Embedded Programming for IoT and Robotics

SPb ETU «LETI» MOEVM

Kirill Krinkin

Jul, 2017

School Topics

- Introduction into Embedded programming
- Introduction into Linux System and Kernel Programming
- Robot Operating System I
- Robot Operating System II

Embedded Programming for IoT and Robotics

Part I. Introduction into Embedded programming

Module 1.1 Intro

- A few questions for participants
- Module overview
- MCU vs CPU
- Electrical engineering, recall
- Controller connecting
- Primitive I/O and

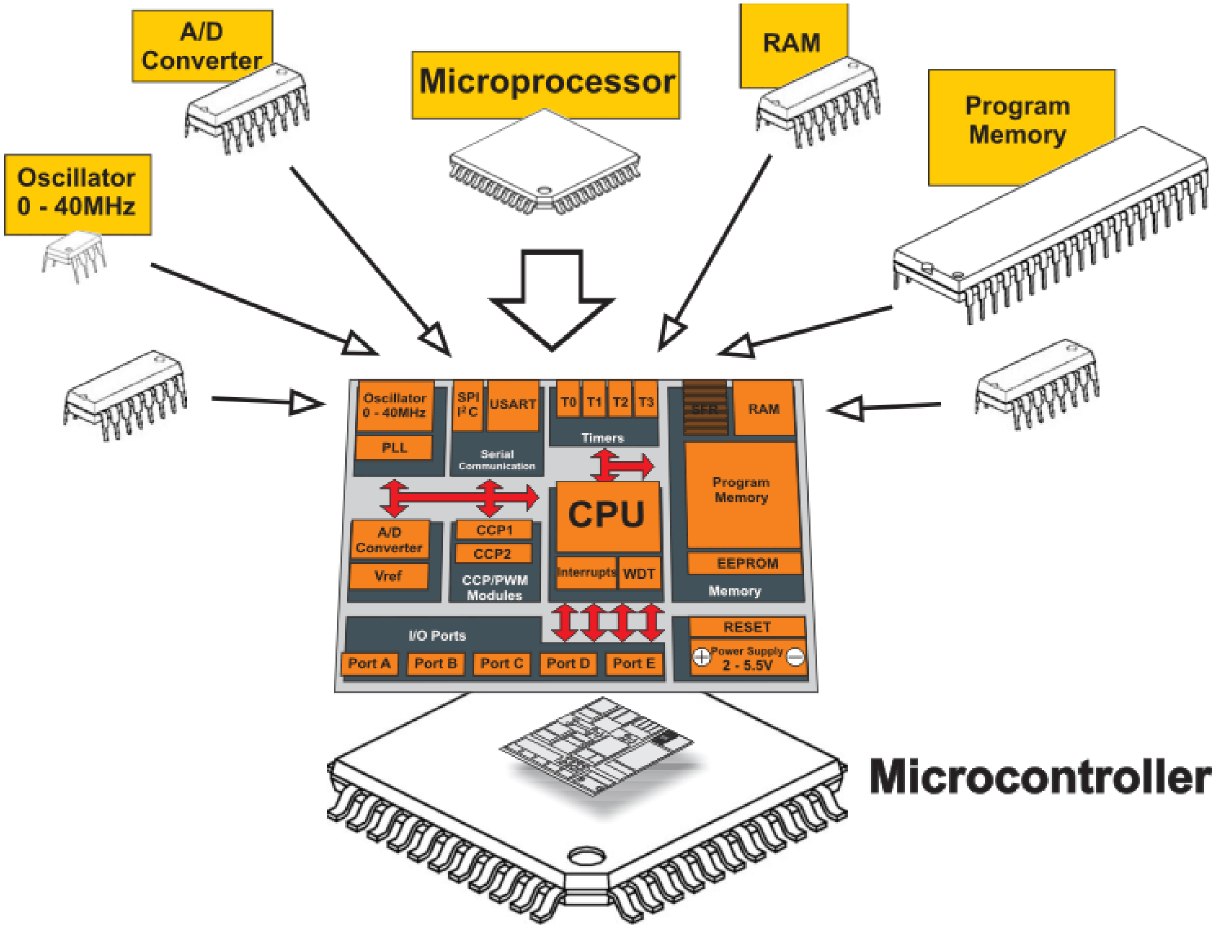
Questions

- Introduce yourself
- Why do you participate?
- Embedded experience?
- C/C++, asm development experience?
- AVR, Arduino other MCU experience?

Part.I Overview

- AVR, MSP-430* Architecture
- Peripheral interfaces
- Wiring[Arduino/Energia], asm programming
  - Input/Output
  - Sensors/actuators overview
  - Interrupts
  - Timers
  - Communication protocols
- Schedule
  - 10.00 – 12.00 Theoretical part
  - 13.00 – 15.00 Practical part

Micro controller Unit

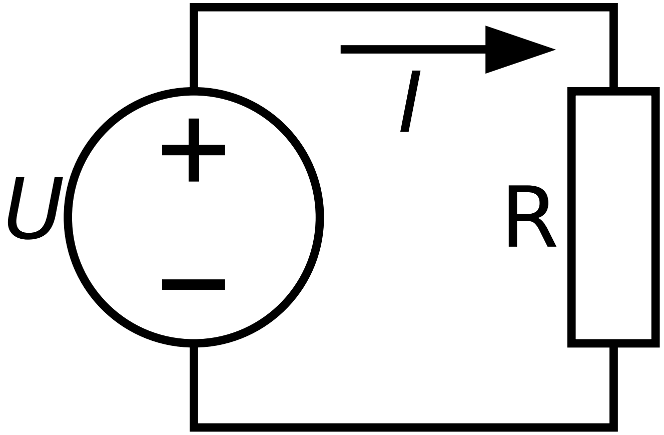


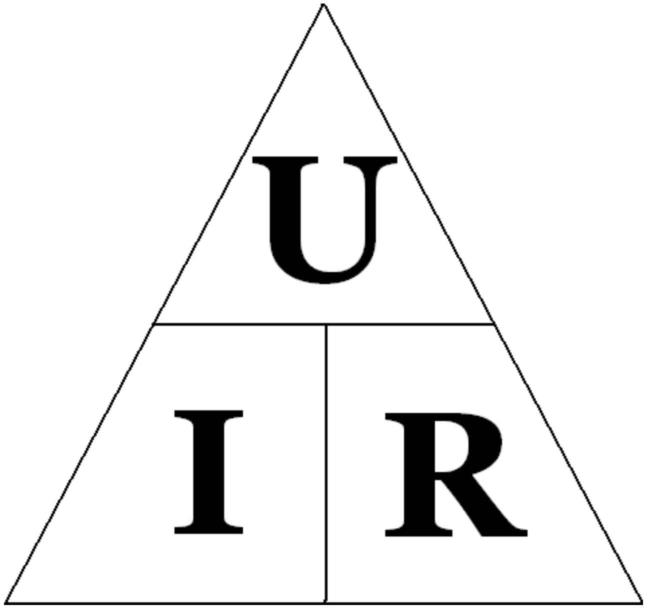
https://cdn.mikroe.com/img/legacy/publication/pic-books/programming-in-c/chapter/01/fig0-1.gif

MCU vs MPU (CPU)

	MCU	MPU
Cost	Cheap	Expensive
Speed	Slow (MHz)	Fast (GHz)
Purpose	Embedded	Computers
Dependency	Single chip with higher integration	External components (RAM, ROM, I/O, etc)

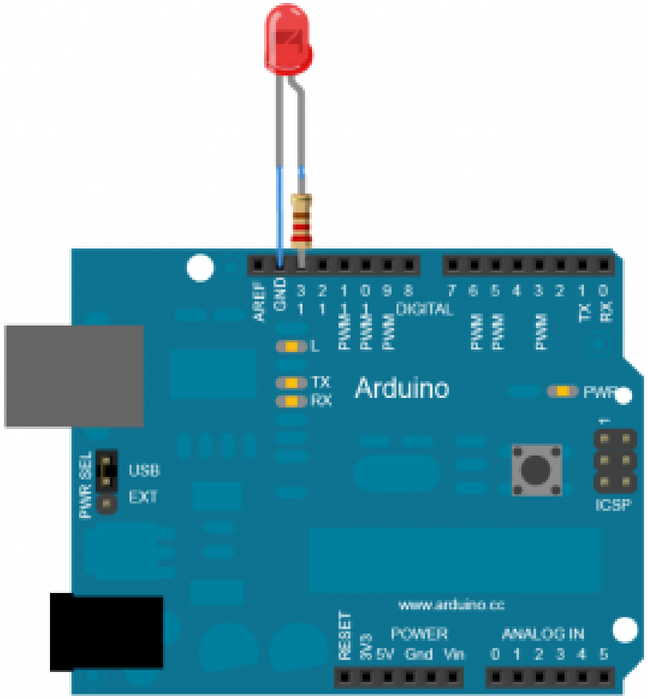
Ohm`s law, and consequences

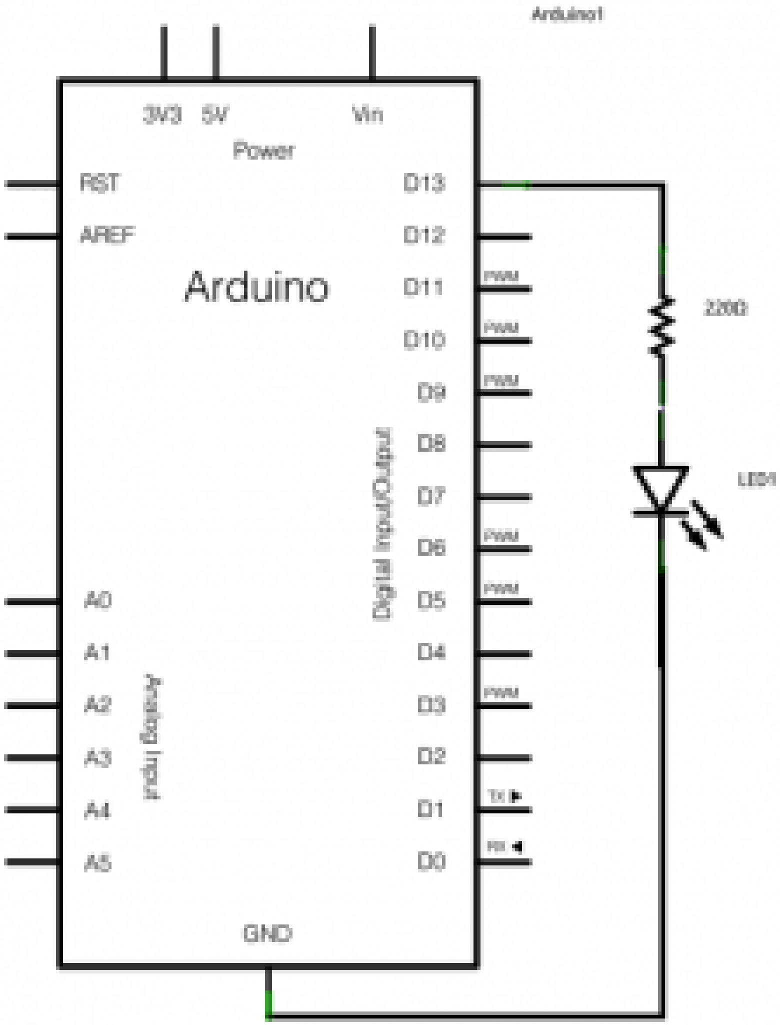




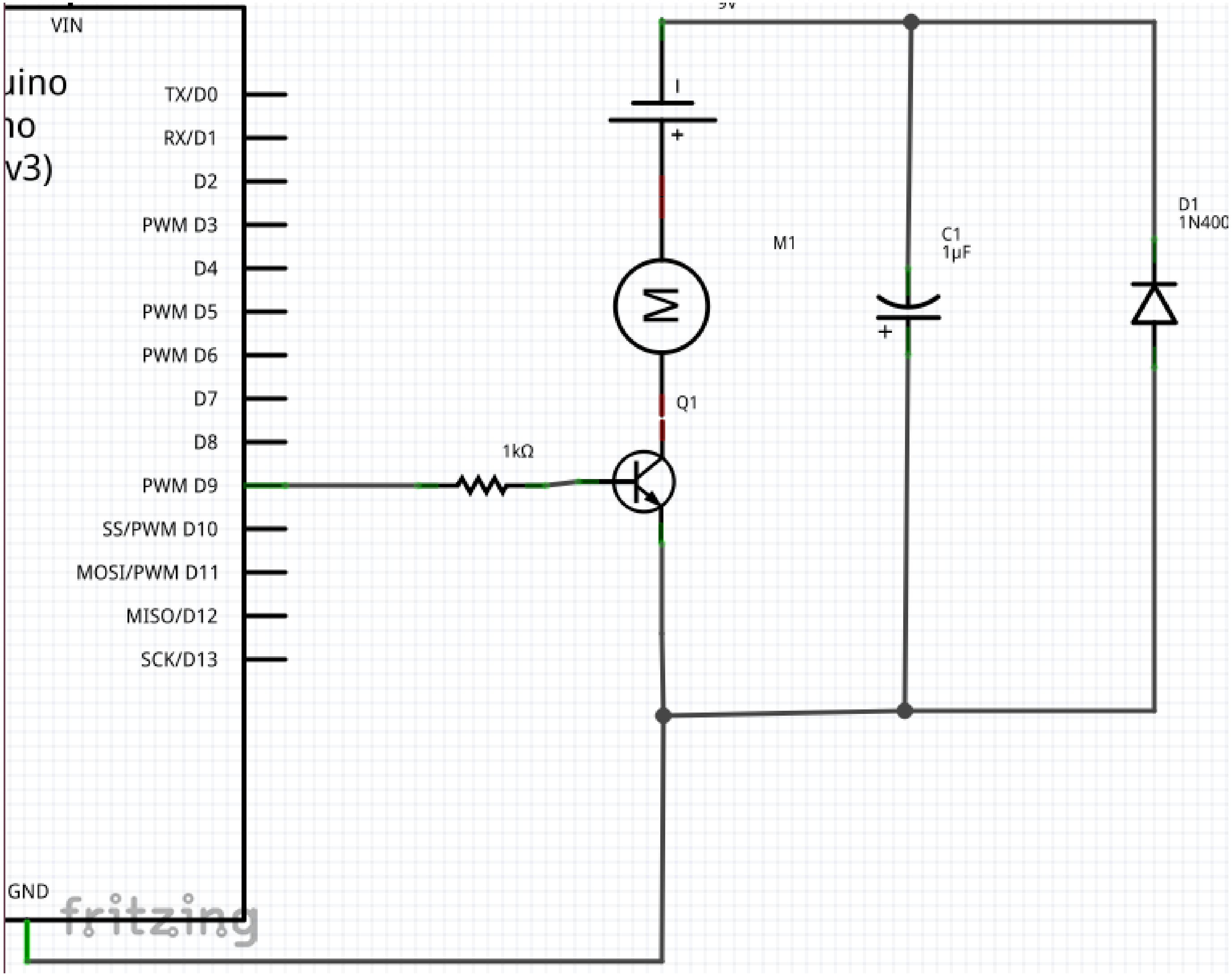
MCU Connecting (1)

- Ohm`s law
- Grounding
- No free contacts

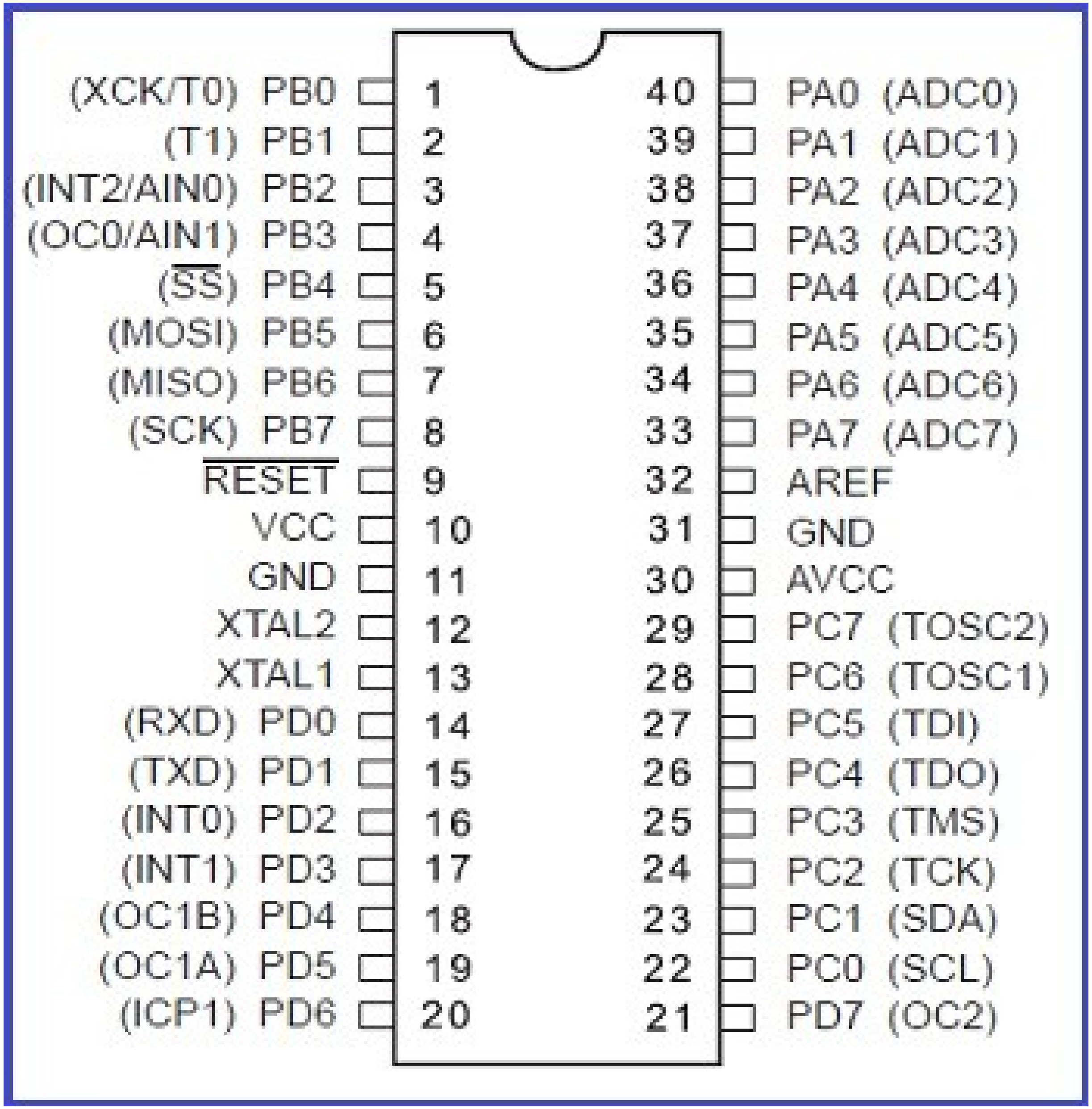




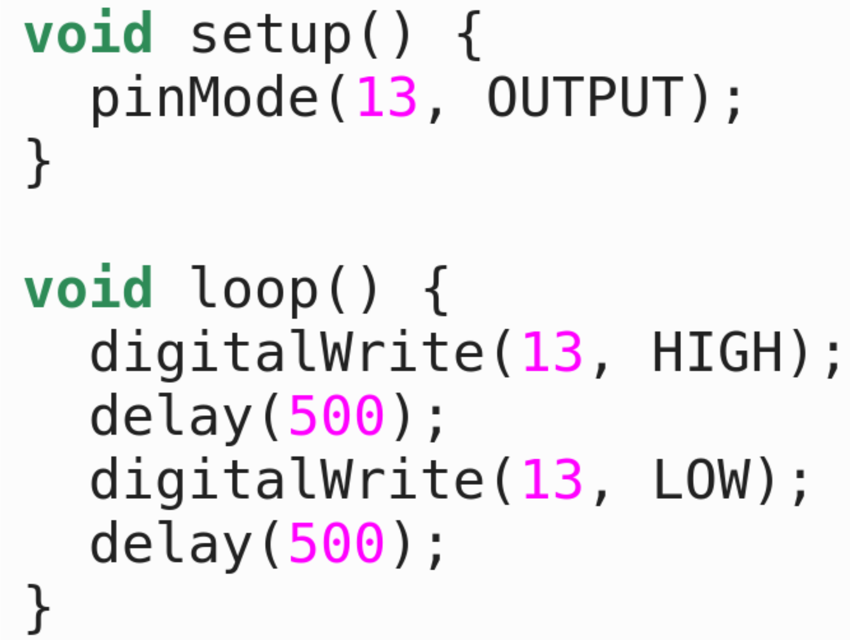
MCU Connecting (2)



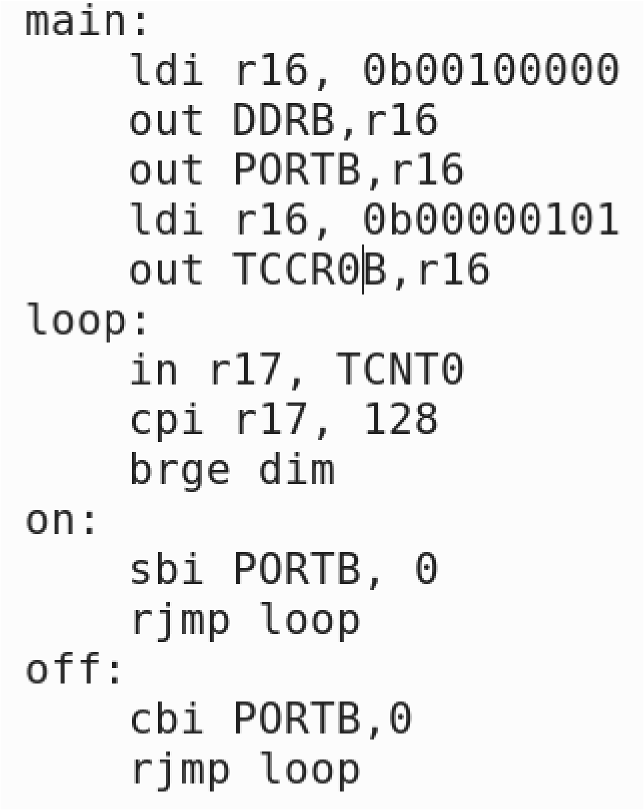
Trivial I/O (1)


(XCK/T0) PB0 1
(T1) PB1 2
(INT2/AIN0) PB2 3
(OC0/AIN1) PB3 4
(SS) PB4 5
(MOSI) PB5 6
(MISO) PB6 7
(SCK) PB7 8
RESET 9
VCC 10
GND 11
XTAL2 12
XTAL1 13
(RXD) PD0 14
(TXD) PD1 15
(INT0) PD2 16
(INT1) PD3 17
(OC1B) PD4 18
(OC1A) PD5 19
(ICP1) PD6 20
40 PA0 (ADC0)
39 PA1 (ADC1)
38 PA2 (ADC2)
37 PA3 (ADC3)
36 PA4 (ADC4)
35 PA5 (ADC5)
34 PA6 (ADC6)
33 PA7 (ADC7)
32 AREF
31 GND
30 AVCC
29 PC7 (TOSC2)
28 PC6 (TOSC1)
27 PC5 (TDI)
26 PC4 (TDO)
25 PC3 (TMS)
24 PC2 (TCK)
23 PC1 (SDA)
22 PC0 (SCL)
21 PD7 (OC2)

Trivial I/O (2)



```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}
```



```
main:
    ldi r16, 0b00100000
    out DDRB,r16
    out PORTB,r16
    ldi r16, 0b00000101
    out TCCR0B,r16
loop:
    in r17, TCNT0
    cpi r17, 128
    brge dim
on:
    sbi PORTB, 0
    rjmp loop
off:
    cbi PORTB,0
    rjmp loop
```

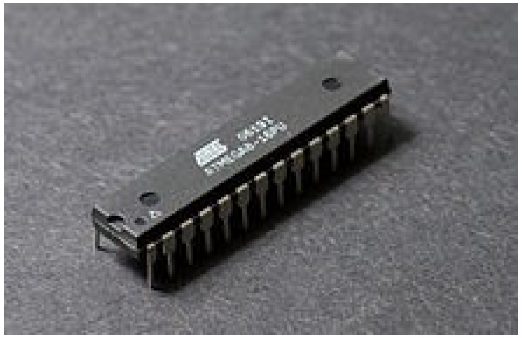
Arduino

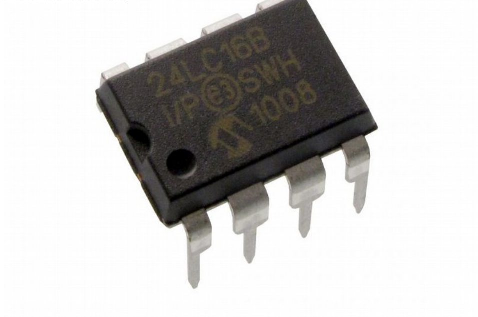
In this module

- AVR MCU
- Arduino boards
- Wiring и Arduino IDE overview
- Simple Circuits
- I/O programming
- Peripheral devices overview

AVR

- AVR is a family of microcontrollers developed by Atmel beginning in 1996
- modified Harvard architecture
- 8-bit RISC single-chip microcontrollers.

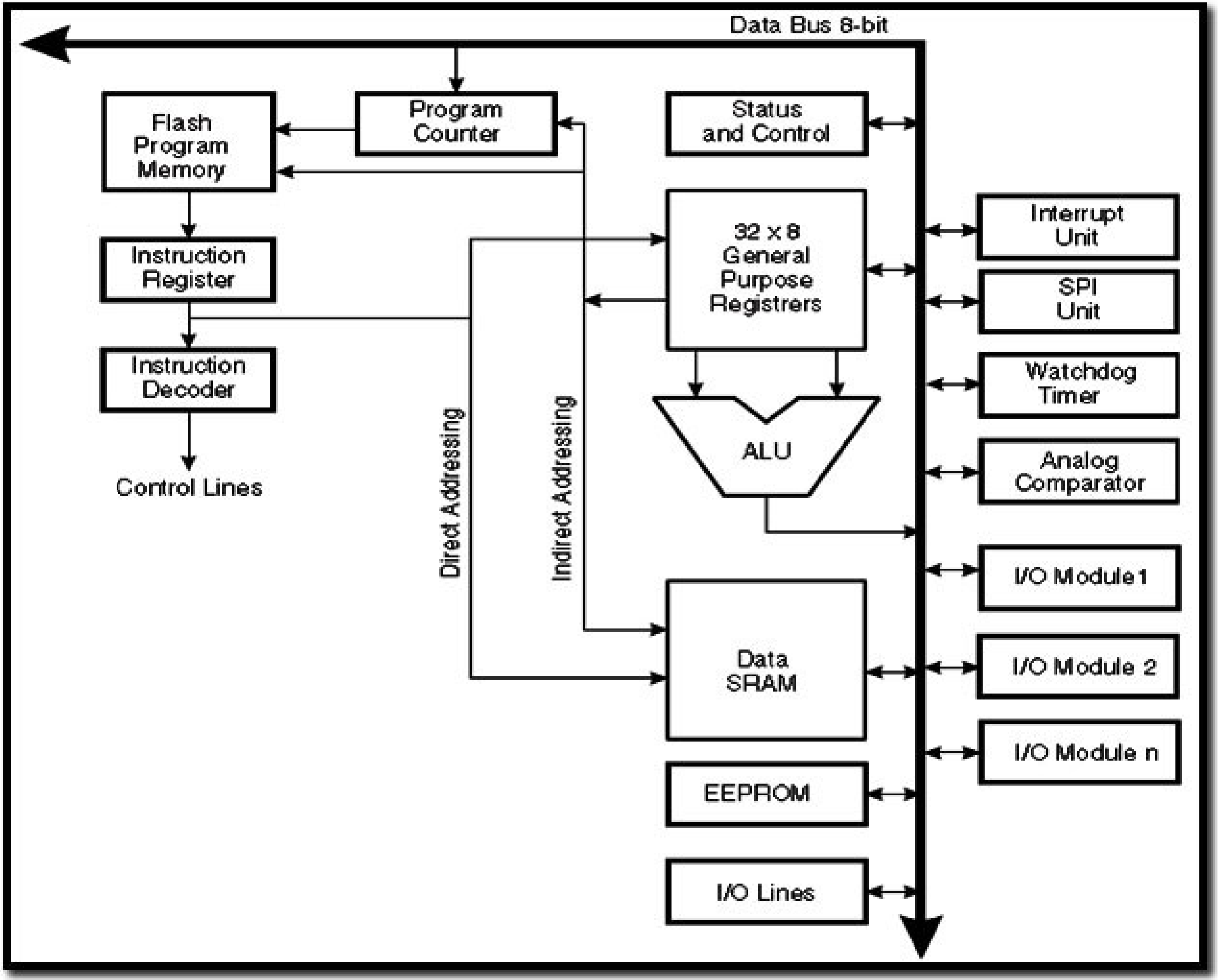




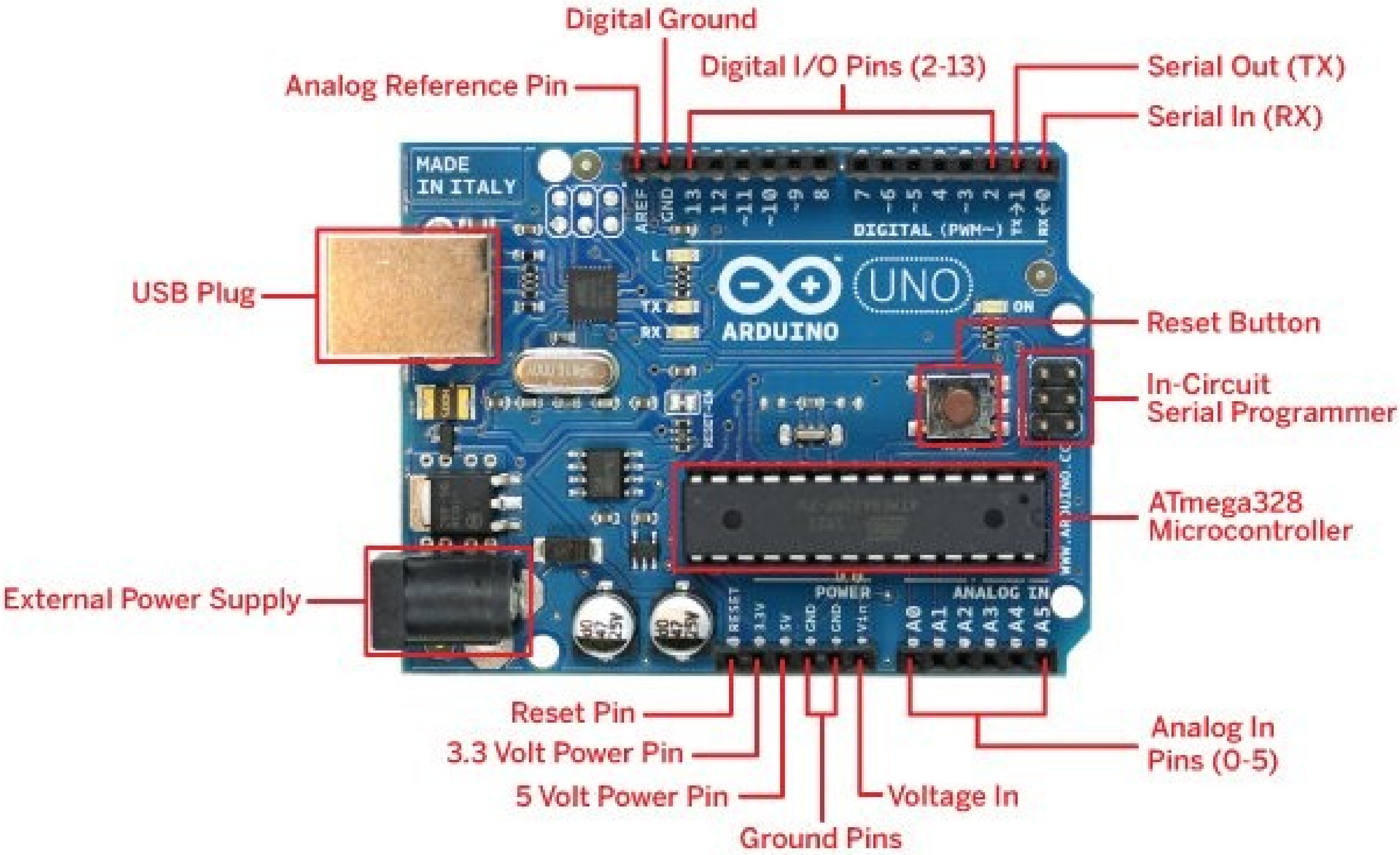
AVR

WIKIPEDIA
The Free Encyclopedia

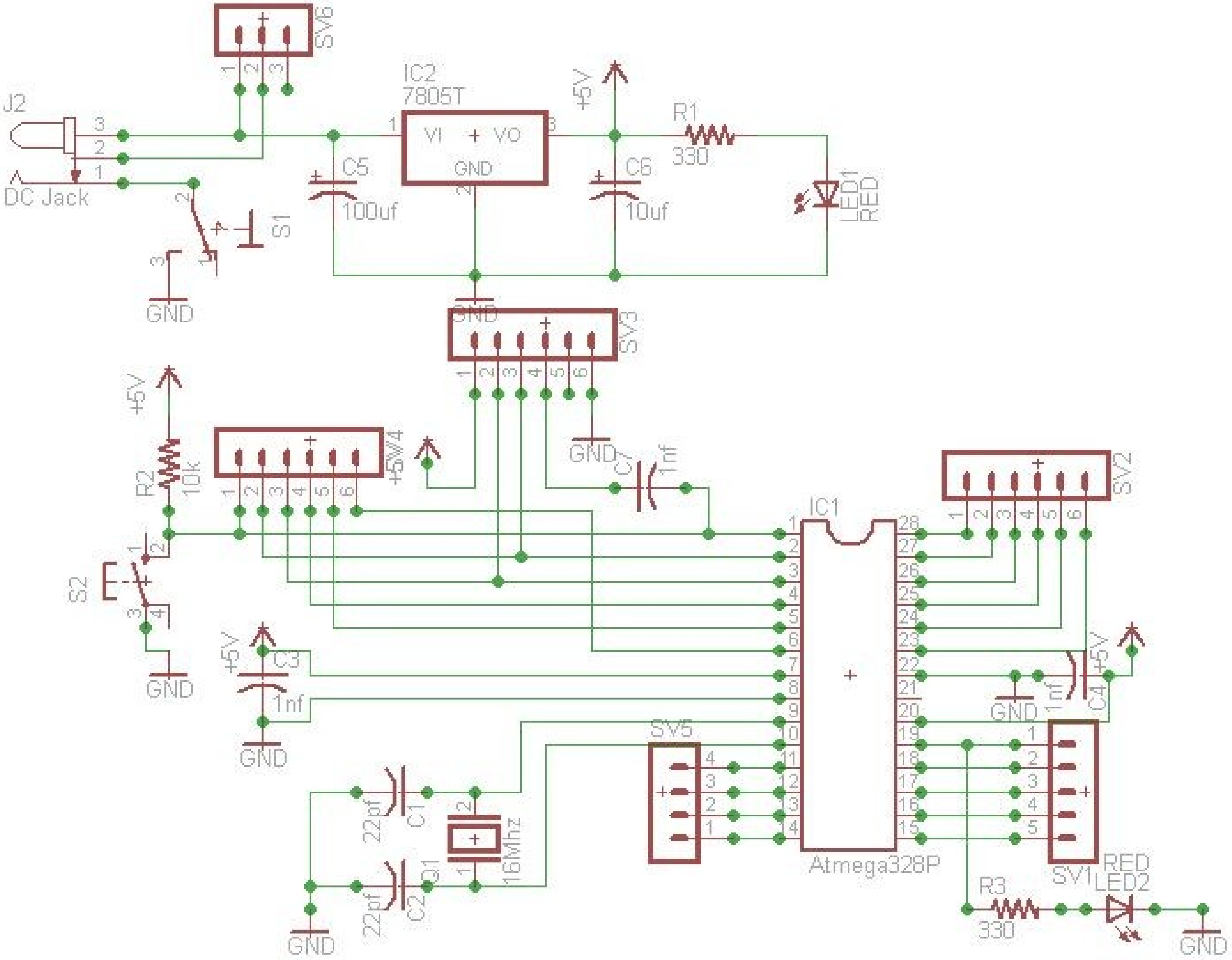
AVR Architecture



Arduino

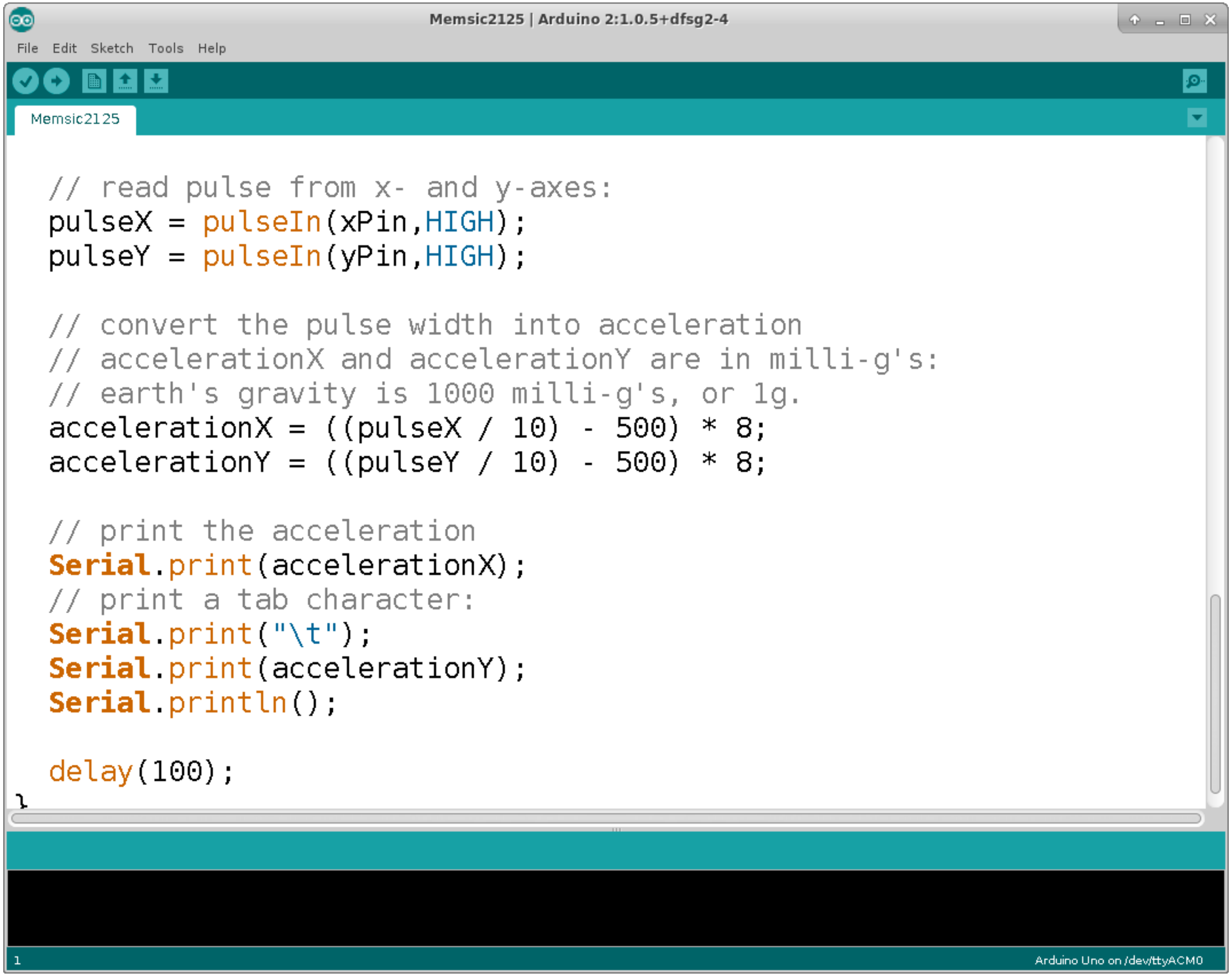

Digital Ground
Analog Reference Pin
Digital I/O Pins (2-13)
Serial Out (TX)
Serial In (RX)
USB Plug
Reset Button
In-Circuit Serial Programmer
ATmega328 Microcontroller
External Power Supply
Reset Pin
3.3 Volt Power Pin
5 Volt Power Pin
Ground Pins
Voltage In
Analog In Pins (0-5)
MADE IN ITALY
ARDUINO
UNO
DIGITAL (PWM~)
POWER
ANALOG IN

Arduino



SV6
IC2
7805T
VI
VO
GND
+5V
R1
330
J2
DC Jack
C5
100uf
C6
10uf
LED1
RED
S1
GND
SV3
+5V
R2
10k
SV4
C7
1nf
SV2
IC1
S2
C3
1nf
C4
SV5
C1
C2
22pf
Q1
16Mhz
Atmega328P
SV1
RED
LED2
R3
330

Wiring, Arduino IDE

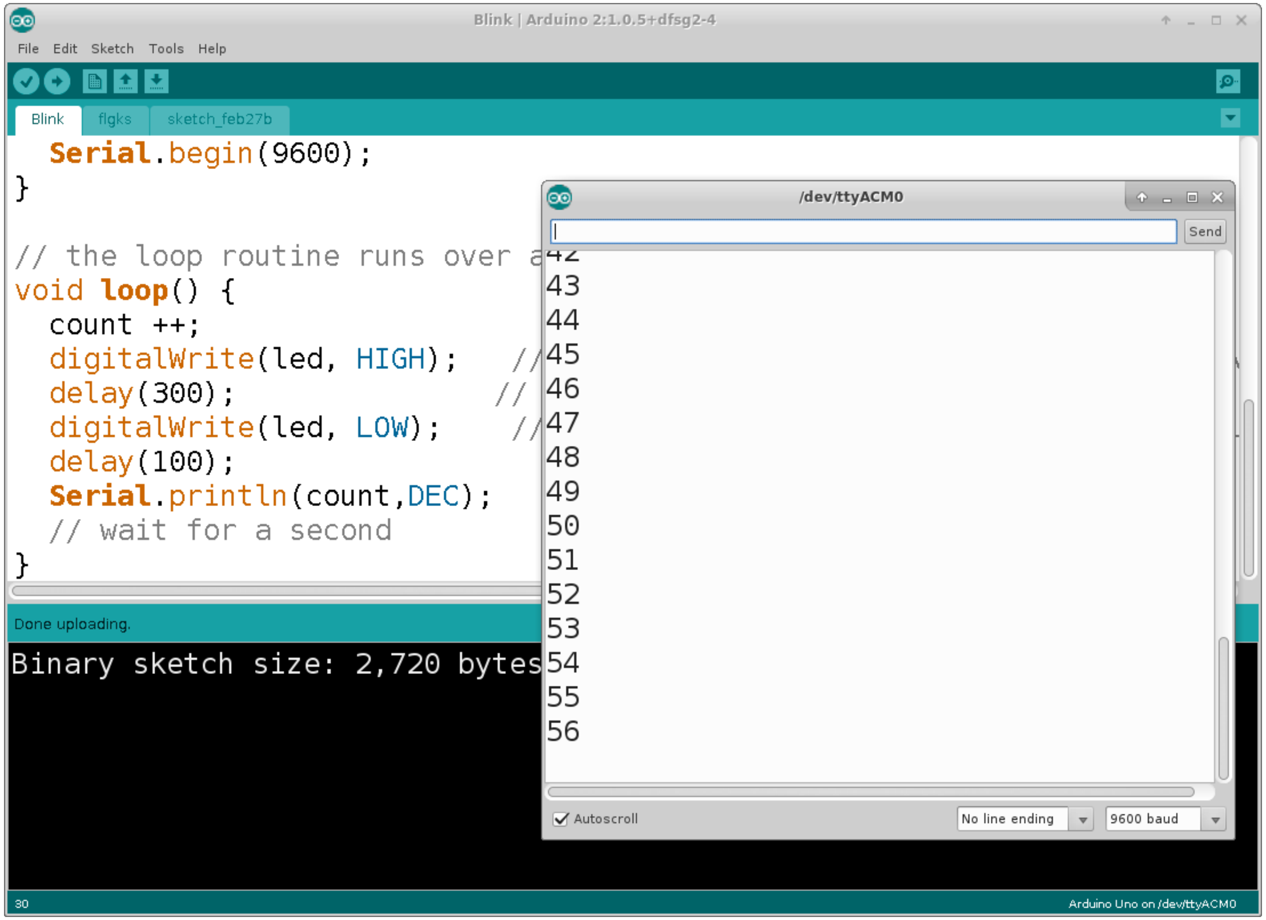


Memsic2125 | Arduino 2:1.0.5+dfsg2-4
File Edit Sketch Tools Help
Memsic2125

```
  // read pulse from x- and y-axes:
  pulseX = pulseIn(xPin,HIGH);
  pulseY = pulseIn(yPin,HIGH);

  // convert the pulse width into acceleration
  // accelerationX and accelerationY are in milli-g's:
  // earth's gravity is 1000 milli-g's, or 1g.
  accelerationX = ((pulseX / 10) - 500) * 8;
  accelerationY = ((pulseY / 10) - 500) * 8;

  // print the acceleration
  Serial.print(accelerationX);
  // print a tab character:
  Serial.print("\t");
  Serial.print(accelerationY);
  Serial.println();

  delay(100);
}
```

Terminology

- Sketches (programms)
- Libraries
- Boards
- Serial Monitor



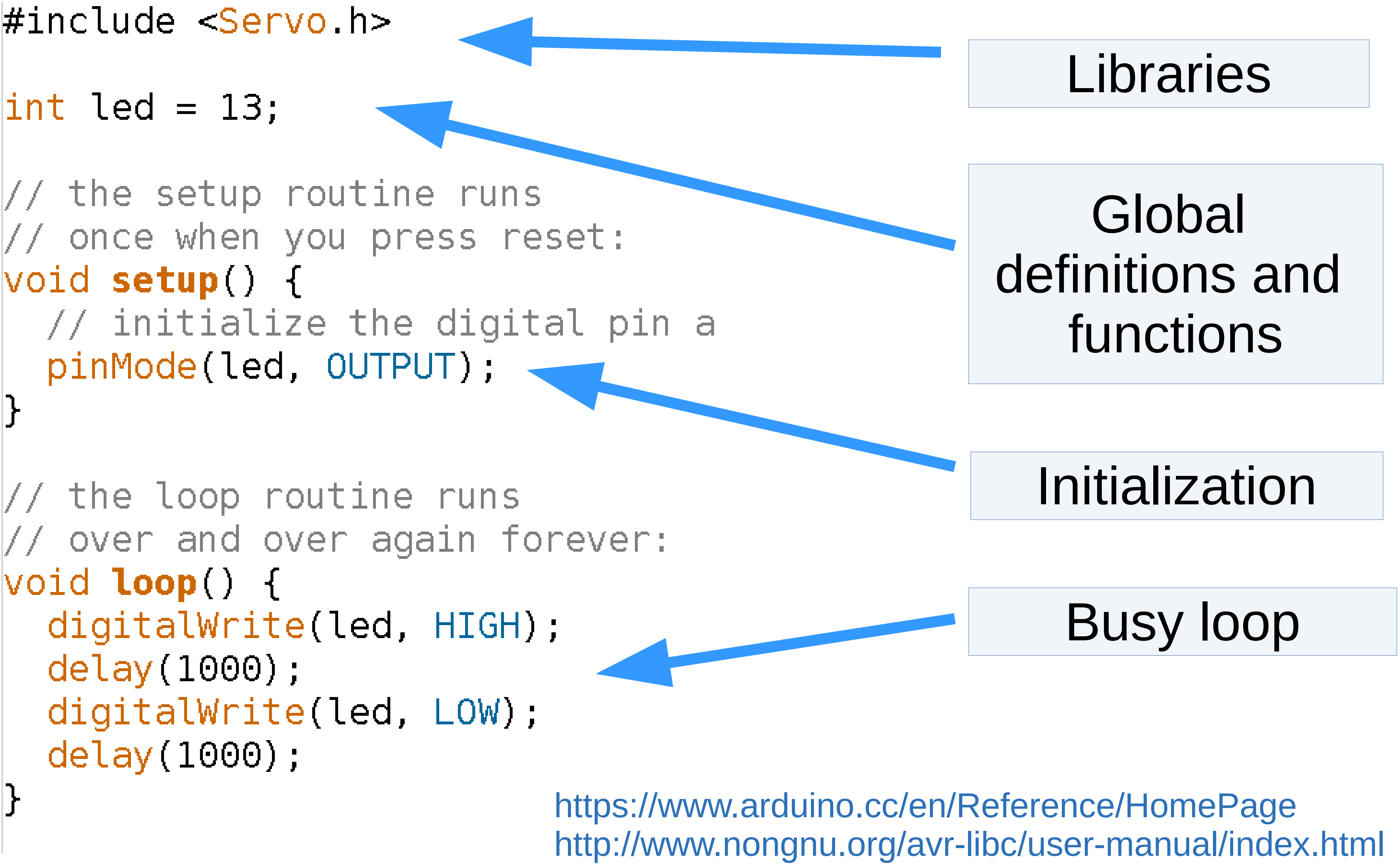
```
  Serial.begin(9600);
}

// the loop routine runs over a
void loop() {
  count ++;
  digitalWrite(led, HIGH);   //
  delay(300);               //
  digitalWrite(led, LOW);    //
  delay(100);
  Serial.println(count,DEC);
  // wait for a second
}
```

Done uploading.

```
Binary sketch size: 2,720 bytes
```

```
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
```

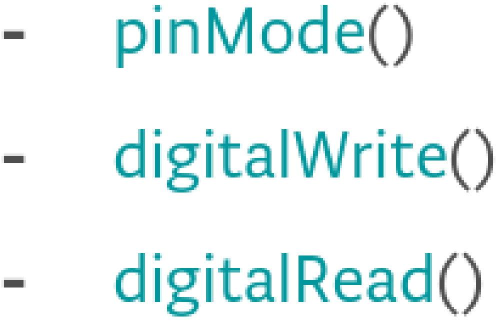
Programm structure



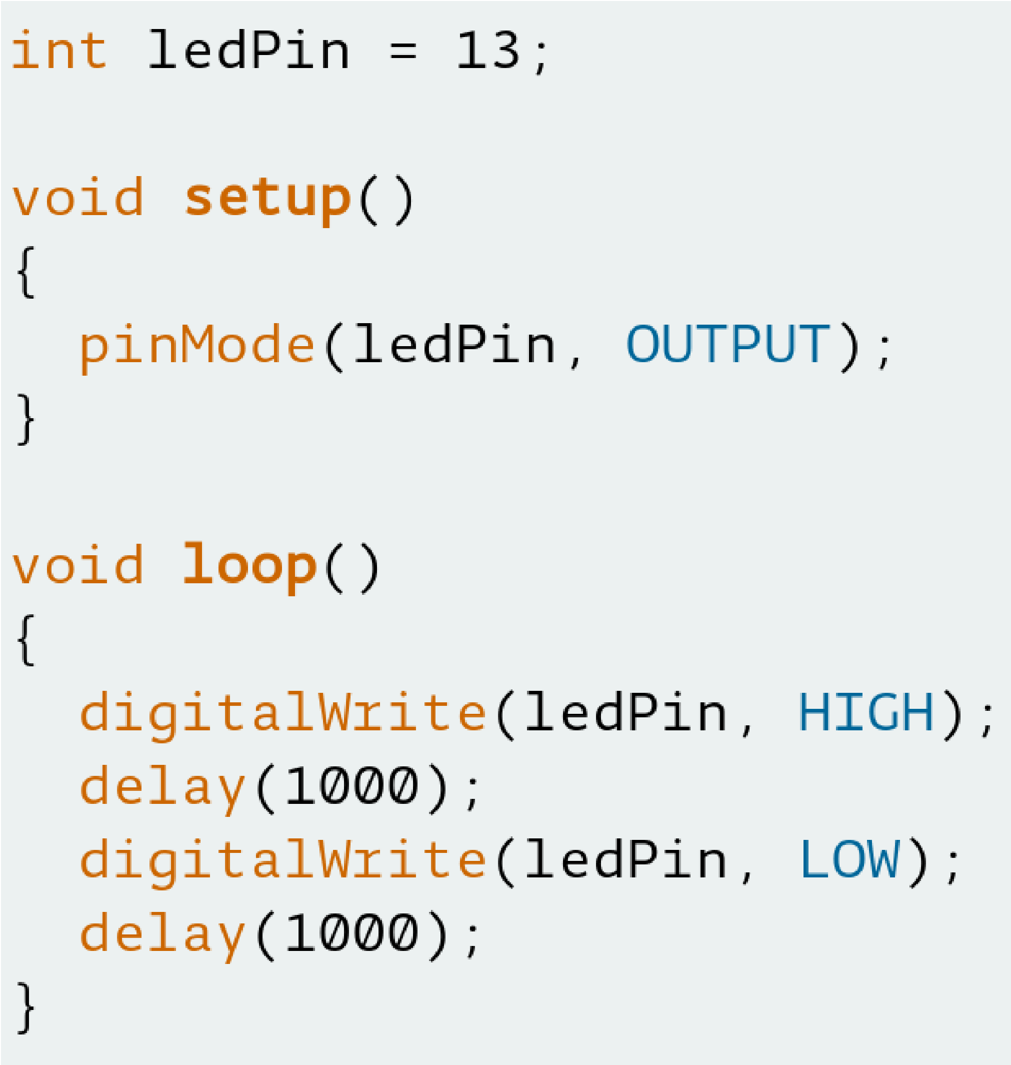
```
#include <Servo.h>

int led = 13;

// the setup routine runs
// once when you press reset:
void setup() {
  // initialize the digital pin a
  pinMode(led, OUTPUT);
}

// the loop routine runs
// over and over again forever:
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

https://www.arduino.cc/en/Reference/HomePage
http://www.nongnu.org/avr-libc/user-manual/index.html

Digital I/O



- pinMode()
- digitalWrite()
- digitalRead()

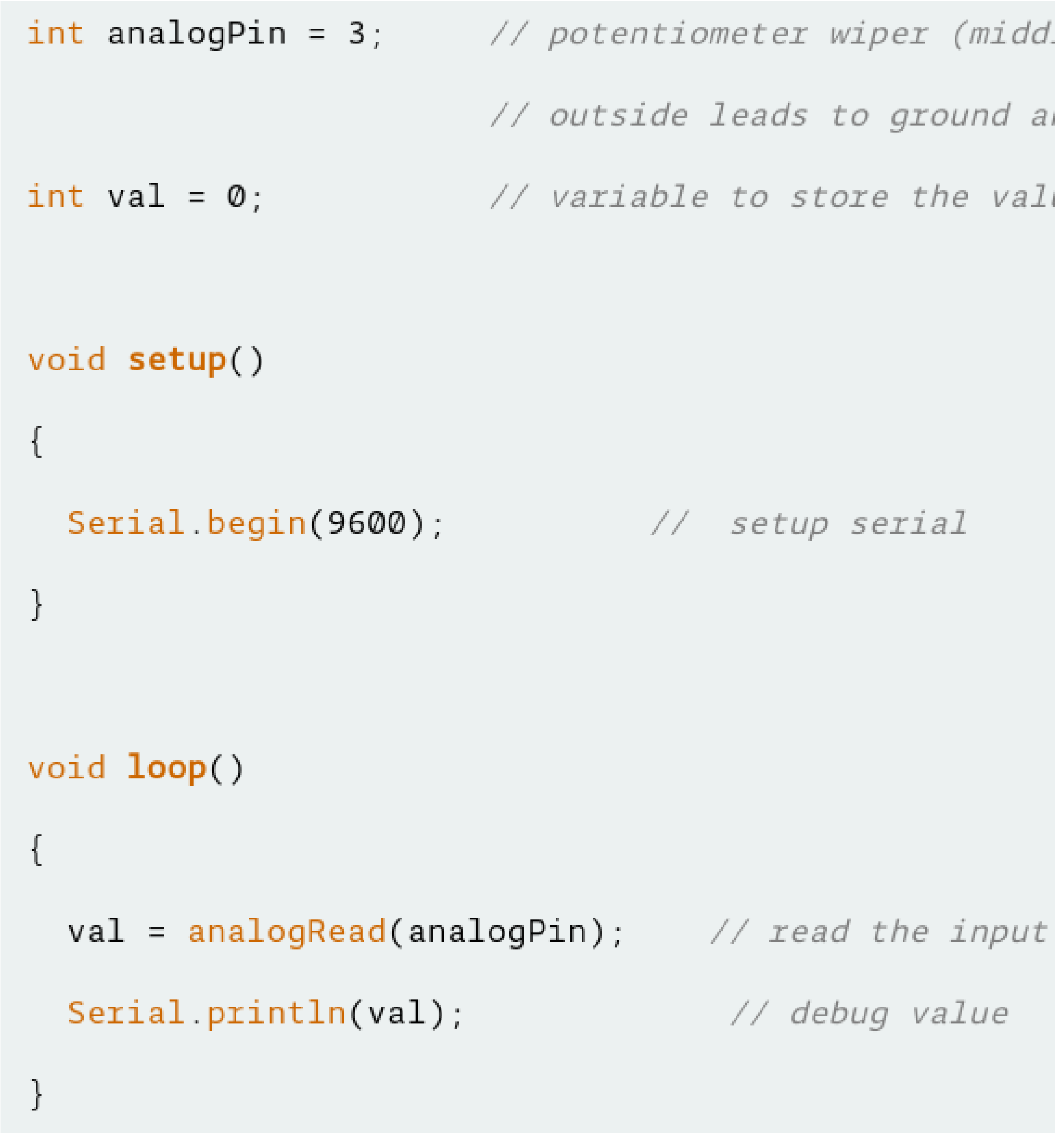


```
int ledPin = 13;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Analog I/O

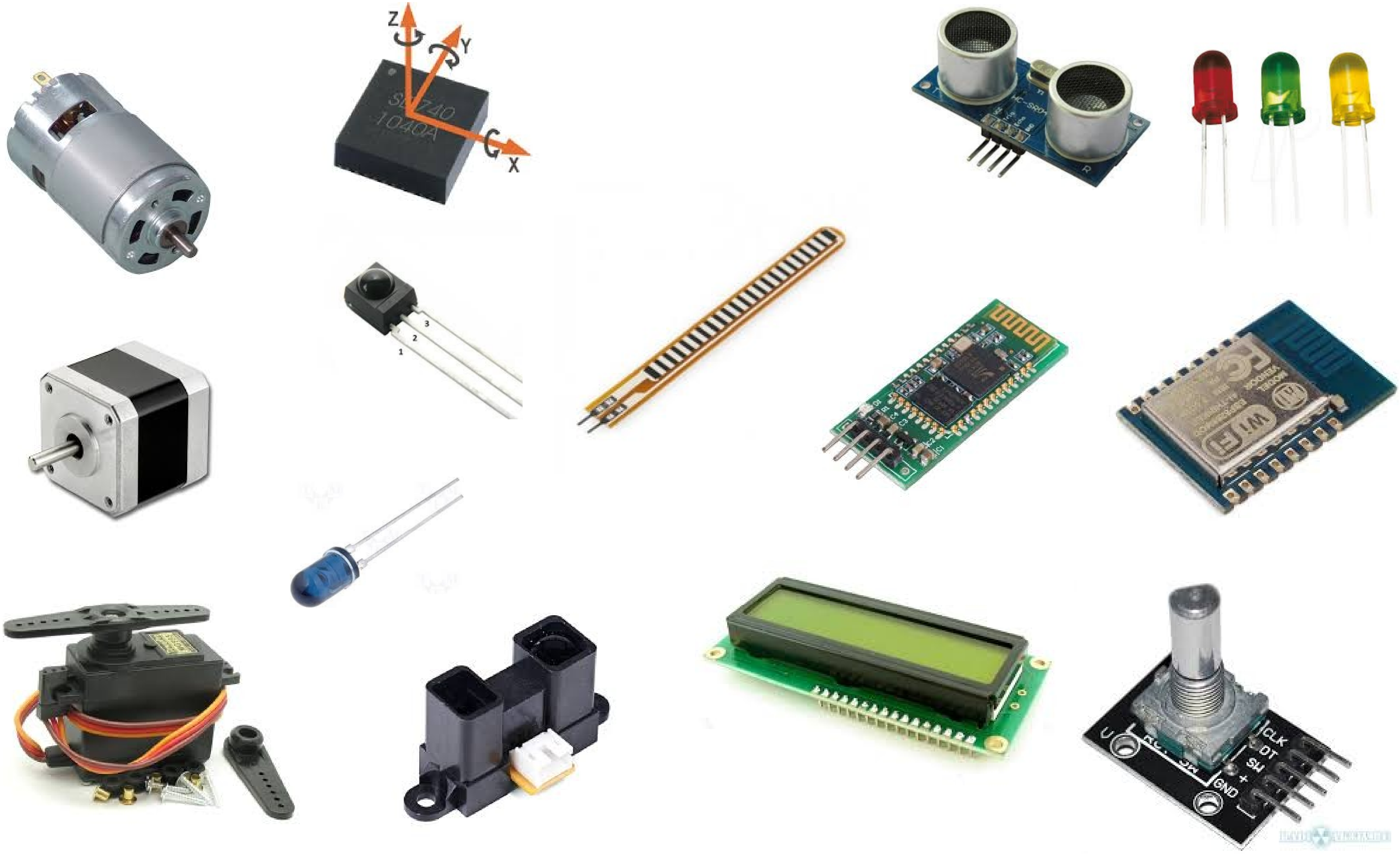
- analogReference()
- analogRead()
- analogWrite() - *PWM*

- A/D Converter 10bit
- Frequency <=10KHz

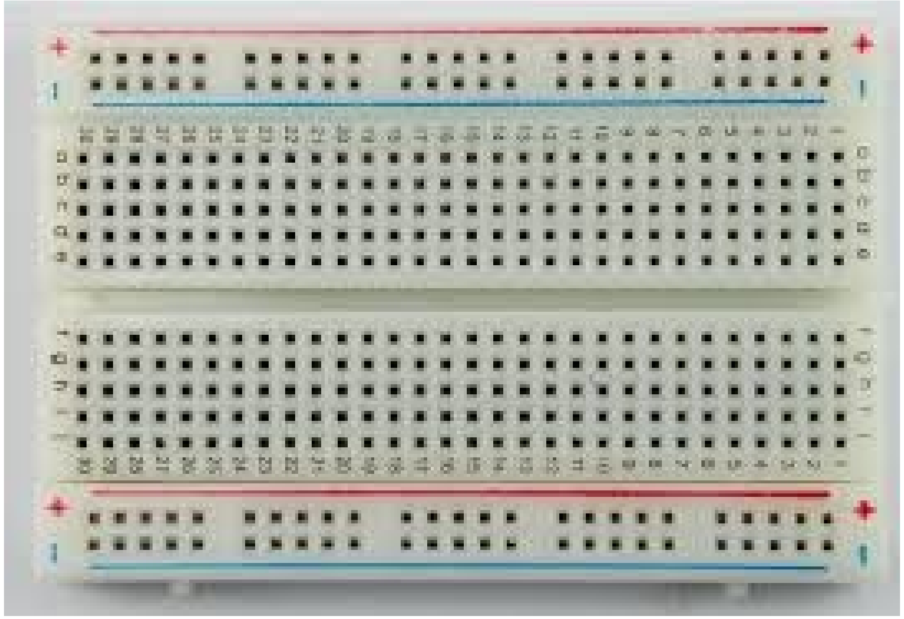


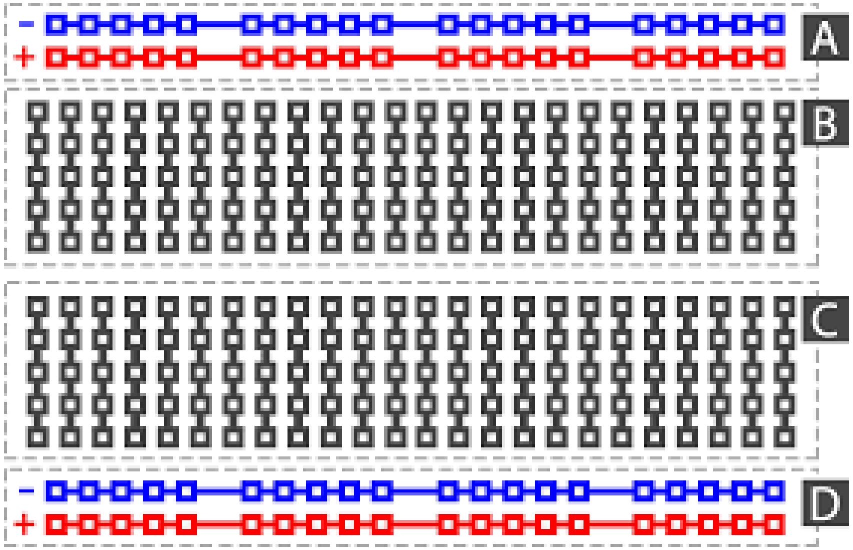
```
int analogPin = 3;     // potentiometer wiper (midd
                       // outside leads to ground a
int val = 0;           // variable to store the val


void setup()
{
  Serial.begin(9600);          //  setup serial
}


void loop()
{
  val = analogRead(analogPin);    // read the input
  Serial.println(val);             // debug value
}
```

Peripheral devices overview

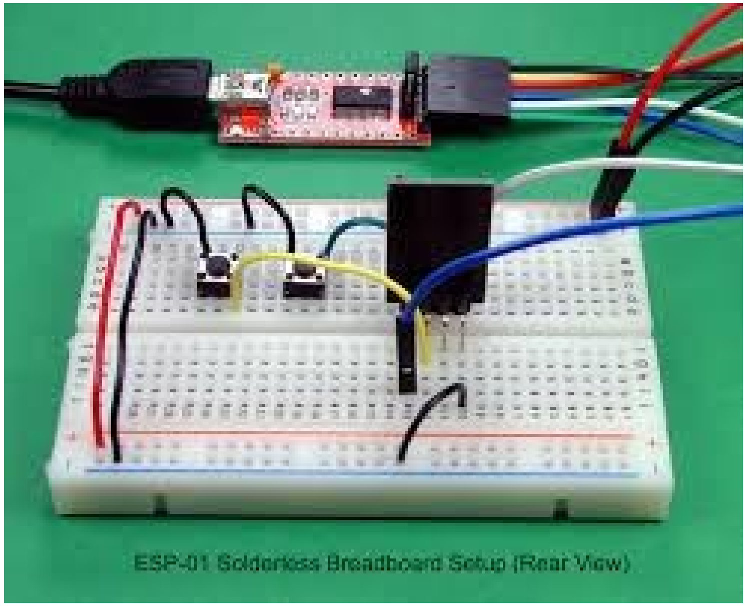


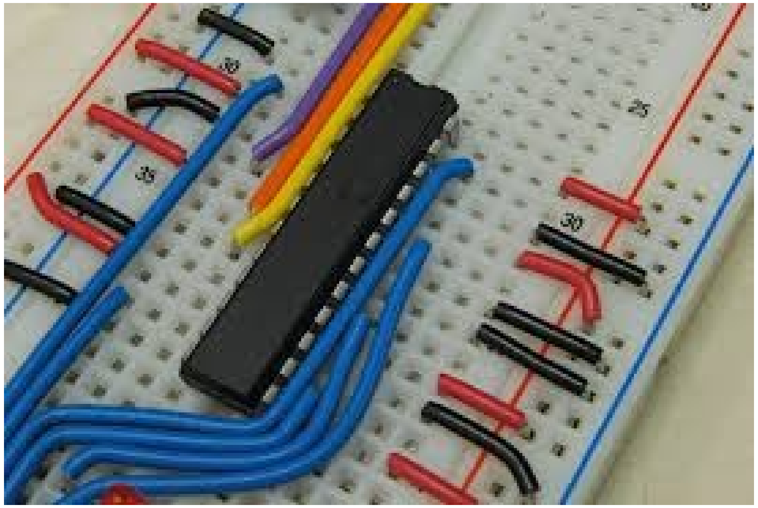
Breadboards

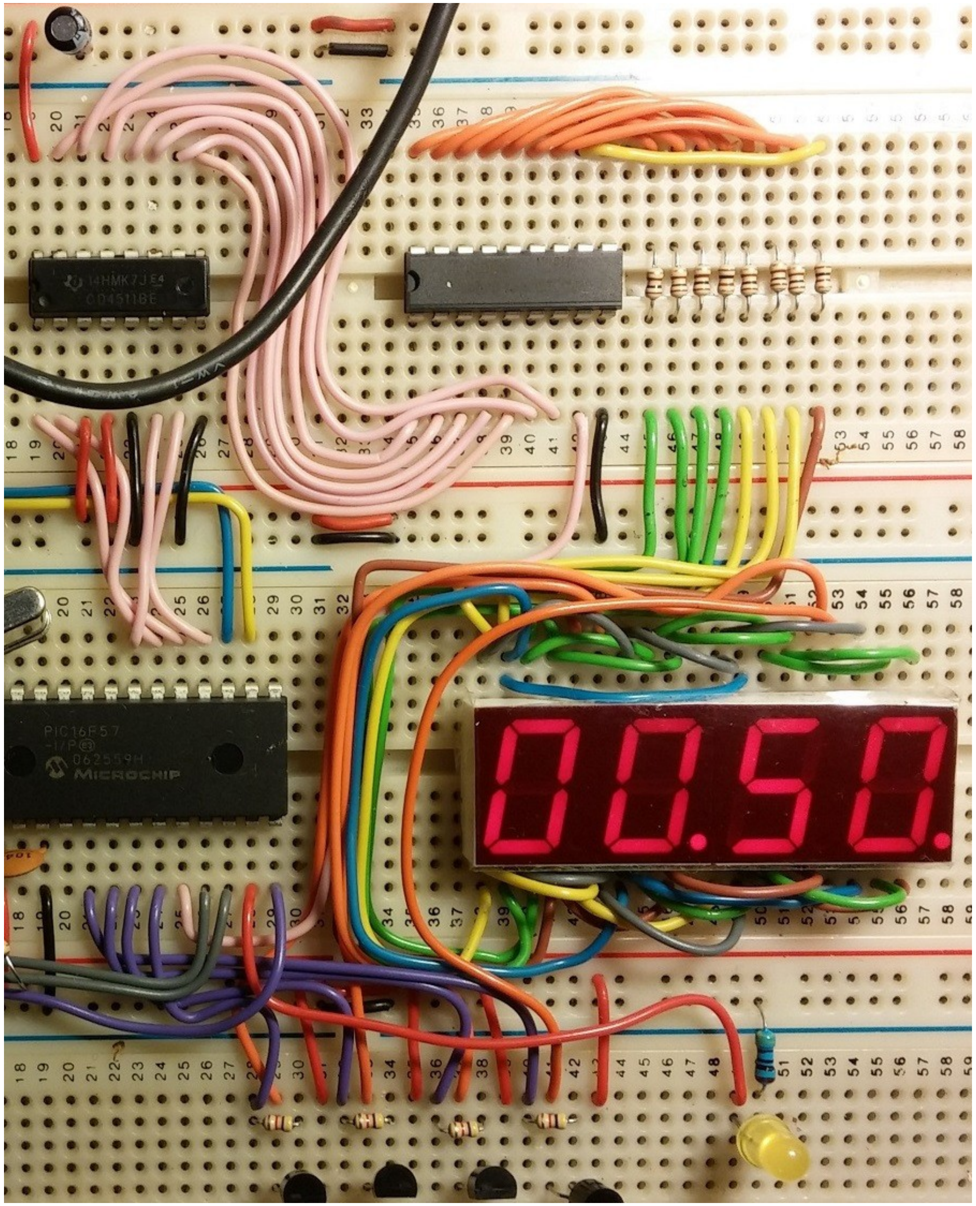




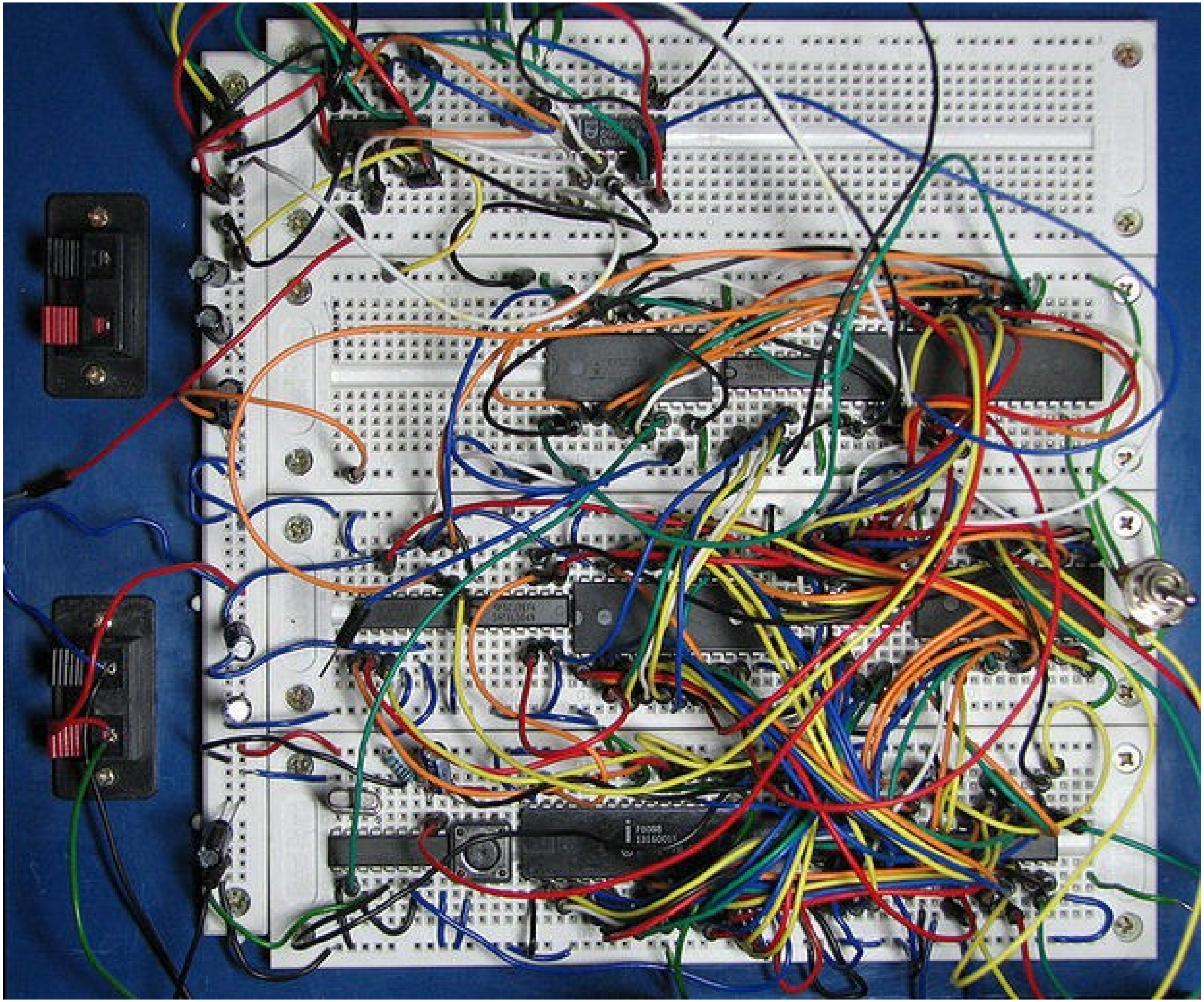
Breadboards







Breadboards



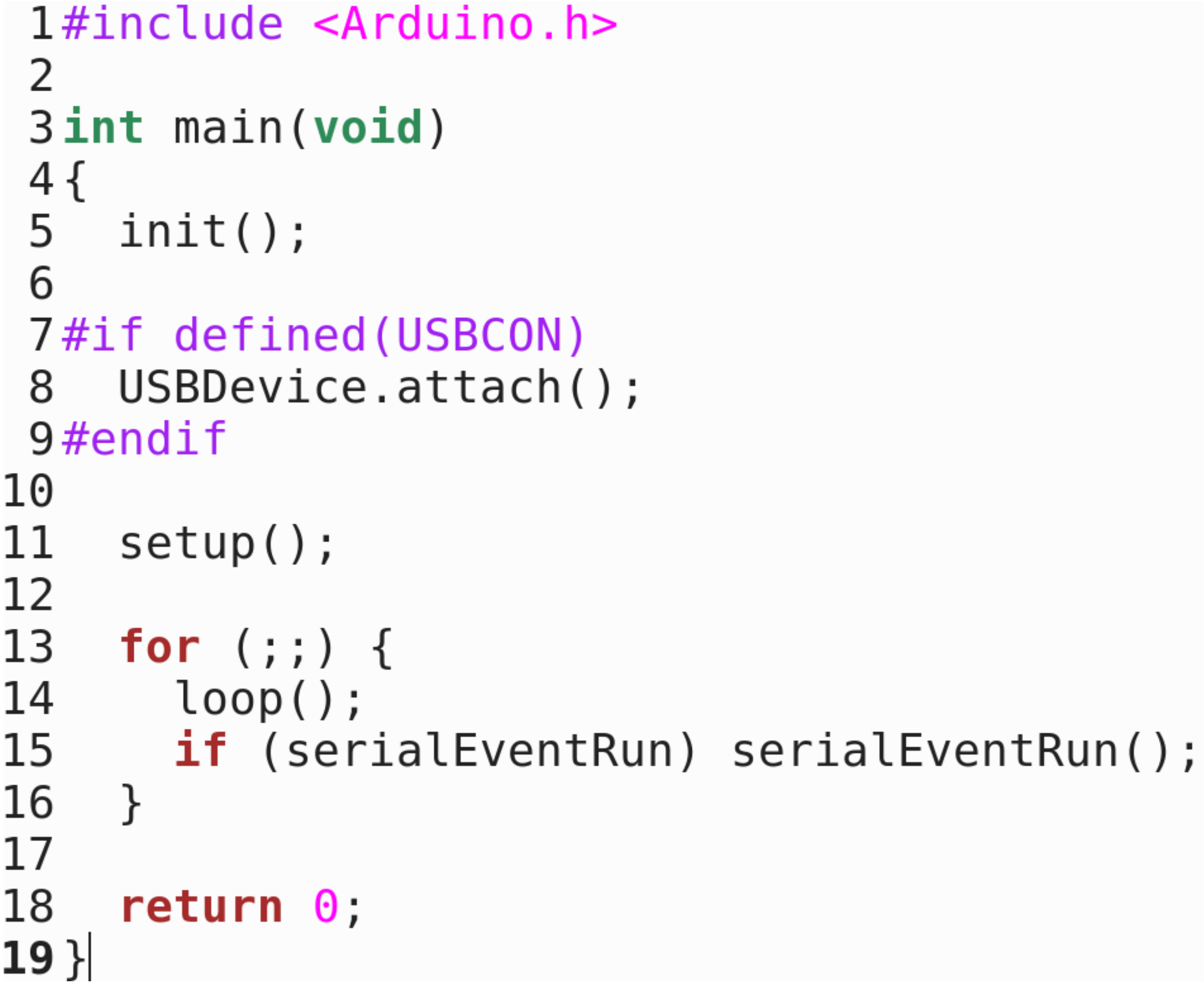
AVR toolchain. AVR-Libc. Bootloader. Timers. Interrupts.

Module overview

- Arduino IDE, under the hood
- Fuses
- Bootloader
- Timers
- Interrupts

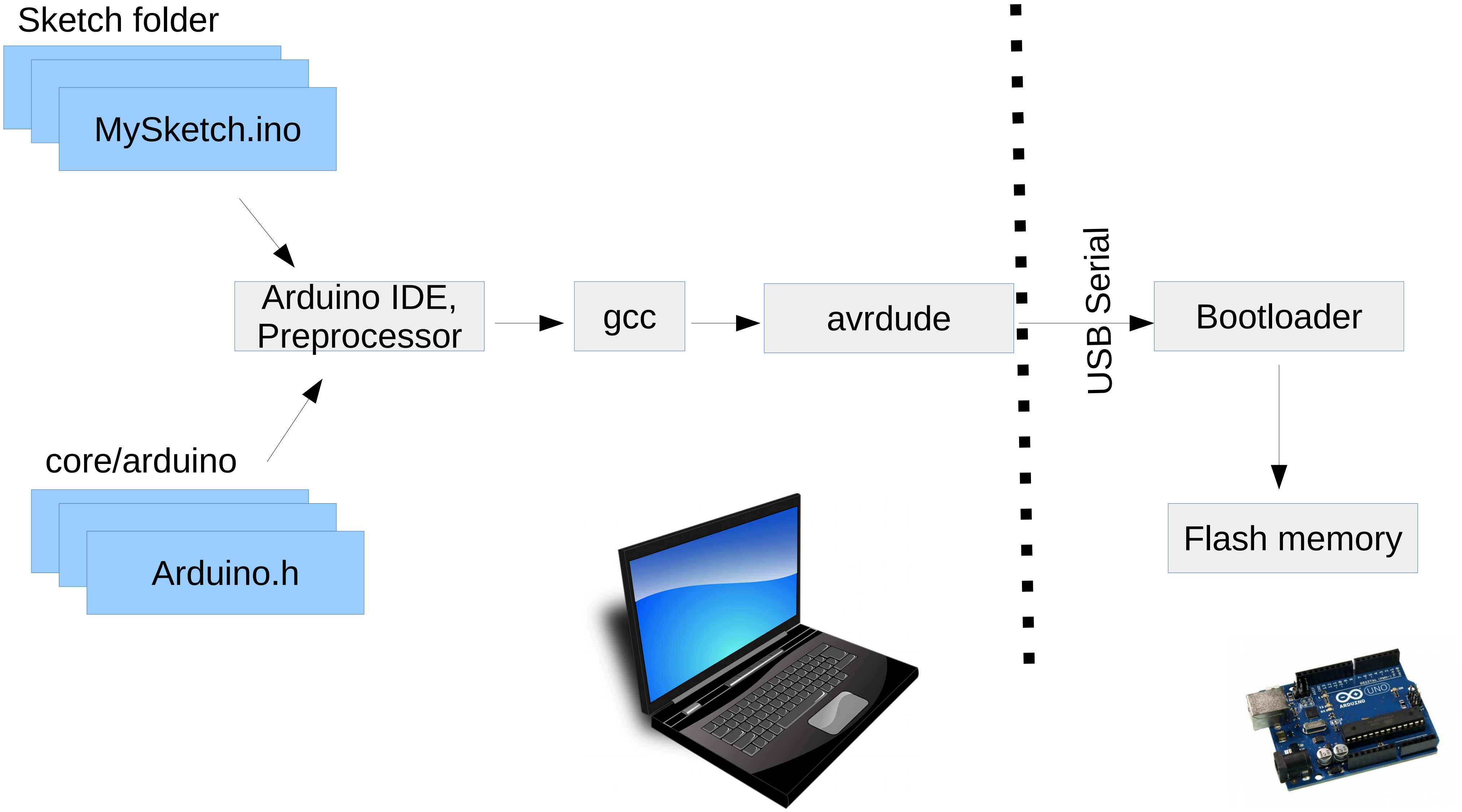
Arduino IDE, under the hood (1)

- cat /usr/share/arduino/hardware/arduino/cores/arduino/main.cpp



```
#include <Arduino.h>

int main(void)
{
  init();

#if defined(USBCON)
  USBDevice.attach();
#endif

  setup();

  for (;;) {
    loop();
    if (serialEventRun) serialEventRun();
  }

  return 0;
}
```

Upload...



Sketch folder

MySketch.ino

core/arduino

Arduino.h

Arduino IDE, Preprocessor

gcc

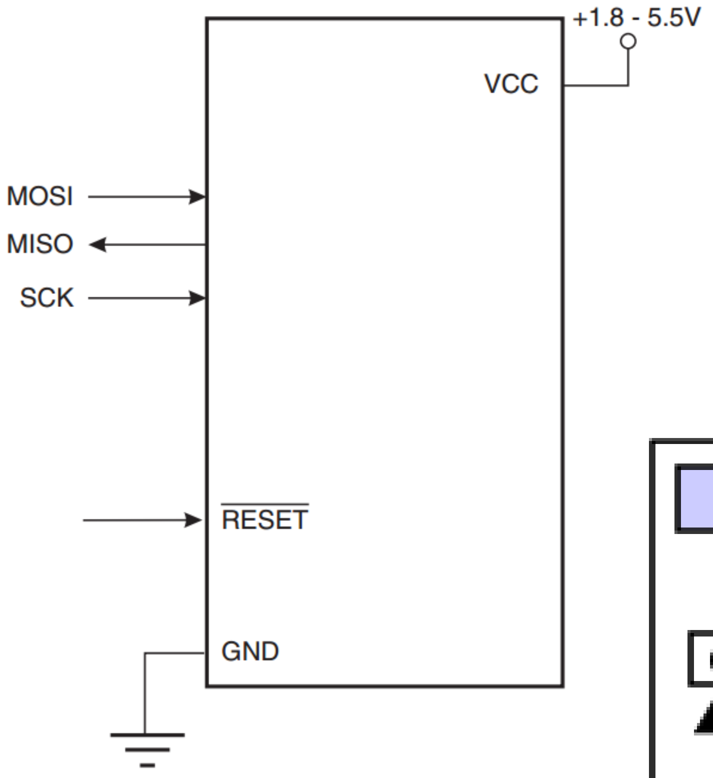
avrdude

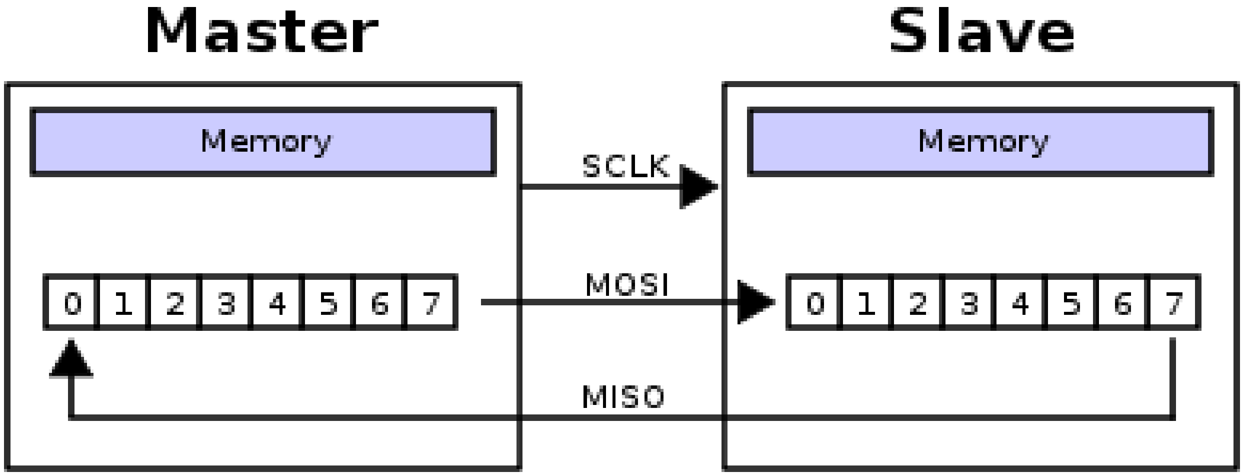
USB Serial

Bootloader

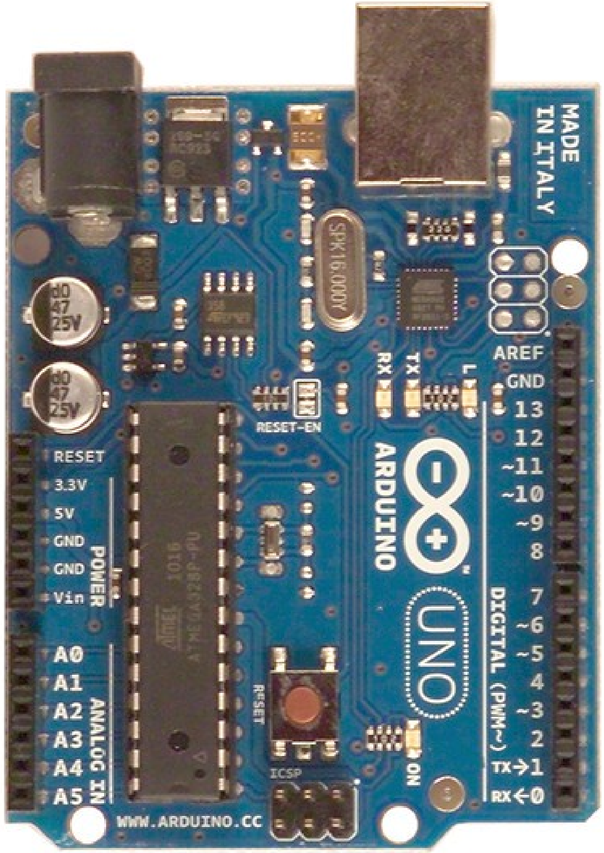
Flash memory

Serial Downloading

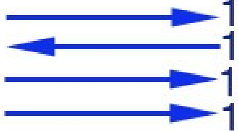




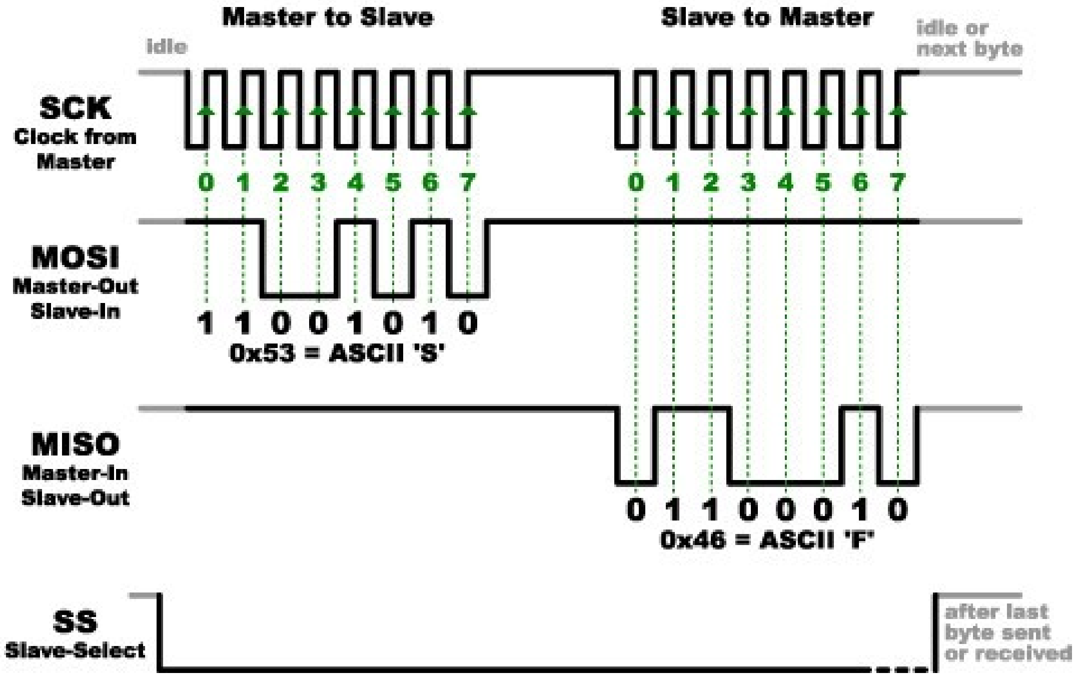
Serial Peripheral Interface (SPI)



SPI pins



13 (SCK)
12 (MISO)
11 (MOSI)
10 (SS)



Fuse bytes (1)

Fuse High Byte	Bit No	Description	Default Value
RSTDISBL [1] [2]	7	External reset disabled	1 (unprogrammed)
DWEN [1] [2] [3]	6	DebugWIRE enabled	1 (unprogrammed)
SPIEN [4]	5	Serial program and data download enabled	0 (programmed) (SPI prog. enabled)
WDTON [5]	4	Watchdog timer always on	1 (unprogrammed)
EESAVE	3	EEPROM preserves chip erase	1 (unprogrammed) (EEPROM not preserved)
BODLEVEL2 [6]	2	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL1 [6]	1	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL0 [6]	0	Brown-out Detector trigger level	1 (unprogrammed)

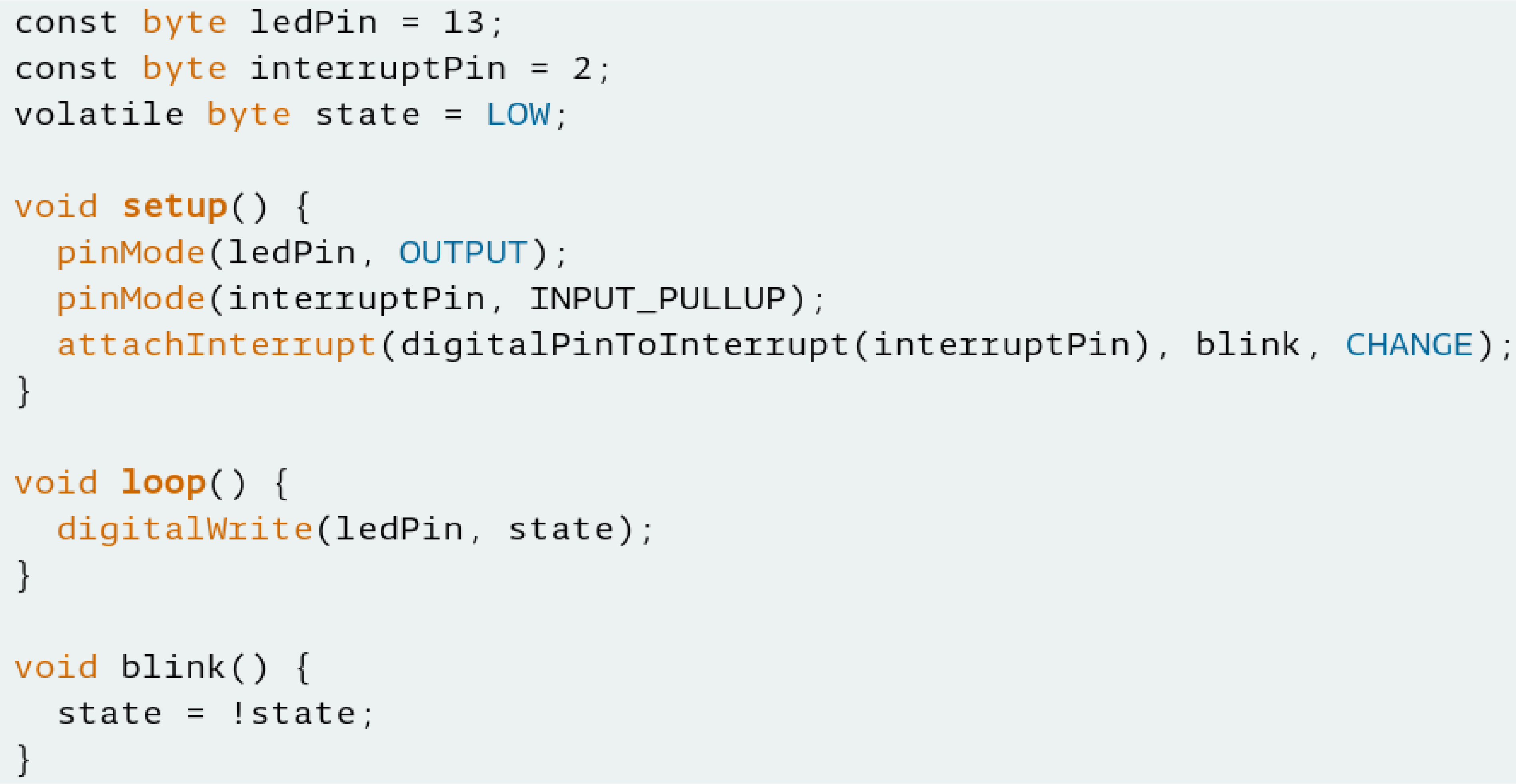
Fuse bytes (2)

Fuse Low Byte	Bit No	Description	Default Value
CKDIV8 (1)	7	Clock divided by 8	0 (programmed)
CKOUT (2)	6	Clock output enabled	1 (unprogrammed)
SUT1 (3)	5	Start-up time setting	1 (unprogrammed)(3)
SUT0 (3)	4	Start-up time setting	0 (programmed)(3)
CKSEL3 (4)	3	Clock source setting	0 (programmed)(4)
CKSEL2 (4)	2	Clock source setting	0 (programmed)(4)
CKSEL1 (4)	1	Clock source setting	1 (unprogrammed)(4)
CKSEL0 (4)	0	Clock source setting	0 (programmed)(4)

In-System Programming (ISP)

- JTAG – protocol/interface for in-system programming
- Programmer (HW) – an electronic equipment that configures programmable non-volatile integrated circuits (called programmable devices)
- bootloader – a computer program that loads an operating system (OS) or runtime environment for the computer after completion of the self-tests

External interrupts

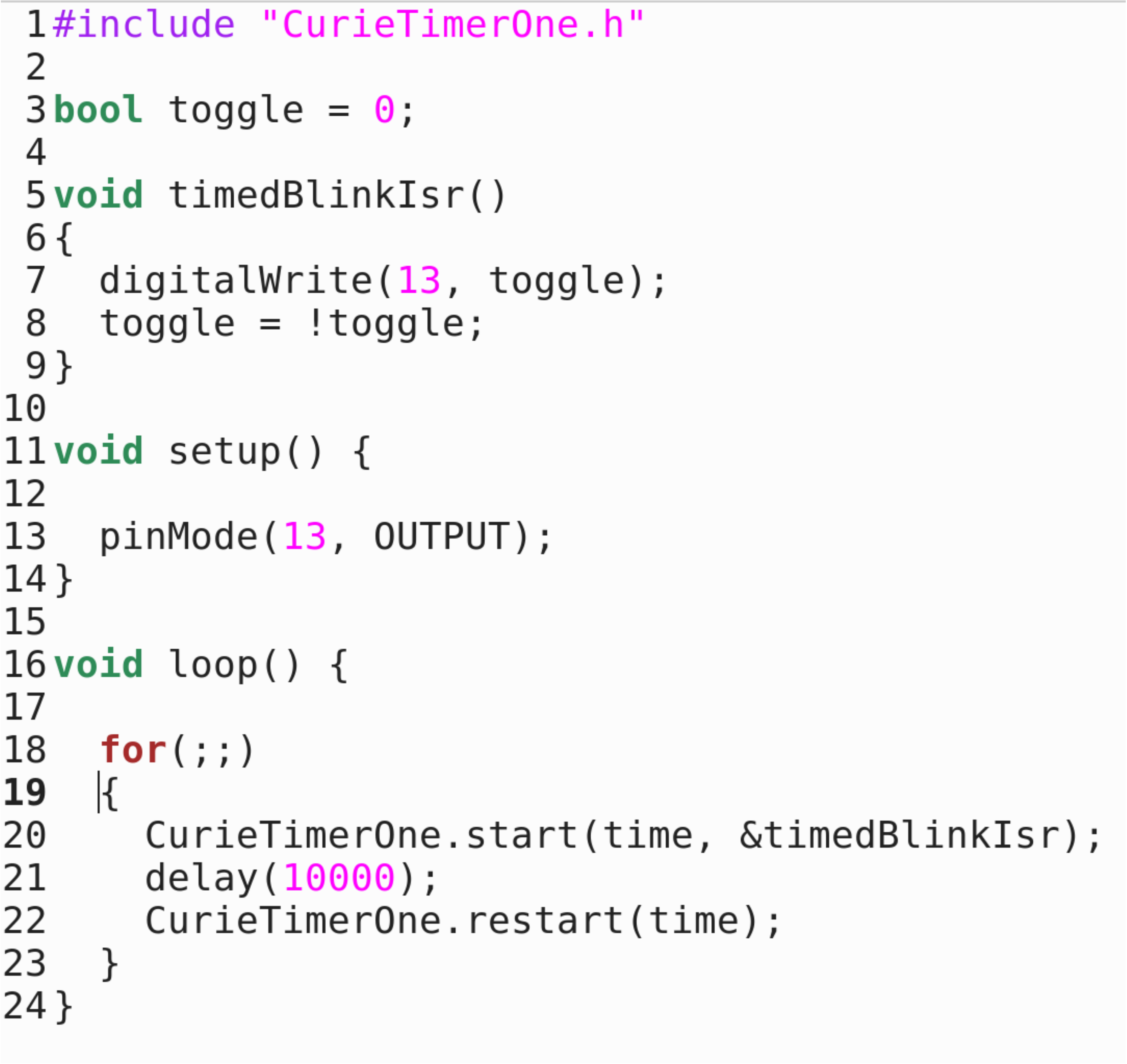


```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

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

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

attachInterrupt()

detachInterrupt()

Timer interrupts

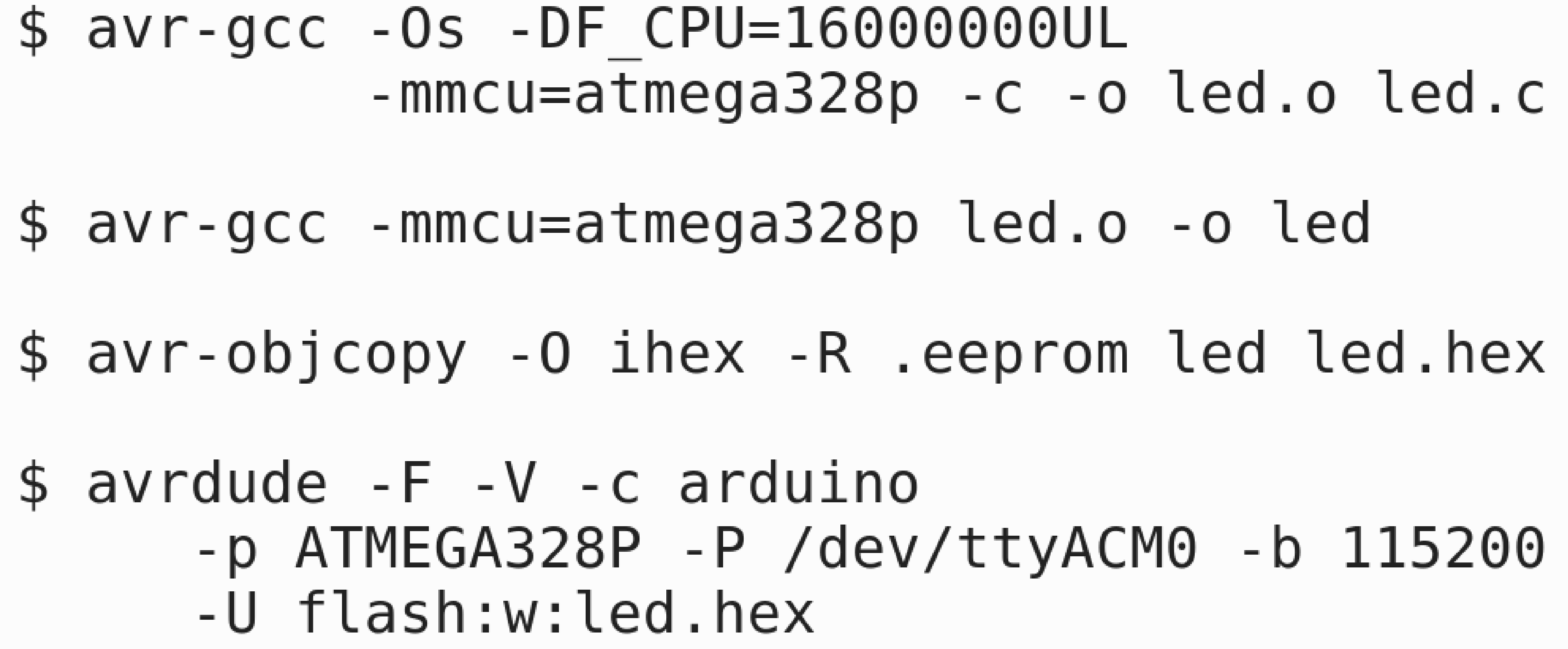


```
#include "CurieTimerOne.h"

bool toggle = 0;

void timedBlinkIsr()
{
  digitalWrite(13, toggle);
  toggle = !toggle;
}

void setup() {

  pinMode(13, OUTPUT);
}

void loop() {

  for(;;)
  {
    CurieTimerOne.start(time, &timedBlinkIsr);
    delay(10000);
    CurieTimerOne.restart(time);
  }
}
```

Installation gcc-avr

- apt-get install
  - gcc-avr - GNU C compiler (cross compiler for avr)
  - avra - assembler for Atmel AVR microcontrollers
  - gdb-avr - GNU Debugger for avr
  - avrdude - software for programming Atmel AVR
  - simulavr - Atmel AVR simulator

Firmware building and uploading



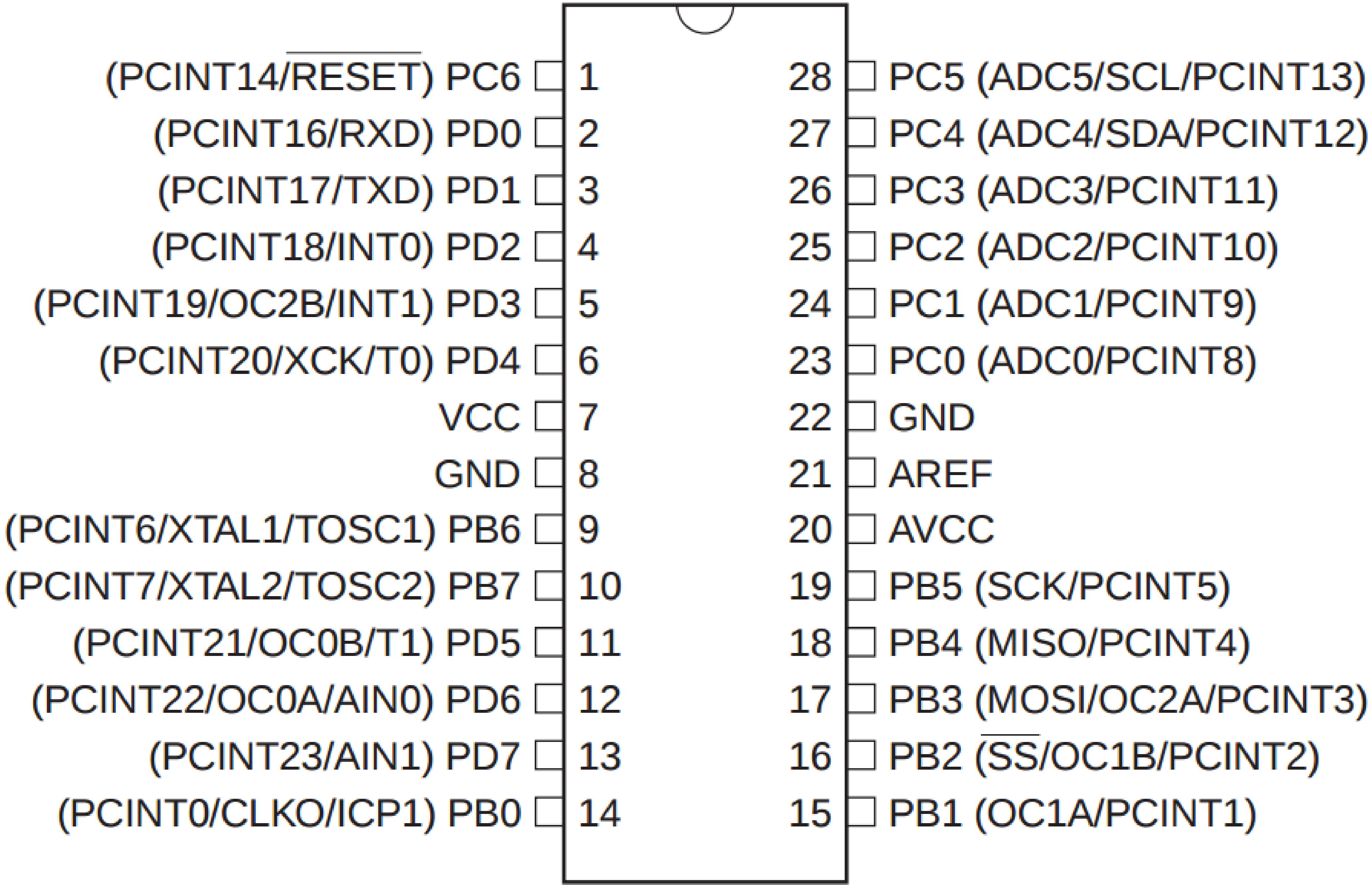
```
$ avr-gcc -Os -DF_CPU=16000000UL
          -mmcu=atmega328p -c -o led.o led.c

$ avr-gcc -mmcu=atmega328p led.o -o led

$ avr-objcopy -O ihex -R .eeprom led led.hex

$ avrdude -F -V -c arduino
     -p ATMEGA328P -P /dev/ttyACM0 -b 115200
     -U flash:w:led.hex
```

Useful links

- http://www.nongnu.org/avr-libc
- S.Monk Programming Arduino. Next Steps
- https://www.arduino.cc/en/hacking/bootloader
- http://www.atmel.com/webdoc/avrlibcreferencemanual/
- http://www.atmel.com/images/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf
- https://learn.sparkfun.com/tutorials/installing-an-arduino-bootloader
- http://chipenable.ru/index.php/programming-avr/item/140-bootloader-avr-xmega.html
- http://www.atmel.com/images/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf

AVR Architecture and assembler.

Pin-outs


(PCINT14/$\overline{\text{RESET}}$) PC6 1
(PCINT16/RXD) PD0 2
(PCINT17/TXD) PD1 3
(PCINT18/INT0) PD2 4
(PCINT19/OC2B/INT1) PD3 5
(PCINT20/XCK/T0) PD4 6
VCC 7
GND 8
(PCINT6/XTAL1/TOSC1) PB6 9
(PCINT7/XTAL2/TOSC2) PB7 10
(PCINT21/OC0B/T1) PD5 11
(PCINT22/OC0A/AIN0) PD6 12
(PCINT23/AIN1) PD7 13
(PCINT0/CLKO/ICP1) PB0 14
28 PC5 (ADC5/SCL/PCINT13)
27 PC4 (ADC4/SDA/PCINT12)
26 PC3 (ADC3/PCINT11)
25 PC2 (ADC2/PCINT10)
24 PC1 (ADC1/PCINT9)
23 PC0 (ADC0/PCINT8)
22 GND
21 AREF
20 AVCC
19 PB5 (SCK/PCINT5)
18 PB4 (MISO/PCINT4)
17 PB3 (MOSI/OC2A/PCINT3)
16 PB2 ($\overline{\text{SS}}$/OC1B/PCINT2)
15 PB1 (OC1A/PCINT1)

Memory

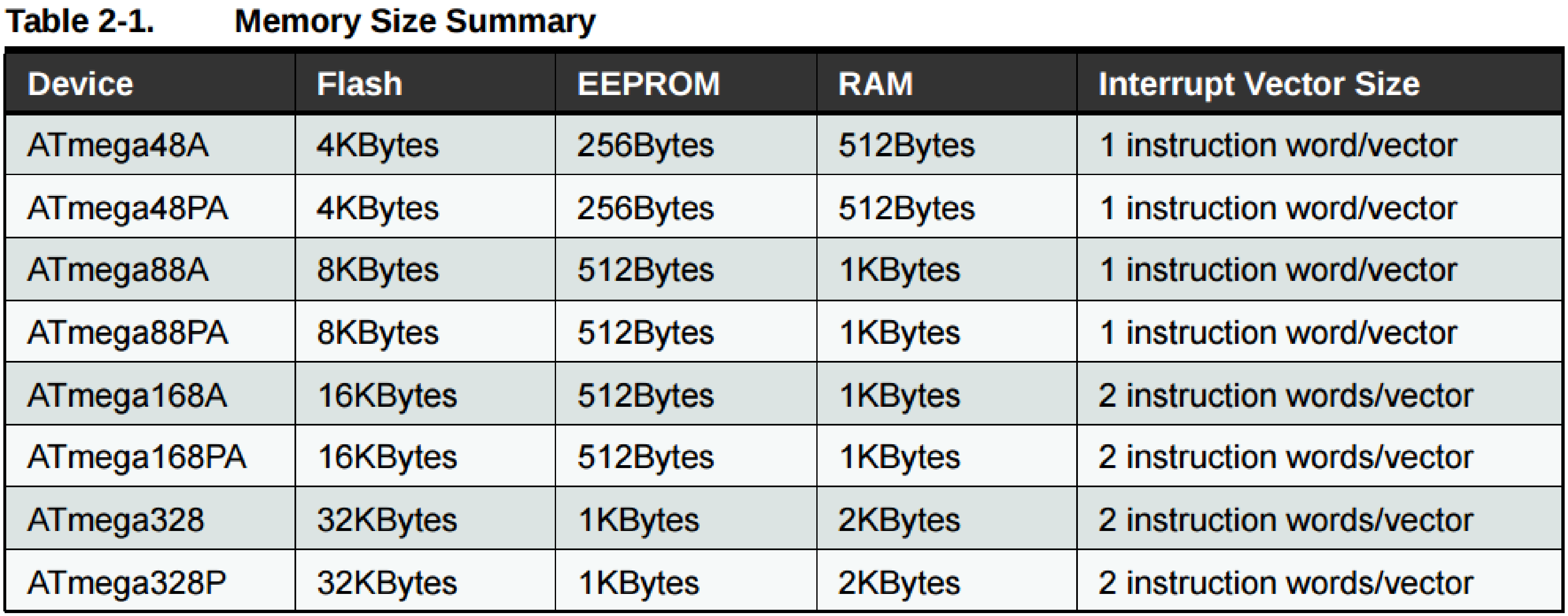
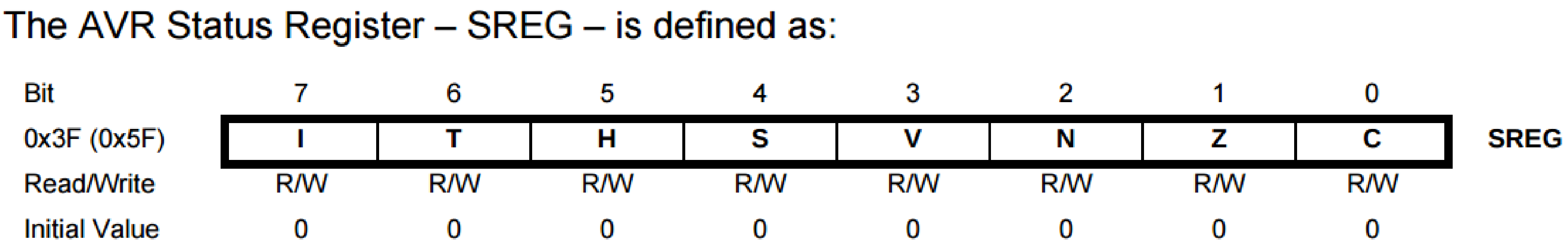


Table 2-1. Memory Size Summary

Device	Flash	EEPROM	RAM	Interrupt Vector Size
ATmega48A	4KBytes	256Bytes	512Bytes	1 instruction word/vector
ATmega48PA	4KBytes	256Bytes	512Bytes	1 instruction word/vector
ATmega88A	8KBytes	512Bytes	1KBytes	1 instruction word/vector
ATmega88PA	8KBytes	512Bytes	1KBytes	1 instruction word/vector
ATmega168A	16KBytes	512Bytes	1KBytes	2 instruction words/vector
ATmega168PA	16KBytes	512Bytes	1KBytes	2 instruction words/vector
ATmega328	32KBytes	1KBytes	2KBytes	2 instruction words/vector
ATmega328P	32KBytes	1KBytes	2KBytes	2 instruction words/vector

Status register



The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

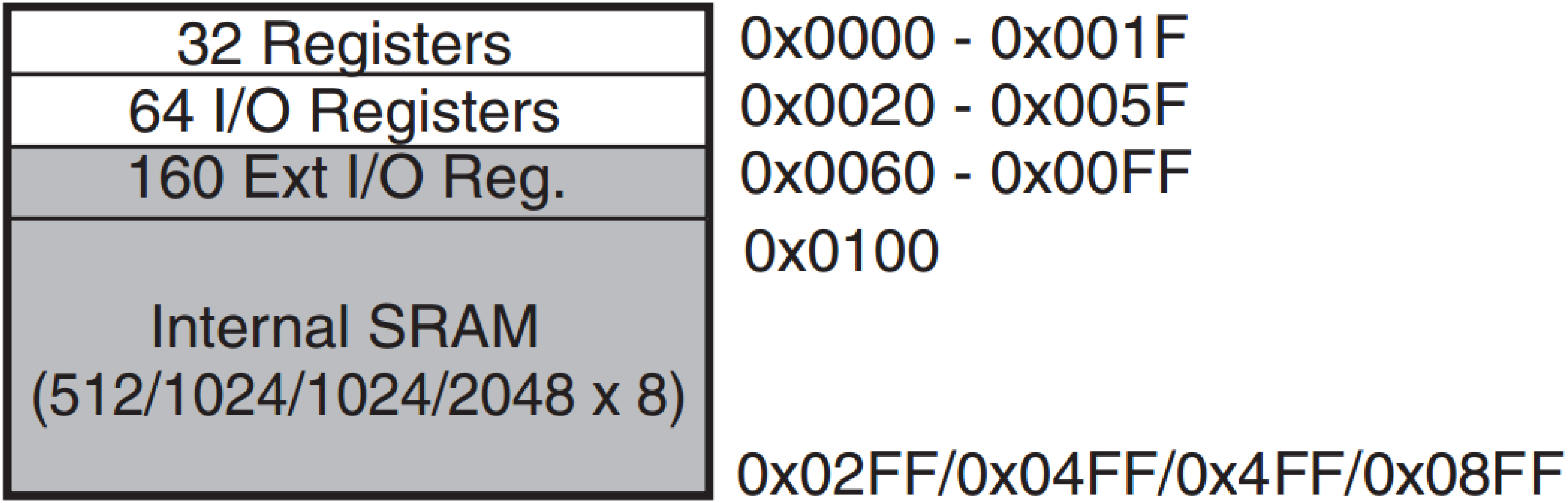
I – global interrupt enable
T – bit copy storage
H – half carry flag
S – sign bit
N – negative flag
Z – zero flag
C – carry flag

General Purpose Registers

General
Purpose
Working
Registers

7 0	Addr.	
R0	0x00	
R1	0x01	
R2	0x02	
…		
R13	0x0D	
R14	0x0E	
R15	0x0F	
R16	0x10	
R17	0x11	
…		
R26	0x1A	X-register Low Byte
R27	0x1B	X-register High Byte
R28	0x1C	Y-register Low Byte
R29	0x1D	Y-register High Byte
R30	0x1E	Z-register Low Byte
R31	0x1F	Z-register High Byte

Data memory map



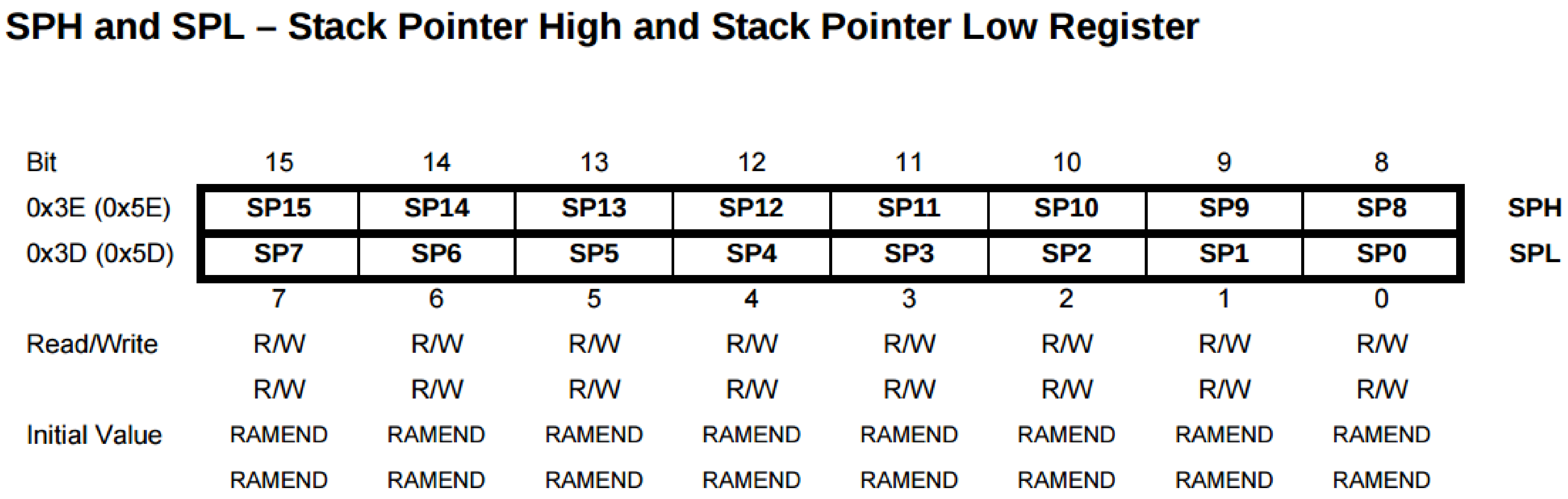
32 Registers — 0x0000 - 0x001F

64 I/O Registers — 0x0020 - 0x005F

160 Ext I/O Reg. — 0x0060 - 0x00FF

Internal SRAM (512/1024/1024/2048 x 8) — 0x0100 … 0x02FF/0x04FF/0x4FF/0x08FF

Stack register



SPH and SPL – Stack Pointer High and Stack Pointer Low Register

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	**SP15**	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
0x3D (0x5D)	**SP7**	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

Stack instructions

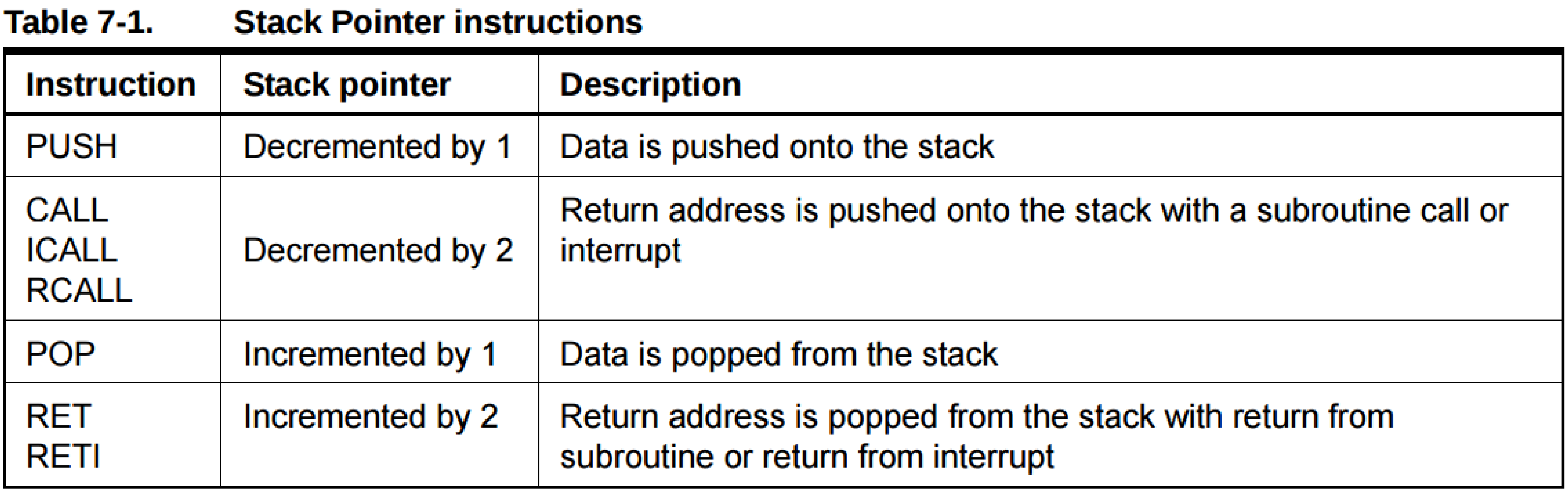
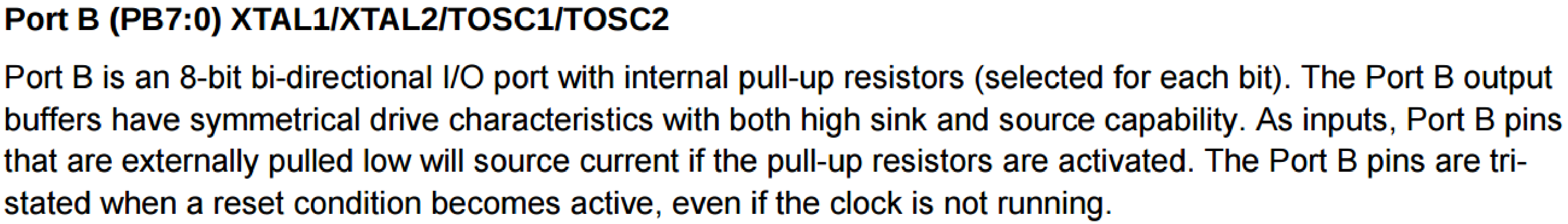


Table 7-1. Stack Pointer instructions

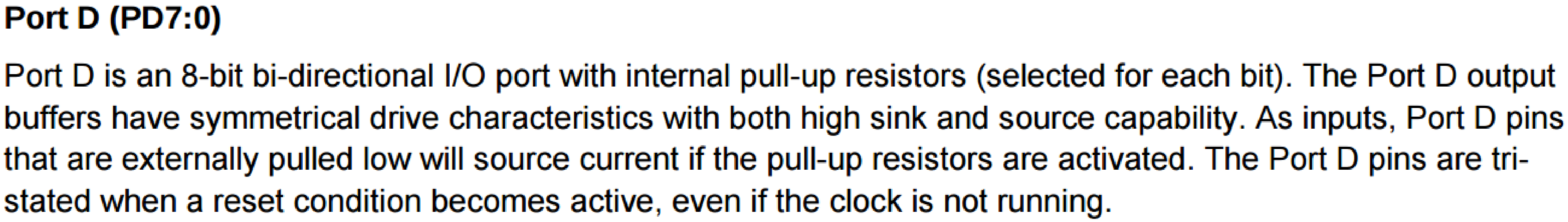
Instruction	Stack pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

Ports



Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

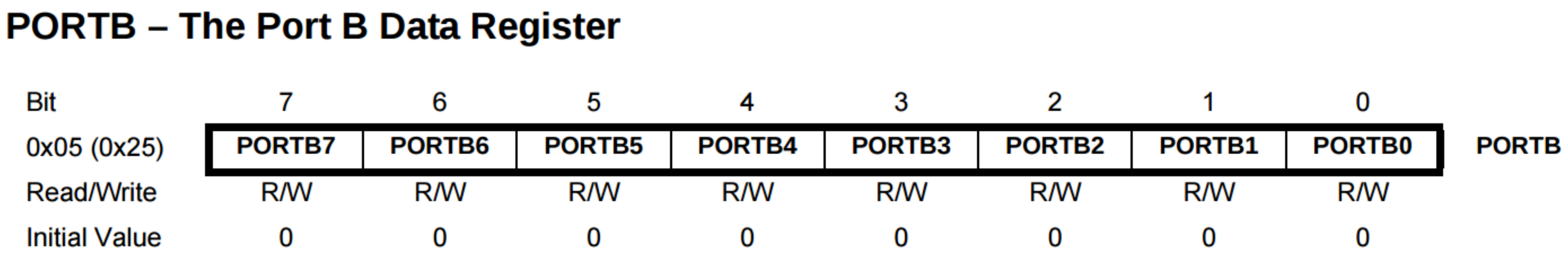
Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.



Port D (PD7:0)

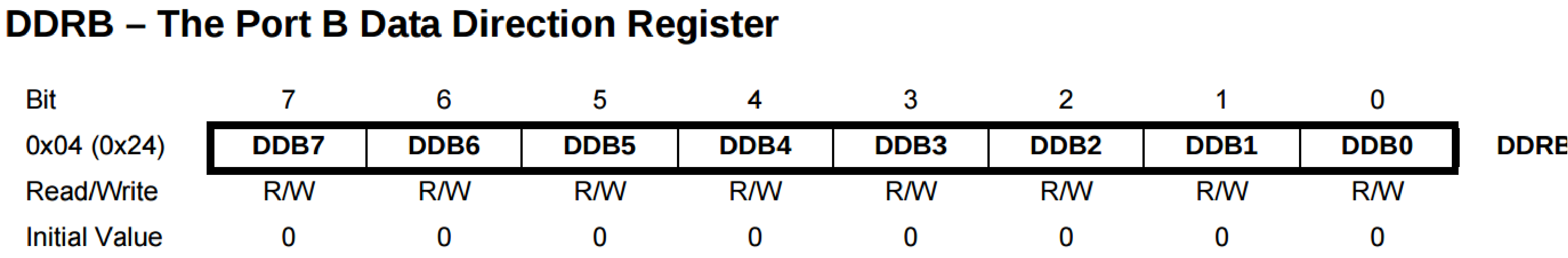
Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

PORTB, DDRB



PORTB – The Port B Data Register

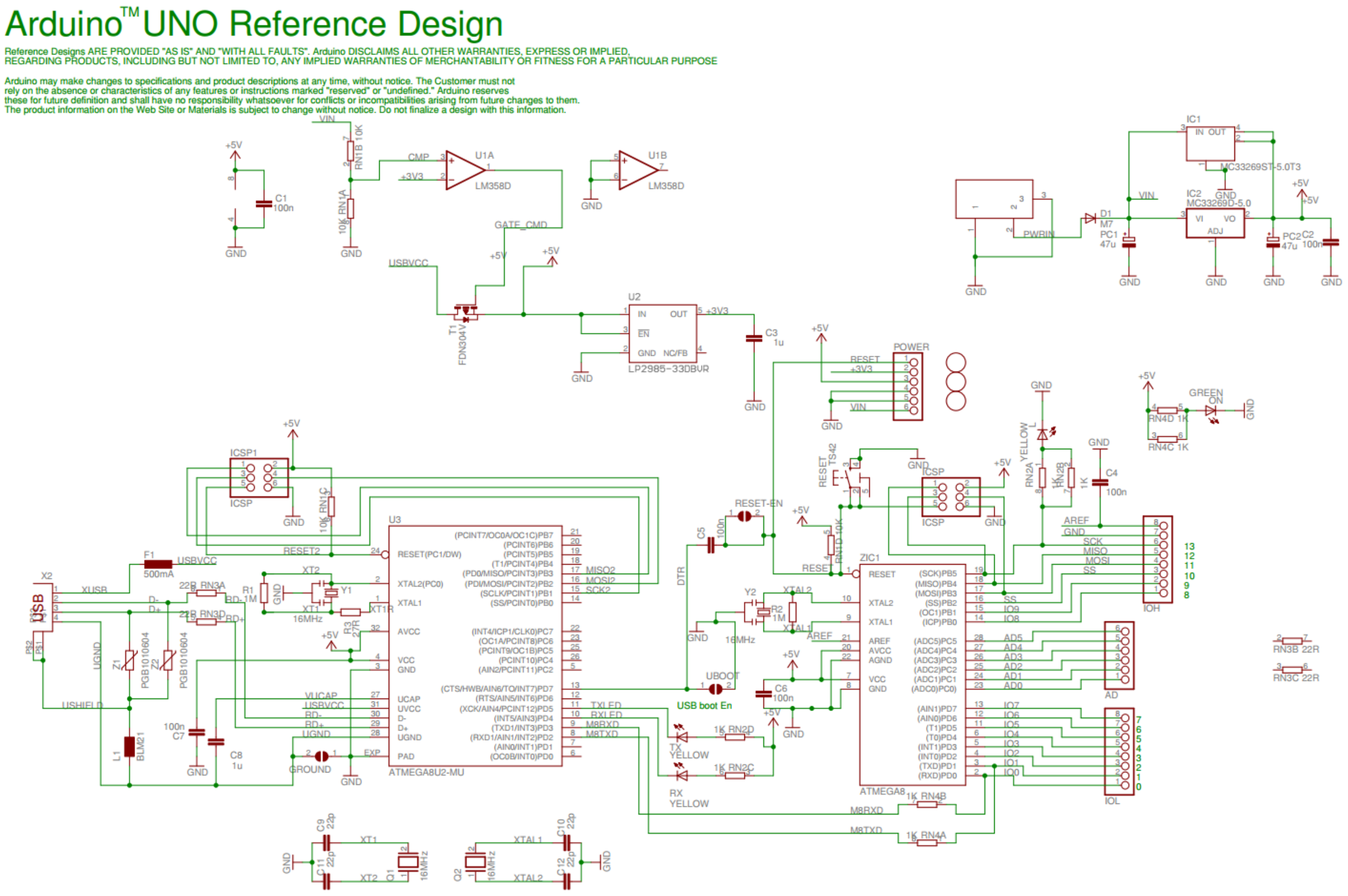
Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	**PORTB7**	**PORTB6**	**PORTB5**	**PORTB4**	**PORTB3**	**PORTB2**	**PORTB1**	**PORTB0**	**PORTB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	**DDB7**	**DDB6**	**DDB5**	**DDB4**	**DDB3**	**DDB2**	**DDB1**	**DDB0**	**DDRB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Arduino Uno Reference Design

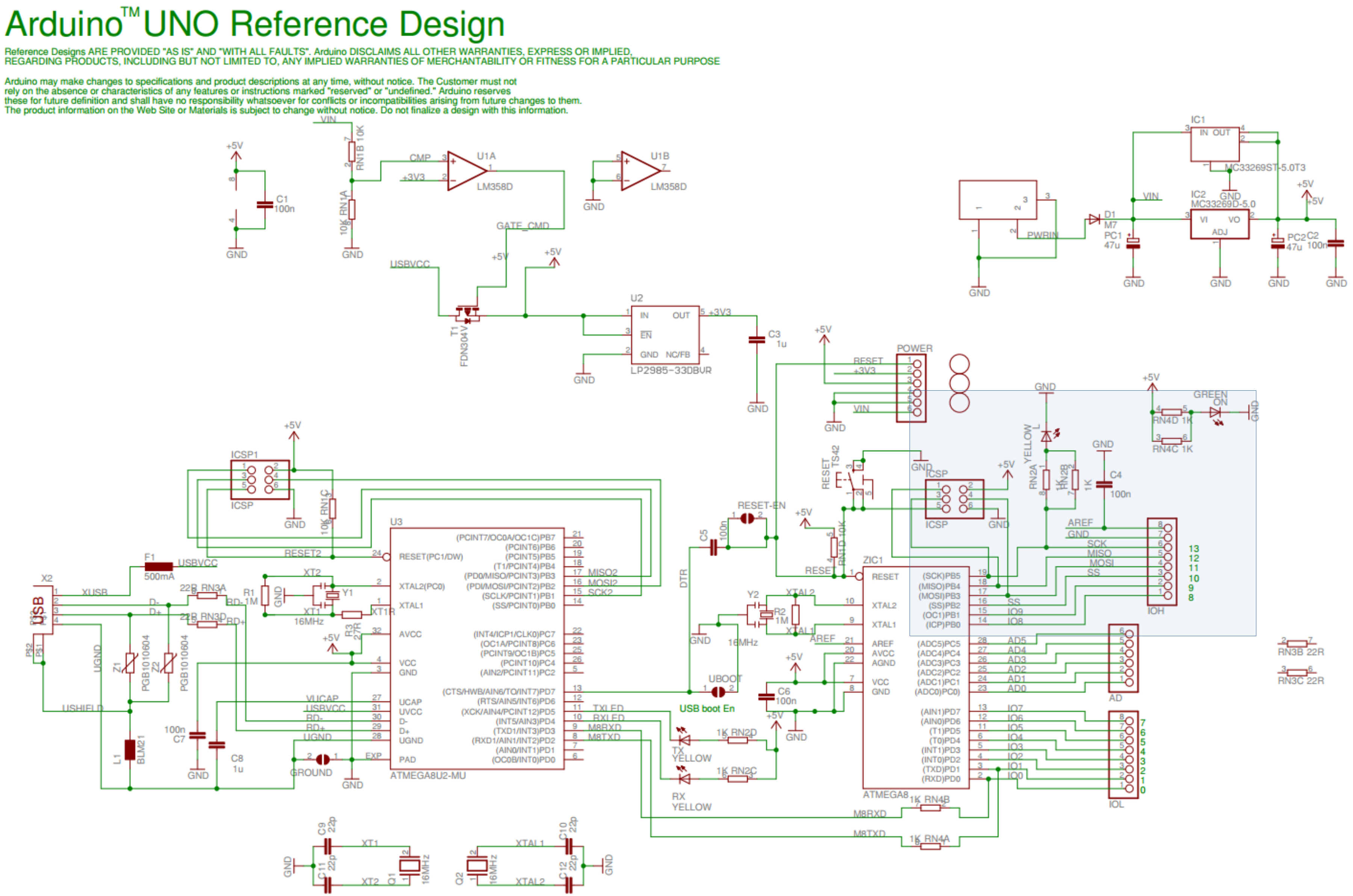


Arduino™ UNO Reference Design

Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Arduino Uno Reference Design

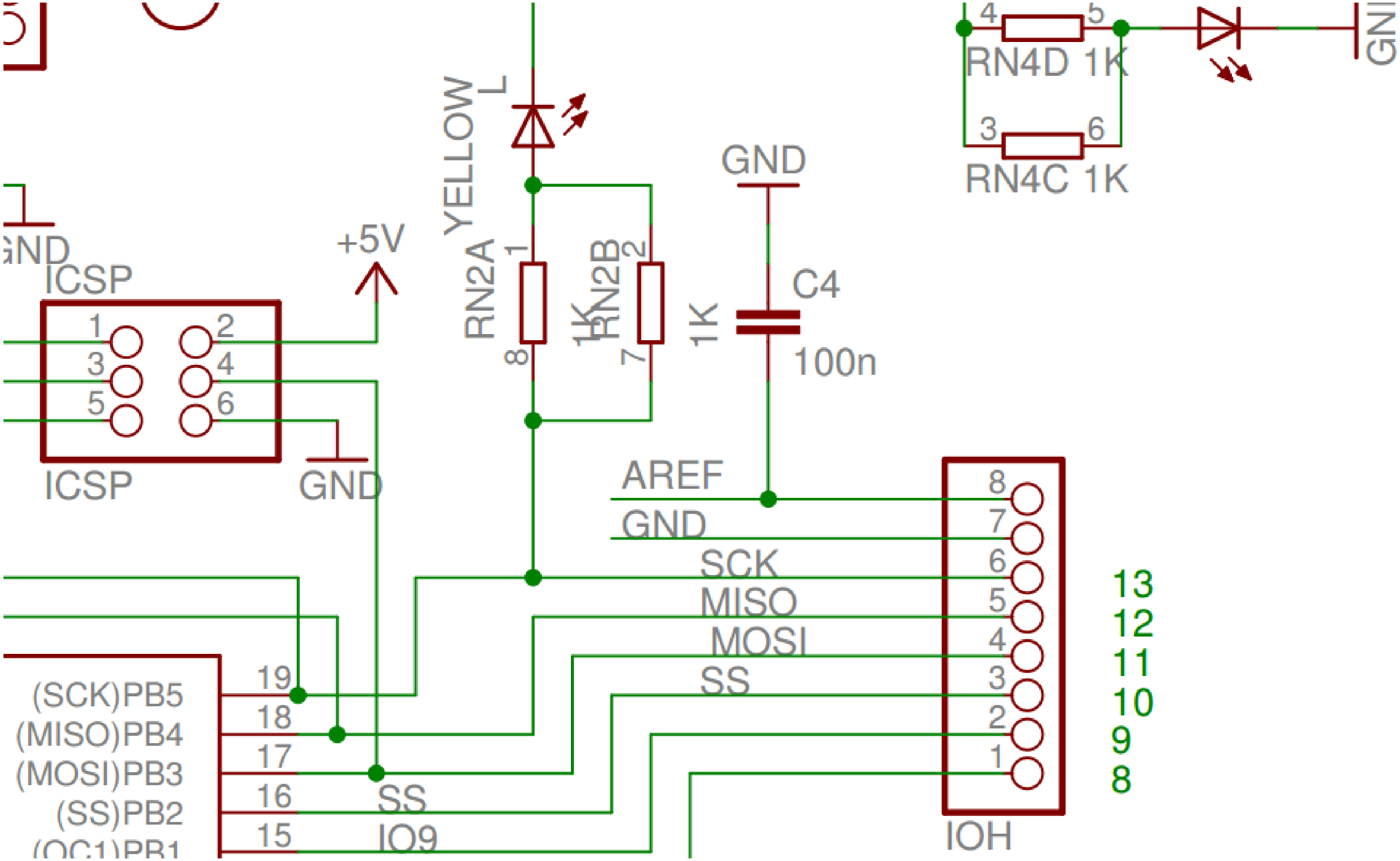


Arduino™ UNO Reference Design

Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

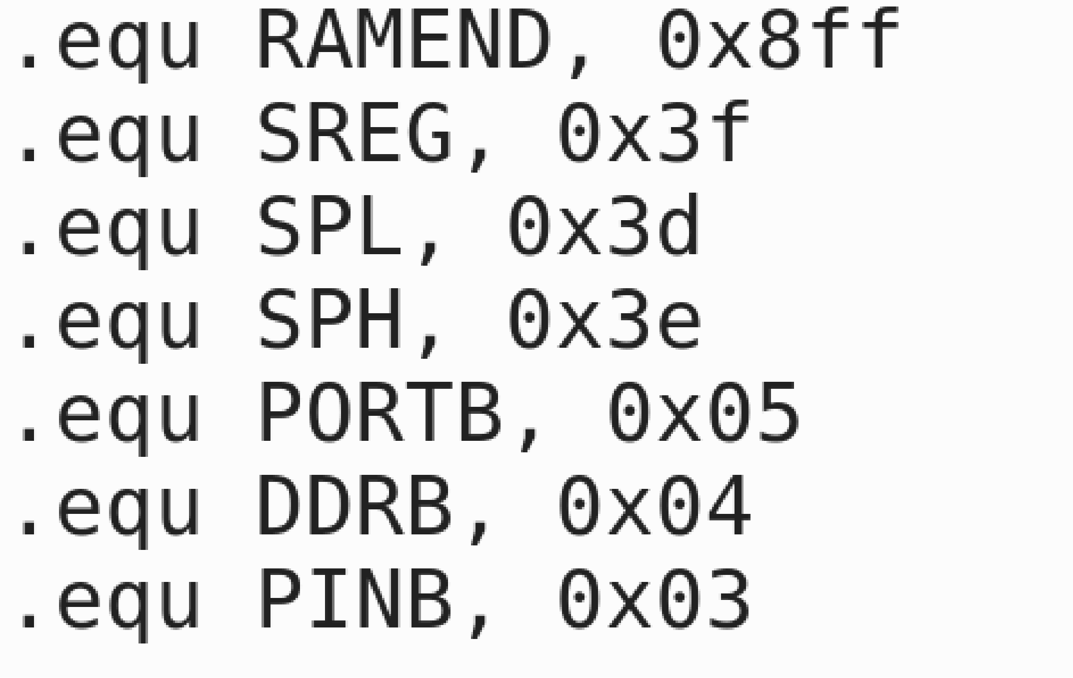
Arduino Uno Reference Design



Program structure

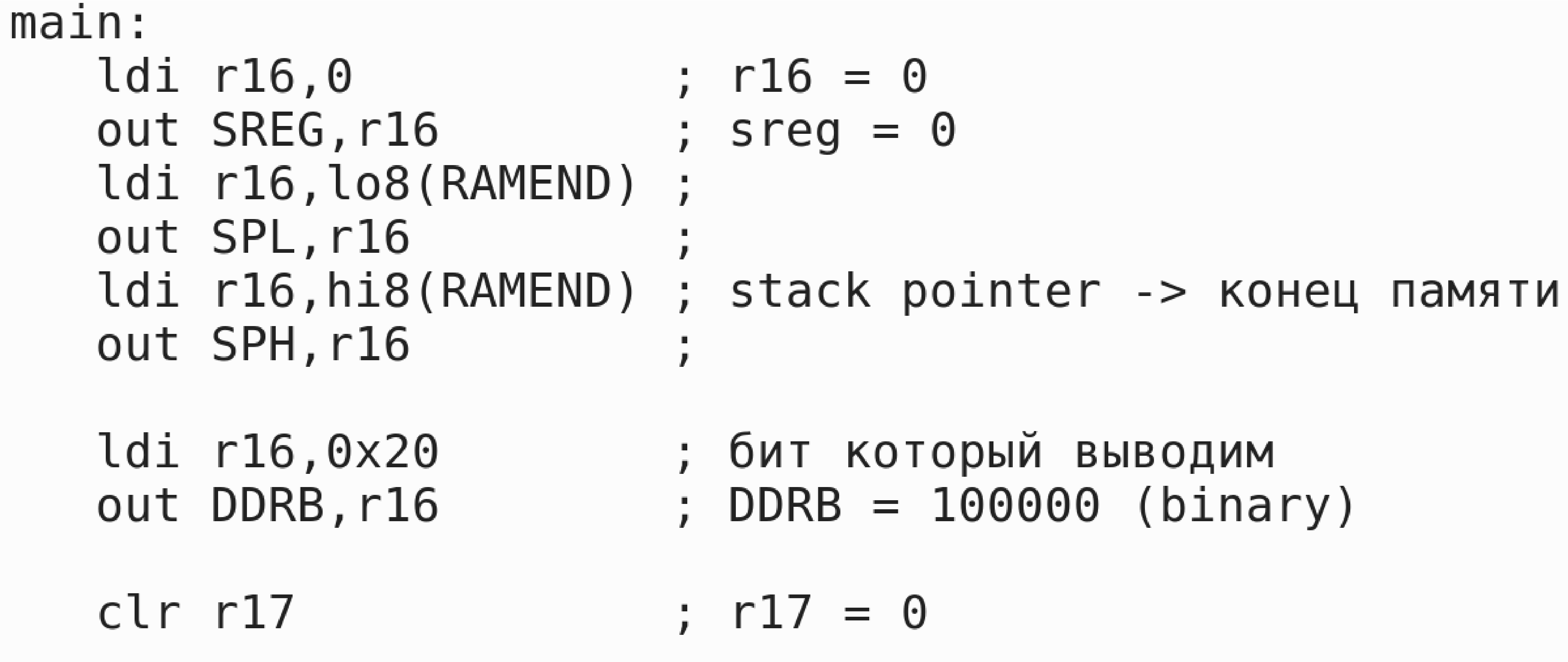
- Definitions
- Initialization
- Busy loop
- Functions (wait)

Definitions



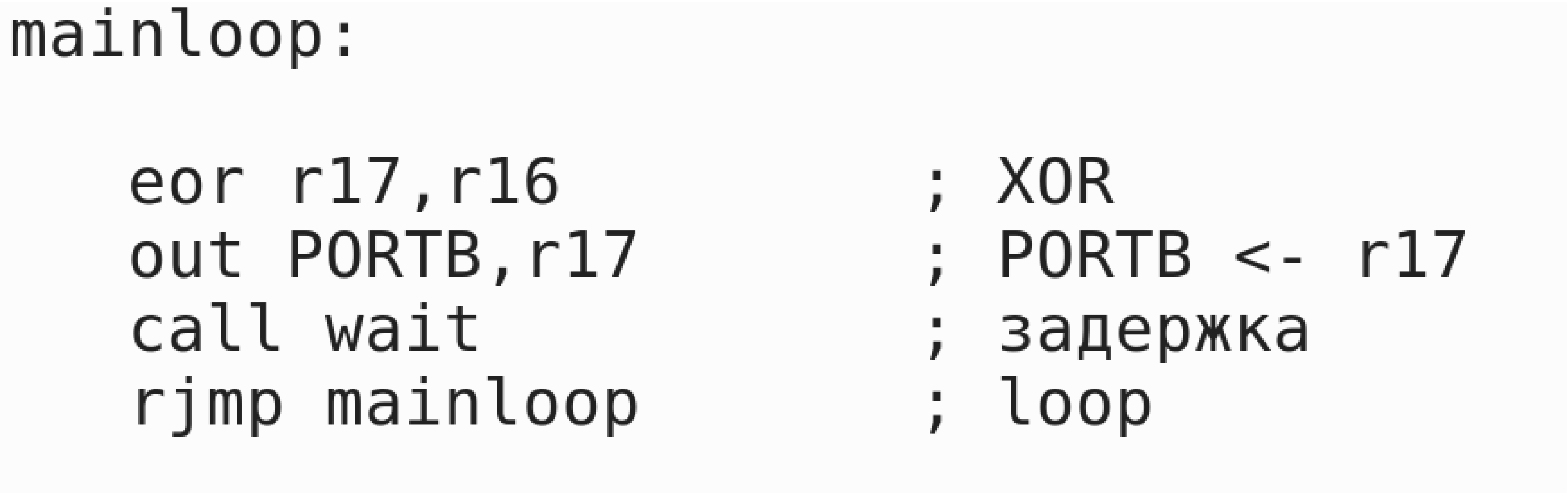
```
.equ RAMEND, 0x8ff
.equ SREG, 0x3f
.equ SPL, 0x3d
.equ SPH, 0x3e
.equ PORTB, 0x05
.equ DDRB, 0x04
.equ PINB, 0x03
```

Initialization



```
main:
   ldi r16,0               ; r16 = 0
   out SREG,r16            ; sreg = 0
   ldi r16,lo8(RAMEND)     ;
   out SPL,r16             ;
   ldi r16,hi8(RAMEND)     ; stack pointer -> конец памяти
   out SPH,r16             ;

   ldi r16,0x20            ; бит который выводим
   out DDRB,r16            ; DDRB = 100000 (binary)

   clr r17                 ; r17 = 0
```

Buzy loop



```
mainloop:

   eor r17,r16                 ; XOR
   out PORTB,r17               ; PORTB <- r17
   call wait                   ; задержка
   rjmp mainloop               ; loop
```

wait

```
wait:
   push r16
   push r17
   push r18

   ldi r16,0x10 ;    loop 0x100000 times
   ldi r17,0x00 ;    ~12 million cycles
   ldi r18,0x00 ;    ~0.7s at 16Mhz

_w0:
   dec r18
   brne _w0
   dec r17
   brne _w0
   dec r16
   brne _w0

   pop r18
   pop r17
   pop r16
   ret
```

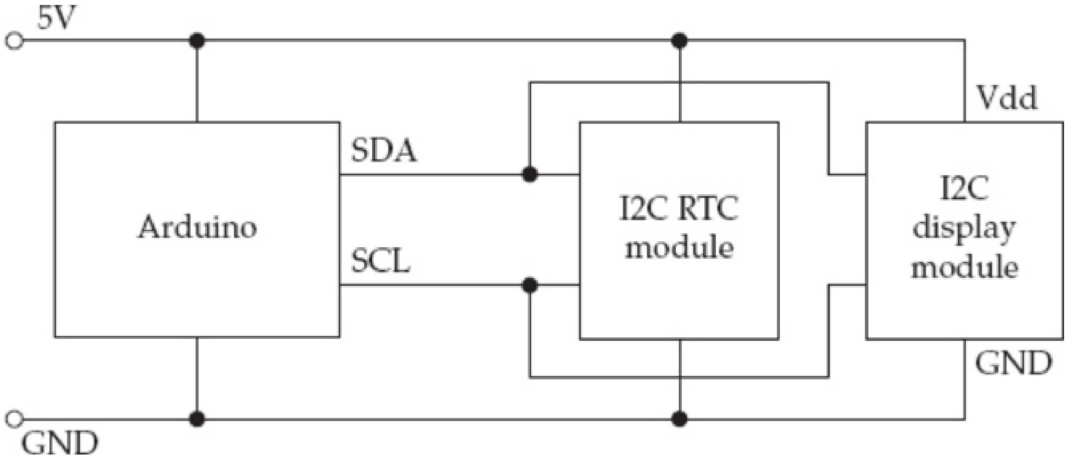
Useful links

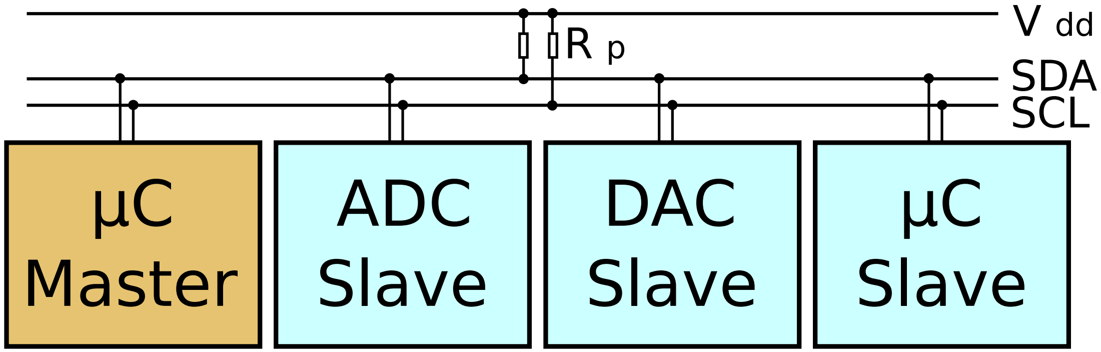
- http://www.instructables.com/id/Command-Line-Assembly-Language-Programming-for-Ard/
- http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf
- http://www.atmel.com/images/atmel-0856-avr-instruction-set-manual.pdf
- http://www.avr-tutorials.com/general/avr-microcontroller-stack-operation-and-stack-pointer
- http://www.avr-asm-tutorial.net/avr_en/beginner/SRAM.html
- https://www.cypherpunk.at/2014/09/native-assembler-programming-on-arduino/
- http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_directives.html
- https://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf

Serial interfaces

I2C (Two Wire Iface, TWI)

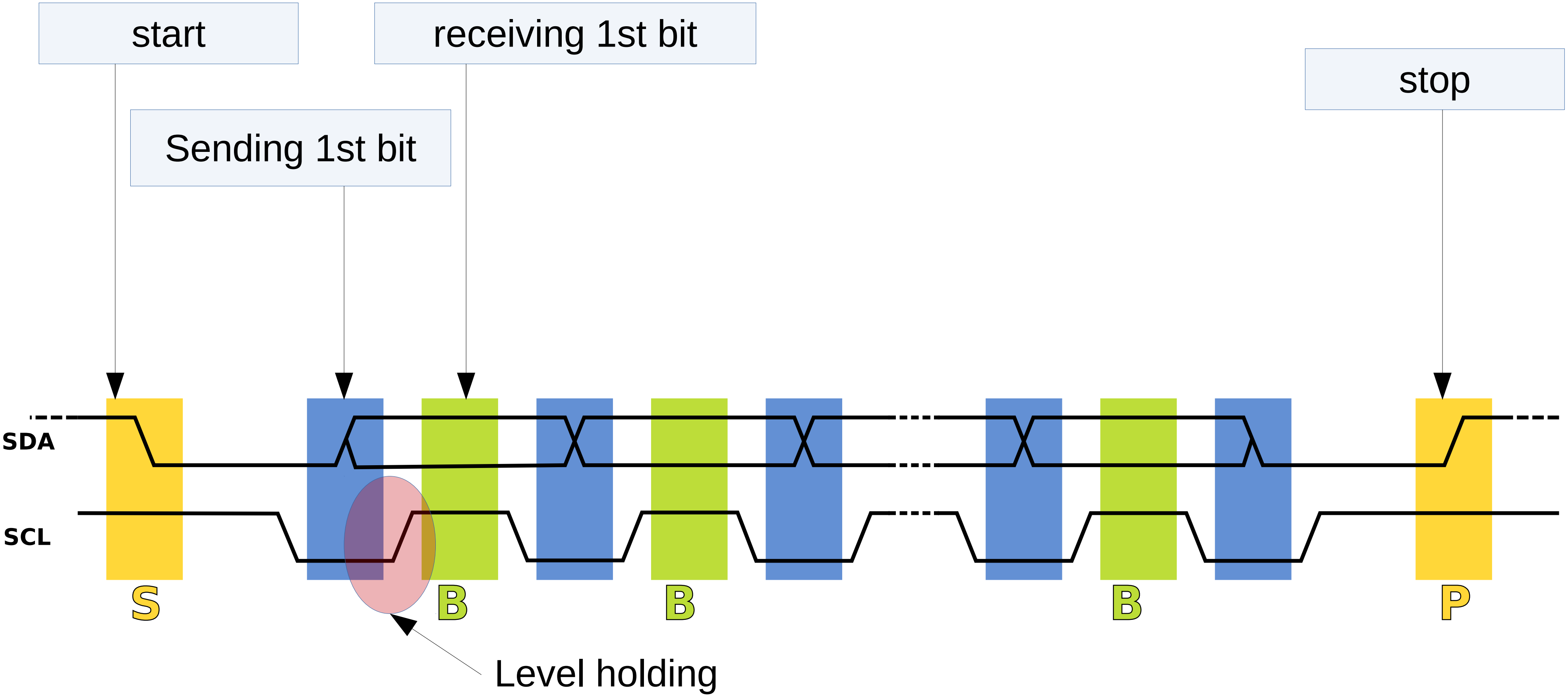
- Inter-Integrated Circuit



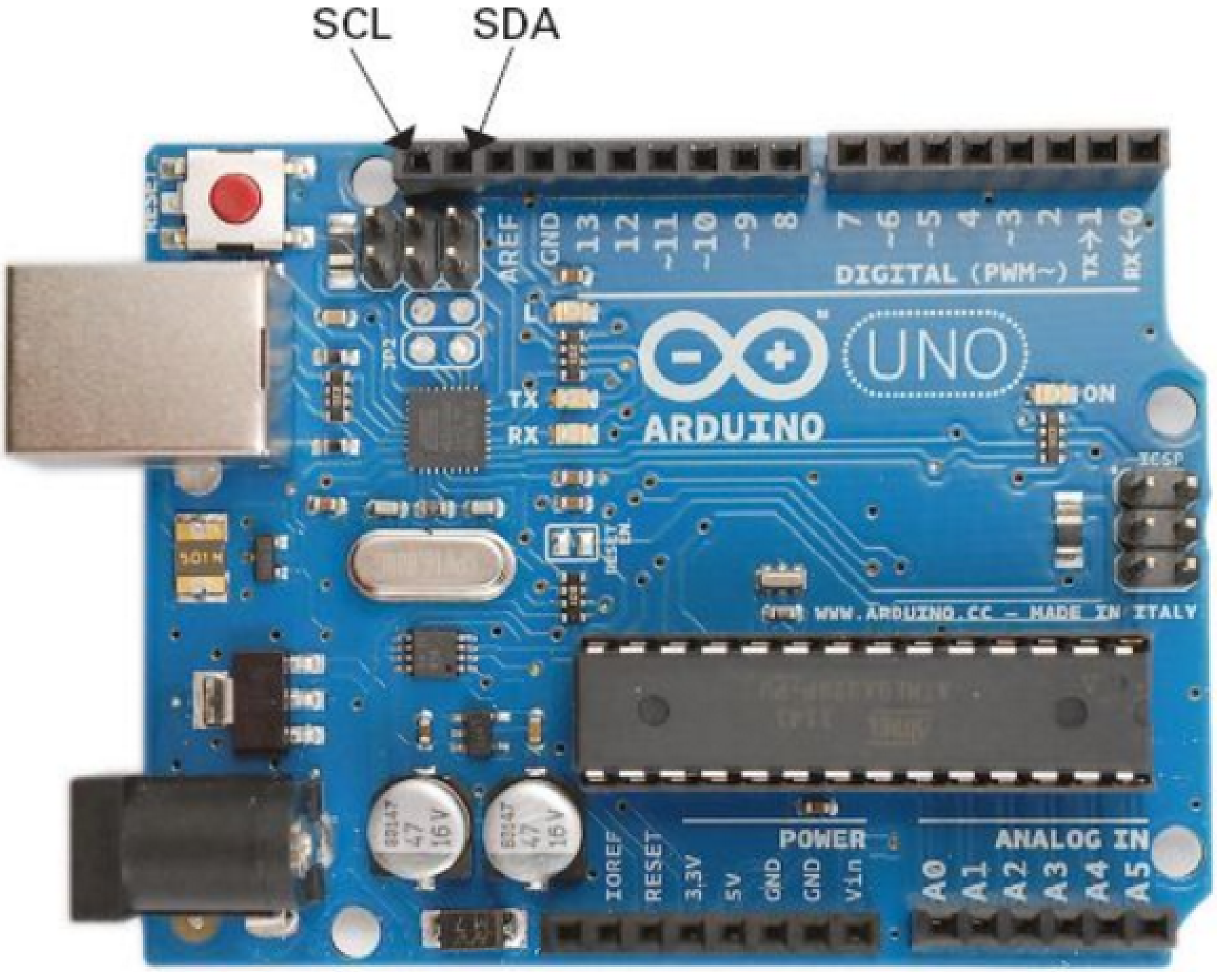


I2C – data transfer

- SDA – Serial Data
- SCL – Serial Clock



Arduino I2C

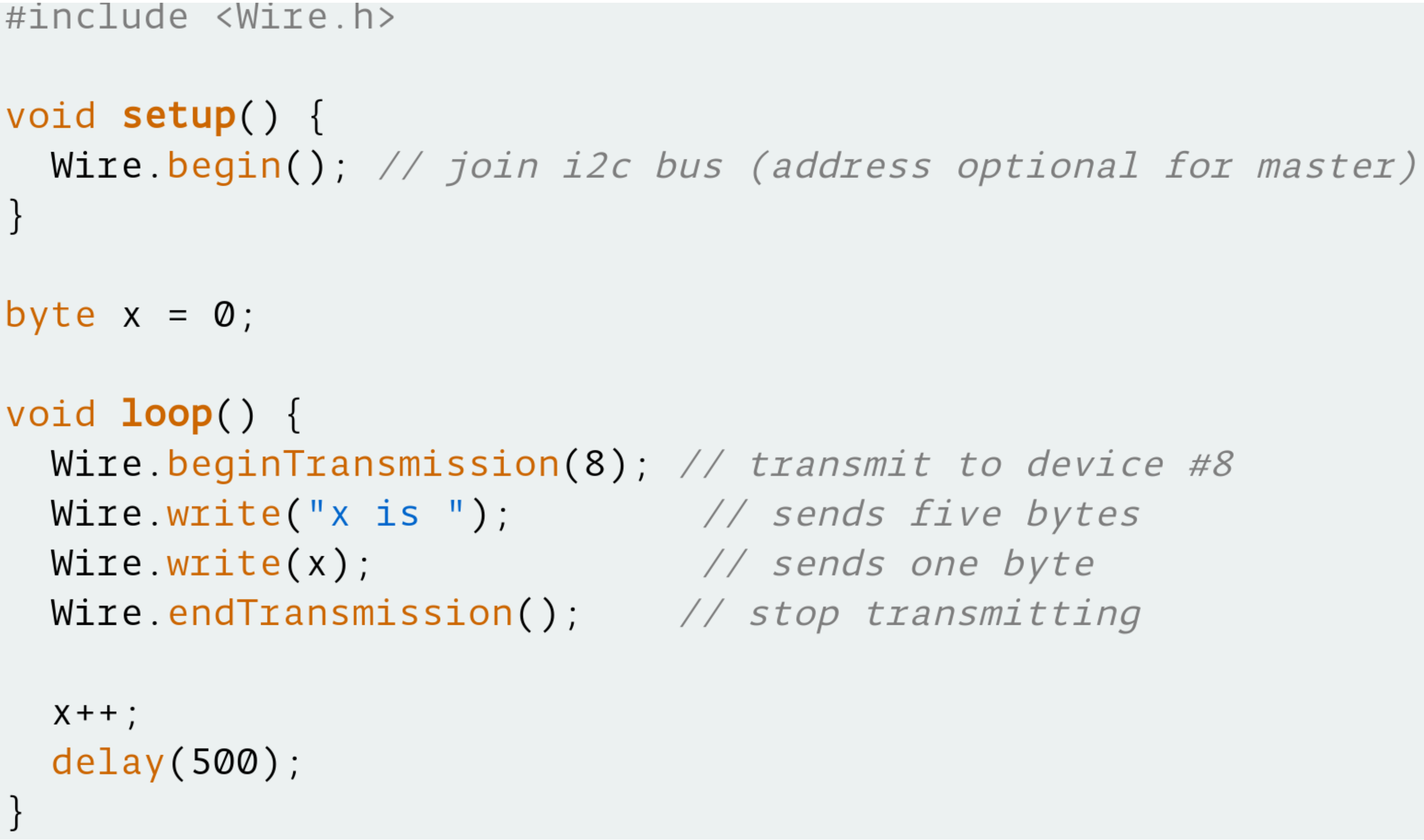

SCL
SDA
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
UNO
ARDUINO
TX
RX
ON
ICSP
WWW.ARDUINO.CC – MADE IN ITALY
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

Wire

Board	I2C / TWI pins
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

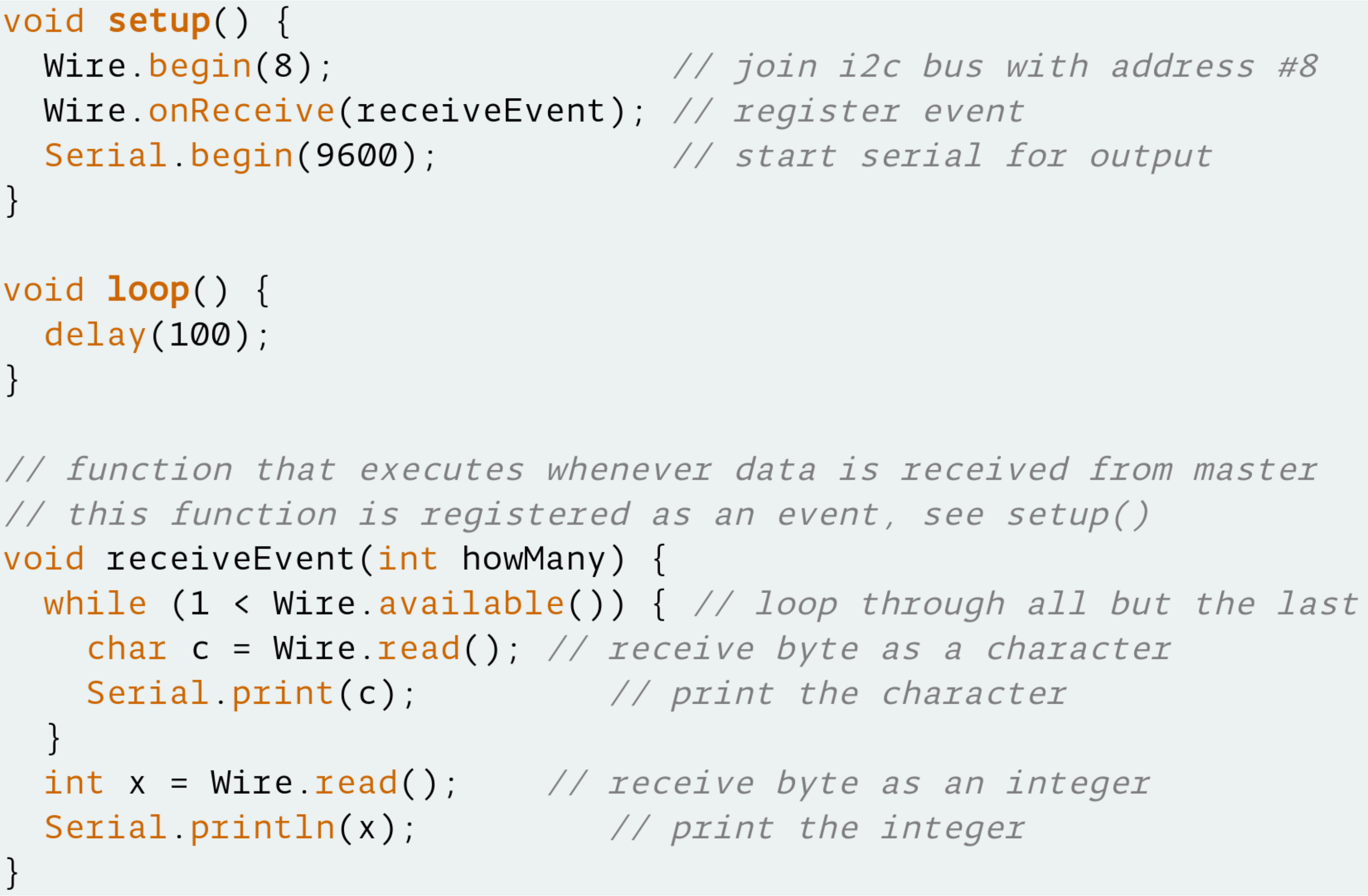
- begin()
- requestFrom()
- beginTransmission()
- endTransmission()
- write()
- available()
- read()
- SetClock()
- onReceive()
- onRequest()

I2C Master Write



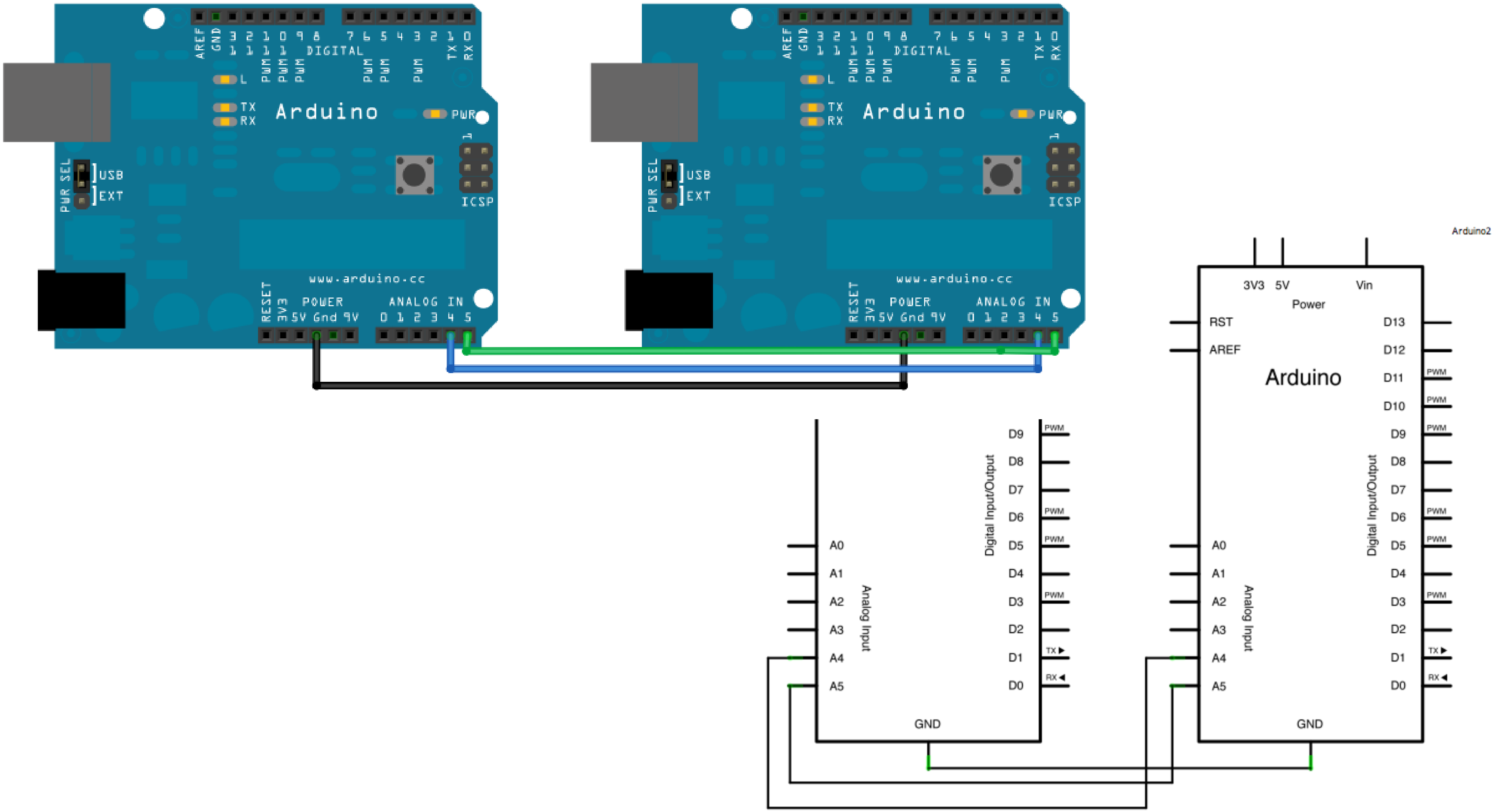
```
#include <Wire.h>

void setup() {
  Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop() {
  Wire.beginTransmission(8); // transmit to device #8
  Wire.write("x is ");        // sends five bytes
  Wire.write(x);              // sends one byte
  Wire.endTransmission();    // stop transmitting

  x++;
  delay(500);
}
```

I2C Slave Read



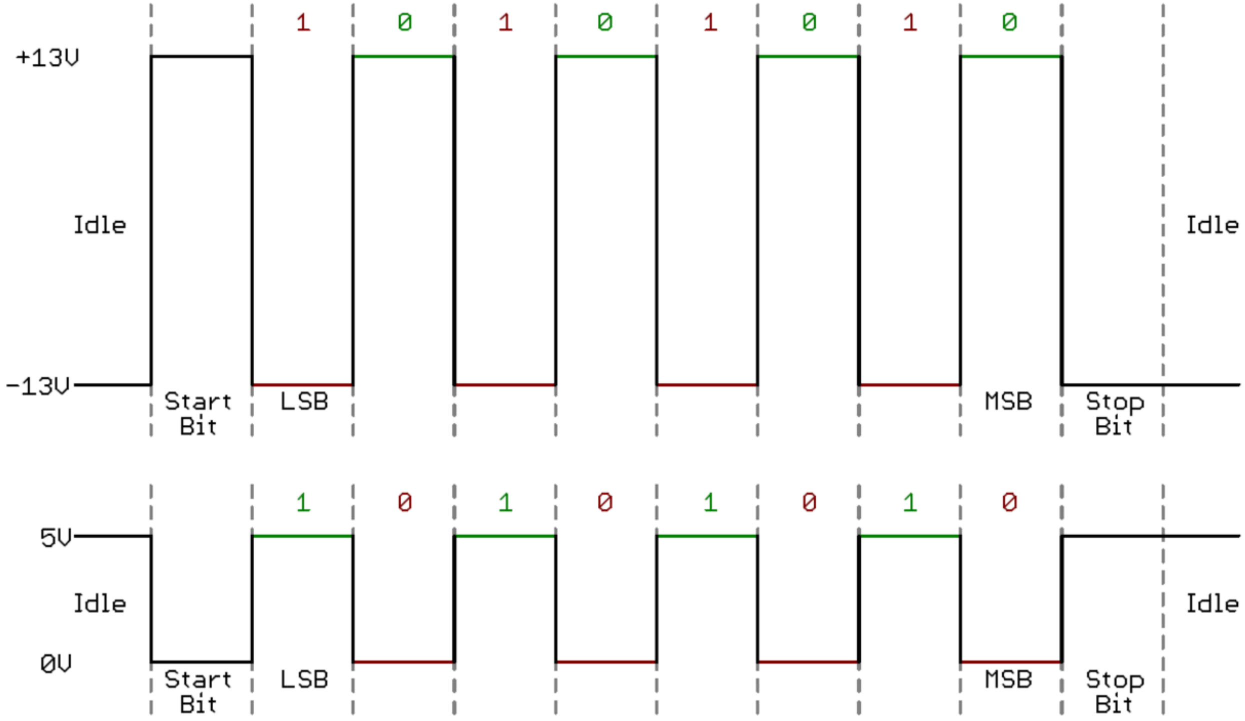
```
void setup() {
  Wire.begin(8);                // join i2c bus with address #8
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600);           // start serial for output
}

void loop() {
  delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany) {
  while (1 < Wire.available()) { // loop through all but the last
    char c = Wire.read(); // receive byte as a character
    Serial.print(c);         // print the character
  }
  int x = Wire.read();    // receive byte as an integer
  Serial.println(x);         // print the integer
}
```

Arduino to Arduino



Universal asynchronous receiver/transmitter (UART)

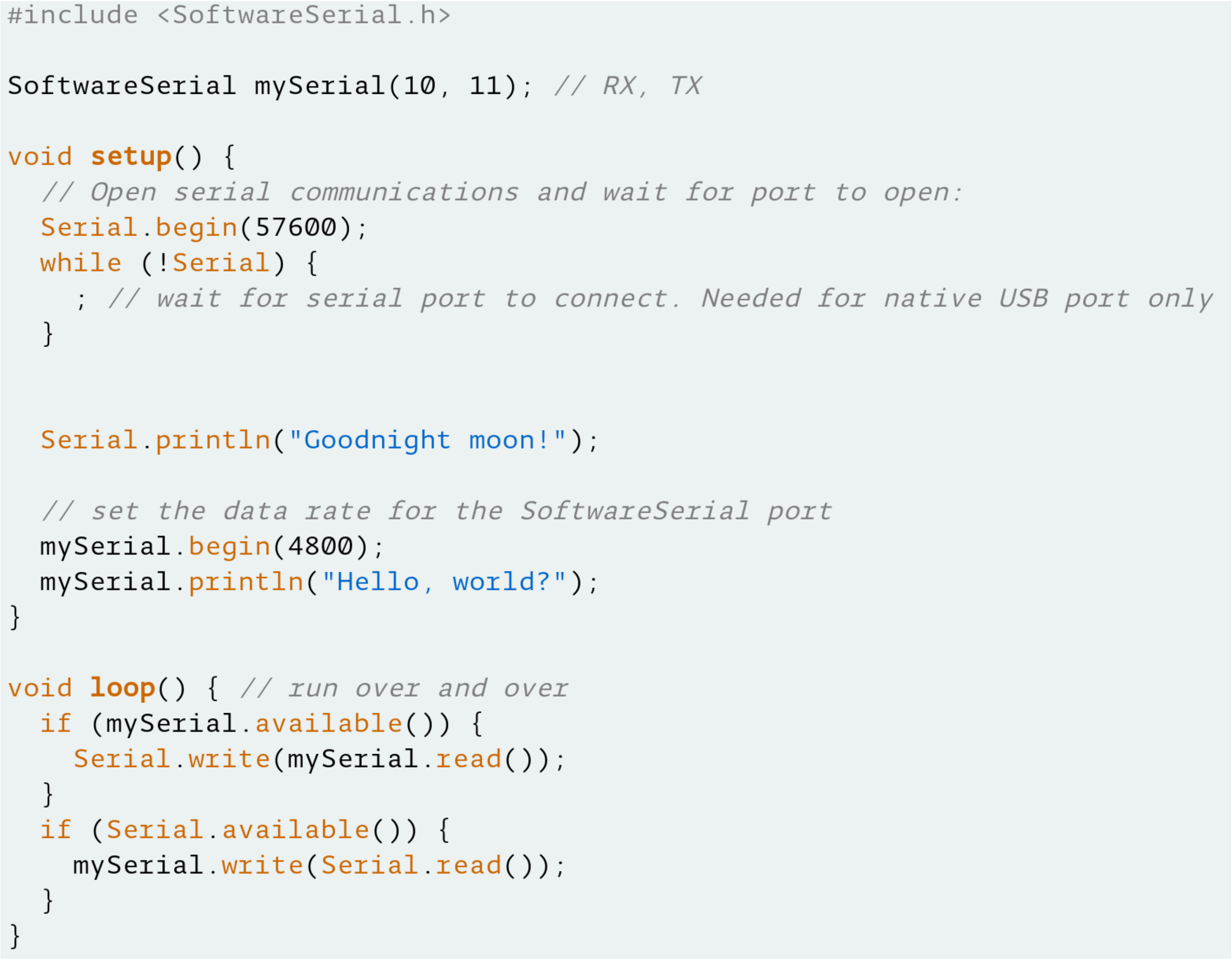
- Speed: 300; 600; 1200; 2400; 4800; 9600; 19200; 38400; 57600; 115200; 230400; 460800; 921600 baud
- Signal levels could be different:



This timing diagram shows both a TTL (bottom) and RS-232 signal sending 0b01010101

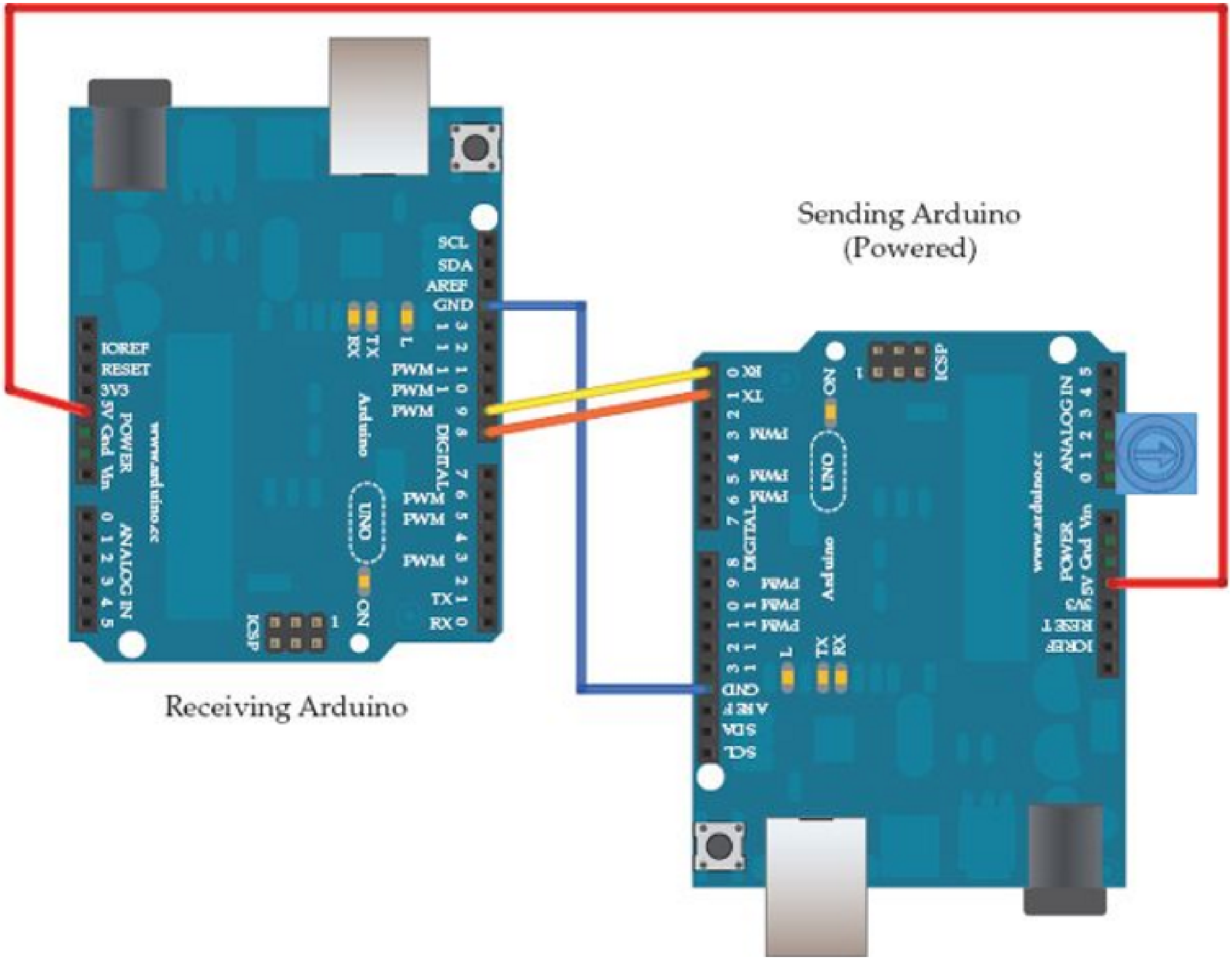
https://www.sparkfun.com/tutorials/215

Arduino SoftwareSerial



```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }


  Serial.println("Goodnight moon!");

  // set the data rate for the SoftwareSerial port
  mySerial.begin(4800);
  mySerial.println("Hello, world?");
}

void loop() { // run over and over
  if (mySerial.available()) {
    Serial.write(mySerial.read());
  }
  if (Serial.available()) {
    mySerial.write(Serial.read());
  }
}
```

UART Arduino-to-Arduino



Useful links

- Simon Monk. Programming Arduino Next Steps.
- https://www.arduino.cc

MSP-430 Overview

MSP-430*

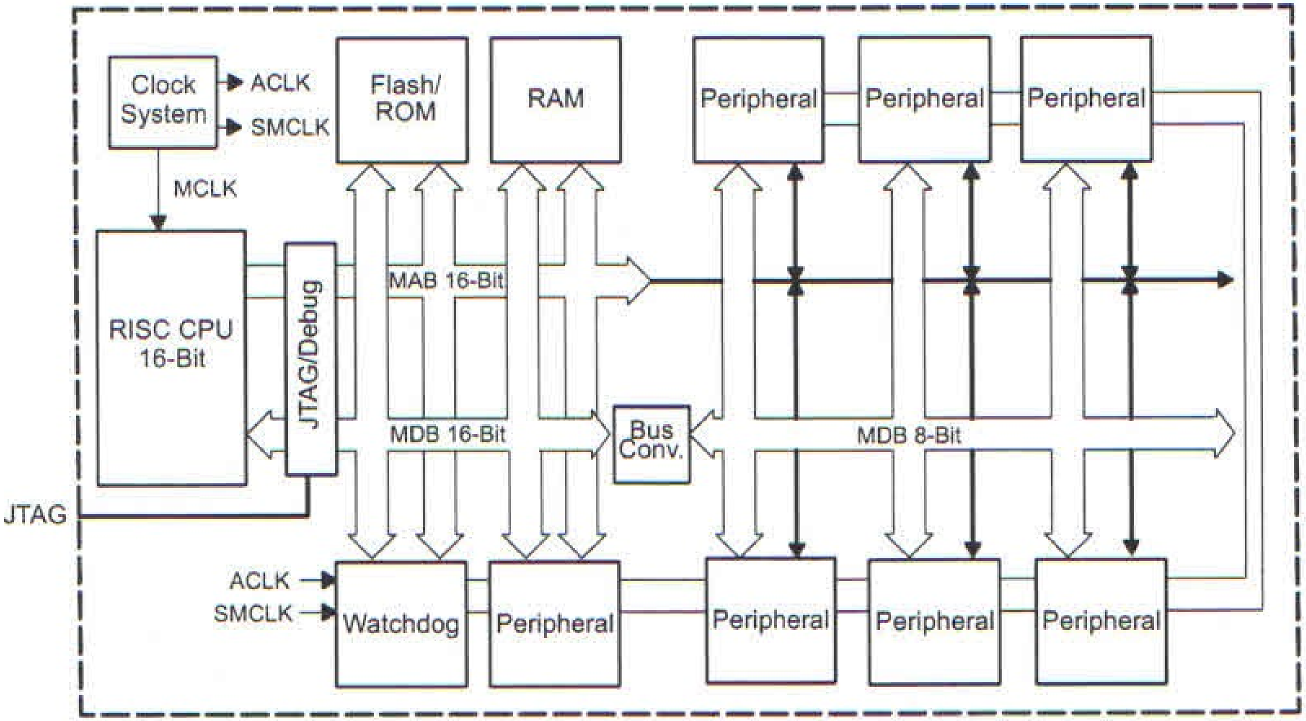
*) Ohio State University ECE 3561 slides have been used

The MSP430

- A von-Neumann style architecture
- Key features
  - Ultralow-power architecture
    - 0.1 uA RAM rentention
    - 0.8 uA real-time clock mode
    - 250 uA/MIPS active
  - High-performance A-to-D conversion
    - 12-bit or 10-bit ADC, 12-bit dual-DAC
    - 200 ksps
  - 16-bit RISC processor features
    - Large Register file
    - Compact code design
    - 27 core instructions
    - 7 addressing modes

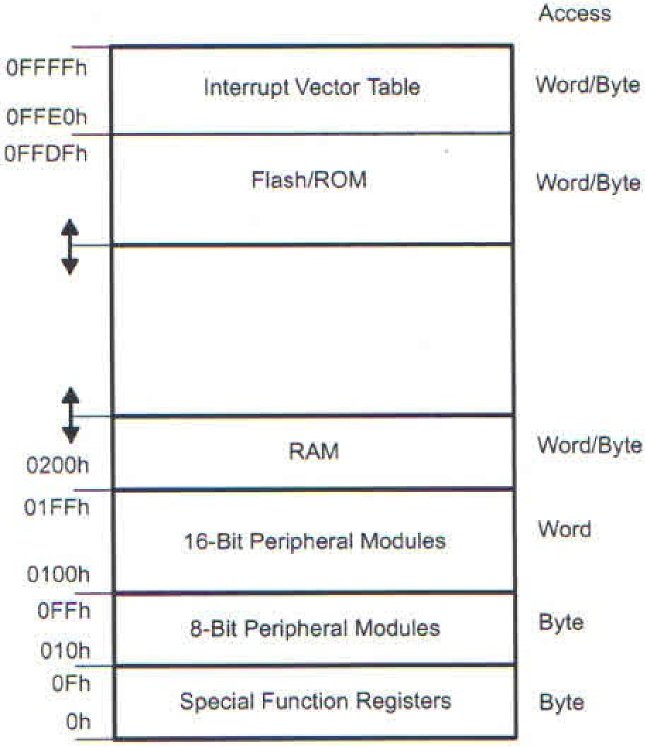
The MSP430

- Block Diagram of internal structure – high level



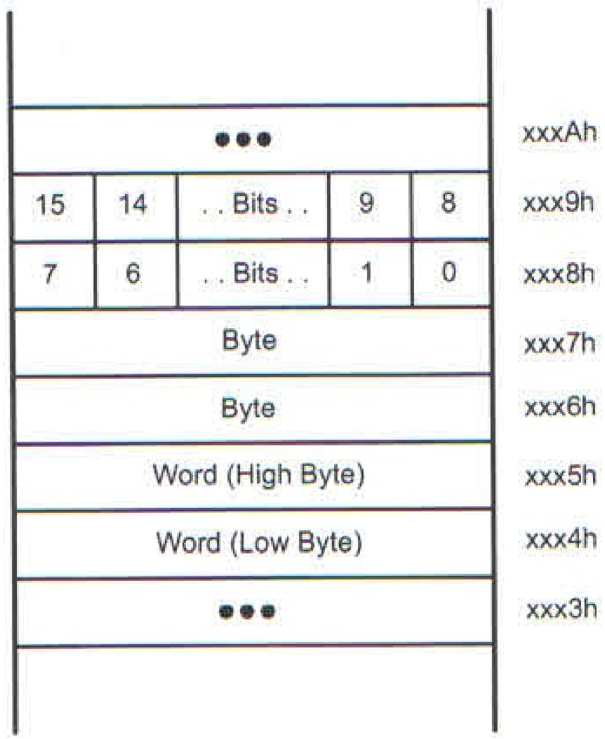
MSP430

- Memory structure – logical
- 16-bit addressable
  - 64K bytes (64KB)
- Amount of Flash/ROM and RAM vary by device
- Last 16 words of Flash/ROM used for the Interrupt Vector Table
- I/O is memory mapped



Memory data organization

- Bytes can be at even or odd addresses
- Words are only at even addresses
  - The low byte of a word is at the even address.
  - The high byte of a word is at the odd address



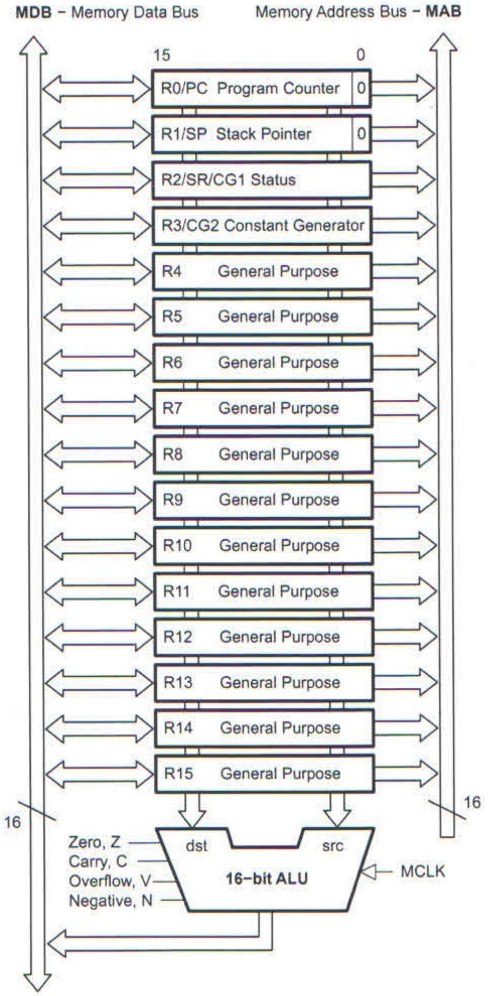
•••					xxxAh
15	14	. . Bits . .	9	8	xxx9h
7	6	. . Bits . .	1	0	xxx8h
Byte					xxx7h
Byte					xxx6h
Word (High Byte)					xxx5h
Word (Low Byte)					xxx4h
•••					xxx3h

The MSP430 CPU

- Incorporates features to support modern programming techniques (Don’t need go to’s)
- The features
  - Calculated branching
  - Table processing
  - 27 RISC instructions
  - 7 addressing modes
  - All instructions use all the addressing modes
  - Full register access
  - Single cycle register operations (RISC)
  - Direct memory-to-memory transfers
  - Constant generator provides most used values

The MSP430 data path

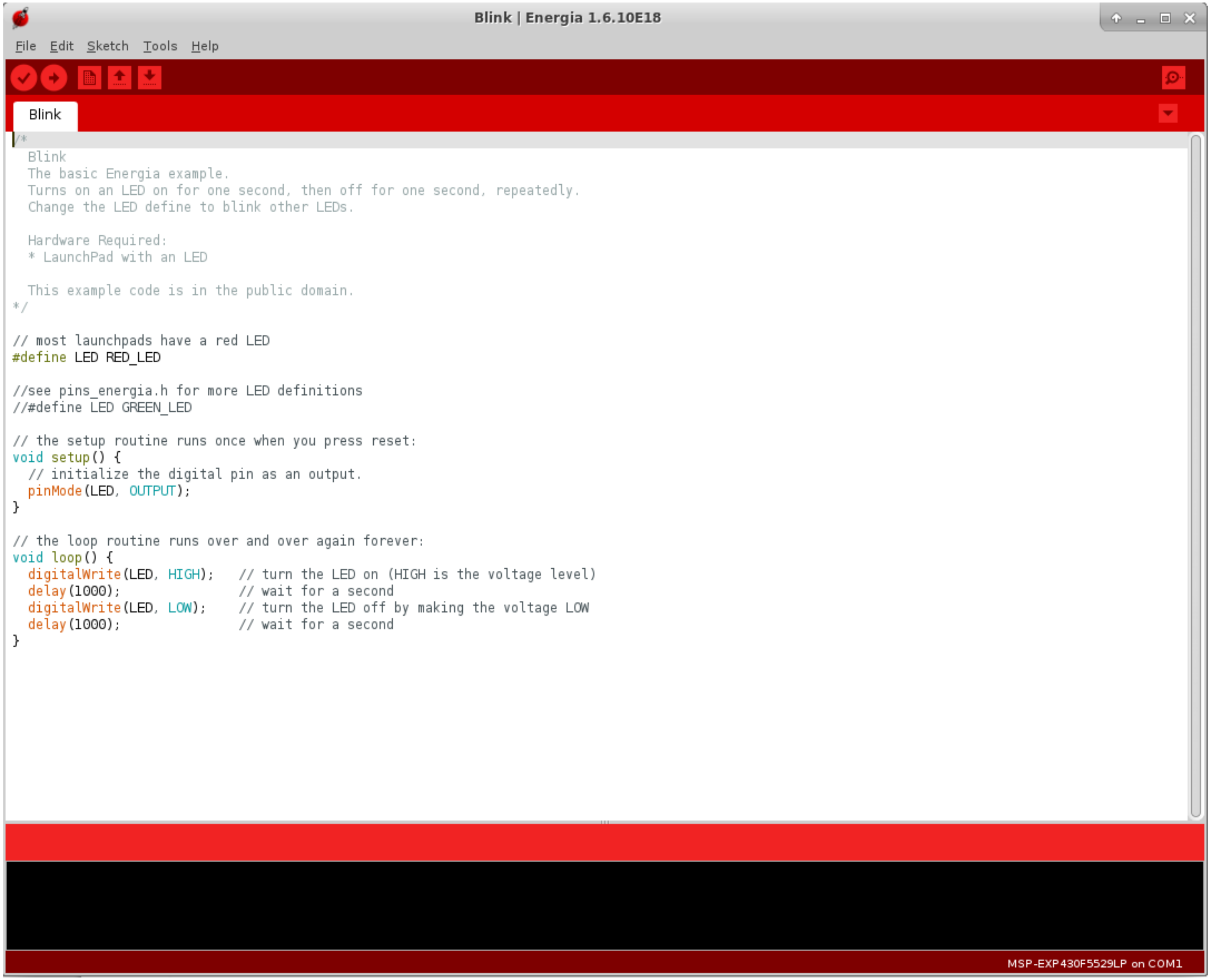
- There are 16 registers
  - Contents are 16-bits
  - User has access to all registers
  - 4 registers are special purpose
- Note bus structure
  - MDB – Memory Data Bus
  - MAB – Memory Address Bus
  - Also have 2 internal bussed to deliver 2 operand to ALU
- Diagram is called the datapath of the processor
- See Users Guide



General purpose registers

- R4 thru R15
  - Registers are indistinguishable
  - Can be used as
    - Data Registers
    - Address Registers
    - Index values
  - Can be accessed with byte or word instructions

Energia IDE



```
/*
  Blink
  The basic Energia example.
  Turns on an LED on for one second, then off for one second, repeatedly.
  Change the LED define to blink other LEDs.

  Hardware Required:
  * LaunchPad with an LED

  This example code is in the public domain.
*/

// most launchpads have a red LED
#define LED RED_LED

//see pins_energia.h for more LED definitions
//#define LED GREEN_LED

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(LED, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```