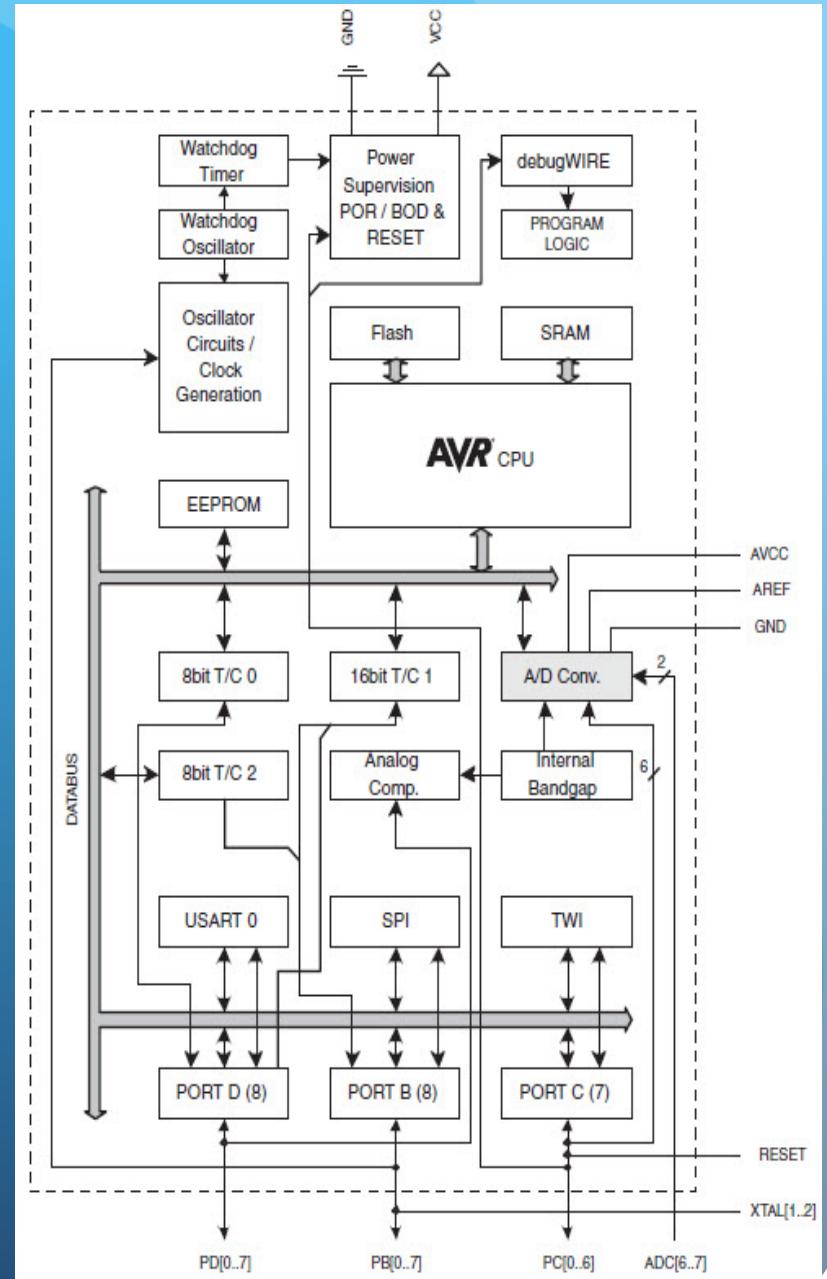
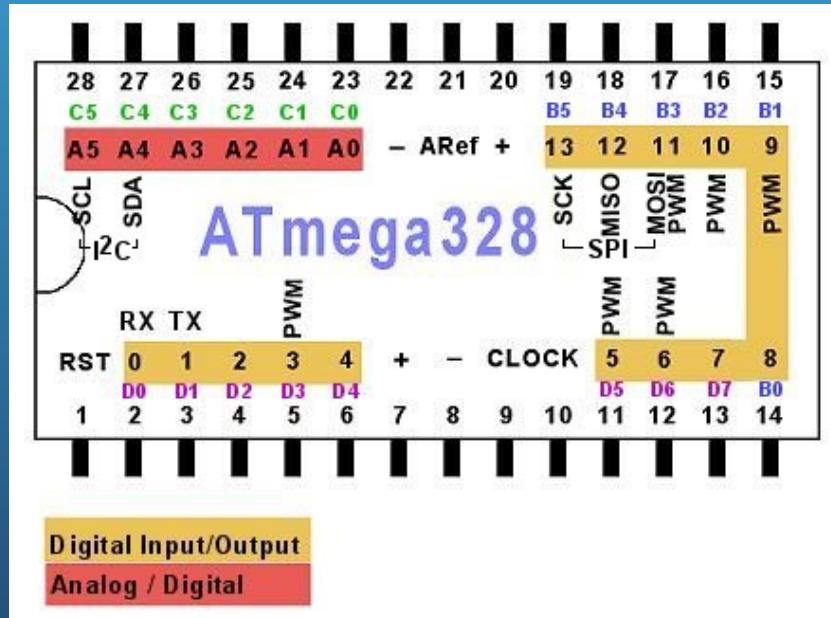
- Situações críticas
 - Acessando portas diretamente
 - Interrupções
 - Eventos externos
 - Eventos temporizados
- Transmissão de dados entre Arduinos
 - UART
 - I2C
 - SPI

Algumas situações são críticas

- Alguns eventos necessitam ação simultânea
 - Fechar válvulas em um sistema de gás
- Alguns eventos exigem resposta imediata
 - Medida precisa da ocorrência do evento
 - Questões de emergência
 - Desligar um circuito
 - Acionar um paraquedas em um drone
 - Desligar um reator nuclear
 - Dependendo da situação não podemos esperar o fluxo normal do programa
- Vamos olhar algumas situações onde tempo importa

ATMEGA328

- Os pinos de I/O e ADCs são mapeados em portas endereçáveis que podem ser acessadas pelo usuário.



Três portas

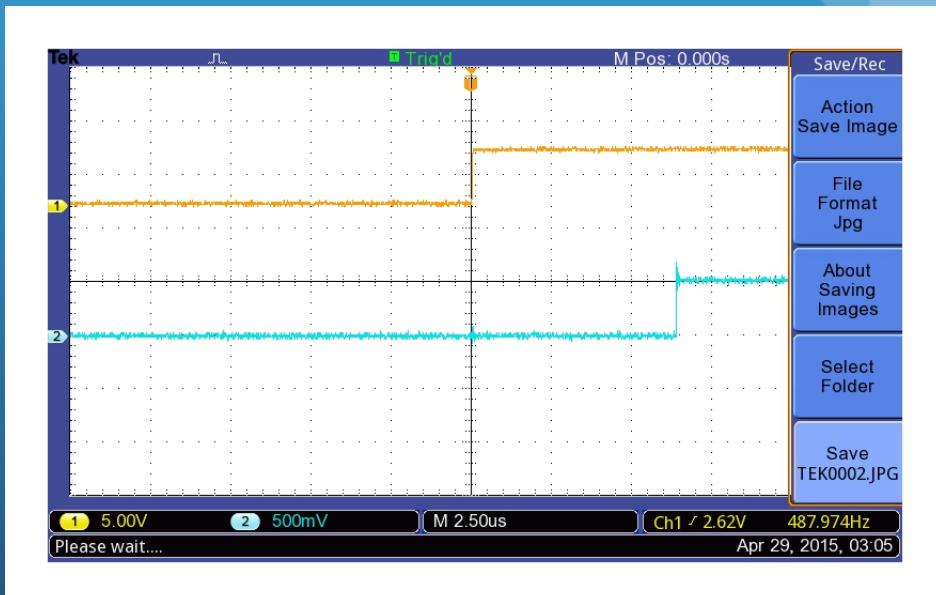
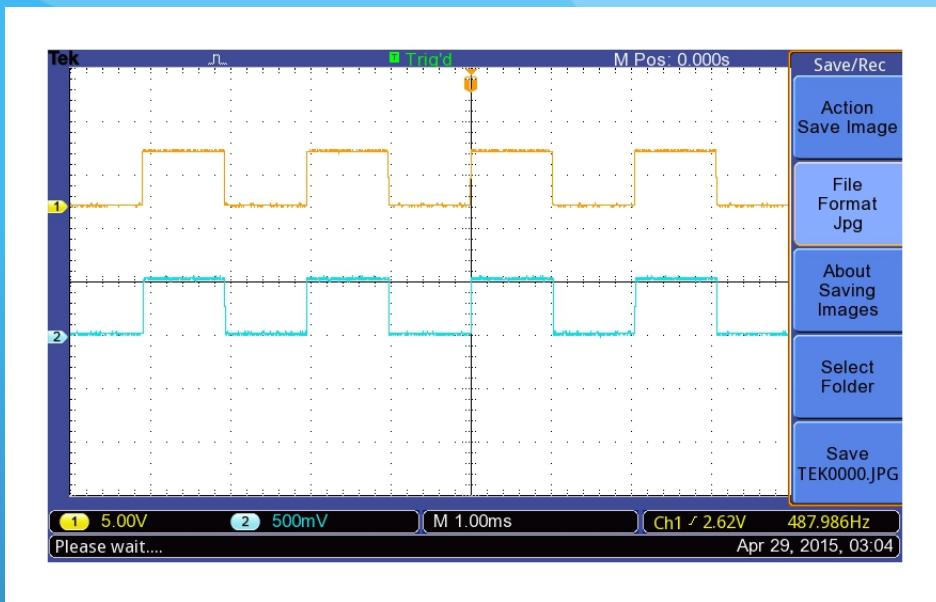
- Porta B - pinos digitais 8 a 13
- Porta C - pinos analógicos
- Porta D - pinos digitais 0 a 7
- Três registradores para cada porta
 - DDRX - Direção dos dados - pode-se ler ou escrever neste registrador
 - PORTX - Valor da porta - pode-se ler ou escrever neste registrador
 - PINX - Registrador dos pinos de input - somente leitura
 - X = B, C ou D
 - Alguns bits dos registradores não são utilizados.

Como manipular portas diretamente (ex: porta D)

- `DDRD = B1111110;`
 - Faz os pinos 1 a 7 como saída e o pino 0 como entrada
- `PORTD = B10101000;`
 - Coloca os pinos 7, 5 e 3 em HIGH e os demais em LOW
- `PIND` contém o resultado da leitura dos pinos.
- Porque fazer desta forma e não usando `pinMode()`, `digitalRead()` ou `digitalWrite()`?

Método 1 tradicional

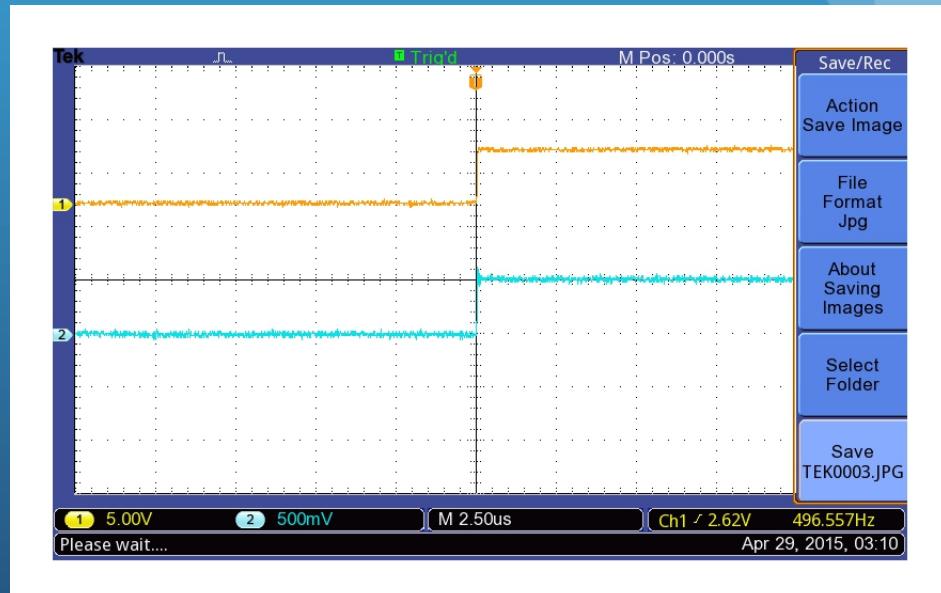
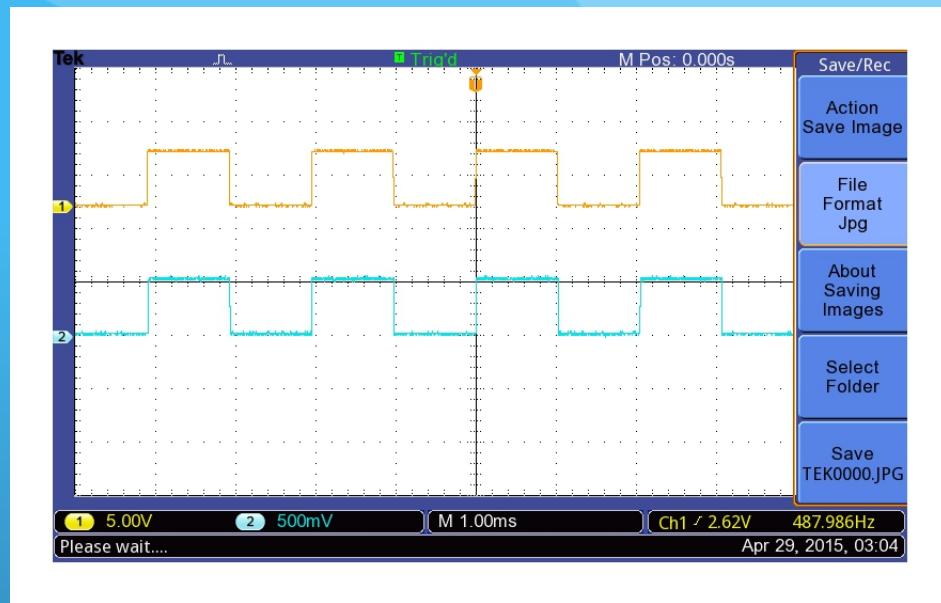
```
void setup()
{
    pinMode(7,OUTPUT);
    pinMode(6,OUTPUT);
}
void loop()
{
    digitalWrite(7,HIGH);
    digitalWrite(6,HIGH);
    delay(1);
    digitalWrite(7,LOW);
    digitalWrite(6,LOW);
    delay(1);
}
```



Método 2 portas

```
void setup()
{
    DDRD = B11000000;
}
```

```
void loop()
{
    PORTD = B11000000;
    delay(1);
    PORTD = 0;
    delay(1);
}
```



Prós e Contras

- Açãoamento da porta é bem mais rápido
 - O digitalWrite() tem cerca de 10-12 linhas de código
- Açãoamento simultâneo de todos os pinos
- Leitura simultânea de todos os pinos
- Programa menor → menos consumo de memória
- Programa mais difícil de ler
- Menos portátil
- Menos seguro, já que estamos manipulando diretamente os valores das portas
- Manipulação de dados requer, em geral, operações binárias

Interrupções

- Função loop() é executada continuamente
- Pode ser muito longa
- O que fazer quando precisamos responder imediatamente a um evento?
- Como que prioridade?
- Microcontroladores possuem mecanismos de interrupção e desvio do fluxo normal de execução de um programa
 - Eventos externos
 - Eventos internos
 - Eventos temporais (periódicos)
 - Vigia do sistema (watchdog)

Interrupções

- Interrupções devem ocorrer com prioridades bem definidas
 - Como proceder se duas interrupções são solicitadas no mesmo instante de tempo?
 - Característico de cada microcontrolador
- Para o ATMEGA328 do Arduino, ver figura ao lado.

Table 12-4. Reset and Interrupt Vectors in ATmega168P

VectorNo.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Interrupções externas

- INT0
 - Ligada ao pino 2
- INT1
 - Ligada ao pino 3
- Podem ser trigeradas nos seguintes eventos
 - RISING: LOW → HIGH
 - FALLING: HIGH → LOW
 - CHANGE: mudar o estado
 - LOW
- `attachInterrupt(int, ISR, mode)`
 - Int = 0 ou 1
 - ISR = nome da função a ser executada se a interrupção for triggerada
 - Mode = RAISING, FALLING, CHANGE ou LOW
- `detachInterrupt(int)`
 - Remove interrupção
- `interrupts()` ou `noInterrupts()`
 - Liga/desliga interrupções

Exemplo

```
int pin = 13;
volatile int state = LOW;

void setup()
{
    pinMode(pin, OUTPUT);
    attachInterrupt(0, blink, CHANGE);
}

void loop()
{
    digitalWrite(pin, state);
}

void blink()
{
    state = !state;
}
```

- Funções para processar interrupções não podem ter argumentos
- Devem ser curtas
- Outras interrupções não são processadas enquanto outra interrupção é processada
 - Outras funções, como delay(), millis(), etc. que dependem de interrupções, não funcionam.
- Variáveis que são alteradas durante interrupções devem ser declaradas como volatile

Desabilitando/habilitando interrupções

```
void setup()
{
}

void loop()
{
    // estou cuidando da minha vida
    noInterrupts();
    // tenho tarefas críticas
    // para processar e não
    // quero que nada as interrompam
    interrupts();
    // já terminei minhas
    // tarefas críticas
}
```

- Em algumas situações críticas não queremos ser interrompidos por interrupções
 - Note que a lista de interrupções possíveis é grande
- Eventos que geram interrupções ficam em fila esperando as interrupções serem habilitadas novamente
- Interrupções que ocorram no mesmo instante são processadas em uma sequência de prioridades (ver figura anterior)

Timers

- O ATMEGA328 possui 3 Timers
- Timer0
 - Contador de 8 bits
 - Utilizado nas funções `delay()` e `millis()`
- Timer1
 - Contador de 16 bits
- Timer2
 - Contador de 8 bits
 - Usado pela função `tone()`
- Funcionamento
 - A cada intervalo de tempo o contador é incrementado de uma unidade
 - No overflow do contador um flag é setado no microcontrolador
 - Posso testar o flag
 - Posso gerar uma interrupção
 - O intervalo de tempo mínimo depende do clock do processador
 - $T = 1/16 \text{ MHz} = 0.0625 \mu\text{s}$

Configurando o timer (registradores)

- TCCR = Timer Counter Control Register
 - TCCRxA e TTCCRxB, com x = 0, 1 ou 2, dependendo do timer utilizado
- TIMSKx = Timer Counter Interrupt Mask Register (x = 0, 1 ou 2)
- TIFRx = Timer Counter Interrupt Flag Register
- TCNTx = Timer Counter Register
 - Armazena o contador do timer utilizado
- OCRxA e OCRxB
 - Output Compare A e B
- No timer de 16 bits os registradores têm nomes ligeiramente diferentes.
- Cada bit nestes registradores estabelece uma propriedade do Timer utilizado (ver datasheet do ATMEGA328)

Ex: TCCRxA e TCCRxB

TCCR1A – Timer/Counter1 Control Register A

Bit (0x80)	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	TCCR1A
Initial Value	0	0	0	0	0	0	0	0	

TCCR1B – Timer/Counter1 Control Register B

Bit (0x81)	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	TCCR1B
Initial Value	0	0	0	0	0	0	0	0	

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$\text{clk}_{\text{IO}}/1$ (No prescaling)
0	1	0	$\text{clk}_{\text{IO}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{IO}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{IO}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{IO}}/1024$ (From prescaler) 64 $\mu\text{s}/\text{tick} \rightarrow \text{max } \sim 8388 \text{ ms}$
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Ex: TIMSKx

15.9.6 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Reserved**

These bits are reserved bits in the ATmega48A/PA/88A/PA/168A/PA/328/P and will always read as zero.

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter Interrupt Flag Register – TIFR0.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

Configurar o timer consome algum tempo

- Por conta disto há bibliotecas disponíveis para download
- TimerOne/TimerThree - bibliotecas para configurar e usar o Timer1 do Arduino
 - Faz toda a configuração dos registradores internos do Timer de 16 bits do Arduino
 - Permite realizar interrupções com base em temporizadores
 - Simples de usar
 - <http://playground.arduino.cc/code/timer1>
 - Baixe e copie em [My Documents]\Arduino\libraries
 - A diferença entre o TimerOne e TimerThree se fazem notar na placa Arduino Mega que possui mais funcionalidades de timers

Algumas funções do TimerOne

```
class TimerOne
{
public:

    // properties
    unsigned int pwmPeriod;
    unsigned char clockSelectBits;
    char oldSREG;// To hold Status Register while ints disabled

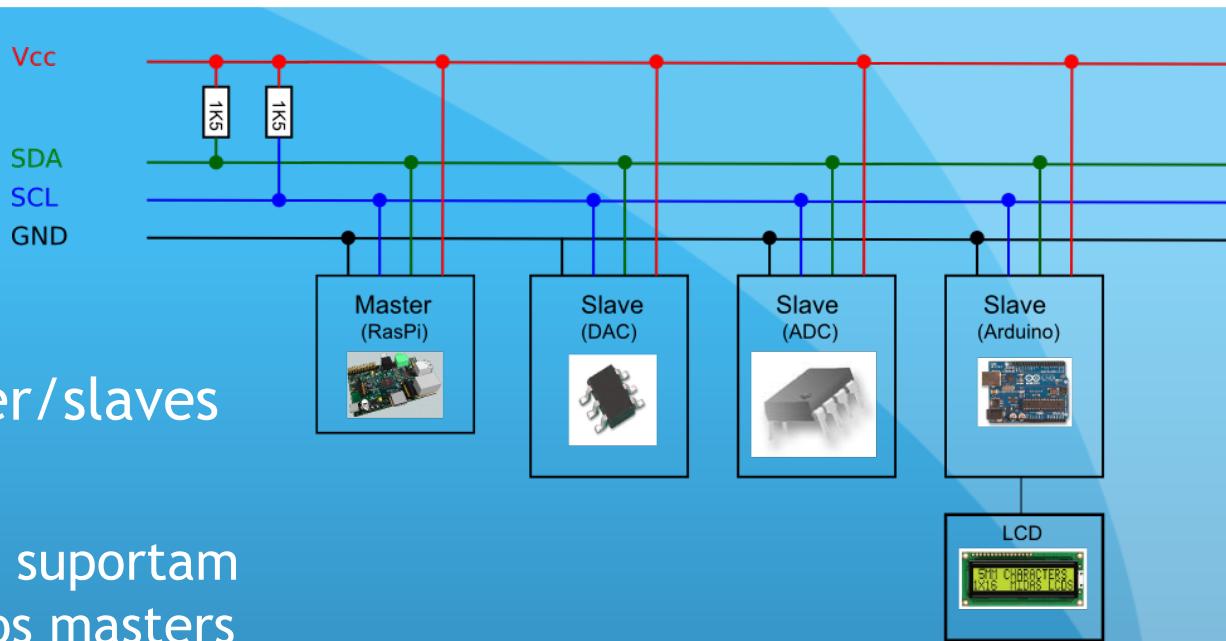
    // methods
    void initialize(long microseconds=1000000);
    void start();
    void stop();
    void restart();
    void resume();
    unsigned long read();
    void pwm(char pin, int duty, long microseconds=-1);
    void disablePwm(char pin);
    void attachInterrupt(void (*isr)(), long microseconds=-1);
    void detachInterrupt();
    void setPeriod(long microseconds);
    void setPwmDuty(char pin, int duty);
    void (*isrCallback)();
};
```

Um é pouco...

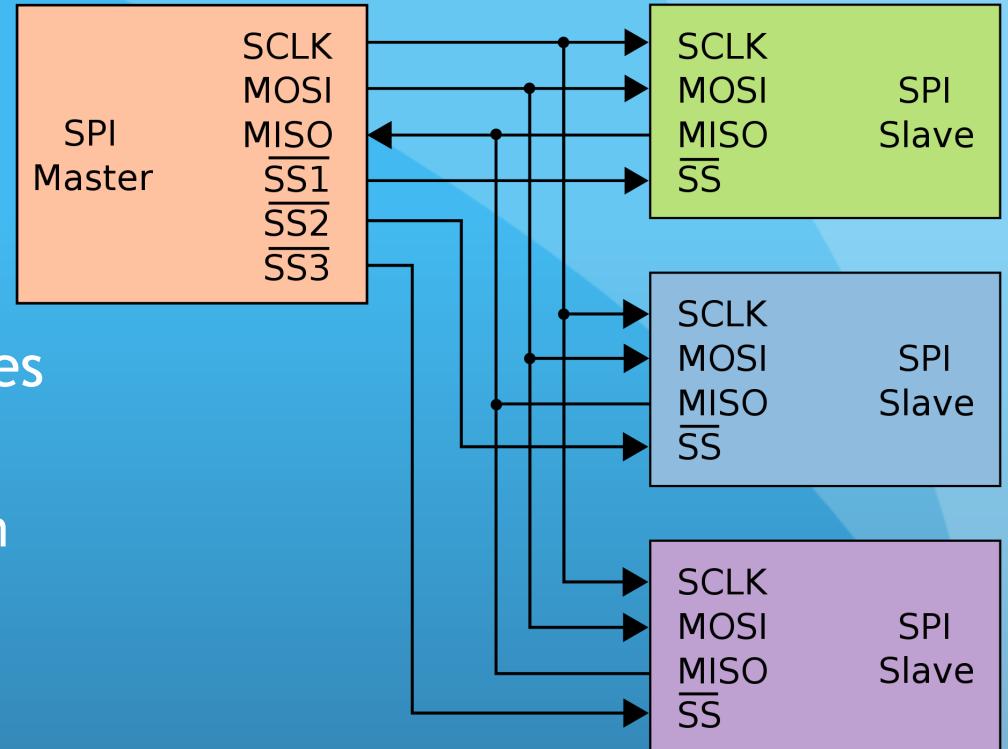
- Muitas vezes precisamos nos comunicar com outro microcontrolador, outro arduino, ou equipamentos complexos.
 - Como ligar dezenas de sensores em um único arduino?
 - Como trocar informações com outros arduinos?
- Métodos de comunicação entre equipamentos
 - UART - Universal Asynchronous Receiver/Transmitter (RS232, RS485, etc.)
 - I2C - Inter Integrated Circuit Communications (Philips)
 - SPI - Serial-Peripheral interface (Motorola)
- A maioria dos microcontroladores atuais suportam estes protocolos de comunicação.

I2C

- Topologia Master/slaves
 - Até 127 slaves
 - Alguns devices suportam rede com vários masters (Arduino é um caso)
- Bus com 4 fios
 - Note os resistores de pullup nas linhas de dados
 - Cada device recebe um ID na rede
 - A comunicação se dá nas linhas SDA e SCL.
 - SDA = dados, SCL = clock (sincronia)
 - Primeiramente o master envia o ID do slave destino
 - 7 bits com ID + 1 bit com informação sobre envio ou recebimento de informação
 - Velocidade típica de ~ 50 kBytes/s



SPI



- Topologia Master/slaves
 - Master único
 - Norma não muito bem definida
 - Muito simples de usar
- Bus com 4 fios
 - SCLK = clock (sincronia)
 - MOSI e MISO - transferência de dados
 - Pino de seleção de device
 - Um pino para cada device no master
 - Alto consumo de pinos de I/O
 - Altas velocidades de transferência ~ 10 MHz

I2C e SPI no Arduino

- Muitos shields se comunicam em um destes protocolos
 - Mas podemos utilizar para comunicação entre Arduinos ou com outros devices (Raspberry Pi)
- Bibliotecas disponíveis para download e fácil uso
 - Wire - Implementação do I2C
 - <http://www.arduino.cc/en/reference/wire>
 - SPI - biblioteca já nativa na IDE do Arduino

Oficinas

- Interrupções para fazer medidas em física
 - O experimento do pêndulo
- Usando interrupções em um experimento de física
- Conectando dois Arduinos usando protocolo I2C



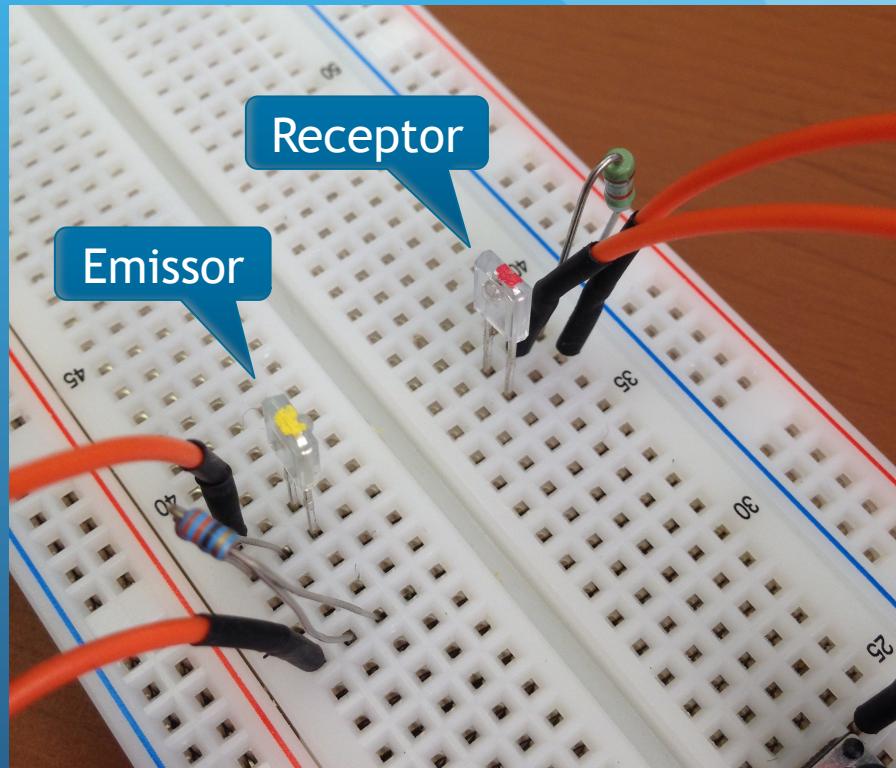
O pêndulo simples

- Medir o período de oscilação de um pêndulo
- Medir a velocidade máxima do pêndulo
- Como fazer isto?
 - Sensor de presença com infravermelho
 - Interrupções programáveis no Arduino



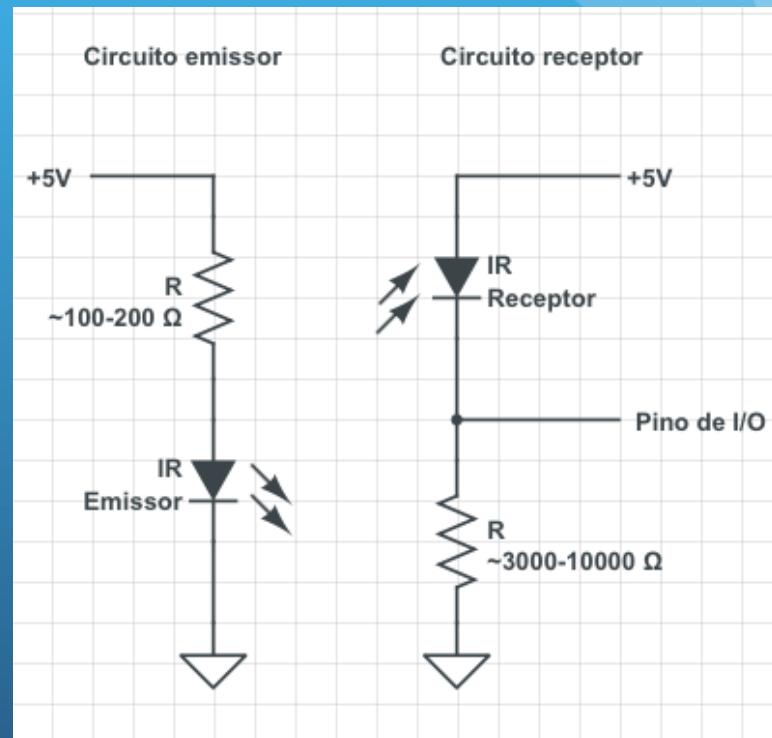
Emissor/receptor de IR

- Medir o período de oscilação de um pêndulo
- Medir a velocidade máxima do pêndulo
- Como fazer isto?
 - Sensor de presença com infravermelho
 - Interrupções programáveis no Arduino



Os circuitos.

- Emissão contínua de IR
- O receptor funciona como um botão.
 - Enquanto estiver recebendo IR ele está passando corrente
 - Pode-se fazer pullup ou pulldown com este receptor



O programa

- Oficina03-IR-01-teste.ino
- O pino 13 do Arduino está conectado a um LED na placa, com o devido resistor de proteção
 - Útil para testar sinais e comportamento de alguns sensores
- O programa mantém o LED aceso enquanto tiver IR sendo detectado pelo receptor
 - Coloque uma coisa entre o emissor e receptor e veja o LED apagar.
- Note que é bem sensível ao posicionamento dos sensores
- Tempo estimado ~ 15 minutos

```
#define IR 3
#define LED 13

void setup() {
    // put your setup code here, to run once:
    pinMode(IR,INPUT);

}

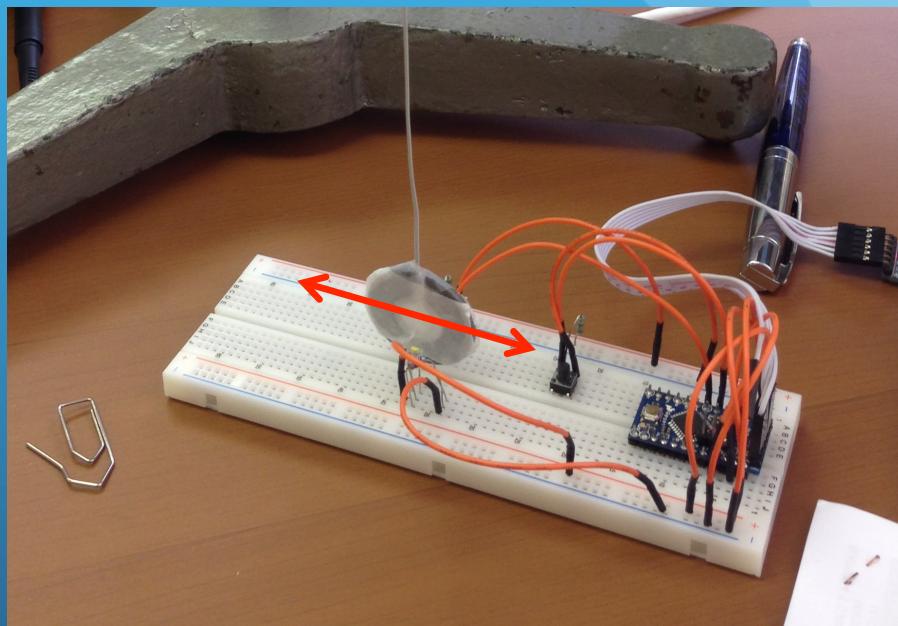
void loop() {

    if(digitalRead(IR)) digitalWrite(LED,HIGH);
    else digitalWrite(LED,LOW);

}
```

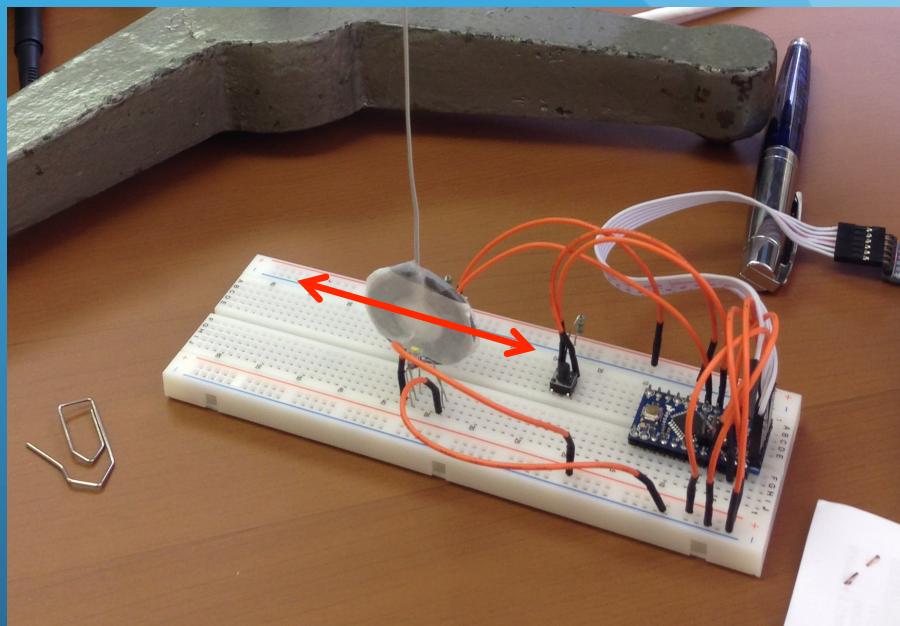
Aperfeiçoando e fazendo o experimento

- O pêndulo oscila entre o emissor e receptor
 - Usar o receptor para determinar o instante de tempo que o pêndulo passou
- Usar um botão para iniciar a tomada de dados
 - Mas devemos iniciar a contagem do tempo só quando o pêndulo passar pela primeira vez
- Uso de interrupções



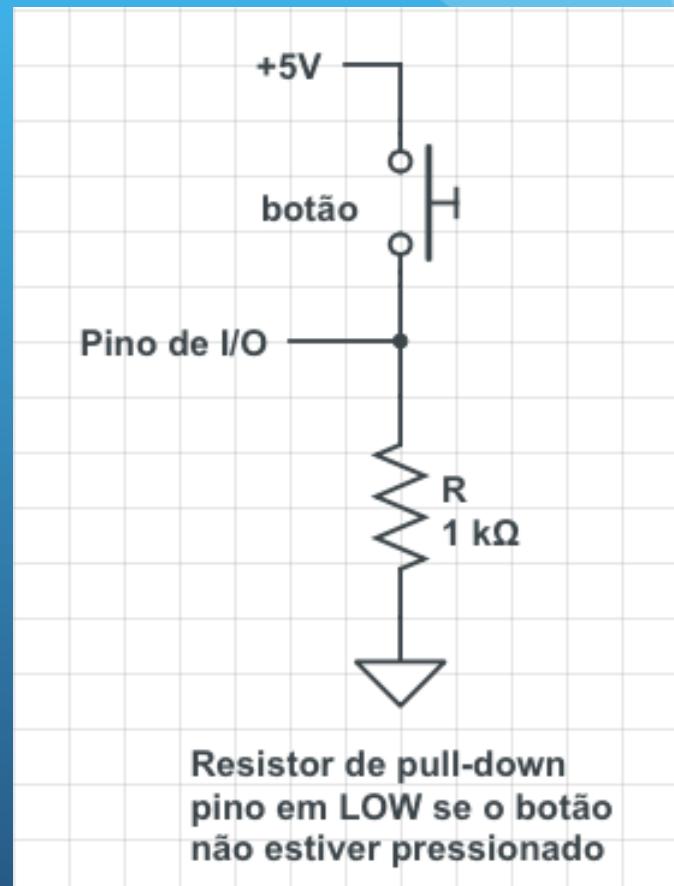
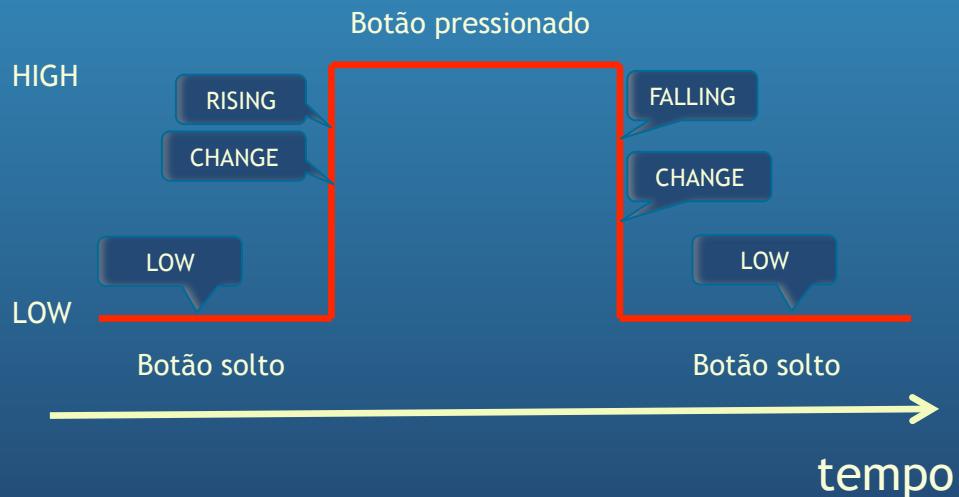
Aperfeiçoando e fazendo o experimento

- Duas interrupções disponíveis
 - Pino 2 - INT 0
 - Pino 3 - INT 1
- Ligar o receptor de IR no pino 3 (INT 1) e o botão no pino 2 (INT 0)
 - Cuidado com o modo da interrupção
- O fluxo do programa é interrompido quando a interrupção é acionada



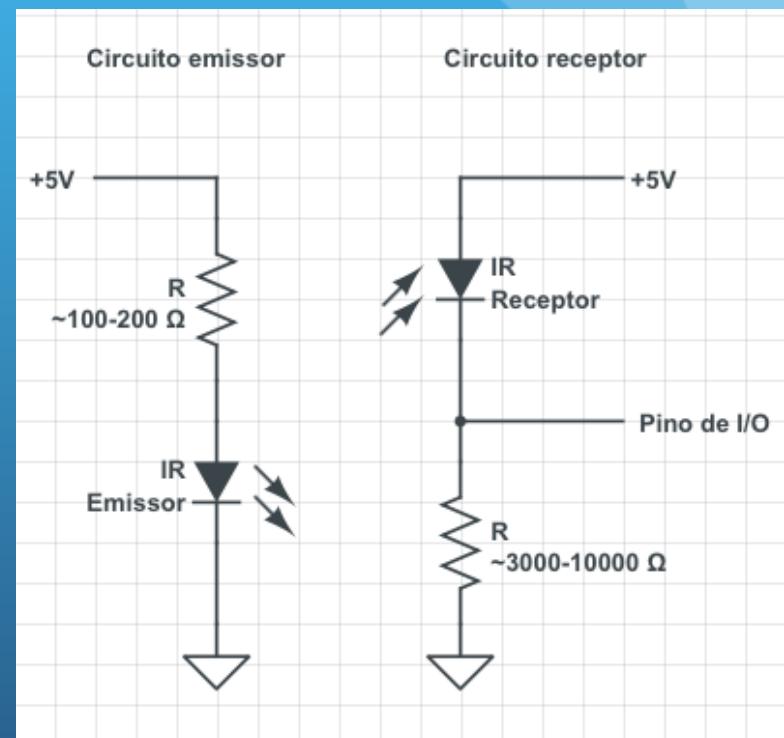
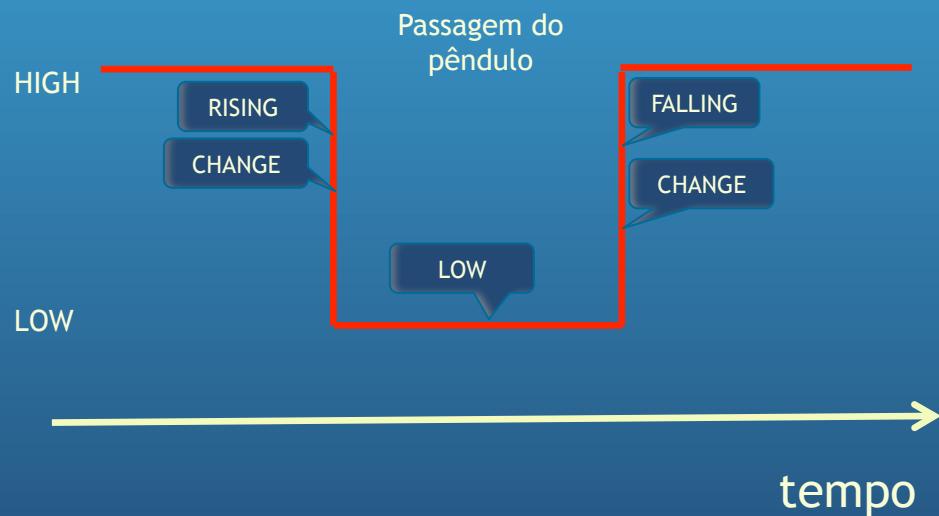
Lembrando o circuito do botão

- O botão será montado com resistor de pulldown
 - Ou seja, botão aberto medimos terra e botão fechado medimos 5V
- Temos 4 modos de interrupção:**RISING, FALLING, CHANGE e LOW.**
 - Como isto aparece no ato de pressionar o botão?



E o sinal no sensor?

- A medida no sensor de IR depende da passagem do pêndulo



O programa

- Quero gerar interrupção no momento que o processo inicia
 - Pressionar o botão
 - Início da passagem do pêndulo
- Ofinica03-IR-02-experimento.ino
- Baixe e instale o programa
- Abra a conexão com a porta serial
- Faça o experimento.
 - Posicione o pêndulo
 - Pressione e solte o botão para zerar
 - Solte o pêndulo e veja os dados fluindo para a tela do computador
- Tempo estimado ~15 minutos

```
#define IR 3
#define LED 13
#define BOTAO 2

bool parado = true;
int n = 0;
float tempo = 0;

void setup() {
    // put your setup code here, to run once:
    pinMode(IR, INPUT);
    pinMode(BOTAO, INPUT);
    attachInterrupt(0, paraTudo, RISING);
    attachInterrupt(1, passouNaFrente, FALLING);
    Serial.begin(115200);
}

void passouNaFrente()
{
    if(parado)
    {
        parado = false;
        tempo = millis();
        n = 0;
    }
    float t = millis();
    Serial.print(n);
    Serial.print(" ");
    Serial.println((t-tempo)/1000,4);
    n++;
}
void paraTudo()
{
    parado = true;
    Serial.println("");
    Serial.println("Estou parado novamente.");
    Serial.println("Esperando alguma coisa passar na minha frente");
}
void loop()
{
    if(digitalRead(IR)) digitalWrite(LED,HIGH);
    else digitalWrite(LED,LOW);
}
```

Medindo a velocidade do pêndulo

- $V = \Delta s / \Delta t$
- Como medir estas grandezas
 - Δs = tamanho do pêndulo
 - Δt = tempo que o pêndulo leva para passar pelo sensor de IR
- Como medir o tempo?
 - Tenho que gerar duas interrupções no mesmo pino e trata-las adequadamente
 - Usar CHANGE
 - Vai interromper no início e no final da passagem
- Oficina03-IR-02-experimento-velocidade.ino
 - Ao lado somente a parte que foi modificada no exemplo anterior
- Tempo estimado ~ 10 minutos

```
#define IR 3
#define LED 13
#define BOTAO 2

bool parado = true;
int n = 0;
float tempo = 0;
bool inicio = true;
float t1, t2;
float dx = 3;

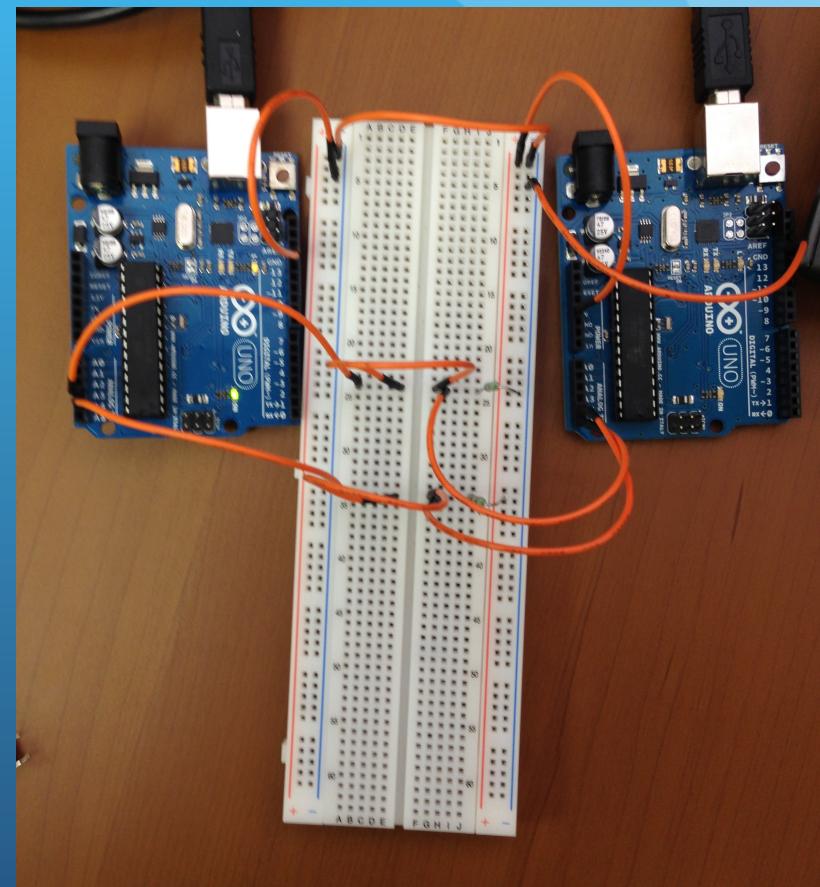
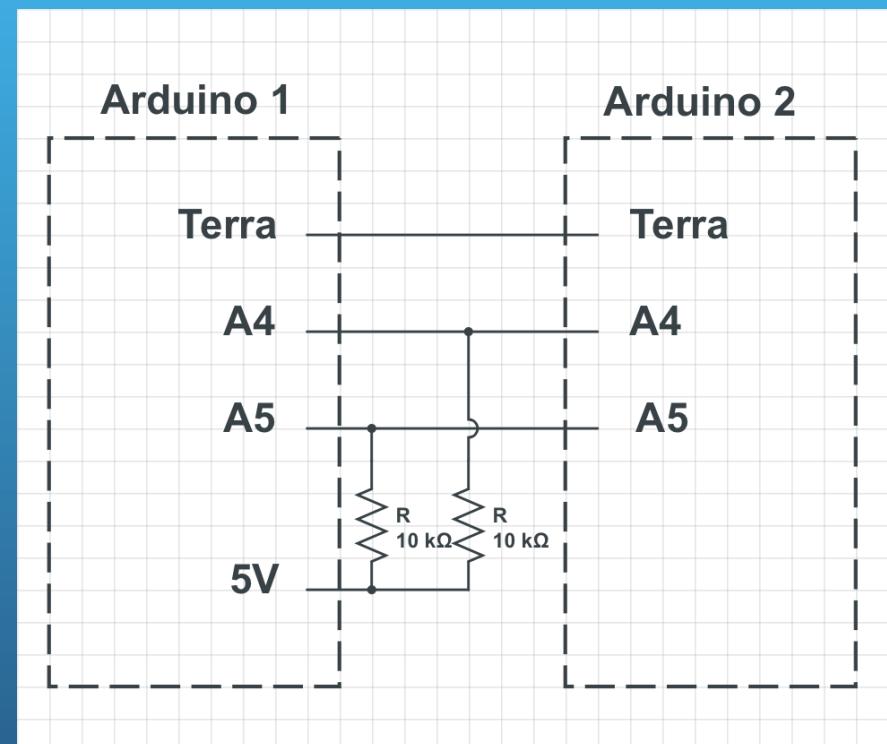
void setup() {
    // put your setup code here, to run once:
    pinMode(IR, INPUT);
    pinMode(BOTAO, INPUT);
    attachInterrupt(0, paraTudo, RISING);
    attachInterrupt(1, passouNaFrente, CHANGE);
    Serial.begin(115200);
}

void passouNaFrente()
{
    inicio = !inicio;
    if(parado)
    {
        parado = false;
        tempo = millis();
        n = 0;
        inicio = true;
    }
    if(inicio) t1 = millis();
    else
    {
        t2 = millis();
        float v = 1000*dx/(t2-t1);
        float t = (t2+t1)/2;
        Serial.print(n);
        Serial.print(" ");
        Serial.print((t-tempo)/1000,4);
        Serial.print(" ");
        Serial.println(v,4);
        n++;
    }
}
```

Comunicação entre arduinos com I2C

- Biblioteca Wire
 - <http://www.arduino.cc/en/reference/Wire>
 - #include <Wire.h>
- No Arduino UNO, os pinos que suportam comunicação I2C são
 - A4 (SDA) e A5 (SCL)
 - Deve-se fazer um pullup para VCC nestes pinos
 - ~2-5 kΩ são suficientes
 - Evite mais de um pullup na linha (diminui a resistência equivalente)
- Métodos disponíveis
 - begin(ADD) - Inicializa. ADD é o endereço ($0 \leq ADD \leq 127$). Valor pode ser emitido para o master (se houver apenas um master)
 - requestFrom
 - beginTransmission
 - endTransmission
 - write
 - Available
 - Read
 - onReceive
 - onRequest

Conectando 2 Arduinos



O Software (ligeiramente diferentes)

```
#include <Wire.h>

#define EU 5
#define OUTRO 4

void setup()
{
    Wire.begin(EU);
    Wire.onReceive(recebe);
    Serial.begin(9600);
}

byte x = 0;

void loop()
{
    Wire.beginTransmission(OUTRO);
    Wire.write("x vale ");
    Wire.write(x);
    Wire.endTransmission();

    x++;
    delay(500);
}
void recebe(int quantos)
{
    while(1 < Wire.available())
    {
        char c = Wire.read();
        Serial.print(c);
    }
    int x = Wire.read();
    Serial.println(x);
}
```

```
#include <Wire.h>

#define EU 4
#define OUTRO 5

void setup()
{
    Wire.begin(EU);
    Wire.onReceive(recebe);
    Serial.begin(9600);
}

byte x = 0;

void loop()
{
    Wire.beginTransmission(OUTRO);
    Wire.write("x vale ");
    Wire.write(x);
    Wire.endTransmission();

    x++;
    delay(500);
}
void recebe(int quantos)
{
    while(1 < Wire.available())
    {
        char c = Wire.read();
        Serial.print(c);
    }
    int x = Wire.read();
    Serial.println(x);
}
```

Atividades

- Atividade em pares de Arduino - junte-se ao seu vizinho.
- Baixe e instale o software
 - Oficina03-I2C-01-Arduino1.ino
 - Oficina03-I2C-01-Arduino2.ino
 - O software é ligeiramente diferente para poder identificar cada Arduino
- Rode o software e observe o comportamento
- Modifique no seu software o que é enviado para o Arduino vizinho
- Desafio: programe o Arduino para ler da sua porta serial e enviar a mensagem para o outro
 - Uma espécie de chat entre Arduinos.

Outras atividades

- Usem o tempo restante da oficina para explorar outras ideias.
- Estou à disposição para discuti-las.