

Além do Arduino

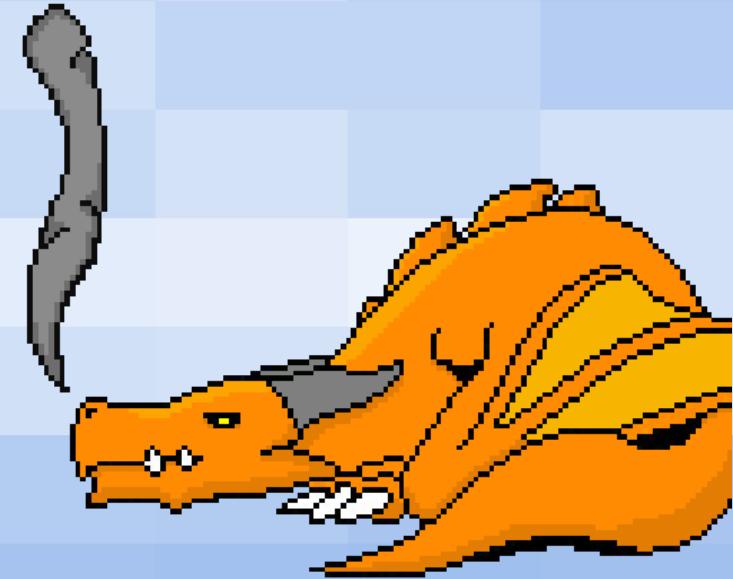
Desenvolvimento de software embarcado com
ferramentas livres

\$ whoami

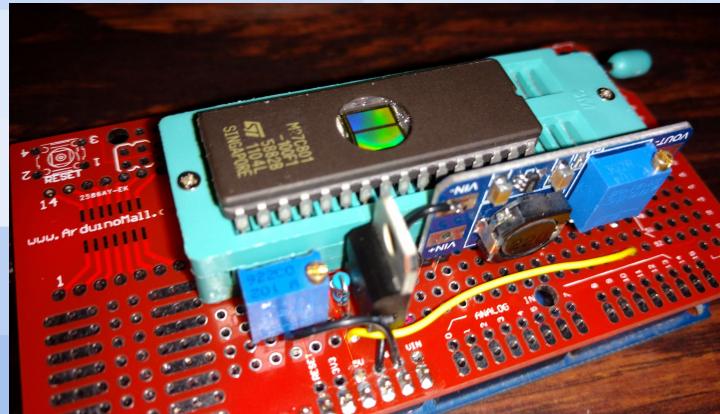
arduino.cc/reference/pt/

dragaosemchama.com/robson/

github.com/robsoncouto



Meus projetos



Conceitos

Software livre [1]

Liberdade 0 - Liberdade para usar o programa para quaisquer propósitos;

Liberdade 1 - Liberdade para estudar o programa e adaptá-lo às suas necessidades. Ter acesso ao código fonte é um requisito.

Liberdade 2 - Liberdade para redistribuir cópias.

Liberdade 3 - Liberdade para melhorar o programa e disponibilizar as melhorias para o público, de forma que toda a comunidade possa se beneficiar disso. Ter acesso ao código fonte é um requisito.



open source
hardware

OSH ou FOSH

Tecnologia física oferecida de forma livre.

O Design do Hardware (desenho mecânico, esquemáticos, lista de materiais, layout da PCI, código HDL e projetos de circuitos integrados), em adição ao software que controla o hardware, distribuídos por licença free/libre.

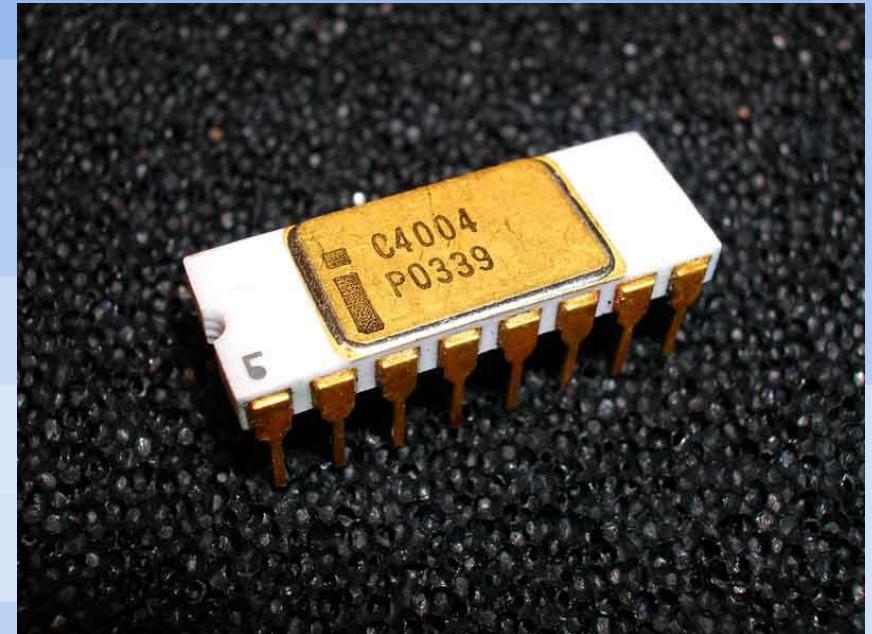
Software embarcado

Software escrito para controlar máquinas ou dispositivos normalmente não vistos como computadores.

- Programa único
- Sistemas operacionais

Microprocessador

Círcuito digital que aceita dados binários como entrada, os processa de acordo com intruções na memória e devolve resultados como saída

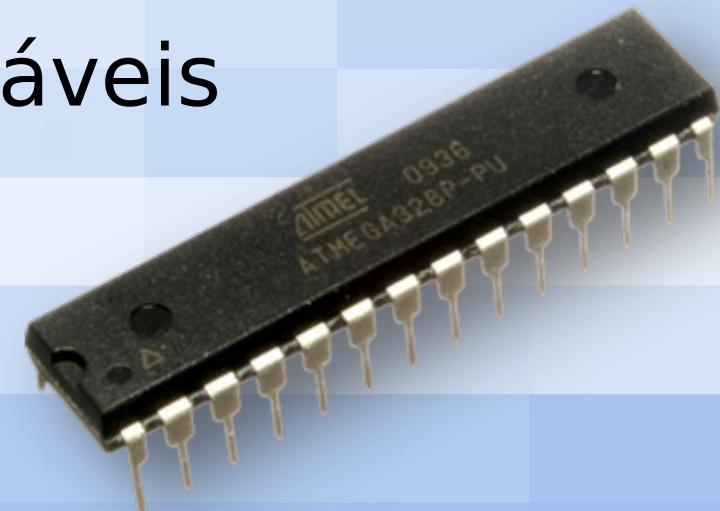


Intel 4004 - 4bits @ 740kHz

Microcontrolador

Um único circuito integrado
com funções computacionais

Inclui CPU, memória, e
periféricos de IO programáveis



SoC

Círcuito integrado com os componentes de um computador.

Inclui componentes avançados como GPUs.

Não necessariamente possui memória integrada.



Arduino

Arduino

Arduino é uma plataforma de computação física open-source baseada em uma placa simples com I/O e uma IDE que implementada nas linguagens Processing/Wiring.

Os sketches são escritos em C++

Compilação do Arduino

Muitas coisas acontecem antes de o seu sketch ser transferido para a placa^[2].

O processo explicado a seguir é válido para as placas Arduino baseadas em micros AVR. Por exemplo: UNO, Nano, Mega, Micro e mini.

Pre-Processamento

- Todos os arquivos .ino na pasta são concatenados em um arquivo cpp;
- Se não estiver presente, “#include <Arduino.h>” é adicionado ao sketch;
- Propótipos são gerados para todas as funções que não os tem.

Compilação

Arquivos com configuração da placa (mcu, memória etc), bibliotecas, e arquivos do core (Serial, IO etc) são utilizados no processo de compilação.

Os arquivos .c e .cpp são compilados separadamente em arquivos objeto .o

Objetos compilados anteriormente são reutilizados para acelerar o processo de compilação.

Linking



Os arquivos .o são linkados ao sketch.

Apenas partes das bibliotecas necessárias ao seu programa são linkadas no arquivo final.

O final final possui a extensão .hex

Esse é o arquivo que vai ser usado para gravação.

Gravação

O arquivo hex^[3] não é um arquivo binário
Este é interpretado pelo avrdude que então
faz a gravação do microcontrolador com o
novo programa.

```
avrdude -v -patmega328p -carduino -P/dev/ttyACM0  
-b115200 -D -Uflash:w:/Blink.ino.hex:i
```

Testes

Arduino vs C puro

Placa Arduino UNO

- Atmega328
- 2KB de SRAM
- 16Mhz

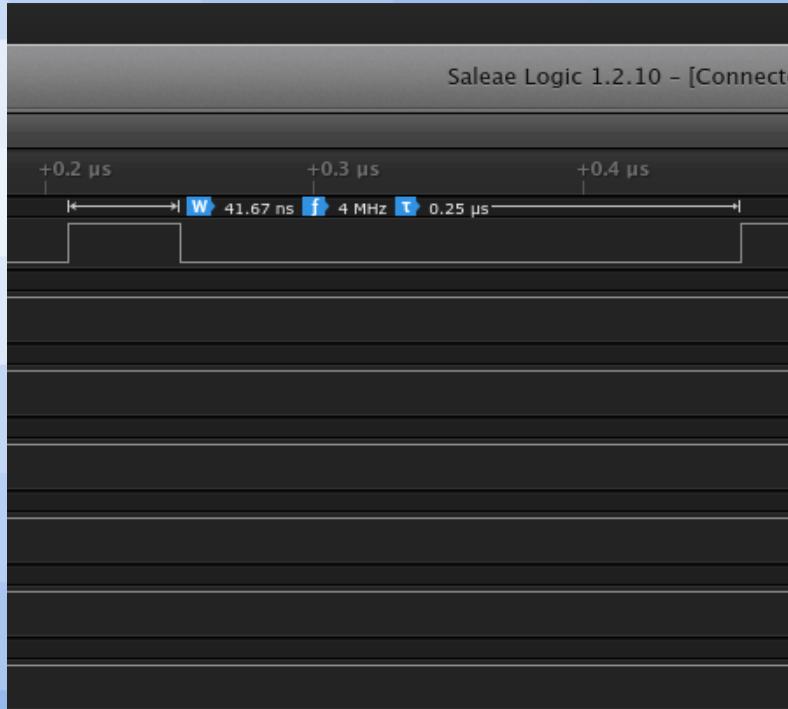
Código

```
void setup() {  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13, HIGH);  
    digitalWrite(13, LOW);  
}
```

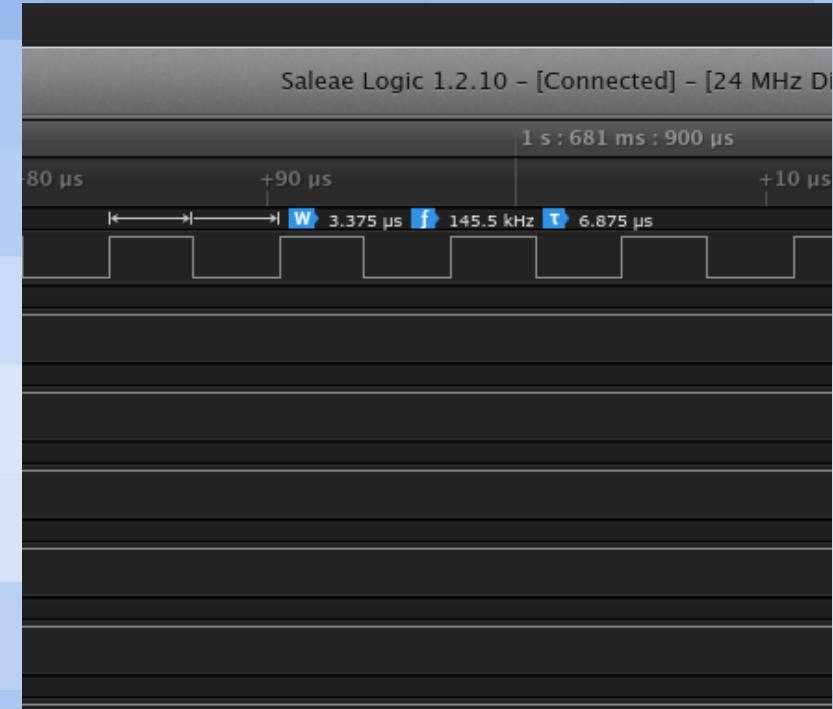
Código

```
void setup() {                                #include<avr/io.h>
    pinMode(13, OUTPUT);                      #include<util/delay.h>
}
int main(void){
    DDRB=(1<<PB5);
    while(1){
        PORTB=(1<<PB5);
        PORTB=(0<<PB5);
    }
}
```

Frequênciá Max de IO



C puro – 4 MHz



Arduino - 145kHz

Espaço na Flash e RAM

avr-size main.hex

text	data	bss	dec	hex	filename
142	0	0	142	8e	main.elf

Sketch uses 734 bytes (2%) of program storage space. Maximum is 32256 bytes.

Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.

```
void digitalWrite(uint8_t pin, uint8_t val)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *out;

    if (port == NOT_A_PIN) return;

    // If the pin that support PWM output, we need to turn it off
    // before doing a digital write.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);

    out = portOutputRegister(port);

    uint8_t oldSREG = SREG;
    cli();

    if (val == LOW) {
        *out &= ~bit;
    } else {
        *out |= bit;
    }

    SREG = oldSREG;
}
```

[4]

- Escrever um programa em C não é simples.
- Pode tomar muito tempo “reinventando a roda”.
- Não há a mesma disponibilidade de bibliotecas.
- Permite controle completo do hardware.
- Acesso a toda a família de microcontroladores.

avr-gcc

Ferramentas

Distro linux

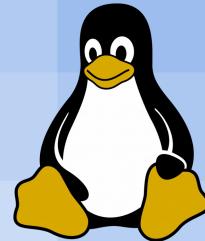
Terminal

Editor de texto

Toolchain (compilador, linker, etc)

Make

Gravador



Distro

Sistema operacional GNU/Linux



Editor de texto

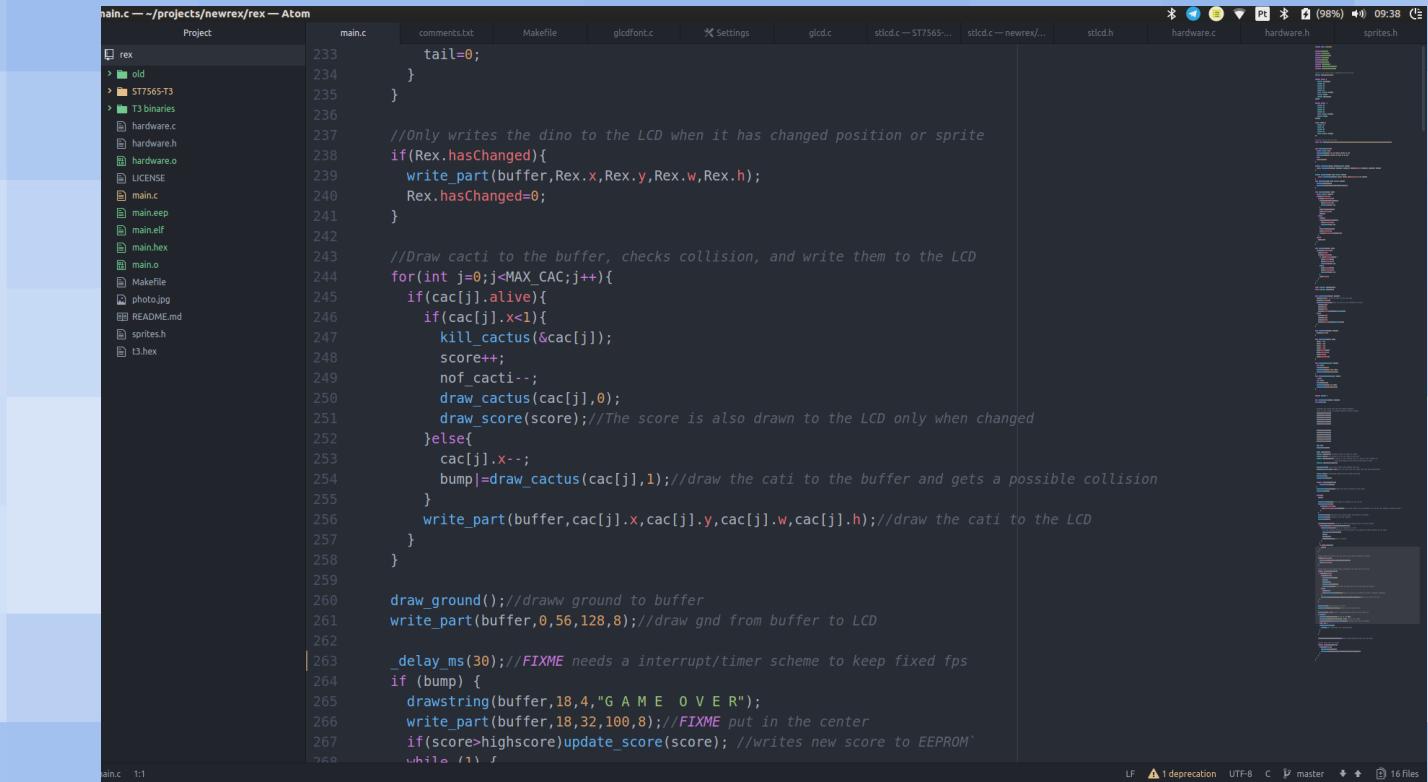
→Nano

→Vim

→Emacs

→Atom

→vscode



The screenshot shows the Atom text editor interface with a dark theme. The main area displays a C program named `main.c`. The code is for a game, likely Flappy Bird, involving a dino, cacti, and a ground. It uses functions like `draw_cactus`, `draw_score`, and `draw_gnd`. The editor includes a sidebar with project files like `Makefile`, `glcdfont.c`, and `hardware.h`. A status bar at the bottom shows file statistics: LF, 1 deprecation, UTF-8, C, master, 16 files.

```
main.c -- ~/projects/newrex/rex -- Atom
Project main.c comments.txt Makefile glcdfont.c Settings glcd.c stlcd.c - ST7365... stlcd.c - newrex/... stlcd.h hardware.c hardware.h sprites.h
└── rex
    ├── old
    ├── ST7365-T3
    ├── T3 binaries
    ├── hardware.c
    ├── hardware.h
    ├── hardware.o
    └── LICENSE
        ├── main.c
        ├── main.eep
        ├── main.elf
        ├── main.hex
        ├── main.o
        └── Makefile
        photo.jpg
        README.md
        sprites.h
        t3.hex

233     tail=0;
234 }
235
236 //Only writes the dino to the LCD when it has changed position or sprite
237 if(ReX.hasChanged){
238     write_part(buffer,ReX.x,ReX.y,ReX.w,ReX.h);
239     ReX.hasChanged=0;
240 }
241
242 //Draw cacti to the buffer, checks collision, and write them to the LCD
243 for(int j=0;j<MAX_CAC;j++){
244     if(cac[j].alive){
245         if(cac[j].x>1){
246             kill_cactus(&cac[j]);
247             score++;
248             nof_cacti--;
249             draw_cactus(cac[j],0);
250             draw_score(score); //The score is also drawn to the LCD only when changed
251         }else{
252             cac[j].x--;
253             bump|=draw_cactus(cac[j],1); //draw the cati to the buffer and gets a possible collision
254         }
255         write_part(buffer,cac[j].x,cac[j].y,cac[j].w,cac[j].h); //draw the cati to the LCD
256     }
257 }
258
259 draw_ground(); //draw ground to buffer
260 write_part(buffer,0,56,128,8); //draw gnd from buffer to LCD
261
262 _delay_ms(30); //FIXME needs a interrupt/timer scheme to keep fixed fps
263 if (bump) {
264     drawString(buffer,18,4,"G A M E   O V E R");
265     write_part(buffer,18,32,100,8); //FIXME put in the center
266     if(score>highscore)update_score(score); //writes new score to EEPROM
267     while(1) s
268 }
```

Toolchain

GNU make

GNU Compiler Collection (GCC)

GNU Binutils: linker, assembler e outros

GNU Debugger (GDB): debug de código

gcc-avr

```
sudo apt-get install gcc-avr binutils-avr  
gdb-avr avr-libc avrdude
```

Em distribuições não baseadas no debian, usar o gerenciador de pacotes respectivo

Pode ser compilado do fonte. Boa sorte!

Código de Exemplo

```
1 #include<avr/io.h>
2 #include<util/delay.h>
3
4 int main(void){
5     DDRB=(1<<PB5); //pino 5 da porta B como saída. 1=saida, 0=entrada
6     while(1){
7         PORTB|=(1<<PB5); //liga o pino PB5 (5V)
8         _delay_ms(1000); //espera 1000ms (1s)
9         PORTB&=~(1<<PB5); // desliga o pino PB5 (0V)
10        _delay_ms(1000);
11    }
12 }
```

Do datasheet^[5]:

14.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description" on page 91, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

Operações Bitwise

```
#define PB5 5
```

```
PORTB |= (1<<PB5);
```

```
DDRB=(1<<PB5);
```

Equivale a:

Equivale a:

```
PORTB = PORTB | (1<<5)
```

```
DDRB = 00100000;
```

Ou:

```
PORTB = PORTB | 00100000;
```

Operações Bitwise

`PORTB&=~(1<<PB5);`

Equivale a:

`PORTB = PORTB & ~ (1<<5)`

Ou:

`PORTB = PORTB & 11011111;`

`PORTB^=(1<<PB5);`

Equivale a:

`PORTB = PORTB ^ (1<<5)`

$X \wedge 1 = \sim X$

Compilação

Programa simples no computador

```
$ cc main.c -o main
```

```
$ ./main
```

Programa simples para micro

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

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

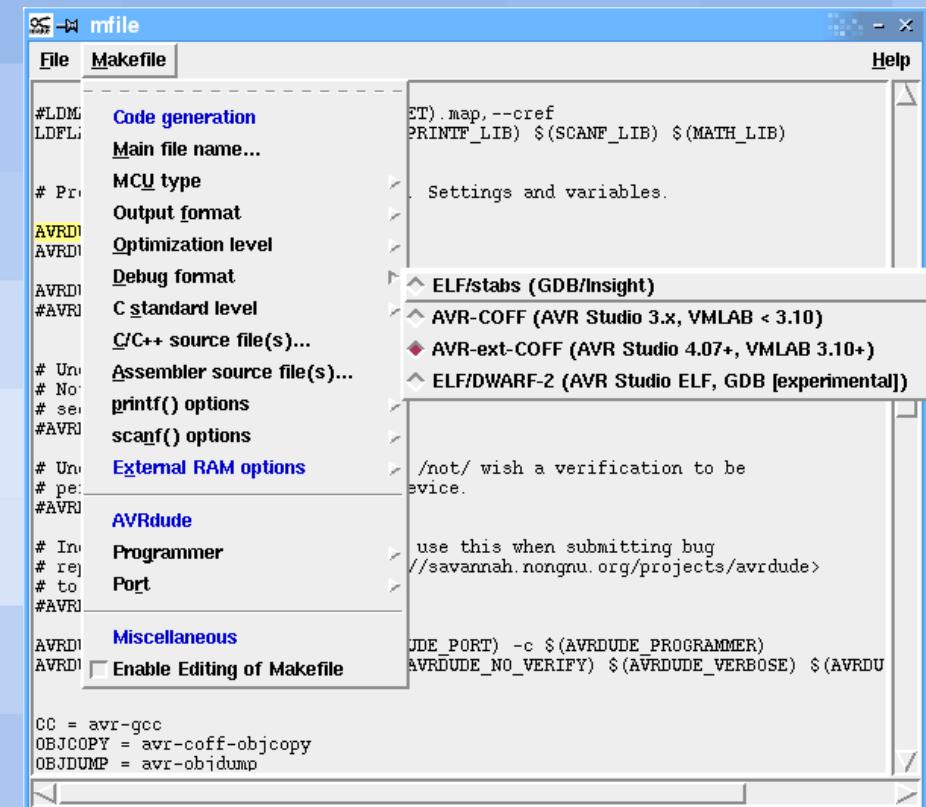
```
$ avr-objcopy -O ihex -R .eeprom led led.hex
```

Make

- Automação do processo de compilação
- Uso de arquivos de texto chamados Makefiles que indicam ao make como compilar e linkar um programa
- A ferramenta make determina automagicamente quais arquivos devem ser recompilados

Exemplo Makefile [6] [7]

```
PKG=led
BIN=${PKG}
OBJS=${PKG}.o
MCU=atmega328p
CC=avr-gcc
OBJCOPY=avr-objcopy
CFLAGS=-Os -DF_CPU=16000000UL
-mmcu=${MCU} -Wall
PORT=/dev/ttyACM0
${BIN}.hex: ${BIN}.elf
    ${OBJCOPY} -O ihex $< $@
${BIN}.elf: ${OBJS}
    ${CC} -mmcu=${MCU} -o $@ $^
```



Gravação

- O avrdude^[8] é uma ferramenta para leitura, escrita e manipulação da memória de micros AVR
- Suporta dezenas de gravadores, incluindo usbasp, usbtiny, o bootloader do arduino, avrisp entre outros

Gravação:

```
avrdude -c usbasp -p atmega328p -U flash:w:main.hex
```

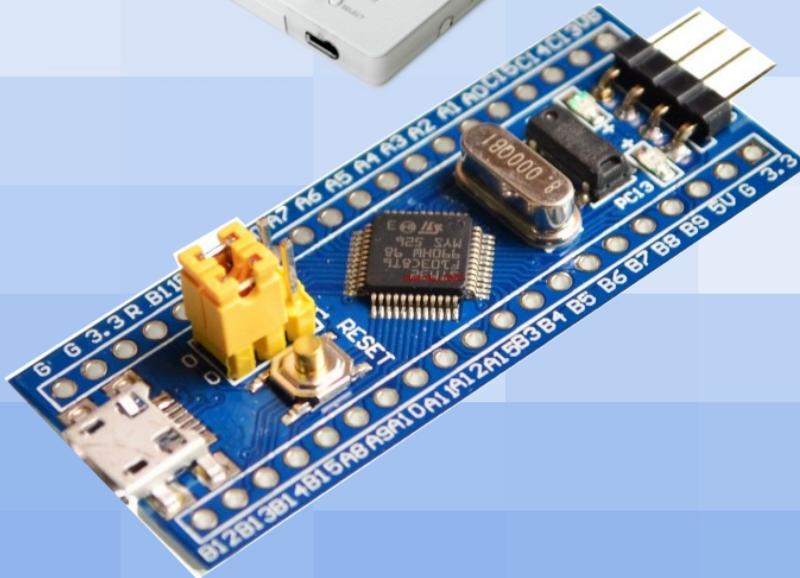
Leitura:

```
avrdude -c usbtiny -p atmega328p -U flash:r:dump.hex:r
```

ARM

Microprocessadores ARM

- Processadores ARM são cores de propriedade intelectual.
- Usados em inúmeros produtos, incluindo smartphones, SBCs (raspberry pi, beaglebone etc) e consoles portáteis
- 32 ou 64 bit
- Centenas de MHz



Apesar de estes processadores serem mais complexos que os anteriores, pode-se utilizar o mesmo processo para se compilar código para os mesmos.

arm-none-eabi-gcc^[9]

```
apt-get install gcc-arm-none-eabi
```

Suporta Cortex-R/Cortex-M, incluindo Cortex-M0, Cortex-M3, Cortex-M4, Cortex-M0+, Cortex-M7, Armv8-M Baseline e Mainline, Cortex-R4, Cortex-R5, Cortex-R7 e Cortex-R8

Libopencm3 [10]

Biblioteca para vários microcontroladores Cortex-M

- ST STM32F0xx/F1xx/F2xx/F30x/F37x/F4xx/F7xx/L0xx/L1xx/L4xx
- Atmel SAM3A/3N/3S/3U/3X e também SAMDxx
- NXP LPC1311/13/17/42/43
- Stellaris LM3S series
- TI (Tiva) LM4F series
- EFM32 Gecko series (only core support)
- Freescale Vybrid VF6xx

Discovering the stm32^[11]

Livro de Geoffrey Brown disponível gratuitamente que trata detalhadamente microcontroladores STM da arquitectura ARM Cortex M3

GCC + Standard Peripheral Library da STM

Material e slide



<https://github.com/robsoncouto/flisol2018>

- [1] - <https://www.gnu.org/philosophy/free-sw.html>
- [2] - <https://github.com/arduino/Arduino/wiki/Build-Process>
- [3] - https://en.wikipedia.org/wiki/Intel_HEX
- [4] - https://github.com/arduino/Arduino/blob/master/hardware/arduino/avr/cores/arduino/wiring_digital.c
- [5] - http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf
- [6] - <https://arduino.stackexchange.com/questions/12114/basic-makefile-for-avr-gcc>
- [7] - <http://www.sax.de/~joerg/mfile/>
- [8] - <http://www.nongnu.org/avrdude/>
- [9] - <https://launchpad.net/gcc-arm-embedded>
- [10] - <https://github.com/libopencm3/libopencm3>
- [11] - <https://www.cs.indiana.edu/~geobrown/book.pdf>