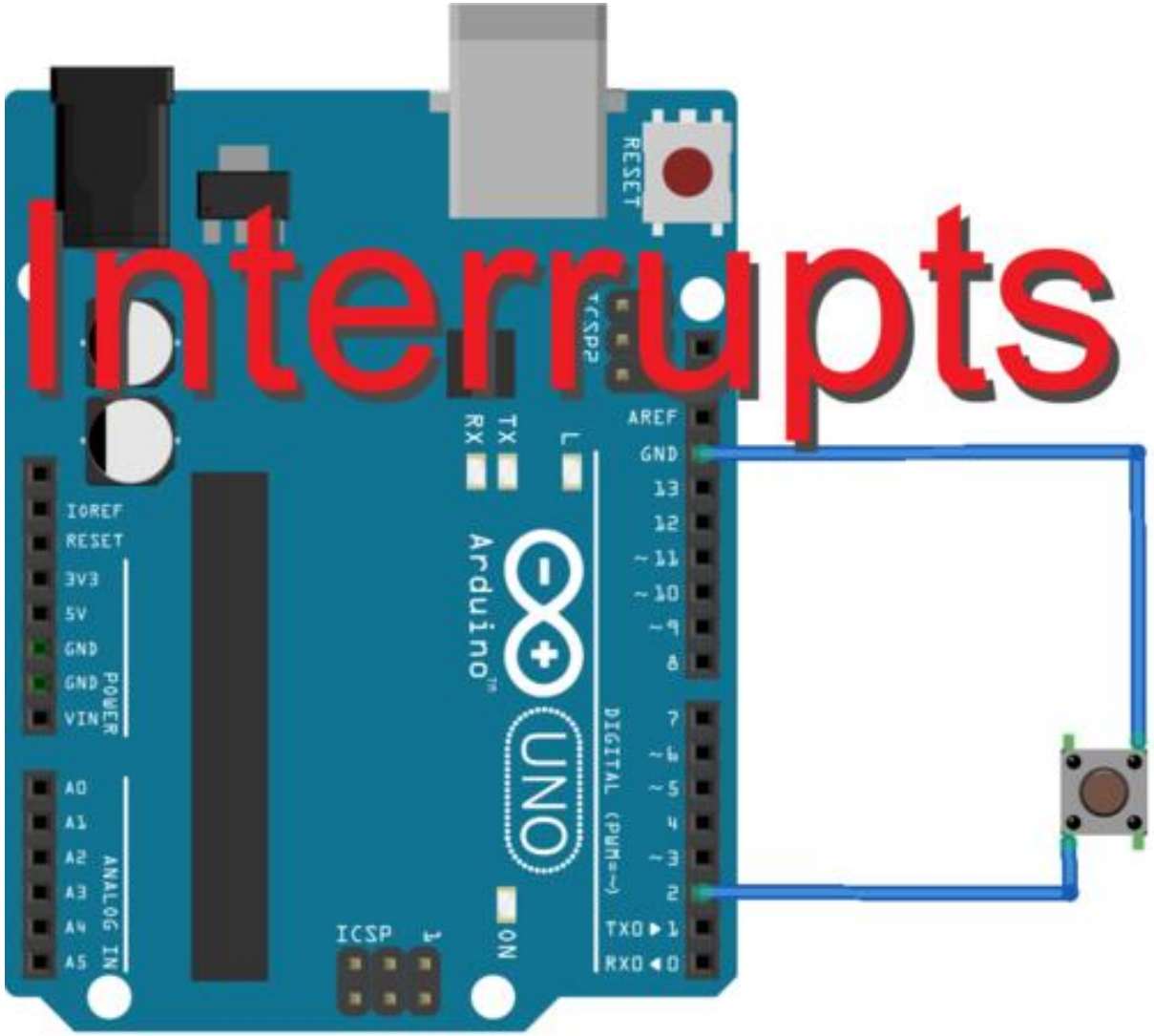
Arduino Interrupts

Karl Wallkum, Precious Plastic Saigon

Arduino Day @DEK Technologies, 09.06.18

Interrupts


RESET
TCSP2
TX
RX
L
AREF
GND
13
12
~11
~10
~9
8
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
Arduino™
UNO
DIGITAL (PWM=~)
7
~6
~5
4
~3
2
TX0 ▶ 1
RX0 ◀ 0
A0
A1
A2
A3
A4
A5
ANALOG IN
ICSP
ON

What is an Interrupt?

- Pause of Workflow
  because of an IRQ (Interrupt **R**e**Q**uest)
- Switching to another Task
- Working on the other task until finished
- Come back to Workflow and continue

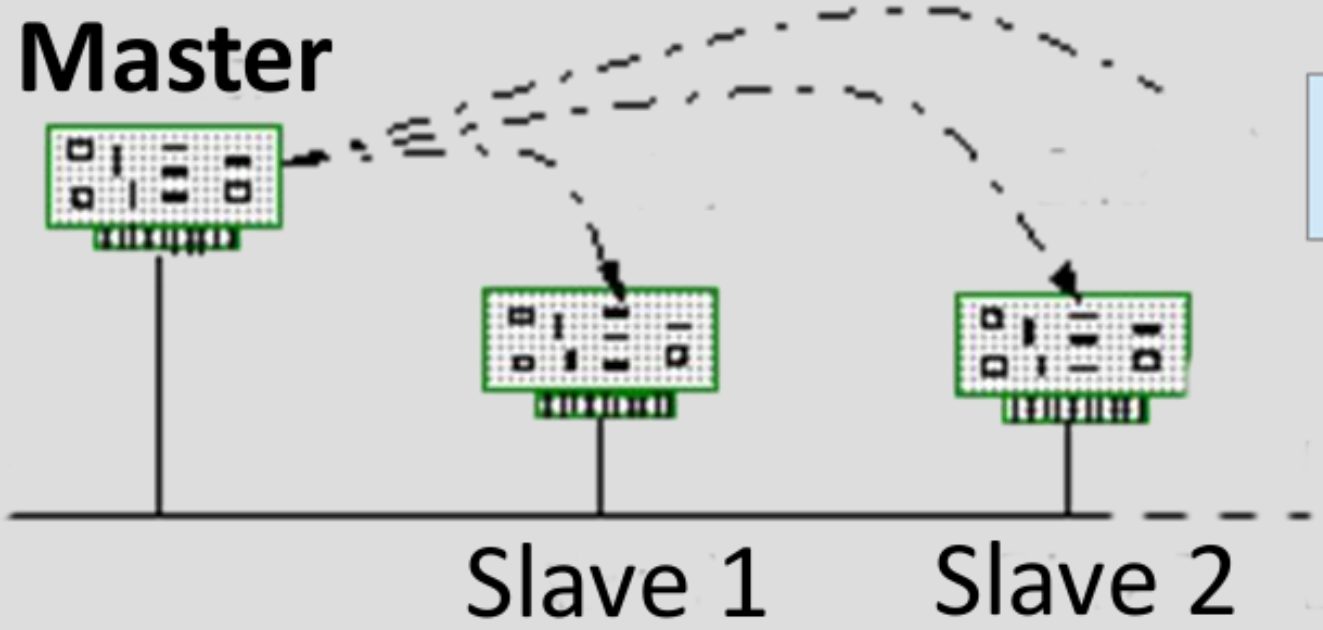
```
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

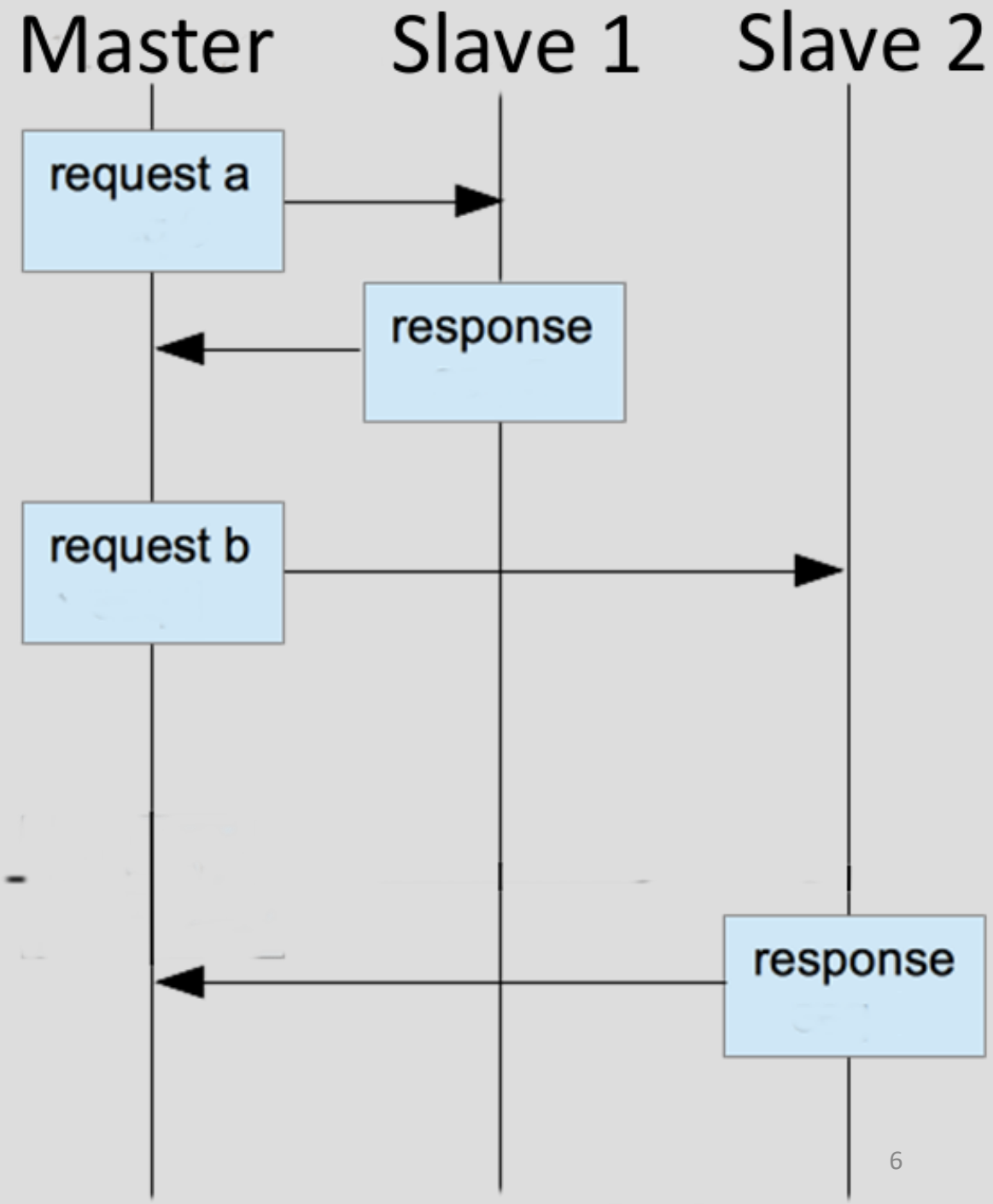
Standard Arduino Program

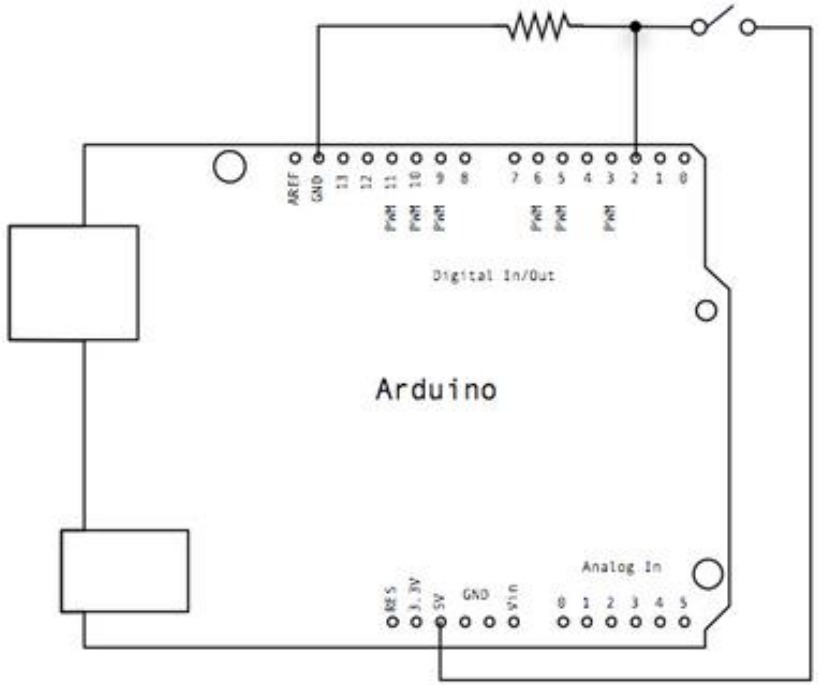
```
1  bool a;
2  void setup() {
3    // put your setup code here, to run once:
4    pinMode(2,INPUT);
5  }
6
7  void loop() {
8    // put your main code here, to run repeatedly:
9    a = digitalRead(2);
10   delay(100)
11 }
```

Simple Polling

Polling


Master
Slave 1
Slave 2

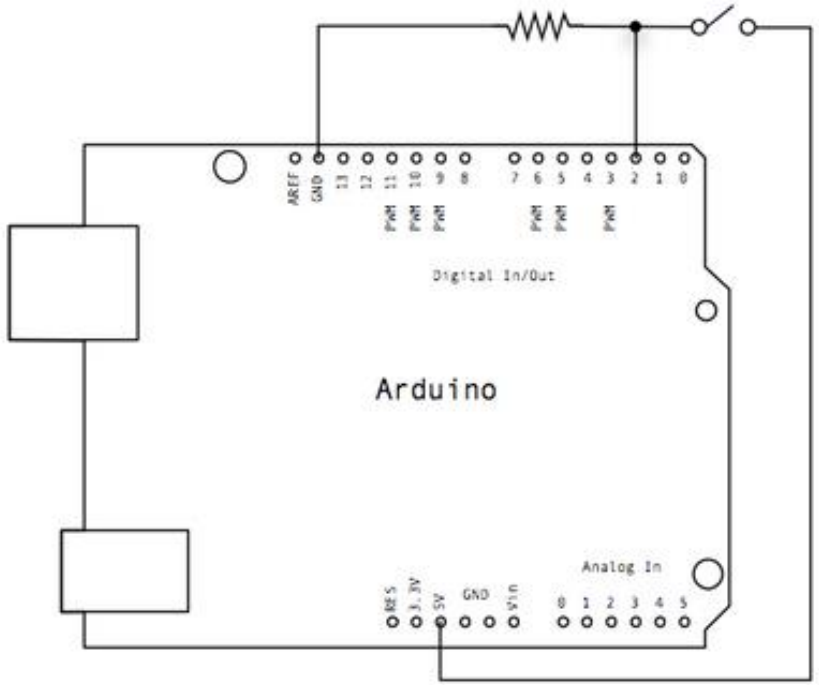

Master
Slave 1
Slave 2
request a
response
request b
response




```
27 const int buttonPin = 2;     // the number of the pushbutto
28 const int ledPin =  13;      // the number of the LED pin
29
30 int buttonState = 0;         // variable for reading the pu
31
32 void setup() {
33   pinMode(ledPin, OUTPUT);// initialize the LED pin as an o
34   pinMode(buttonPin, INPUT);// initialize the pushbutton pi
35 }
36
37 void loop() {
38   buttonState = digitalRead(buttonPin);// read the state of
39
40   // check if the pushbutton is pressed. If it is, the butt
41   if (buttonState == HIGH) {
42     digitalWrite(ledPin, HIGH);    // turn LED on:
43   } else {
44     digitalWrite(ledPin, LOW);    // turn LED off:
45   }
46 }
```

Read a button:

Arduino Button example




```
1 const byte ledPin = 13;
2 const byte interruptPin = 2;
3 volatile byte state = LOW;
4
5 void setup() {
6   pinMode(ledPin, OUTPUT);
7   pinMode(interruptPin, INPUT_PULLUP);
8   attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
9 }
10
11 void loop() {
12   digitalWrite(ledPin, state);
13 }
14
15 void blink() {
16   state = !state;
17 }
```

Read a button through Interrupt:

Arduino Button example with Interrupt

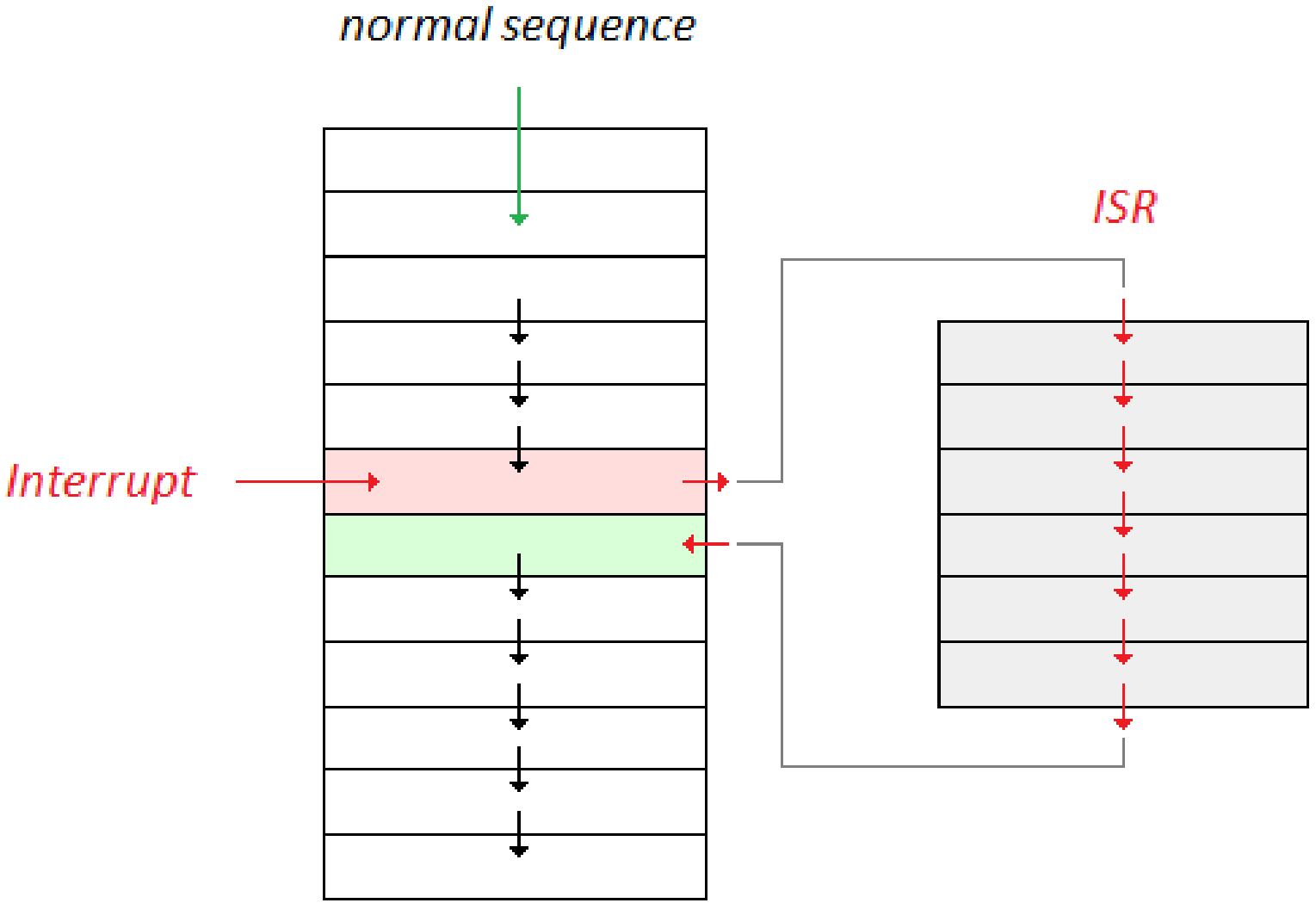
Interrupt-driven programming

- Similar to Callback, but on a deeper Hardware level.

- Interrupts are stored in an interrupt vector table
- contains the addresses of all the ISR (**I**nterrupt **S**ervice **R**outine)

Interrupt-driven programming

- Design-principle
- Get rid of delay() as much as possible
- main loop contains as little code as possible
- all events are just triggered by timers, and Hardware Interrupts



attachInterrupt()

```
attachInterrupt(digitalPinToInterrupt(interruptPin), ISR, Trigger);
```

- **use digitalPinToInterrupt(pin)**
- delay() won't work inside Interrupt
- value returned by millis() will not increment.
- Serial data received may be lost.
  Serial communication inside an ISR is not possible.
- global variables are used to pass data between an ISR and the main program.
  Declare them as volatile.

Turning interrupts off

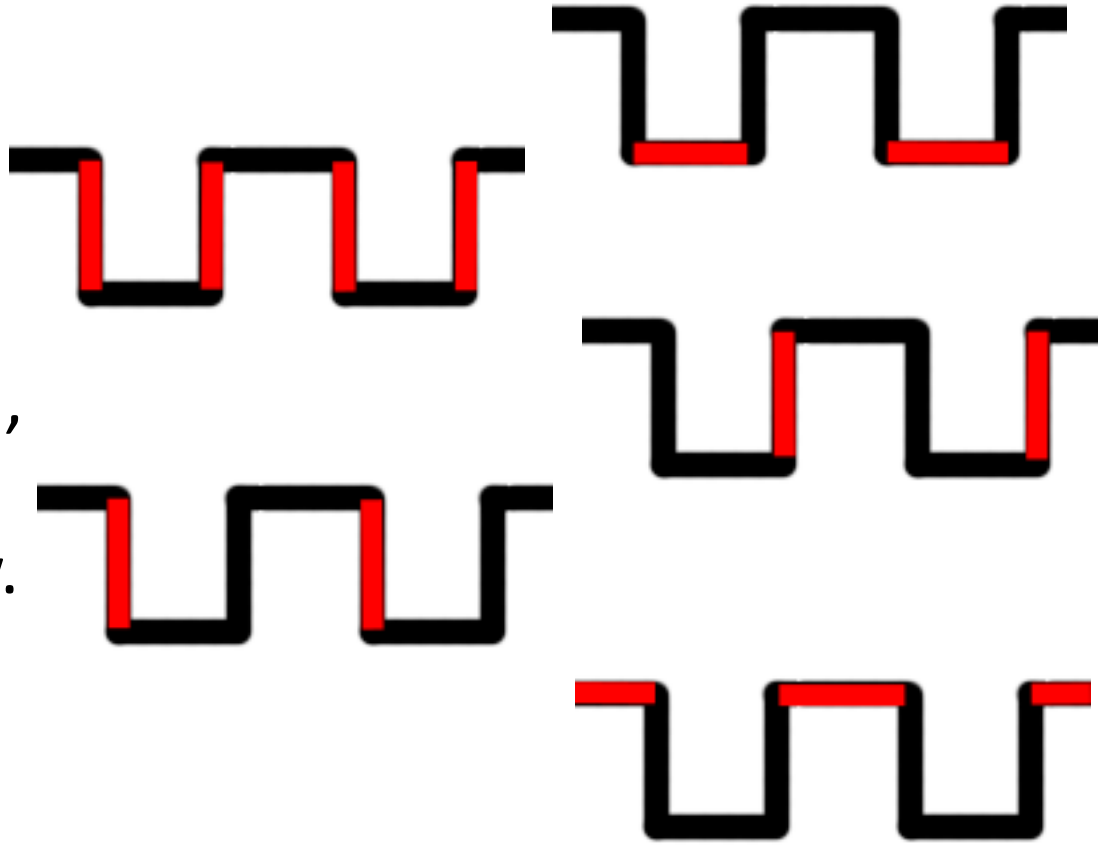
detachInterrupt() Turns off the given interrupt.
Can be re-enabled with **attachInterrupt()**

noInterrupts() Disables ALL interrupts
Can be re-enabled with **interrupts()**

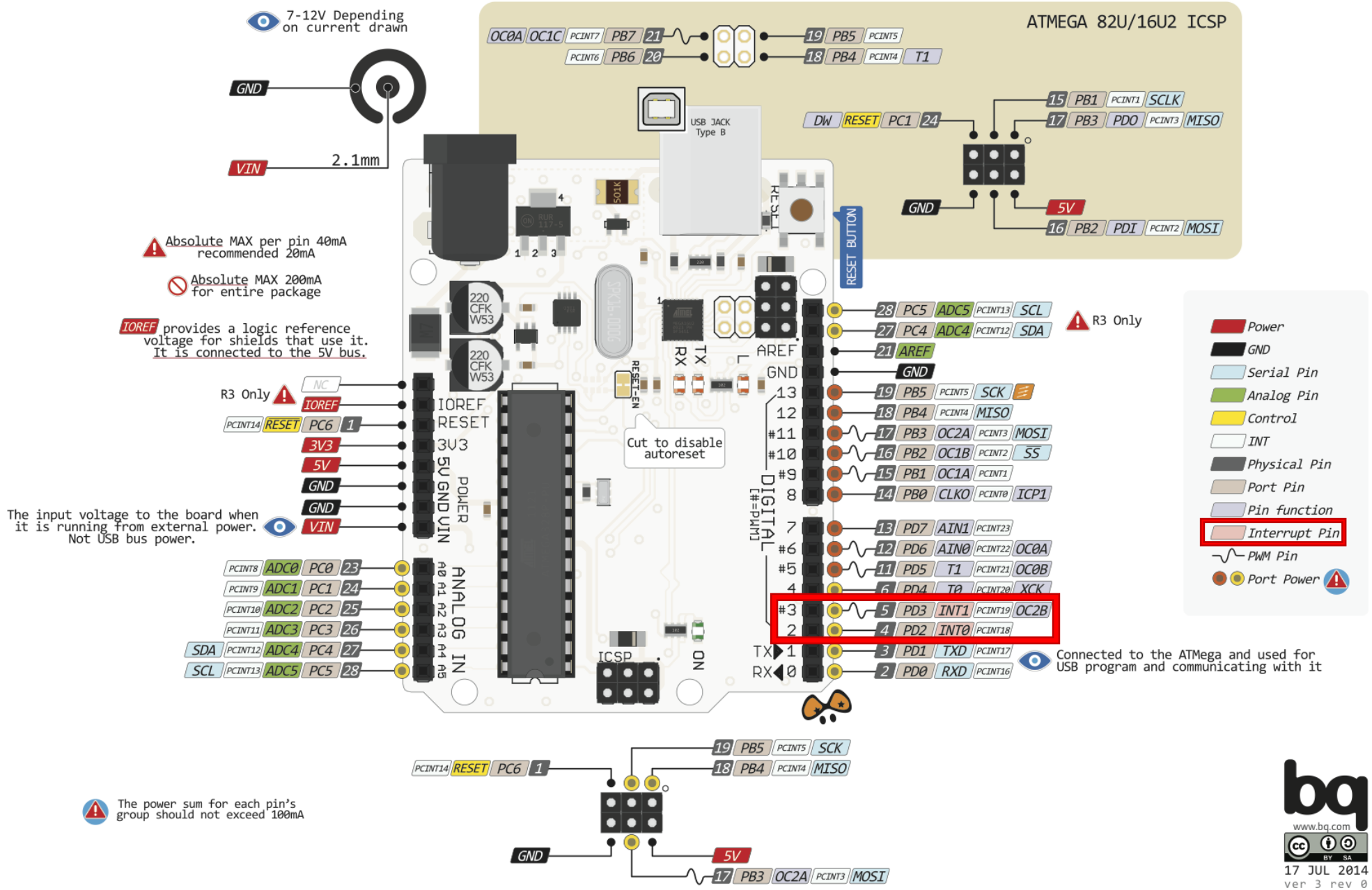
Hardware IRQ‘s

```
attachInterrupt(digitalPinToInterrupt(interruptPin), ISR, Trigger);
attachInterrupt(digitalPinToInterrupt(2), button1_pressed,HIGH);
```

- **LOW** pin is low,
- **CHANGE** pin changes value
- **RISING** pin goes from low to high,
- **FALLING** pin goes from high to low.
- **HIGH** the pin is high.



UNO PINOUT



17 JUL 2014
ver 3 rev 0

ATmega328 Interrupt table

Sorted by priority

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/Coutner1 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

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/Coutner1 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

Hardware Interrupts for different Boards

Uno			
attachInterrupt	Name	Pin on chip (PDIP)	Pin on board
0	INT0	4	D2
1	INT1	5	D3

Mega2560			
attachInterrupt	Name	Pin on chip (TQFP)	Pin on board
0	INT4	6	D2
1	INT5	7	D3
2	INT0	43	D21
3	INT1	44	D20
4	INT2	45	D19
5	INT3	46	D18

Leonardo			
attachInterrupt	Name	Pin on chip (TQFP)	Pin on board
0	INT0	18	D3
1	INT1	19	D2
2	INT2	20	D0
3	INT3	21	D1
4	INT6	1	D7

Timer

AVR microcontrollers have Timers with counting-registers:

Size: **8** or **16**bits.
Registers start with an initial value of **0**.
increment (counting up) is done automatically and peridocally.
When a counter has an **overflow**, a Timer-overflow-Interrupt is thrown.

The Timers of an AVR-Microcontroller run independent of the CPU.

For good resolution, incrementing the counter could be done syncronously to the CPU-clock:

Problem:
On an Arduino UNO, the Atmega328 runs with 16Mhz.
8-bit-timer would **overflow** after 16 microseconds,
the 16-bit-Register after ca. 4.1 Miliseconds.

Solution: **Prescaler**
Increment the counter after a defined amount of clock-counts.
Usual values: **8, 64, 256** oder **1024**.
A Prescale of 1024 increments a counter after 64 microseconds.
This allows a **16**-bit counter to run for ca. 4.2 seconds until it overflows.

Timer example

- Every 0,5 seconds, the timer should overflow:

- 16-Bit-Timer: *bits* = 16 => ***maxcount*** = 2^{16} = **65536.**
- 1 timeroverflow per 0.5 seconds: *deltaT* = 0,5 sec = *1 / timerfreq*
- The CPU-clock of an Arduino-Board is: *cpufreq* = 16 MHz = **16.000.000 Hz**
- by default: *prescale* = **256**
- The timer starts with 0 with *initcount* = 65.536 - 8.000.000/256 = **34.286**

```
#define ledPin 13

void setup()
{
  pinMode(ledPin, OUTPUT);  // Ausgabe LED festlegen

  // Timer 1
  noInterrupts();           // Alle Interrupts tempo
  TCCR1A = 0;
  TCCR1B = 0;

  TCNT1 = 34286;            // Timer nach obiger Rec
  TCCR1B |= (1 << CS12);    // 256 als Prescale-Wert
  TIMSK1 |= (1 << TOIE1);   // Timer Overflow Interr
  interrupts();             // alle Interrupts schar
}
// Hier kommt die selbstdefinierte Interruptbehandlu
// für den Timer Overflow
ISR(TIMER1_OVF_vect)
{
  TCNT1 = 34286;              // Zähler erneut vorbel
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1); //
}

void loop()
{
  // Wir könnten hier zusätzlichen Code integrieren
}
```

Default timer use on Arduinos

An Arduino doesn‘t only have a single Timer, but several of them:

- Timer 0 (8 Bit) used for functions like *delay(), millis(), micros()*
- Timer 1 (16 Bit) used in the Servo-Library
- Timer 2 (8 Bit) used in the Tone-Library

PWM:

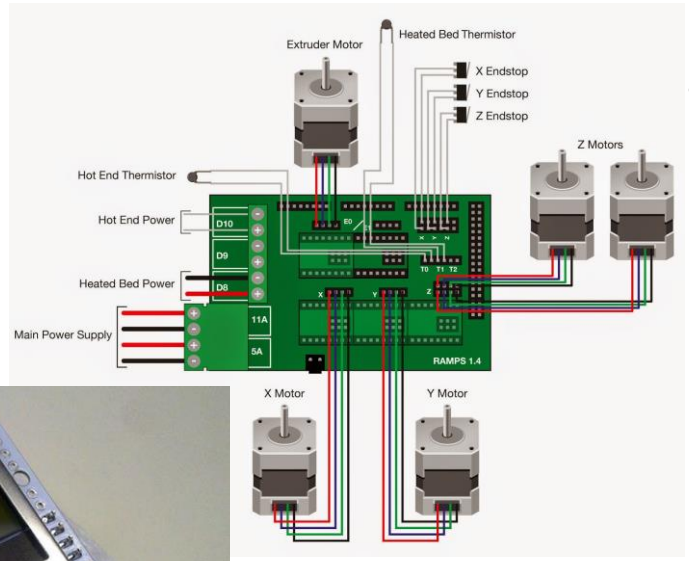
- Timer 0 : Pins **5** + **6**
- Timer 1 : Pins **9** + **10**
- Timer 2 : Pins **3** + **11**

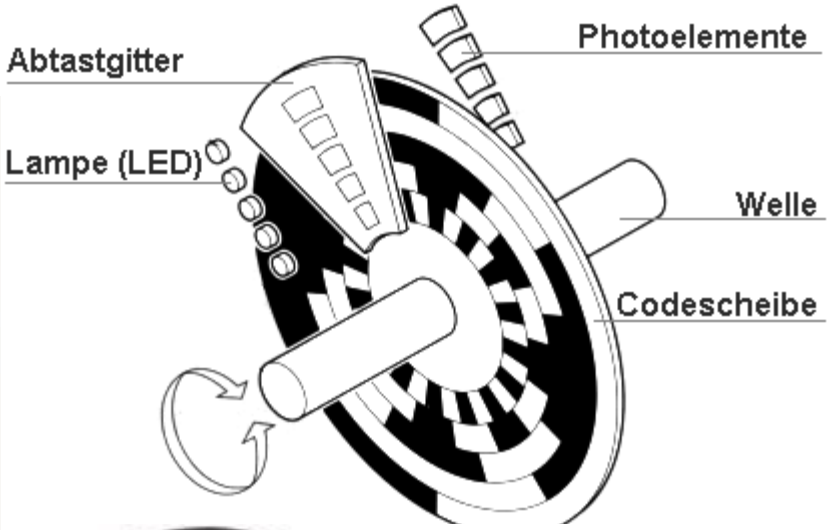
Be careful, when changing Timers!

Take care, how much your Microcontroller can handle

- Interrupts will pause the regular program flow
- Only handle easy stuff during an interrupt
- KISS (*Keep it short and simple)*
- *Do workload-heavy stuff outside of interrupts*
- *Cache Data produced in an interrupt (e.g. with a FIFO/Queue) and save/send it during normal program flow*

Application of interrupts





Pc-Interrupts

With Linux Interrupts can be shown with the following command:

cat /proc/interrupts

With Windows Interrupts can be shown with:

msinfo32.exe → Hardwareressourcen → IRQs