Microcontroladores: Laboratorio 1

1st Mateo Lecuna
Ingeniería en Mecatrónica
Universidad Tecnológica (UTEC)
Fray Bentos, Uruguay
mateo.lecuna@estudiantes.utec.edu.uy

2nd Mateo Sanchez
Ingeniería en Mecatrónica
Universidad Tecnológica (UTEC)
Maldonado, Uruguay
mateo.sanchez@estudiantes.utec.edu.uy

***Resumen*—Se presenta el diseño e implementación de cuatro subsistemas sobre el microcontrolador ATmega328P: (1) un plotter cartesiano programado en C, (2) un selector/detector de color que explora un círculo cromático, (3) un piano electrónico, y (4) una cerradura electrónica. Cada módulo ejercita competencias clave de sistemas embebidos: planificación de trayectorias y temporización (plotter), adquisición/filtrado analógico y mapeo a acciones (LDR→servo→LED), generación de señales y lectura confiable de entradas (piano), y control secuencial por máquina de estados con validación de credenciales (cerradura). Se verificó el funcionamiento estable de los módulos, con lecturas analógicas consistentes, posicionamiento del servo acorde a la detección, generación de tonos precisa y lógica de acceso confiable. Como líneas de mejora se identifican: perfiles de movimiento (trapezoidal/S-curve) para el plotter, calibración y normalización de la LDR frente a iluminación ambiente, antirrebote/antighosting en el teclado del piano, y mecanismos adicionales de seguridad en la cerradura (bloqueo por intentos, almacenamiento en EEPROM).**

Keywords: ATmega328P, sistemas embebidos, lenguaje C, plotter cartesiano, detección de color, servomotor, piano electrónico, cerradura electrónica.

I. Introducción

Objetivos específicos

II. Marco Teórico

III. Metodología

IV. Resultados

IV-A. *Plotter*

IV-B. *Seleccionador de colores*

IV-C. *Piano*

IV-D. *Cerradura electrónica*

V. Conclusiones

V-A. *Plotter*

V-B. *Seleccionador de colores*

V-C. *Piano*

V-D. *Cerradura electrónica*

Referencias

[1] ARXterra. Addressing modes — 8-bit avr instruction set. [Online]. Available: https://www.arxterra.com/3-addressing-modes/

[2] Carpeta del laboratorio (google drive). Carpeta compartida del laboratorio. [Online]. Available: https://drive.google.com/drive/u/0/folders/1fP0aILozXeapRgDPDNWT1TRhAYr1PPPT

[3] Microchip Technology Inc. Atmega328p datasheet. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[4] circuito.io. (2018) Arduino uno pinout diagram. [Online]. Available: https://www.circuito.io/blog/arduino-uno-pinout/

[5] Software Particles. (2024, Apr.) Learn how an 8×8 led display works and how to control it using an arduino. Figura: 8×8 LED matrix pin mapping (imagen del artículo). [Online]. Available: https://softwareparticles.com/learn-how-an-8x8-led-display-works-and-how-to-control-it-using-an-arc

[6] Microchip Community (AVR Freaks). Avr freaks — comunidad de desarrolladores avr. Foros técnicos y soluciones prácticas sobre AVR. [Online]. Available: https://www.avrfreaks.net/

[7] J. Ganssle. A guide to debouncing. Referencia clásica para anti-rebote en pulsadores/encoders. [Online]. Available: https://www.ganssle.com/debouncing.htm

[8] G. Schmidt. (2021, Sep.) Beginners introduction to the assembly language of atmel-avr-microprocessors. Tutorial de avr-asm-tutorial.net (revisión septiembre 2021). [Online]. Available: https://kitsandparts.com/tutorials/assemblers/BeginnersAVRasm.pdf

[9] T. Redelberger. (2019, Apr.) Avr assembler for complex projects. Versión 0.4 (2019-04-06). [Online]. Available: https://web222.webclient5.de/doc/swdev/avrasm/advavrasm2/AdvancedUseAVRASM2_en_20190406.pdf

[10] Laboratorio de Microcontroladores, UTEC, "Repositorio de laboratorio de microcontroladores (tec.micro)," https://github.com/MateoLecuna/Tec.Micro, 2025, accedido el 26 de septiembre de 2025.

VI. Anexos

VI-A. *Carpeta de laboratorio*

Enlace de acceso a la carpeta de Google Drive con simulaciones y evidencias del laboratorio.

VI-B. *Repositorio del Laboratorio*

Repositorio de Laboratorio de Microcontroladores (Tec.Micro) [10]

VI-C. *Microcontrolador ATmega328P*

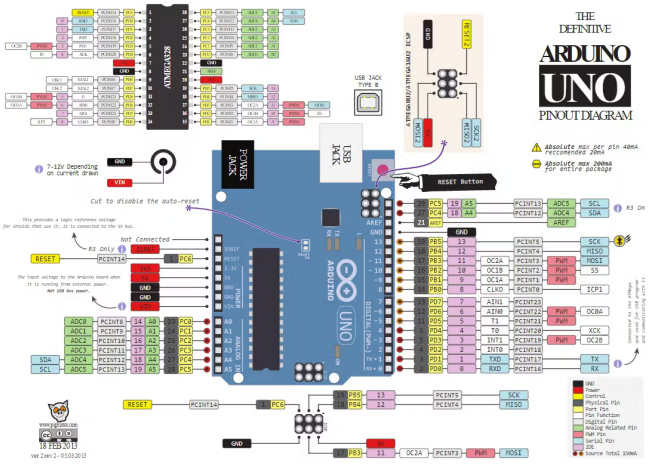

Figura 1. Diagrama de pines del Arduino Uno. Fuente: [4].

VI-D. *Matriz de LEDs 1088AS*

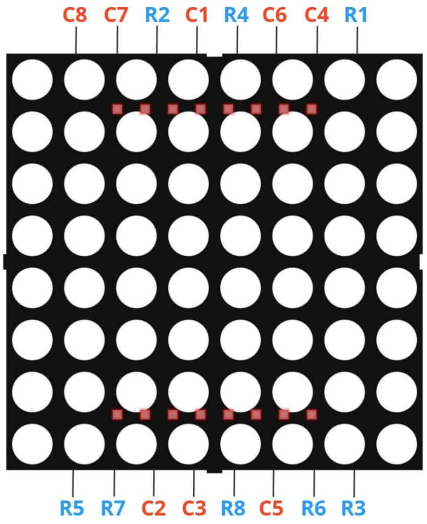

Figura 2. Pinout de Matriz de LEDs 1088AS. Fuente: [5].

VI-E. Comandos Plotter

Pin Digital	Conexión
D2	Bajar solenóide X0
D3	Subir solenóide X1
D4	Movimiento hacia abajo X5
D5	Movimiento hacia arriba X6
D6	Movimiento hacia la izquierda X7
D7	Movimiento hacia la derecha X10

Figura 3. Comandos del Plotter del laboratorio de Mecatrónica. Fuente: Hoja de laboratorio.

VI-F. Señal para conversor digital-analógico

```
0x00,0x08,0x10,0x18,0x20,0x28,0x30,0x38,
0x40,0x48,0x50,0x58,0x60,0x68,0x70,0x78,
0x80,0x88,0x90,0x98,0xa0,0xa8,0xb0,0xb8,
0xc0,0xc8,0xd0,0xd8,0xe0,0xe8,0xf0,0xf8,
0xff,0xf7,0xef,0xe7,0xdf,0xd7,0xcf,0xc7,
0xbf,0xb7,0xaf,0xa7,0x9f,0x97,0x8f,0x87,
0x7f,0x77,0x6f,0x67,0x5f,0x57,0x4f,0x47,
0x3f,0x37,0x2f,0x27,0x1f,0x17,0x0f,0x07
```

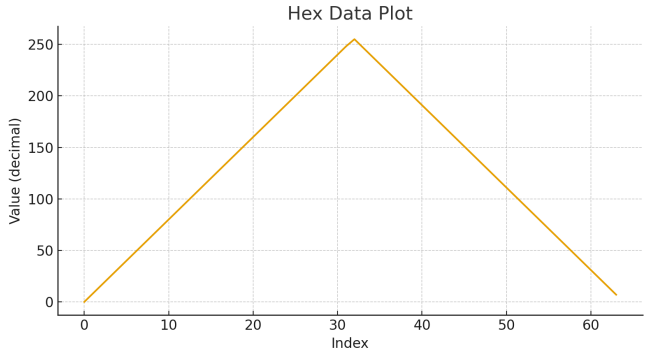

Figura 4. Graficado de señal para conversor. Fuente: Elaboración Própia

VI-G. Stack Pointer

Incialización del Stack Pointer:

```
RESET:
  cli ldi r16, high(RAMEND)
  out SPH, r16
  ldi r16, low(RAMEND)
  out SPL, r16 sei
  ; ...
```

Usos del Stack Pointer: rcall, call, interrupciones preservacion de datos entre subrutinas e ISRs

VI-H. Interrupt service routines

```
MI_ISR:
  push r16
  in r16, SREG
  push r16
  ; ...
  pop r16
  out SREG, r16
  push r16
  reti
```

VI-I. Delay activo

```
ACTIVE_DELAY:
  push r18 push r19 push r20

  ldi  r18, 1
  ldi  r19, 10
  ldi  r20, 229
L1:
  dec  r20
  brne L1
  dec  r19
  brne L1
  dec  r18
  brne L1
  nop

  pop r20 pop r19 pop r18
  ret
```

VI-J. Vectores de interrupción

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Figura 5. Vectores de interrupciones en el ATmega328P. Fuente: hoja de datos del ATmega328P [3].

VI-K. Configuración Timer 1

El prescaler del Temporizador 1 es configurado a través del registro TCCR1B el cual posee la siguiente configuración:

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 6. Registro de control B (TCCR1B) para Timer/Counter1. Fuente: hoja de datos del ATmega328P [3].

Los bits CS12 CS11 y CS10 configuran el prescaler del timer conforme a la siguiente tabla:

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Figura 7. Clock Select (CS12:0) opciones de prescaler para Timer/Counter1. Fuente: hoja de datos del ATmega328P [3].

El máximo valor de tiempo que admite el Timer 1 (de 16 bits) con el prescaler de 1024 es de 4.19 segundos (aproximadamente). Para valores más grande de delay será necesario crear un contador de overflow aparte utilizando registros de uso general o SRAM

Este es un ejemplo de inicialización de timer 1 para un overflow de 1s

```
ldi r16, 0b101        sts TCCR1B, r16
ldi r16, HIGH(49911)  sts TCNT1H, r16
ldi r16, LOW(49911)   sts TCNT1L, r16
```

El primer registro (TCCR1B) determina el prescaler del reloj (según la figura 7)

Utilizando la tabla de vectores de interrupcion que se muestra en la figura 5 se mapea el vector de interrupción con la etiqueta del ISR correspondiente:

```
.org 0x001A rjmp TIMER1_OVF_ISR

TIMER1_OVF_ISR:
    push r16
    out r16, SREG
    push r16
    ; ...
    pop r16
    in SREG, r16
    push r16
    reti
```

VI-L. Interrupciones Externas

VI-L1. EINT: Interrupciones externas

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Figura 8. Tabla de configuraciones para EICRA. Fuente: hoja de datos del ATmega328P [3].

12.2.1 EICRA – External Interrupt Control Register A

The external interrupt control register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 9. Registro de configucación de interrupciones externas EICRA. Fuente: hoja de datos del ATmega328P [3].

12.2.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 10. Registro de configucación de máscaras de interrupciones externas EICRA. Fuente: hoja de datos del ATmega328P [3].

VI-L2. PCINT: Interrupciones por cambio en PIN

12.2.4 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 11. Registro de configucación interrupciones PCICR (PCINT). Fuente: hoja de datos del ATmega328P [3].

12.2.8 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

Figura 12. Registro de configucación de mascara de pines para interrupciones PCIMSK0 (PCINT). Fuente: hoja de datos del ATmega328P [3].

VI-M. SRAM

```
.dseg
variable: byte 1
arreglo: byte 100
```

VI-N. FLASH

```
.cseg
.org 0x300 TABLA:
    .db 0xFF, 0x30, 0x30, 0xFF

GET_DATA:
    ldi ZH, high(TABLA<<1)
    ldi ZL, low(TABLA<<1)
    lpm r16, Z+
    ret
```

VI-Ñ. USART Asíncrono

VI-Ñ1. Inicialización:

```
.equ _F_CPU = 16000000
.equ _BAUD = 9600
.equ _BPS = (_F_CPU/16/_BAUD) - 1

.org 0x0024 rjmp USART_RX_ISR

RESET:
; Configurar baudios
sts UBRR0H, high(_BPS)
sts UBRR0L, low(_BPS)

; Habilitar: receptor, transmisor
; e interrupciones por RX
ldi r16, 0b10011000
sts UCSR0B,r16

; Establecer formato:
; 8 data bits, 2 stop bits
ldi r16, (1<<USBS0)|(3<<UCSZ00)
sts UCSR0C,r16

sei
```

VI-Ñ2. Transmisión:

```
ldi r16, 0x3f ; Cargar r16
sts  UDR0, r16 ; Transmitir
```

VI-Ñ3. Recepción:

```
USART_RX_ISR:
push r16
in r16, SREG
push r16

; Datos recibidos en r16
lds r16, UDR0
; ...

pop r16
out SREG, r16
pop r16
reti
```

VI-O. Ring Buffer

Reservando RAM para el buffer

```
.dseg
tx_buffer: .byte TX_BUF_SIZE
tx_head:   .byte 1
tx_tail:   .byte 1
```

Tamaño de buffer y máscara de clamping

```
.cseg
.equ TX_BUF_SIZE = 16 ; Pot. de 2
.equ TX_BUF_MASK = TX_BUF_SIZE - 1
```

Avanzar head, y clampear con BUF MASK. Tail se puede avanzar utilizando la mísma lógica

```
lds  r17, tx_head
mov  r19, r17 ; Copiar para despues
inc  r19
andi r19, TX_BUF_MASK
```

El truco de utilizar un múltiplo de 2 y una máscara tiene la ventaja de poder limitar el rango de una variable (clamping) con tan solo una instrucción: “andi”.

Si luego de incrementar head, en el caso de que head == tail (buffer lleno), se realiza una espera activa.

```
wait_space:
; buffer lleno? next == tail
lds  r18, tx_tail
cp   r19, r18
breq wait_space
```

Guardar un valor en el buffer

```
; Z -> Cabeza de buffer
ldi  ZL, low(tx_buffer)
ldi  ZH, high(tx_buffer)
add  ZL, r17
adc  ZH, r1
st   Z, r16 ; Guardar r16 en buffer
```

En este caso se decide utilizar una espera activa, pero podría llegar a no ser una buena práctica. Otra alternativa sería simplemente descartar el dato si el buffer está lleno.

VI-P. USART Asíncrono con Ring Buffer

VI-P1. Inicialización:

```
.dseg ; Ring buffer ram alocation
tx_buffer: .byte TX_BUF_SIZE
tx_head:   .byte 1
tx_tail:   .byte 1

.cseg
.equ TX_BUF_SIZE = 256
.equ TX_BUF_MASK = TX_BUF_SIZE - 1

.equ _F_CPU = 16000000
.equ _BAUD = 57600
.equ _BPS = (_F_CPU/16/_BAUD) - 1

; Recieved USART data
.org 0x0024 rjmp USART_RX_ISR
; USART Data register clear
.org 0x0026 rjmp USART_UDRE_ISR

RESET:
  clr r1

  ; Stack
```

```
  ldi r16, high(RAMEND) out SPH, r16
  ldi r16, low(RAMEND)  out SPL, r16

  ; Init USART
  ldi r16, low(_BPS)
  ldi r17, high(_BPS)
  rcall USART_INIT

  sei
  ; ...

USART_INIT:
  sts  tx_head, r1
  sts  tx_tail, r1

  sts UBRR0H, r17
  sts UBRR0L, r16

  ldi r16, 0b10011000
  sts UCSR0B,r16

  ldi r16, (1<<USBS0)|(3<<UCSZ00)
  sts UCSR0C,r16
  ret
```

VI-P2. Subrutina USART WRITE BYTE:

```
USART_WRITE_BYTE:
  push r17
  push r18
  push r19
  push ZH
  push ZL

  ; head/tail
  lds  r17, tx_head
  lds  r18, tx_tail

  ; r16 -> next = (head + 1) & MASK
  mov  r19, r17
  inc  r19
  andi r19, TX_BUF_MASK
  ; Clamping:
  ; Con 256 es 0xFF: no cambia,
  ; pero deja claro el patrón

  wait_space:
  ; buffer lleno? next == tail
  lds  r18, tx_tail
  cp   r19, r18
  breq wait_space ; espera activa

  have_space:
  ; Z -> Cabeza de buffer
  ldi  ZL, low(tx_buffer)
  ldi  ZH, high(tx_buffer)
  add  ZL, r17
  adc  ZH, r1

  st   Z, r16 ; Guardar en buffer

  ; Cabeza = next
  sts  tx_head, r19

  ; Habilitar interrupcion UDRE
  ; así el ISR comienza/continúa
  ; drenando el buffer
  cli
  lds  r18, UCSR0B
  ori  r18, (1<<UDRIE0)
  sts  UCSR0B, r18
  sei

  pop  ZL
  pop  ZH
  pop  r19
  pop  r18
  pop  r17
ret
```

VI-P3. Interrución USART UDRE ISR:

```
USART_UDRE_ISR:
  push r16
  in   r16, SREG
  push r16
  push r17
  push r18
  push r20
  push ZH
  push ZL

  ; r17 = head, r18 = tail
  lds  r17, tx_head
  lds  r18, tx_tail

  ; buffer vacío? head == tail
  cp   r17, r18
  brne usart_udre_send

  ; vacío: deshabilitar UDRIE0
  lds  r20, UCSR0B
  andi r20, ~(1<<UDRIE0)
  sts  UCSR0B, r20
  rjmp usart_udre_exit

  usart_udre_send:
  ; Z -> Cola de buffer
  ldi  ZL, low(tx_buffer)
  ldi  ZH, high(tx_buffer)
  add  ZL, r18
  adc  ZH, r1
```

```
  ; Transmitir byte
  ld   r16, Z
  sts  UDR0, r16

  ; cola = (cola + 1)
  inc  r18
  andi r18, TX_BUF_MASK
  ; Clamping:
  ; Con 256 es 0xFF: no cambia,
  ; pero deja claro el patrón

  sts  tx_tail, r18

  usart_udre_exit:
  pop  ZL
  pop  ZH
  pop  r20
  pop  r18
  pop  r17
  pop  r16
  out  SREG, r16
  pop  r16
  reti
```

VI-P4. Interrución USART RX ISR:

```
USART_RX_ISR:
  push r16
  in r16, SREG
  push r16

  ; Leer datos recibidos en r16
  lds r16, UDR0
  ; ...

  pop r16
  out SREG, r16
  pop r16
  reti
```

VI-P5. Enviar mensajes por USART:

```
SEND_MESSAGE:
  push r16

  SEND_MESSAGE_LOOP:
  lpm r16, Z+
  cpi r16, 0

  breq SEND_MESSAGE_END
  rcall USART_WRITE_BYTE
  rjmp SEND_MESSAGE_LOOP

  SEND_MESSAGE_END:
  pop r16
  ret
```

Ejemplo de mensaje guardado en FLASH:

```
MSG_MENU:
.db "E. Actual: Standby", 0x0A, 0x0A
.db "Elija una opcion:", 0x0A
.db "[1] -> Conf. carga ligera ", 0x0A
.db "[2] -> Conf. carga mediana", 0x0A
.db "[3] -> Conf. carga pesada ", 0x0A
.db "[A] -> Iniciar", 0x0A, 0x0A, 0 ;Fin
```

VI-Q. Bit Masks

VI-Q1. SET BIT:

```
SET_BIT:
  push r16
  push r17
  push ZL
  push ZH

  ld  r17, Z     ; read current value
  or  r17, r16   ; set bit
  st  Z, r17     ; write back

  pop ZH
  pop ZL
  pop r17
  pop r16
  ret
```

VI-Q2. CLEAR BIT:

```
CLEAR_BIT:
  push r16
  push r17
  push ZL
  push ZH

  ld  r17, Z    ; read current value
  com r16       ; invert mask (11110111)
  and r17, r16  ; clear bit
  st  Z, r17    ; write back

  pop ZH
  pop ZL
  pop r17
  pop r16
  ret
```

VI-R. Look Up Table (LUT)

```
; Config. de puertos filas
.org 0x300
ROW_PORTS:
  .db 0x25, 0x25, 0x25, 0x25
  .db 0x28, 0x25, 0x28, 0x28

; Config. de pines de puertos
ROW_MASKS:
  .db 0b00001000, 0b00000010
  .db 0b00010000, 0b00000100
  .db 0b00001000, 0b00100000
```

```
  .db 0b00010000, 0b00100000

; Config. de puertos columnas
COL_PORTS:
  .db 0x2B, 0x2B, 0x2B, 0x28
  .db 0x25, 0x28, 0x28, 0x2B

; Config. de pines de puertos
COL_MASKS:
  .db 0b00010000, 0b00100000
  .db 0b10000000, 0b00000100
  .db 0b00000001, 0b00000001
  .db 0b00000010, 0b01000000
```

VI-S. Máquina de estados

```
.def current_state = r20

STATE_MACHINE:
  cpi current_state, 0
  breq STATE_MACHINE_STATE_0
  cpi current_state, 1
  breq STATE_MACHINE_STATE_1
  cpi current_state, 2
  breq STATE_MACHINE_STATE_2
  cpi current_state, 3

  rjmp STATE_MACHINE_DEFAULT


  STATE_MACHINE_STATE_0:
    ; ... state logic
    rjmp STATE_MACHINE_END

  STATE_MACHINE_STATE_1:
    ; ... state logic
    rjmp STATE_MACHINE_END

  STATE_MACHINE_STATE_2:
    ; ... state logic
    rjmp STATE_MACHINE_END

  STATE_MACHINE_DEFAULT:
    ; ... fail state logic
    rjmp STATE_MACHINE_END


  STATE_MACHINE_END:
    ; ...
    ret
```

VI-T. Macros

Definir macros en archivo “.inc”:

```
.macro DISABLE_TIMER_1
 push r16
 ldi r16, 0    sts TCCR1B, r16
 ldi r16, (0<<TOIE1)  sts TIMSK1, r16
 ldi r16, (1<<TOV1)   out TIFR1,  r16
 pop r16
.endmacro

.macro ENABLE_TIMER_1
 ; @0 Timer seconds
 push r16
 mov timer1_ovf_counter, @0
 ldi r16, 0b101   sts TCCR1B, r16
 ldi r16, HIGH(49911) sts TCNT1H, r16
 ldi r16, LOW(49911)  sts TCNT1L, r16
 ldi r16, (1<<TOV1)   out TIFR1,  r16
 ldi r16, (1<<TOIE1)  sts TIMSK1, r16
 pop r16
.endmacro
```

Importar macros a “main.asm”:

```
.include "m328pdef.inc"
.include "macros.inc"
```

VI-U. Plotter: Generador de círculos con Python

```
import math

# Number of steps per quadrant
divisions = 42
# Maximum cycles per motor
max_cycles = 100
solenoid = "SOLENOID_DOWN"

# Define quadrants with corresponding
# directions and wave signs
quadrants = [
  ("RIGHT", "UP", 1, 1),      # Q I
  ("DOWN", "RIGHT", -1, 1),   # Q II
  ("LEFT", "DOWN", -1, -1),   # Q III
  ("UP", "LEFT", 1, -1)       # Q IV
]

# Function to scale
# wave values to cycles
def get_cycles(value1,
               value2,
               max_cycles):
  max_val = max(value1, value2)
  if max_val == 0:
    return 0, 0

  scale = max_cycles / max_val
  return int(round(value1 * scale)),
  int(round(value2 * scale))

# Generate commands
for quad_index, (dir1,
                 dir2,
                 sign1,
                 sign2)
in enumerate(quadrants):
```

```
print(f"\n; Quadrant
{quad_index + 1}: {dir1} & {dir2}")

for step in range(divisions):
  # Calculate sine/cosine wave (0..1)
  angle = (math.pi / 2) *
  (step / (divisions - 1))

  wave1 = math.sin(angle) * sign1
  wave2 = math.cos(angle) * sign2

  # Take absolute value because
  # we only care about time cycles
  wave1, wave2 = abs(wave1),
                 abs(wave2)

  # Scale to cycles
  cycles1, cycles2 =
  get_cycles(wave1,
             wave2,
             max_cycles)

  # Both motors on for min(cycles1,
  #                        cycles2)
  i_cycles = min(cycles1, cycles2)
  # Only the \extra" cycles
  # for the motor that is ahead
  j_cycles = abs(cycles1 - cycles2)

  # Determine which motor
  # gets the extra cycles
  if cycles1 > cycles2:
    extra_motor = dir1
  else:
    extra_motor = dir2

  # Output commands
  if i_cycles > 0:
    print(f" .db {i_cycles},
    {solenoid} + {dir1} + {dir2}")

  if j_cycles > 0:
    print(f" .db {j_cycles},
    {solenoid} + {extra_motor}")
```