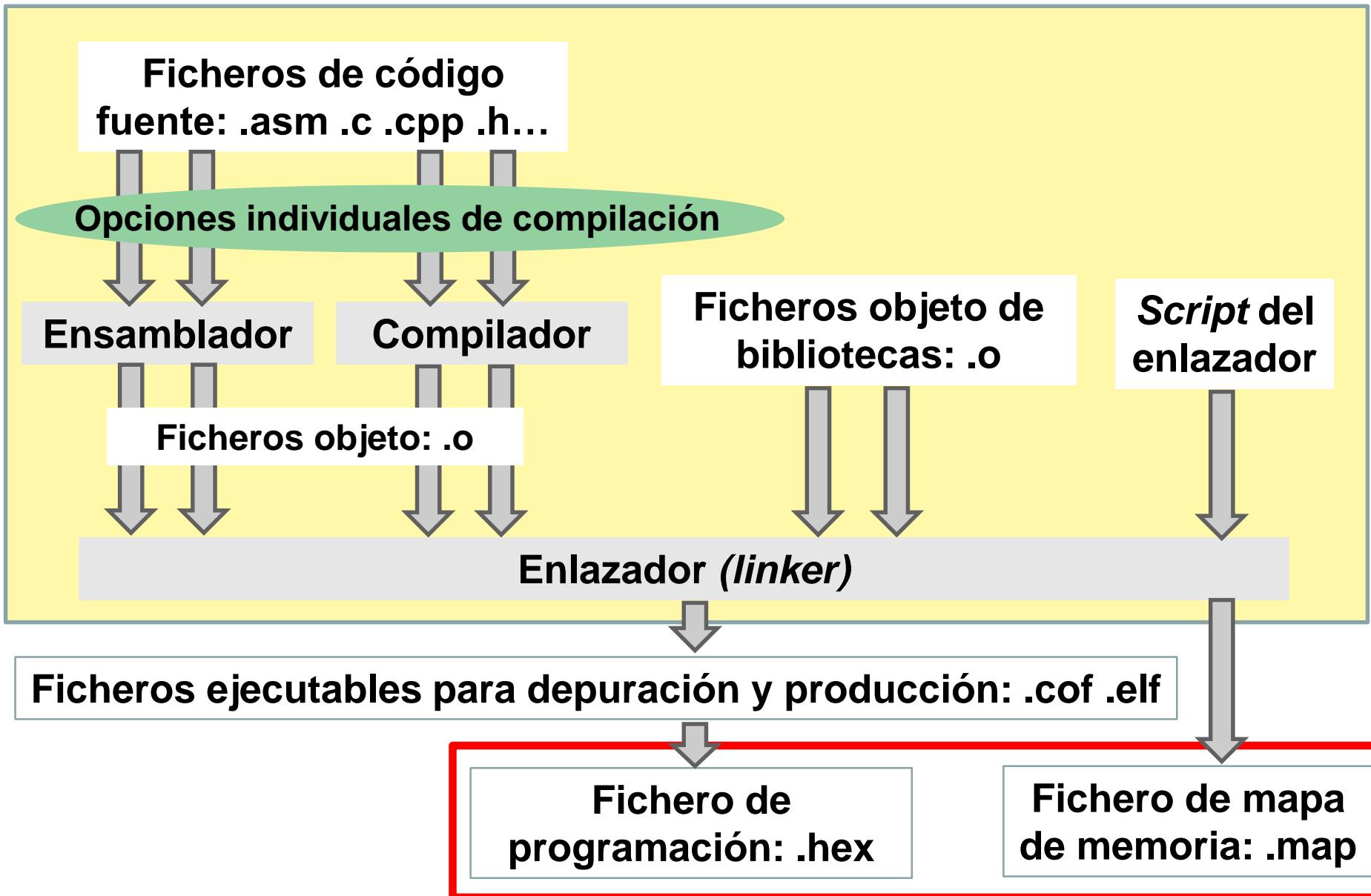




## Sistemas con Microprocesadores

### 5. Construcción de aplicaciones y programación del microcontrolador

# Construcción (*build*) de una aplicación



# Ejemplo compilación: parpadeo-main.c

```
#include <avr/io.h>
#define F_CPU 16000000UL
#ifndef __DELAY_BACKWARD_COMPATIBLE__
#include <util/delay.h>

// LED conectado a patilla PB5 en Arduino UNO
#define DDR_PUERTO_LED DDRB
#define PUERTO_LED PORTB
#define BIT_LED (1 << PB5) // 0b00100000

uint8_t Retardos[] = {1,2,4,6,8};
uint8_t Indice_retardo;

void retardo_x100ms(uint8_t centenas_ms)
{
    uint8_t ret_ind;
    for(ret_ind=0; ret_ind<centenas_ms; ret_ind++)
        _delay_ms(100);
}
```

```
void parpadea(void)
{
    PUERTO_LED = BIT_LED;
    retardo_x100ms(Retardos[Indice_retardo]);
    PUERTO_LED = 0;
    retardo_x100ms(Retardos[Indice_retardo]);

    Indice_retardo++;
    if(Indice_retardo >= sizeof(Retardos))
        Indice_retardo = 0;
}

int main(void)
{
    DDR_PUERTO_LED = BIT_LED;
    Indice_retardo = 2;
    while (1)
        parpadea();
}
```

# parpadeo.X.production.map

- Este fichero de texto es útil para comprobar el tamaño ocupado en memoria por variables, constantes y código.
- En este caso el fichero tiene 520 líneas y varias partes:

Memory Configuration				
Name	Origin	Length	Attributes	
text	0x0000000000000000	0x0000000000000000	<b>8000</b>	xr
data	0x000000000800100	0x0000000000000000	<b>800</b>	rw !x
eeprom	0x000000000810000	0x0000000000000000	<b>400</b>	rw !x
fuse	0x000000000820000	0x0000000000000000	<b>3</b>	rw !x
lock	0x000000000830000	0x0000000000000000	400	rw !x
signature	0x000000000840000	0x0000000000000000	400	rw !x
user_signatures	0x000000000850000	0x0000000000000000	400	rw !x
*default*	0x0000000000000000	0xffffffffffffffffffff		

- En esta parte se informa de los tamaños de las distintas regiones de memoria, atributos y direcciones de inicio (asumidos por el enlazador):

text: almacena código (FLASH)

data: almacena variables (RAM)

eeprom: espacio de la EEPROM

fuse: bits de configuración

lock: bits config. bloqueo acces. mem.

signature: identifica el dispositivo

# parpadeo.X.production.map

- En esta parte se detalla la salida del *script* del enlazador (no mostrada) y cómo se han mapeado variables y código en memoria.
  - Variables colocadas en secciones:

## Linker script and memory map

```
[...]  
.bss.Indice_retardo  
          0x00800105      0x1  
.bss.Indice_retardo  
          0x00800105      0x1  [...] /parpadeo-main.o  
          0x000000000800105           Indice_retardo  
[...]  
.data.retardos 0x00800100      0x5 load address 0x00000158  
.data.retardos  
          0x00800100      0x5  [...] /parpadeo-main.o  
          0x00800100           retardos
```

bss: contiene variables estáticas/globales declaradas sin valor (ocupa RAM)

progmemx: contiene constantes (ocuparán solo FLASH)

data: contiene variables estátic./glob. declaradas con valor (RAM y FLASH)

# parpadeo.X.production.map

- Mapeo del código: Dirección y tamaño de funciones, rutinas y vectores de interrupción:

## Linker script and memory map

[...]

.text.parpadea	0x000000c8	0x3c
.text. <b>parpadea</b>	0x000000 <b>c8</b>	<b>0x3c</b> [...] /parpadeo-main.o
	0x000000c8	parpadea

[...]

.text.retardo_x100ms	0x00000104	0x20
.text. <b>retardo_x100ms</b>	0x00000 <b>104</b>	<b>0x20</b> [...] /parpadeo-main.o
	0x00000104	retardo_x100ms

[...]

.text.main	0x0000013a	0x10
.text. <b>main</b>	0x00000 <b>13a</b>	<b>0x10</b> [...] /parpadeo-main.o
	0x0000013a	main

[...]

(Tamaños y direcciones en bytes (no en las palabras de 16 bits de la FLASH))

# parpadeo.X.production.hex

- Este tipo de fichero almacena el contenido a programar en microcontroladores, EEPROM, y otros dispositivos.
- Codifica información binaria usando texto.
- Fue diseñado por Intel en 1973.
- Existen otros formatos alternativos. Ej.: el SREC de Motorola.
- Cada línea codifica una serie de bytes en hexadecimal usando el siguiente formato:

:<cuenta de bytes><dirección><tipo de registro><datos><suma de control>

<u>Campo</u>	<u>Tamaño</u>	<u>Descripción</u>
:	1 carácter	Código de comienzo del registro
cuenta de bytes	1 byte (2 caract.)	Número de bytes en el campo de datos
dirección	2 bytes	Dirección de memoria (relativa) de comienzo
tipo de registro	1 byte	Código entre 0 y 5. 0=Regist. datos 1=Fin de fichero
datos	n bytes	Secuencia de n bytes
suma de control	1 byte	Valor para comprobar que el fichero no tiene errores

# parpadeo.X.production.hex

- 29 líneas de texto codifican el binario del programa parpadeo-main.c

# Construcción de una aplicación

ARDUINO

Fichero .ino

```
v  
arduino-builder  
|  
| Fichero .ino.cpp  
| |  
| | avr-g++  
| v  
| Fichero .ino.cpp.o  
| |  
| | avr-gcc  
| v  
| Fichero .ino.elf  
| |  
| | avr-objcopy  
| v  
| Fichero .ino.hex  
v
```

avrduude (programación)

MPLAB

Makefile

```
v  
make  
|  
| Fichero .c  
| |  
| | xc8-cc  
| v  
| Fichero .o  
| |  
| | xc8-cc  
| v  
| Fichero .elf  
| |  
| | avr-objcopy  
| v  
| Fichero .hex  
v
```

atprogram (programación)



# Programación del microcontrolador: avrdude

- Programa de línea de comandos para descargar/cargar a la memoria de microcontroladores AVR. Incluyendo:
  - FLASH, EEPROM, *fuse bytes* y *lock bits*.
  - Parámetros:

-C <fich-conf>	Fich. infor. de micros y programadores (avrdude.conf)
-p <microcont>	Identificador del microcontrolador (atmega328p)
-c <programador>	Identificador del programador (arduino)
-P <puesto>	Identificador del puerto paral./serie (virtual) (usb)
-b <baudios>	Velocidad del puerto serie (115200)
-D	Desactiva el borrado automática inicial de la FLASH
-U <mem>:<oper>:<fich>:<fmt>	Especifica la operación <oper> a realizar sobre la memoria <mem> usando el fichero <fich> con formato <fmt> (flash:w:fichero.hex:i)

# avrdude

- Ejemplo de ejecución (parámetros copiados de Arduino IDE):
  - Las rutas y el identificador del puerto serie varían según sistema operat.

```
/[...]/avrdude -C [...]/avrdude.conf -p atmega328p -c arduino -P /dev/cu.usbmodem14101 -b 115200 -D -U flash:w:uart.X.production.hex:i  
avrdude: AVR device initialized and ready to accept instructions  
Reading | ##### | 100% 0.00s  
avrdude: Device signature = 0x1e950f (probably m328p)  
avrdude: reading input file "uart.X.production.hex"  
avrdude: writing flash (12866 bytes):  
Writing | ##### | 100% 2.08s  
avrdude: 12866 bytes of flash written  
avrdude: verifying flash memory against uart.X.production.hex:  
avrdude: load data flash data from input file uart.X.production.hex:  
avrdude: input file uart.X.production.hex contains 12866 bytes  
avrdude: reading on-chip flash data:  
Reading | ##### | 100% 1.65s  
avrdude: verifying ...  
avrdude: 12866 bytes of flash verified  
avrdude: safemode: Fuses OK (E:00, H:00, L:00)  
avrdude done. Thank you.
```

# Bits de configuración (*fuse bits*)

- Configuran opciones que la aplicación no suele cambiar.
- En ATmega328P estos bits están contenidos en 3 bytes.
- Estos bits valen 1 por defecto (cuando están sin programar).

**Byte de configuración extendido (*Extended fuse byte*):**

<u>Bit</u>	<u>Descripción</u>
7	<no usado>
6	<no usado>
5	<no usado>
4	<no usado>
3	<no usado>
2	Nivel de voltaje de activación del <i>Brown-Out Detector</i>
1	"
0	"

# Bits de configuración (*fuse bits*)

## Byte de configuración alto (*High fuse byte*):

<u>Bit</u>	<u>Descripción</u>
7	Desactiva el <i>reset</i> externo (a través de la patilla PC6)
6	Habilita la interfaz debugWIRE
5	Habilita la programación y descarga de datos serie (SPI)
4	Deja siempre activo el perro guardián (reinicia el microcontrolador)
3	Conserva la memoria EEPROM durante el borrado del dispositivo
2	Selecciona el tamaño de memoria reservada para caragador arranque ( <i>boot loader</i> )
1	“”
0	Cambia la posición del vector de <i>reset</i> en memoria

Cuidado con programar los bits 5 y 7 sin querer: pueden dejar el microcontrolador inutilizable.

# Bits de configuración (*fuse bits*)

## Byte de configuración bajo (*Low fuse byte*):

- Selecciona el origen de la señal de reloj y detalles sobre su funcionamiento.

<u>Bit</u>	<u>Descripción</u>
7	Activa la división de la frecuencia de la señal de reloj entre 8
6	Activa la salida de la señal de reloj a través de la patilla PB0
5	Tiempo de puesta en marcha del reloj
4	"
3	Selecciona el origen del reloj (oscilador de cristal, oscilador RC...)
2	"
1	"
0	"

Estos bytes de configuración pueden ser leídos y escritos mediante un cable programador.

A través de código en el microcontrolador solo pueden ser leídos.

# Bytes de firma (signature bytes)

- Todos los microcontroladores Atmel tienen un código de firma de tres bytes que identifica el dispositivo.
- Residen en un espacio de direcciones separado.
- Pueden ser leídos por avrdude o por código en el microcontr.

**Para el ATmega328P  
los bytes de firma son:**

Dirección	Valor
0x000	0x1E
0x001	0x95
0x002	0x0F

## Otros bytes programables

- *Lock bits*: Limitan el acceso a memoria EEPROM y FLASH.
- *Calibration byte*: Valor de calibración para el oscilador RC interno. Reside en espacio de dir. de bytes de firma (solo lect.)

# Comunicación con el PC: usart-main.c

```
#include <avr/io.h>
#define F_CPU 16000000UL
#define BAUD 9600
#include <util/setbaud.h>
#include <util/delay.h>
#include <stdio.h>
void ini_usart(void)
{ // Establece la velocidad
    UBRR0H = UBRRH_VALUE;
    UBRR0L = UBRRL_VALUE;
#if USE_2X
    UCSR0A |= (1 << U2X0);
#else
    UCSR0A &= ~(1 << U2X0);
#endif
    // Activa la transmisión y recepción
    UCSR0B = (1 << TXEN0) | (1 << RXEN0);
    // Configura el carácter a 8 bits y 1 bit d parada
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}
```

```
void envia_car_usart(char car)
{ // Espera a que se libere el buffer
    loop_until_bit_is_set(UCSR0A, UDRE0);
    UDR0 = car; // Envía el carácter
}
char recibe_car_usart(void)
{ // Espera a que haya un caráct. en el buffer
    loop_until_bit_is_set(UCSR0A, RXC0);
    return(UDR0); // Lee el carácter
}
int envia_car_usart_stream(char car, FILE *s)
{
    if (car == '\n')  envia_car_usart('\r');
    envia_car_usart(car);
    return(0);
}
int recibe_car_usart_stream(FILE *s)
{
    return((int)recibe_car_usart());
}
```

# uart-main.c (2)

```
FILE usart_dev = FDEV_SETUP_STREAM(envia_car_usart_stream, recibe_car_usart_stream,
_FDEV_SETUP_WRITE);
```

```
int main(void)
{
    ini_usart();
    stdout = &usart_dev;
    stdin = &usart_dev;

    // Configura la patilla del led como salida
    DDRB = (1 << PB5);
    while(1)
    {
        int valor_led = 0;
        printf("Valor para el led (0 o 1): ");
        scanf("%i", &valor_led);
        PORTB = (valor_led==0)? 0 : (1 << PB5);
        _delay_ms(1);
        printf("\nLed a: %x\n", (PINB & (1 << PB5)) != 0);
    }
}
```

```
> screen /dev/cu.usbmodem14101 9600
```

Valor para el led (0 o 1):

Led a: 1

Valor para el led (0 o 1):

Led a: 0

Valor para el led (0 o 1):