



Embedded Programming for Beginners

Implementing an embedded application
using Arduino

Session 3

Course Goals

- Objective 1
 - Understand the Arduino world and history
 - Learn Arduino internals, peripherals, project options
- Objective 2
 - Understand how to identify components for an embedded project
 - Introduction to datasheets
 - Integrate registers and bitwise operations into the mix
- Objective 3
 - Building applications with open source software
 - Building temperature reading application and integrate it with the data acquisition and the rest of kit components

Hello world: C vs C++

```
0 ~/workspace
$ cat test.c
/*
 * =====
 *
 *      Filename:  test.c
 *
 *      Description:
 *
 *      Version:   1.0
 *      Created:   11/06/2021 10:43:51 AM
 *      Revision:  none
 *      Compiler:  gcc
 *
 *      Author:    Vaduva Jan Alexandru (),
 *      Organization:
 *
 * =====
 */

#include <stdio.h>

int main()
{
    printf("Hello World");

    return 0;
}
```

```
0 ~/workspace
$ g++ -S test.cpp
```

```
0 ~/workspace
$ gcc -S test.c
```

```
0 ~/workspace
$ g++ -S test2.cpp
```

```
0 ~/workspace
$ cat test.cpp
/*
 * =====
 *
 *      Filename:  test.c
 *
 *      Description:
 *
 *      Version:   1.0
 *      Created:   11/06/2021 10:43:51 AM
 *      Revision:  none
 *      Compiler:  gcc
 *
 *      Author:    Vaduva Jan Alexandru (),
 *      Organization:
 *
 * =====
 */

#include <stdio.h>

int main()
{
    printf("Hello World");

    return 0;
}

0 ~/workspace
$ cat test2.cpp
/*
 * =====
 *
 *      Filename:  test2.cpp
 *
 *      Description:
 *
 *      Version:   1.0
 *      Created:   11/06/2021 10:56:01 AM
 *      Revision:  none
 *      Compiler:  gcc
 *
 *      Author:    Vaduva Jan Alexandru (),
 *      Organization:
 *
 * =====
 */

#include <iostream>

using namespace std;

int main()
{
    cout<<"Hello World";

    return 0;
}
```

Hello world: C vs C++ assembly diff

```
.file "test.c"
.text
.section .rodata
.LC0:
.string "Hello World"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
leaq .LC0(%rip), %rdi
movl $0, %eax
call printf@PLT
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 7.5.0-3ubuntu1-18.04) 7.5.0"
.section .note.GNU-stack,"",@progbits

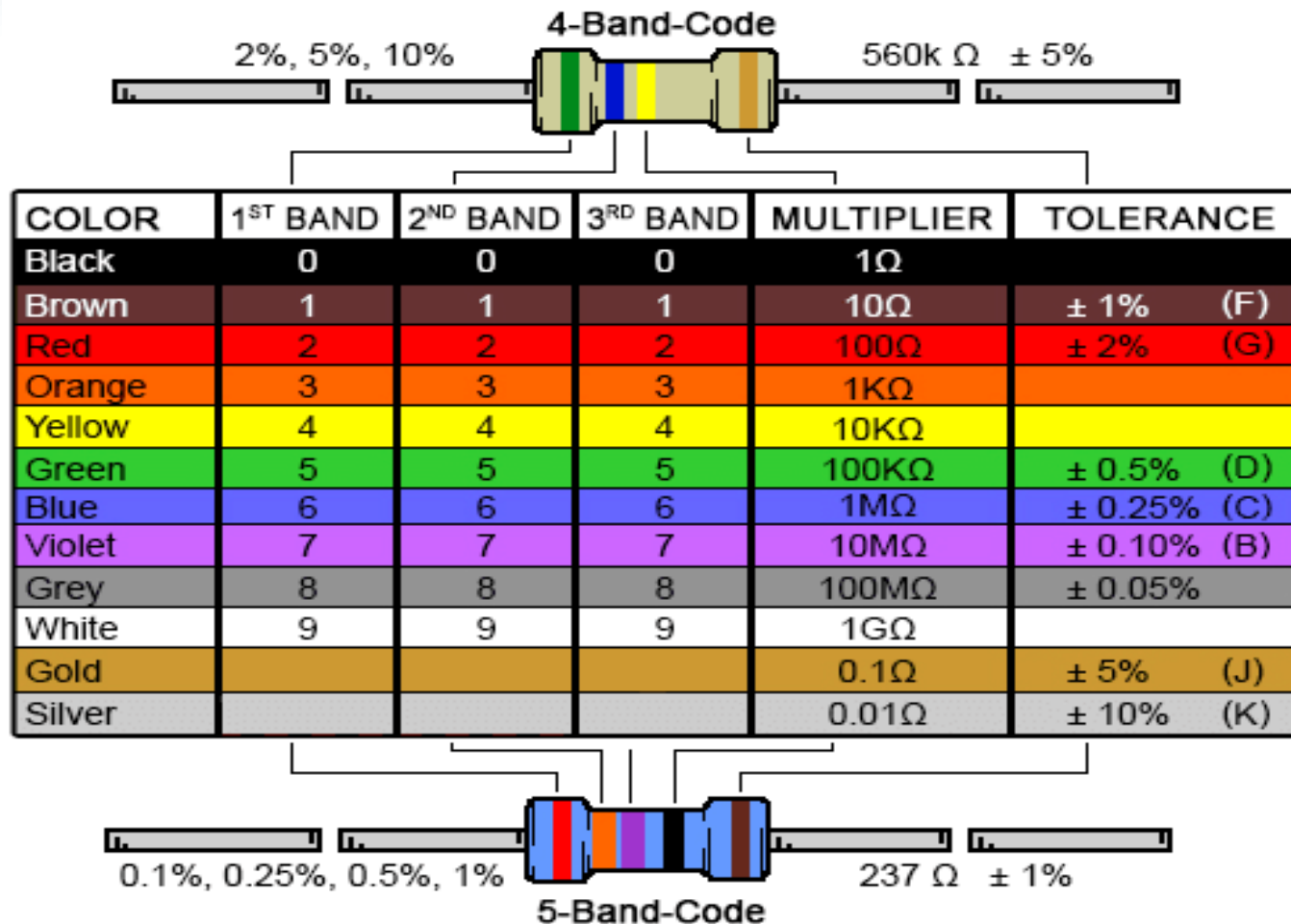
.file "test.cpp"
.text
.section .rodata
.LC0:
.string "Hello World"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
leaq .LC0(%rip), %rdi
movl $0, %eax
call printf@PLT
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 7.5.0-3ubuntu1-18.04) 7.5.0"
.section .note.GNU-stack,"",@progbits

.file "test2.cpp"
.text
.section .rodata
.type _ZStL19piecewise_construct, @object
.size _ZStL19piecewise_construct, 1
_ZStL19piecewise_construct:
.zero 1
.local _ZStL8_iolinit
.comm _ZStL8_iolinit,1,1
.LC0:
.string "Hello World"
.text
.globl main
.type main, @function
main:
.LFB1493:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
leaq .LC0(%rip), %rsi
leaq _ZSt4cout(%rip), %rdi
call _ZStI1st1char_traitsIcEERSt13basic_ostreamIcT_ES5_PKC@PLT
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE1493:
.size main, .-main
.type _Z41_static_initialization_and_destruction_0ii, @function
_Z41_static_initialization_and_destruction_0ii:
.LFB1977:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl %edi, -4(%rbp)
movl %esi, -8(%rbp)
cmpl $1, -4(%rbp)
jne .L5
cmpl $0x535, -8(%rbp)
jne .L5
leaq _ZStL8_iolinit(%rip), %rdi
call _ZNSt8ios_base4InitC1Ev@PLT
leaq __dso_handle(%rip), %rdx
leaq _ZStL8_iolinit(%rip), %rsi
movq _ZNSt8ios_base4InitD1Ev@GOTPCREL(%rip), %rax
movq %rax, %rdi
call __cxa_atexit@PLT
.L5:
nop
leave
```

Hello world: C++ assembly extra

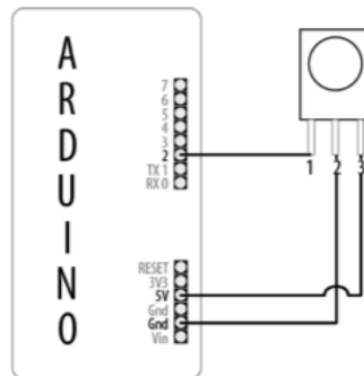
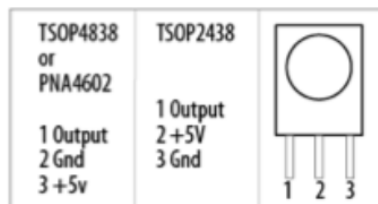
```
.cfi_endproc
.LFE1493:
.size main, .-main
.type _Z41__static_initialization_and_destruction_0ii, @function
_Z41__static_initialization_and_destruction_0ii:
.LFB1977:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl %edi, -4(%rbp)
movl %esi, -8(%rbp)
cmpl $1, -4(%rbp)
jne .L5
cmpl $65535, -8(%rbp)
jne .L5
leaq _ZStL8__ioint(%rip), %rdi
call __ZNSt8ios_base4InitC1Ev@PLT
leaq __dso_handle(%rip), %rdx
leaq _ZStL8__ioint(%rip), %rsi
movq __ZNSt8ios_base4InitD1Ev@GOTPCREL(%rip), %rax
movq %rax, %rdi
call __cxa_atexit@PLT
.L5:
nop
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE1977:
.size _Z41__static_initialization_and_destruction_0ii, .-_Z41__static_ini
.type _GLOBAL__sub_I_main, @function
_GLOBAL__sub_I_main:
.LFB1978:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $65535, %esi
movl $1, %edi
call _Z41__static_initialization_and_destruction_0ii
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE1978:
.size _GLOBAL__sub_I_main, .-_GLOBAL__sub_I_main
.section .init_array,"aw"
.align 8
.quad _GLOBAL__sub_I_main
.hidden __dso_handle
.ident "GCC: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0"
.section .note.gnu-stack,"",@progbits
```

How to read resistors value



Arduino interrupts

- We have an IR detector connected to pin 2
- The program monitors pulse on this pin and stores the duration of each pulse in an array
- When the array is appended, the duration is displayed on the Serial Monitor



```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```

Volatile keyword

- Its a variable qualifier; it is used before the datatype of a variable, to modify the way in which the compiler and subsequent program treats the variable.
- Compiler will load the variable from RAM and not from a storage register.
- Under certain conditions, such as through interrupts, the value for a variable stored in registers can be inaccurate.

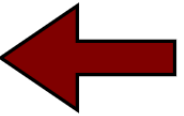
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```



Changing variables

- The variables that are changed in an interrupt function are declared as volatile.
- This lets the compiler know that the values could change at any time (by an interrupt handler).
- Without using the volatile keyword, the compiler would think these variables are not being changed by any code getting called and would replace these variables with constant values.

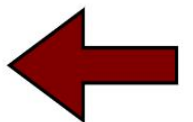
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```



attachInterrupt() function

- The attachInterrupt(0, analyze, CHANGE); call enables the program to handle interrupts.
- The first number in the call specifies which interrupt to initialize.
- On a standard Arduino board (such as UNO), two interrupts are available: number 0, which uses pin 2, and number 1 on pin 3.
- Interrupt 0 and interrupt 1 have the same priorities (with Wiring).

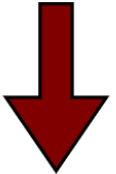
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```



List of interrupts

Vector no.	Program address	Source	Interrupt definition
1	0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset and JTAG AWR Reset
2	0002	INT0	External Interrupt Request 0
3	0004	INT1	External Interrupt Request 1
4	0006	PCINT0	Pin Change Interrupt Request 0
5	0008	PCINT1	Pin Change Interrupt Request 1
6	000A	PCINT2	Pin Change Interrupt Request 2
7	000C	WDT	Watchdog Time-out Interrupt
8	000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0010	TIMER2_COMPB	Timer/Counter2 Compare Match B
10	0012	TIMER2_OVF	Timer/Counter 2 Overflow
11	0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0018	TIMER1_COMPB	Timer/Counter1 Compare Match B
14	001A	TIMER1_OVF	Timer/Counter1 Overflow
15	001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	001E	TIMER0_COMPB	Timer/Counter0 Compare Match B
17	0020	TIMER0_OVF	Timer/Counter0 Overflow
18	0022	SPI_STC	SPI Serial Transfer Complete
19	0024	USART0_RX	USART0 Rx Complete
20	0026	USART0_UDRE	USART0 Data Register Empty
21	0028	USART0_TX	USART0 Tx Complete
22	002A	ADC	ADC Conversion Complete
23	002C	EE_READY	EEPROM Ready
24	002E	ANALOG_COMP	Analog Comparator
25	0030	TWI	Two-Wire Serial Interface
26	0032	SPM_READY	Store Program Memory Ready

attachInterrupt() function arguments

- The second parameter specifies what function to call (interrupt handler) when the interrupt event happens.
- The final parameter specifies what should trigger the interrupt:
 - CHANGE: whenever the pin level changes (low to high or high to low).
 - LOW: when the pin is low.
 - RISING: when the pin goes from low to high.
 - FALLING: when the pin goes from high to low.

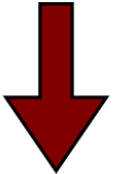
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```



Main loop

- The main loop just checks the index variable to see if all the entries have been set by the interrupt handler.
- It will print the contents of the array results only once.
- Nothing in loop changes the value of index. The index is changed inside the analyze function when the interrupt condition occurs.

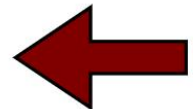
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```



Termination condition

- The code stays in the while loop at the end of the inner block, so you need to reset the board when you want to do another run.

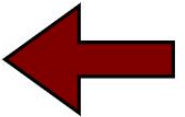
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```



micros() function

- The micros() function returns the number of micro-seconds since the Arduino began running the current program.
- This number will overflow (go back to zero), after approximately 70 minutes.
- On 16 MHz Arduino boards (e.g. UNO), this function has a resolution of four microseconds (i.e. the value returned is always a multiple of four).

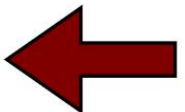
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```



analyze() function

- The index value is used to store the time since the last state change into the next slot in the results array.
- The time is calculated by subtracting the last time, the state changed from the current time in microseconds.
- The current time is then saved as the last time a change happened.

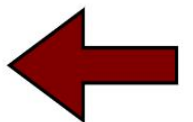
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = microseconds - microseconds;
    }
    index = index + 1;
  }
  microseconds = microseconds;
}
```



Source code overview

- Each time an interrupt is triggered, index is incremented and the current time is saved.
- The time difference is calculated and saved in the array (except for the first time the interrupt is triggered, when index is 0).
- When the maximum number of entries has occurred, the inner block in loop runs, and it prints out all the values to the serial port.

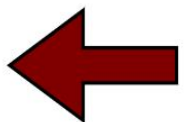
```
const int irReceiverPin = 2;           //pin the receiver is connected to
const int numberOfEntries = 64;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long results[numberOfEntries];

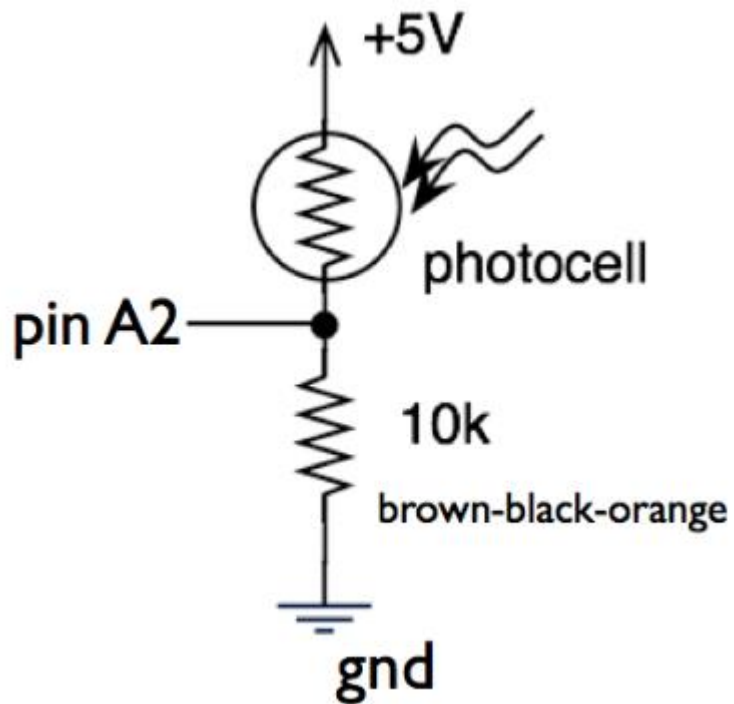
void setup()
{
  pinMode(irReceiverPin, INPUT);
  Serial.begin(9600);
  attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)
  {
    Serial.println("Durations in Microseconds are:");
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    while(1)
    ;
  }
  delay(1000);
}

void analyze()
{
  if(index < numberOfEntries )
  {
    if(index > 0)
    {
      results[index] = micros() - microseconds;
    }
    index = index + 1;
  }
  microseconds = micros();
}
```



Hands-on exercise: What's the code doing?



```
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;
}
```

Arduino wiring C limitations

- Inside the interrupt function, **delay()** won't work
- Values returned by the **millis()** and **micros()** functions will not increment.
- Serial communications while in the function may be lost!
- You should declare as volatile any variables that you modify within the attached function.
- By default, interrupts are atomic.

Atomic sections in Arduino

- We can enable/disable interrupts on certain sections of code with **interrupts()** and **noInterrupts()**.
- Interrupts in Arduino are enabled by default.
- Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.

```
void setup() {}

void loop()
{
    noInterrupts();
    // critical, time-sensitive code here
    interrupts();
    // other code here
}
```

Interrupts default

- By default, you can not have interrupts inside interrupts.
- With C-Wiring both interrupt 0 and interrupt 1 are assigned the same priority.
- If you have a way to have interrupts inside interrupts, then when an interrupt is issued, you immediately leave the current interrupt and execute the new interrupt.
- Enabling interrupts inside interrupts is a “hack” : Is not recommended as it raises all sorts of issues with preserving the state of the machine before the interrupting interrupt is serviced.

Enabling interrupts inside interrupts

```
volatile int i,j,z;
```

```
void artificialdelay()
```

```
{
  for (i=0; i<900; i++){
    for (j=0; j<900; j++){
      z=i*10;}}
}
```

```
void setup()
```

```
{
  Serial.begin(9600);
  attachInterrupt(0, interrupt0, CHANGE);
  attachInterrupt(1, interrupt1, CHANGE);
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
}
```

```
void loop()
```

```
{
  Serial.println("entering main loop");
  artificialdelay();
  Serial.println("leaving main loop");
}
```

```
void interrupt0()
```

```
{
  interrupts();
  digitalWrite(13,HIGH);
  artificialdelay();
  digitalWrite(13,LOW);
  noInterrupts(); //not really needed
}
```

```
void interrupt1()
```

```
{
  interrupts();
  digitalWrite(12,HIGH);
  artificialdelay();
  digitalWrite(12,LOW);
  noInterrupts(); //not really needed
}
```

- If I press pin2 will go into `interrupt0()`.
- While it is processing, I can press pin3 and jump into function `interrupt1()`.

Hands-on exercise: Operation scenarios (1)

```
volatile int i,j,z;
```

```
void artificialdelay()
```

```
{
  for (i=0; i<900; i++){
    for (j=0; j<900; j++){
      z=i*10;}}
}
```

```
void setup()
```

```
{
  Serial.begin(9600);
  attachInterrupt(0, interrupt0, CHANGE);
  attachInterrupt(1, interrupt1, CHANGE);
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
}
```

```
void loop()
```

```
{
  Serial.println("entering main loop");
  artificialdelay();
  Serial.println("leaving main loop");
}
```

```
void interrupt0()
```

```
{
  interrupts();
  digitalWrite(13,HIGH);
  artificialdelay();
  digitalWrite(13,LOW);
  noInterrupts(); //not really needed
}
```

```
void interrupt1()
```

```
{
  interrupts();
  digitalWrite(12,HIGH);
  artificialdelay();
  digitalWrite(12,LOW);
  noInterrupts(); //not really needed
}
```

- What happens if you trigger **interrupt0** when inside **interrupt0**?

Hands-on exercise: Operation scenarios (2)

```
volatile int i,j,z;

void artificialdelay()
{
    for (i=0; i<900; i++){
        for (j=0; j<900; j++){
            z=i*10;}}
}

void setup()
{
    Serial.begin(9600);
    attachInterrupt(0, interrupt0, CHANGE);
    attachInterrupt(1, interrupt1, CHANGE);
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
}

void loop()
{
    Serial.println("entering main loop");
    artificialdelay();
    Serial.println("leaving main loop");
}
```

```
void interrupt0()
{
    //interrupts() ;
    digitalWrite(13,HIGH);
    artificialdelay();
    digitalWrite(13,LOW);
    //noInterrupts() ;
}

void interrupt1()
{
    //interrupts() ;
    digitalWrite(12,HIGH);
    artificialdelay();
    digitalWrite(12,LOW);
    //noInterrupts() ;
}
```

- What happens if you trigger **interrupt1** when inside **interrupt0**?

Arduino internal interrupts

- Start the interrupt handling mechanism

```
sei ();                // enable interrupts
cli ();                // disable interrupts
```

- Configure the device that will send the interrupts

```
OCR1A = 31249; // compare match register 16MHz / 256 / 2Hz-1
TCCR1B | = (1 << WGM12); // CTC mode
TCCR1B | = (1 << CS12); // 256 prescaler
```

- Write the interrupt handling routine

```
ISR (TIMER1_COMPA_vect) {
    // interrupt code from Timer1
}
```

- Activate the interrupt and test the program

```
TIMSK1 | = (1 << OCIE1A); // enable interrupt TIMER1_COMPA
sei ();                    // enable interrupts globally
```

Arduino using the interrupts

- No return required
- In handler the interrupt inhibits the execution of other interruptions
- Short execution time
- Volatile variables
- Other wrappers (around AVR core instructions) offered by the interface are:
 - sei () - sets to bit I of the SREG, activating global interrupts. Call the assembler sei instruction
 - cli () - sets bit I in SREG to 0, disabling global interrupts. Call the assembler cli instruction
 - reti () - return from an interrupt handling routine, activates global interrupts. It should be the last instruction called in a routine that uses the ISR_NAKED flag.

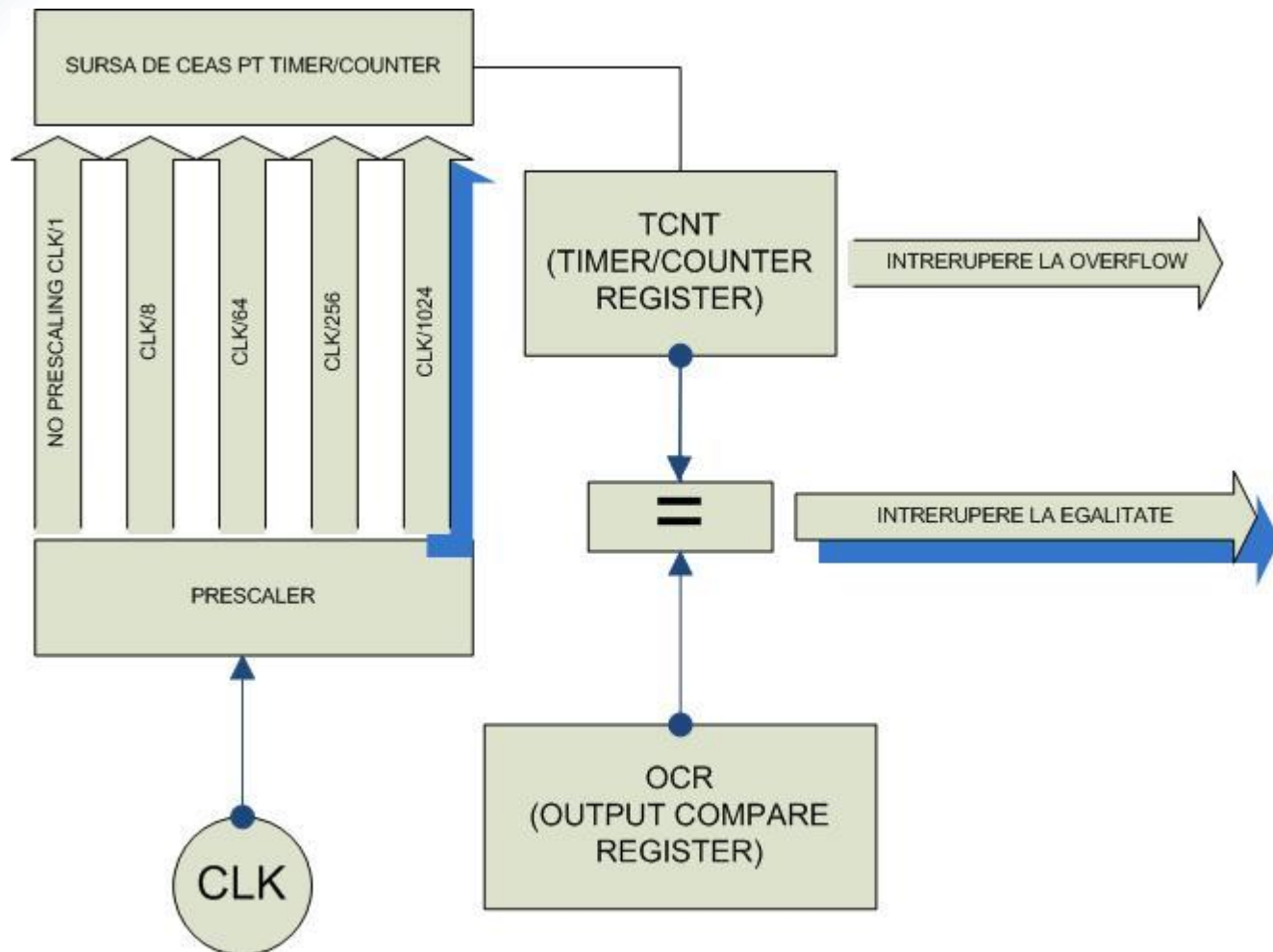
Hands-on exercise: Timers and interrupts

- Using Timers and Interrupts, you will achieve an improved version of Hello World (blink) and a tone generator using the buzzer. Thinkercad and Arduino.
 - **Task 0:** Configure Timer 1 using registers and the TIMER1_COMPA interrupt to light an LED once per second:

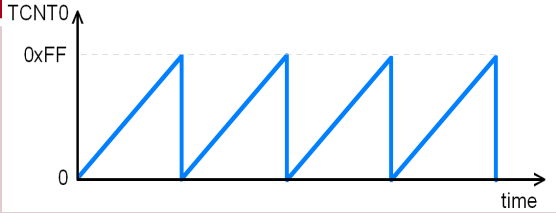
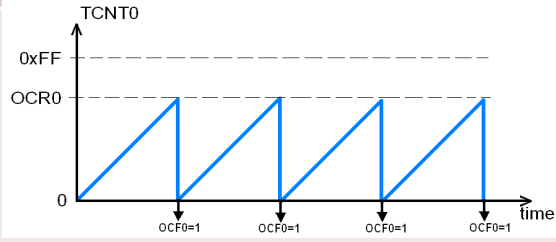
```
ISR(TIMER1_COMPA_vect) {  
    // blink led  
}  
  
void setup() {  
    // configurați pin-ul de output  
  
    // inițializare timer 1  
    cli();  
  
    TCCR1A = 0;  
    TCCR1B = 0;  
    TCNT1 = 0;  
  
    OCR1A = 31249;           // compare match register 16MHz/256/2Hz-1  
    TCCR1B |= (1 << WGM12); // CTC mode  
    TCCR1B |= (1 << CS12);  // 256 prescaler  
    TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt  
    sei();  
}  
  
void loop() {  
  
}
```

- **Task 1:** Configure Timer 1 using registers and the TIMER1_COMPA interrupt to command a buzzer at an audible frequency (e.g. a musical tone / note between 440 and 880 Hz). See this link for the frequency of musical notes: <https://www.intmath.com/trigonometric-graphs/music.php>

Timer: principle of operation



Operating modes

Mode	Description	Image counter	Properties
Normal	<ul style="list-style-type: none"> - starts at 0 - counts up to 0xFFFF 		the frequency is set to a few predefined values, given by the clock frequency and the prescaler
CTC Clear Timer on Compare	<ul style="list-style-type: none"> - starts at 0 - counts until a threshold is reached (OCRnA for Timer 0/2, OCR1A or ICR1 for Timer 1) 		the frequency is variable, determined by the threshold value, the clock frequency and the prescaler

Registers

Timer	Registers	Role
Timer0 8 bytes	TCNT0	Timer counter register 0 (counting)
	TCCR0A, TCCR0B	Timer control registers 0 (various bits for timer configuration)
	OCR0A, OCR0B	Threshold registers for timer 0 (countdown threshold at which interruption can be triggered, depending on configuration)
	TIMSK0, TIFR0	Registers with activation bits for timer interrupts 0 / activation flags (activate interrupts)
Timer1 16 bytes	TCNT1	Timer counter register 1 (same as timer0, only 16-bit)
	TCCR1A TCCR1B TCCR1C	Timer 1 control registers (same as timer0)
	OCR1A , OCR1B	16-bit threshold registers of timer 1 (same as timer0)
	TIMSK1, TIFR1	(same as timer0)
	ICR1	Register used to retain the value of the counter when an external event occurs on the ICP pin or as a threshold for one of the CTC modes
Timer2 8 bytes	the same registers as Timer0	The difference from Timer 0 is the possibility to use a separate external oscillator for Timer 2, on pins TOSC1 and TOSC2
	ASSR, GTCCR	Registers related to the asynchronous operation of this timer compared to the rest of the system

Working with timers

- Setting the operating mode

```
TCCR0A |= (1 << WGM01);  
OCR0A = 5;
```

- Setting the prescaler

```
TCCR2B |= (1 << CS22) | (1 << CS21);
```

- Interrupt configuration

```
ISR(TIMER1_COMPA_vect) {  
    // cod întrerupere  
}  
  
void configure_timer1() {  
    // exemplu de configurare pentru Timer 1 în mod CTC  
    // care va genera întreruperi cu frecvența de 2Hz  
    OCR1A = 31249; // compare match register 16MHz/256/2Hz-1  
    TCCR1B |= (1 << WGM12); // CTC mode  
    TCCR1B |= (1 << CS12); // 256 prescaler  
}  
  
void init_timer1() {  
    TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt  
}  
  
void setup() {  
    // dezactivăm întreruperile globale  
    cli();  
    configure_timer1();  
    init_timer1();  
    // activăm întreruperile globale  
    sei();  
}  
  
void loop() {  
}
```

ATmega Timer calculator

- ATmega Timer/Counter/Prescaler calculator:
https://web.archive.org/web/20180312235028/https://et06.dunweber.com/atmega_timers/

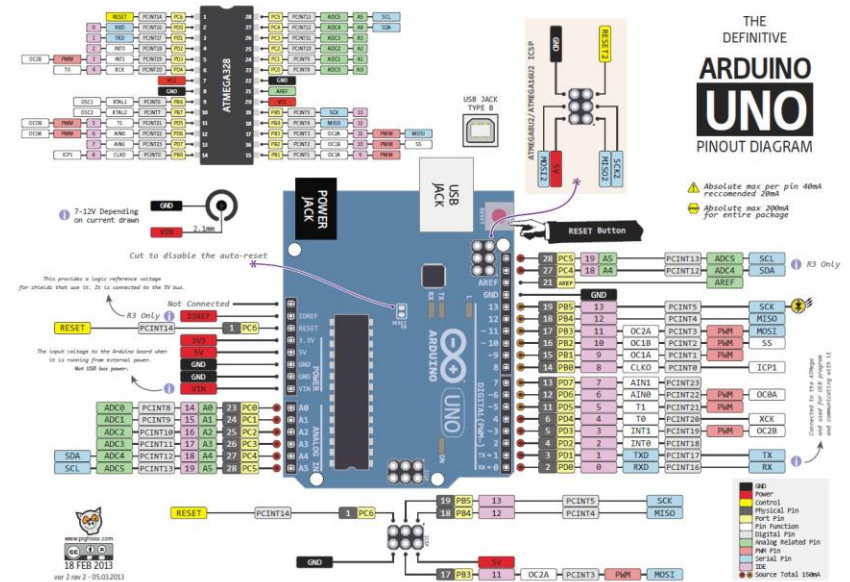
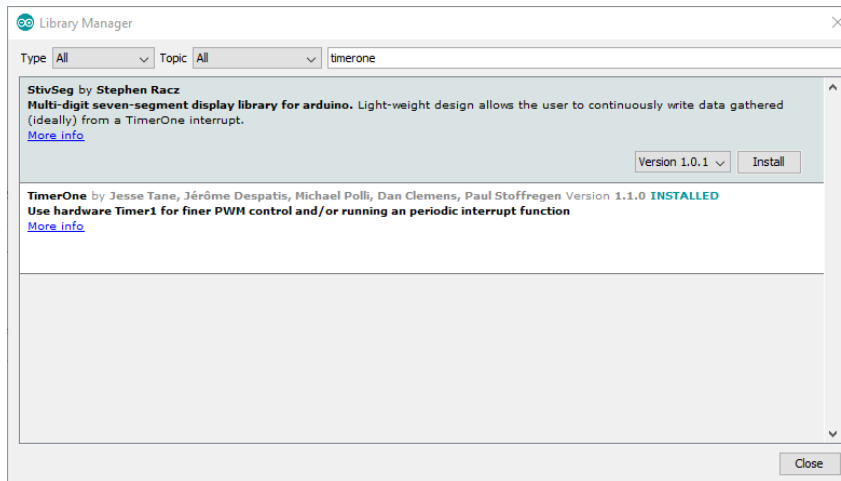
ATMEGA TIMER/COUNTER/PRESCALER CALCULATOR

Calculate ATmega Timer/Counter and prescaler values directly on this page.

Timer/Counter0 (8 bit) and Timer/Counter1 (16 bit)

Clocks		Prescalers		Registers								
$f_{\text{clock}} =$	<input type="text" value="10 M"/>	Prescaler	<input type="text" value="1"/> <input type="text" value="8"/> <input type="text" value="64"/> <input type="text" value="256"/> <input type="text" value="1024"/>	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Hz	$f_{\text{timer}} =$	Decimal	<input type="text"/>									OCR0
$f_{\text{timer}} =$	<input type="text" value="1 k"/>	Hexadec.	<input type="text"/>									
Hz		Error [%]	<input type="text"/>									
Factor	<input type="text" value="10e3"/>											
<input type="button" value="Calculate"/>												
		Green	- Timer/Counter0.	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
		Blue	- Timer/Counter1.	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
		Red	- Cannot be used.									OCR1AH
												OCR1AL
				OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK

Exercises prerequisites



Hands-on exercise: Timers and interrupts

- Tinkercad and Arduino continued ...
 - **Task 2:** Run the following program that configures Timer 0 to turn on the LED periodically without interruption:

```
void setup()
{
  DDRD |= 1 << PD6;

  TCCR0A = 0;
  TCCR0B = 0;
  TCNT0 = 0;

  TCCR0A |= 1 << COM0A0;
  TCCR0A |= 1 << WGM01;

  OCR0A = 0x80;
  TCCR0B |= 3 << CS00;
}

void loop()
{
  // your program here
}
```

Hands-on exercise: Timers and interrupts

- Arduino specific exercises:
 - **Task 3:** Set up a Timer using the Arduino TimerOne library so that you turn on an LED once a second

```
#include "TimerOne.h"

void setup()
{
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);      // initialize timer1, and set a 1/2 second period
  Timer1.attachInterrupt(callback); // attaches callback() as a timer overflow interrupt
}

void callback()
{
  // toggle pin
}

void loop()
{
  // your program here
}
```

- **Task 4:** Use the tone () library function to generate sound on a buzzer connected to pin 8. Find an example in the Arduino IDE at file / examples / digital.

Closing remarks

- C vs C++
- Arduino interrupts
- Volatile
- Reading resistors value
- sei()
- cli()
- reti()
- micros()
- attachInterrupt()
- Timers
- Prescaler
- TimerOne
- tone