

# Diseño e implementación con microcontrolador ATmega328P para la automatización de procesos de transporte, punzonado, visualización y generación de señales

1º Lucas Elizalde

*Ingeniería en Mecatrónica  
UTEC*

Fray Bentos, Río Negro

lucas.elizalde@estudiantes.utec.edu.uy

2º Juan Manuel Ferreira

*Ingeniería en Mecatrónica  
UTEC*

Fray Bentos, Río Negro

juan.ferreira.m@estudiantes.utec.edu.uy

3º Felipe Morrudo

*Ingeniería en Mecatrónica  
UTEC*

Fray Bentos/, Río Negro

felipe.morrudo@estudiantes.utec.edu.uy

**Resumen**—Se presenta el diseño e implementación de cuatro prototipos basados en ATmega328P: cinta transportadora con punzonadora, matriz de LEDs 8x8 con UART, DAC R-2R de 8 bits y control de plotter integrado a PLC. Se programaron rutinas en ensamblador con temporizadores CTC y comunicación serie para monitoreo e interacción. La cinta logró el ciclo automatizado y telemetría, quedando pendiente optimizar sensores y PWM. La matriz mostró texto desplazable y figuras con velocidad ajustable. El DAC generó una rampa descendente de 256 niveles verificada en osciloscopio, y el plotter trazó triángulo, círculo aproximado y cruz, evidenciando limitaciones mecánicas. Los resultados validan la propuesta y apuntan mejoras en interrupciones, PWM, tamaño de LUT y filtrado. En conjunto, se comprobó la capacidad del ATmega328P para integrar control, comunicación y generación de señales en distintos sistemas mecatrónicos.

**Index Terms**—ATmega328P; Assambler, Cinta transportadora, Punzonadora, Comunicación UART, Matriz de LEDs, DAC R-2R, Plotter.

## I. INTRODUCCIÓN

El presente trabajo tiene como finalidad desarrollar y analizar distintos sistemas mecatrónicos basados en el microcontrolador ATmega328P, integrando conceptos de control digital, comunicación serial y conversión digital-análogica. El estudio se centra en la construcción y programación de prototipos que permiten vincular los contenidos teóricos de la unidad curricular con aplicaciones prácticas en laboratorio, asegurando una visión integral del diseño, simulación y validación de sistemas embebidos.

En una primera etapa, se abordará el montaje de un modelo de cinta transportadora con punzonadora, utilizando los componentes del kit Fischertechnik. Este sistema será controlado por el microcontrolador, gestionando estados definidos (espera, alimentación, posicionado, punzonado, descarga y fin de ciclo), modos de operación para diferentes cargas, así como la comunicación mediante USART para monitoreo y control remoto.

Posteriormente, se implementará un sistema de visualización con matriz de LEDs, donde se desarrollará un código en

ensamblador para mostrar mensajes desplazables de forma continua, además de habilitar la selección de figuras a través de la comunicación serial. De forma complementaria, se explorará el funcionamiento de un conversor digital a analógico (DAC) de tipo R-2R, aplicando una tabla de búsqueda (LUT) para la generación de señales específicas que serán analizadas con osciloscopio.

Finalmente, se abordará el control de un plotter monocromático mediante la integración del ATmega328P y un PLC, empleando comunicación serial para la ejecución de figuras predeterminadas. Este proceso permitirá comprender la interacción entre actuadores, sensores y rutinas de control en tiempo real, evaluando las limitaciones mecánicas y de sincronización.

## II. OBJETIVOS

### II-A. Objetivo General

- Diseñar, implementar y analizar sistemas mecatrónicos basados en el microcontrolador ATmega328P, integrando montaje físico, programación en ensamblador y comunicación serial, para automatizar procesos de transporte, punzonado, visualización de mensajes y generación de señales analógicas.

### II-B. Objetivos Específicos

- Construir y poner en funcionamiento el modelo de cinta transportadora con punzonadora, controlado mediante el ATmega328P y gestionado por estados definidos para diferentes tipos de carga.
- Implementar la comunicación USART para el monitoreo en tiempo real, la recepción de comandos de inicio y la selección remota de modos de operación.
- Programar en ensamblador la matriz de LEDs para mostrar mensajes desplazables y permitir la elección de imágenes predefinidas a través de la comunicación serial.

- Generar señales analógicas específicas mediante una tabla de búsqueda (LUT) y un DAC R-2R de 8 bits, verificando su forma de onda en el osciloscopio.
- Controlar el plotter monocromático mediante el ATmega328P y un PLC, automatizando el dibujo de figuras predeterminadas dentro de los límites mecánicos del sistema.

### III. MATERIALES

- ATmega328p
- Fuente DC.
- Kit Fischertechnik Robotics
- LEDs
- Matriz de LEDs 8x8 1088AS
- Multímetro digital.
- Osciloscopio.
- Plotter
- Protoboard.
- Resistencias:
  - 320Ω (8),
  - 1 kΩ (8),
  - 2 kΩ (8),

### IV. MARCO TEÓRICO

#### IV-A. ATmega328p

El ATmega328P es un microcontrolador de la familia AVR desarrollado por Atmel (actualmente parte de Microchip), ampliamente utilizado como núcleo de la placa Arduino Uno debido a su versatilidad y bajo consumo. Se basa en una arquitectura Harvard con un conjunto reducido de instrucciones (RISC) y un pipeline de un ciclo, lo que le permite ejecutar la mayoría de sus 131 instrucciones en un único ciclo de reloj y alcanzar hasta 20 MIPS a una frecuencia de 20 MHz.

Este dispositivo cuenta con una memoria Flash de 32 KB para almacenar programas, 2 KB de SRAM para datos y 1 KB de EEPROM para almacenamiento no volátil, lo que lo hace adecuado para aplicaciones embebidas que requieren autonomía y confiabilidad. Dispone de 23 pines de entrada y salida digitales configurables (GPIO), varios de los cuales pueden cumplir funciones alternas como comunicación serial, conversión analógica-digital o generación de señales PWM.

Entre sus periféricos más relevantes se encuentran tres temporizadores (dos de 8 bits y uno de 16 bits), un convertidor analógico-digital (ADC) de 10 bits con seis canales, un comparador analógico, así como interfaces de comunicación USART, SPI y TWI (compatible con I2C). Además, incorpora un watchdog timer, soporte para interrupciones internas y externas, y diferentes modos de bajo consumo que lo hacen eficiente para aplicaciones portátiles o de larga duración.

Gracias a estas características, el ATmega328P es ampliamente utilizado en entornos educativos y de prototipado, pero también en aplicaciones industriales y comerciales,

ya que ofrece un equilibrio entre simplicidad, bajo costo y capacidad de procesamiento.

#### IV-B. Assembler

El lenguaje ensamblador, o Assembler, es un lenguaje de programación de bajo nivel que permite dar instrucciones de forma directa al procesador de un sistema. A diferencia de los lenguajes de alto nivel, utiliza mnemónicos que representan operaciones básicas como transferir datos, realizar cálculos o controlar el flujo de ejecución. Estas instrucciones son traducidas a código máquina mediante un ensamblador, lo que hace posible que el microcontrolador ejecute las órdenes. En el caso del ATmega328P, este lenguaje resulta especialmente relevante ya que permite aprovechar al máximo sus recursos internos, ofreciendo un control detallado sobre su arquitectura y facilitando la programación de rutinas optimizadas para tareas específicas (MSMK, 2025).

#### IV-C. USART

El USART (Universal Synchronous/Asynchronous Receiver/Transmitter) es un periférico de comunicación serial integrado en microcontroladores como el ATmega328P que permite la transmisión y recepción de datos tanto en modo asíncrono como en modo síncrono. En el modo asíncrono funciona de forma equivalente a un UART, utilizando bits de inicio y parada para organizar la comunicación sin necesidad de una señal de reloj compartida. En el modo síncrono, en cambio, emisor y receptor comparten una señal de reloj que permite una transferencia de datos más rápida y precisa. Esta flexibilidad lo convierte en un recurso fundamental para establecer enlaces entre el microcontrolador y otros dispositivos digitales (USART Vs. UART: What's the Difference?, 2023).

#### IV-D. Fischertechnik Robotics TXT 4.0 Base Set

El kit fischertechnik Robotics TXT 4.0 Base Set está compuesto por 244 piezas que permiten construir hasta 12 modelos funcionales, entre ellos semáforos, escáneres de códigos y robots móviles. Sus componentes principales incluyen el controlador TXT 4.0, dos motores con codificador, una cámara USB, un sensor ultrasónico, un sensor de pista, un fototransistor, botones y LEDs. El controlador cuenta con 512 MB de RAM, 4 GB de memoria eMMC, múltiples puertos de entrada/salida para motores, sensores digitales y analógicos, servos, así como conectividad Wi-Fi y Bluetooth. Además, ofrece compatibilidad con el entorno de programación ROBO Pro Coding y con Python, lo que habilita la implementación de lógica de control, procesamiento de señales, mediciones y proyectos de automatización aplicados a prototipos reales (fischertechnik, 2025). En Fig. 1 se muestra el kit.



Fig. 1: Fischertechnik Robotics TXT 4.0 Base Set

#### IV-E. Arduino UNO F5

El Arduino UNO F5 Adapter como se muestra en Fig. 2 es un módulo diseñado específicamente para facilitar la integración del Arduino UNO con los componentes del sistema fischertechnik, lo que lo convierte en una herramienta ideal para proyectos de robótica educativa y prototipado rápido. Su principal ventaja radica en permitir que estudiantes, aficionados y profesionales puedan aprovechar la flexibilidad del Arduino con la robustez mecánica y modularidad de fischertechnik (EduTechnik, 2025).



Fig. 2: Arduino UNO F5

#### IV-F. Matriz de Leds 1088AS

La matriz LED 8x8 modelo 1088AS es un dispositivo de visualización compuesto por 64 diodos emisores de luz organizados en ocho filas y ocho columnas. Su configuración de cátodo común permite controlar cada LED de manera individual mediante el uso de multiplexado, lo que la hace adecuada para representar caracteres, símbolos o patrones gráficos. Gracias a su sencillez de manejo y a su integración con microcontroladores como el ATmega328P, se convierte en un recurso didáctico y práctico para proyectos de electrónica, prototipos y sistemas embebidos (Matriz LED 8x8 1088AS De 3 Mm, s.f.). A continuación, en Fig. 3 se muestra la Matriz.

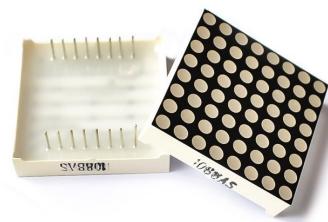


Fig. 3: Matriz de Leds AS1088AS

#### IV-G. Conversor digital-analógico R-2R

Un conversor digital-analógico (DAC, por sus siglas en inglés) es un dispositivo encargado de transformar una secuencia de valores digitales discretos en una señal analógica continua. Entre las distintas arquitecturas posibles, la red resistiva R-2R se destaca por su simplicidad y bajo costo, ya que únicamente requiere resistencias de dos valores:  $R$  y  $2R$ .

El principio de funcionamiento del DAC R-2R se basa en el uso de divisores de tensión en cascada, donde cada bit de entrada contribuye con un peso binario en la suma total. De este modo, el valor digital de  $n$  bits se convierte en un nivel de tensión proporcional, de acuerdo con:

$$V_{out} = V_{ref} \cdot \frac{D}{2^n}, \quad (1)$$

donde  $V_{ref}$  es la tensión de referencia del sistema,  $D$  es el valor digital aplicado y  $n$  es el número de bits del DAC. En el presente trabajo se utilizó un DAC R-2R de 8 bits ( $n = 8$ ), por lo que el valor de salida abarca un rango entre 0 y  $V_{ref}$ .

Para garantizar el ponderado binario correcto, las resistencias deben cumplir la relación  $2R = 2 \cdot R$  con buena coincidencia (*matching*). El tamaño del paso (LSB) que separa niveles adyacentes es

$$\Delta = \frac{V_{ref}}{2^n}, \quad (2)$$

lo cual explica la apariencia escalonada observada en las mediciones cuando se emplea una LUT de 64 muestras.

#### IV-H. Plotter

Un plotter de dibujo de tres ejes es un sistema de control de movimiento que permite representar figuras gráficas mediante el desplazamiento coordinado de un cabezal sobre los ejes X, Y y Z. A diferencia de los plotters bidimensionales tradicionales, este tipo de dispositivo incorpora un eje vertical que posibilita levantar o bajar el útil de trabajo, lo cual brinda mayor versatilidad en la ejecución de trazos y en la simulación de procesos industriales de mecanizado. El sistema se compone de una estructura tipo pórtico, motores paso a paso o servomotores, guías lineales y un controlador electrónico que interpreta coordenadas y genera las señales de movimiento. Este tipo de plotter se utiliza con fines educativos y experimentales, ya que facilita el aprendizaje de conceptos de control de movimiento, cinemática cartesiana

y programación de trayectorias, integrando hardware de control y software para la ejecución de rutinas de dibujo automatizado (Shinde, s.f.).

#### IV-I. Algoritmo de Bresenham

El algoritmo de Bresenham es una técnica de dibujo digital que permite representar figuras geométricas en una pantalla o matriz de puntos utilizando únicamente operaciones con enteros, lo que lo hace muy eficiente en sistemas de recursos limitados como los microcontroladores. Su principio consiste en calcular, de manera incremental, qué puntos de la retícula deben encenderse para aproximar la forma ideal. Aunque fue desarrollado inicialmente para el trazado de líneas rectas, también se adapta al dibujo de circunferencias y elipses, generando sus puntos a partir de la simetría de la figura y evitando cálculos complejos de raíz cuadrada o números en coma flotante (Hurtado, s.f.).

#### V. PROCEDIMIENTO

##### V-A. Cinta Transportadora con Punzonadora

En esta sección, se planea controlar una cinta transportadora con una punzonadora incorporada. Esto se va a programar mediante el microcontrolador ATMEGA328P, la cinta transportadora con punzonadora es un modelo perteneciente a un kit de Fischertechnik, pero no será controlada mediante su placa original.

Para este laboratorio, se usa una placa Arduino UNO con un adaptador, este es el UNO F5, el cual es compatible con Fischertechnik por lo que simplifica las conexiones. A continuación, en la Fig.4 se observa el adaptador UNO F5 con sus respectivas conexiones.

**Arduino UNO - Uno-F5 board**  
fischertechnik compatible

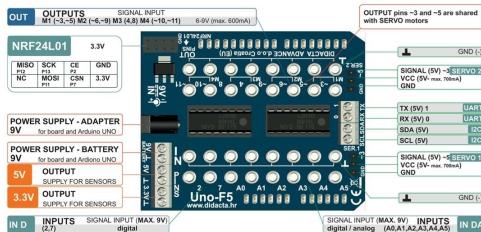


Fig. 4: Adaptador UNO F5

Mediante el manual, se fueron ensamblando las piezas necesarias para replicar el modelo. A continuación, en la Fig.5 se observa cómo debería quedar el modelo una vez finalizado su montaje.

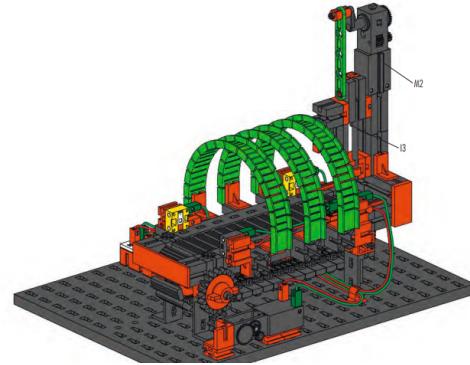


Fig. 5: Modelo Fischertechnik Cinta Transportadora con Punzonadora

Este modelo necesita dos motores para su correcto funcionamiento, un motor para activar la cinta transportadora, y otro para hacer bajar y subir la punzonadora. En la cinta transportadora se colocarán cargas, las cuales posteriormente serán punzonadas. Estas cargas se clasifican en 3 tipos: Ligera, Mediana y Pesada. Cada carga se distingue por tener sus propios tiempos durante el proceso, o sea, tiempos de traslación, espera, punzadora y descarga de la carga. Mediante LEDs, se indica el tipo de carga transportada y en qué estado se encuentra, dichos estados son: Sistema en espera, Sistema Funcionado y Final del ciclo. A continuación, en el Cuadro.I se observan las conexiones de motores y LEDs, y en el Cuadro.II se observan los tiempos por tipo de carga.

Cuadro I: Conexiones ARDUINO UNO F5

MOTOR/LED	UNO F5	Arduino UNO
MOTOR CINTA	M1	PD3 y PD5
MOTOR PUNZONADORA	M3	PD6 y PD9
ESPERA (AMARILLO)	2	PC0
FUNCIONANDO (VERDE)	7	PC1
FINAL (ROJO)	A0	PC2
LIGERA (BLANCO)	A1	PC3
MEDIANA (AZUL)	A2	PC4
PESADA (ROJO)	A3	PC5

Cuadro II: Tiempos de ciclo por carga

Modo	Alimentación		Punzonado		Descarga
	Avance	Pausa	Bajar y subir	Presión	Cinta en reversa
Ligera	3s	2s	2s	2s	3s
Mediana	4s	2s	2s	3s	4s
Pesada	5s	3s	2s	4s	5s

Por último, se debe incorporar un monitoreo en tiempo real mediante comunicación USART. Además, el sistema se debe inicializar mediante este medio, permitiendo al usuario seleccionar el tipo de carga y la cantidad de cargas a mandar. Este monitoreo cuenta con otros estados a los

mostrados anteriormente, para una mayor exactitud del rastreo de la carga. A continuación, en la Fig.6 se observa el diagrama de estados para el monitoreo a tiempo real.

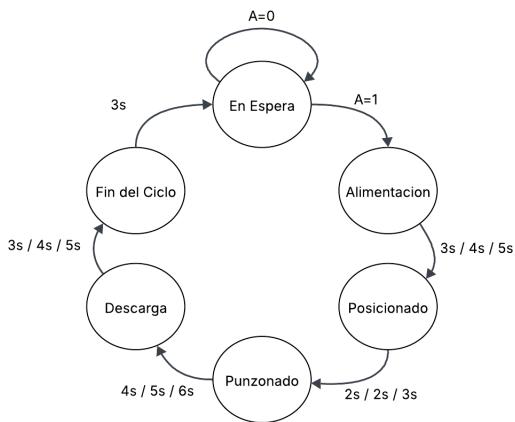


Fig. 6: Diagrama de Estados de Monitoreo a tiempo real

Para la programación y simulación se utilizó la siguiente distribución de pines entre la matriz de LEDs y el microcontrolador ATmega328P. El Cuadro. III muestra la conexión de las filas de la matriz, mientras que el Cuadro. IV detalla la conexión de las columnas donde en cada una se usara una resistencia de  $320\ \Omega$  para limitar la corriente.

Cuadro III: Conexión de filas de la matriz al ATmega328P

Filas matriz	ATmega328P
F1	PB1
F2	PC0
F3	PB0
F4	PB4
F5	PC0
F6	PD7
F7	PD2
F8	PD5

#### V-B. Manejo de Matriz de LEDs mediante UART

El objetivo de este apartado es programar el microcontrolador ATmega328P para controlar una matriz de LEDs, de forma que sea capaz de mostrar un mensaje desplazándose de derecha a izquierda. Este mensaje debe tener al menos 12 caracteres y moverse de manera fluida a lo largo de la pantalla. Además, se debe incorporar una comunicación UART que permita mostrar un mensaje de bienvenida al usuario, junto con un menú interactivo desde el cual se podrá seleccionar entre diversas opciones.

El menú deberá ofrecer varias opciones de visualización, incluyendo el mensaje desplazándose, así como figuras predefinidas: una cara feliz, una cara triste, un rombo, un corazón y un alien similar al de Space Invaders. Asimismo, se debe permitir la opción de mostrar las imágenes de forma secuencial, cambiando entre ellas después de un intervalo de tiempo determinado.

Además, la velocidad de desplazamiento del mensaje debe ser ajustable, con el fin de garantizar que el contenido sea legible para el usuario en todo momento. Para ello, se debe implementar un mecanismo que permita modificar la velocidad de desplazamiento del texto según las preferencias del usuario.

Finalmente, el código debe estar bien documentado, lo que permitirá realizar ajustes futuros en el contenido del mensaje o en la velocidad de desplazamiento de manera sencilla.

Una vez comprendido el funcionamiento, se procedió a realizar la programación mientras de manera simultánea se efectuaba la simulación del circuito en el software Proteus. Para llevar a cabo este proceso, la programación se dividió en tres partes: en primer lugar, la implementación del mensaje desplazable; en segundo lugar, la configuración del monitor serial con su respectivo menú; y, finalmente, la integración del circuito completo.

Cuadro IV: Conexión de columnas de la matriz al ATmega328P

Columnas matriz	ATmega328P
C1	PB5
C2	PD3
C3	PD4
C4	PB2
C5	PD6
C6	PB3
C7	PB1
C8	PB0

Además, en la simulación se emplean el PB0 que se utiliza como línea de transmisión de datos (TX), junto con el pin PB1 que se configura como línea de recepción (RX) del ATmega328P para establecer la comunicación serie con el módulo Virtual Terminal de Proteus. De esta forma, el microcontrolador puede enviar y recibir datos a través de la interfaz UART, lo que permite implementar el menú de opciones.

Una vez concluida la simulación en el software, se implementó el circuito en físico utilizando la misma distribución de pines definida en la etapa de prueba. Cabe destacar que en la simulación la matriz de LEDs se representa con las columnas en los ocho pines superiores y las filas en los ocho pines inferiores, lo cual difiere de la disposición real de la Matriz de LEDs 1088AS que se utilizará en el circuito físico.

A continuación en los Cuadros V y VI se muestran las conexiones de filas y columnas al ATmega328p

Cuadro V: Conexión de Filas de la Matriz al ATmega328P

Filas Matriz	Pines Matriz	ATmega328P
F1	9	PB1 (pin 15)
F2	14	PC0 (pin 23)
F3	8	PB0 (pin 14)
F4	12	PB4 (pin 18)
F5	1	PC0 (pin 23)
F6	7	PD7 (pin 13)
F7	2	PD2 (pin 4)
F8	5	PD5 (pin 11)

Cuadro VI: Conexión de Columnas de la Matriz al ATmega328P

Columnas Matriz	Pines Matriz	ATmega328P
C1	13	PB5 (pin 19)
C2	3	PD3 (pin 5)
C3	4	PD4 (pin 6)
C4	10	PB2 (pin 16)
C5	6	PD6 (pin 12)
C6	11	PB3 (pin 17)
C7	15	PB1 (pin 15)
C8	16	PB0 (pin 14)

#### V-C. DAC R-2R con LUT

Para la implementación del conversor digital a analógico (DAC) se utilizó una red R-2R de 8 bits, construida con resistencias de 1 kΩ para las ramas  $R$  y 2 kΩ para las ramas  $2R$ . Esta red fue conectada directamente al puerto *PORTD* del microcontrolador ATmega328P (mapeo PD7 = MSB, PD0 = LSB), encargado de entregar los valores digitales secuenciales de la tabla precargada (Look-Up Table, LUT).

El trabajo comenzó con el diseño y simulación en el software Proteus, donde se incorporó el ATmega328P programado con el archivo *.hex* generado en Microchip Studio. De esta forma se verificó que la LUT cargada en el microcontrolador producía la señal digital esperada y que la red R-2R la convertía correctamente en su equivalente analógico. Una vez validado el funcionamiento, el circuito se implementó físicamente en protoboard utilizando el mismo microcontrolador y valores de resistencias, conectando la salida de la red R-2R a un osciloscopio para la verificación experimental de la forma de onda.

La señal asignada al grupo fue la **Señal 4**, conformada por **256** muestras que generan una **rampa descendente**. En el Cuadro VII se muestran los niveles digitales de salida utilizados (de 0xFF a 0x00).

Cuadro VII: Valores digitales de la Señal 4 (256 muestras)

0xFF	0xFE	0xFD	0xFC	0xFB	0xFA	0xF9	0xF8	0xF7	0xF6	0xF5	0xF4	0xF3	0xF2	0xF1	0xF0
0xEF	0xEE	0xED	0xEC	0xEB	0xEA	0xE9	0xE8	0xE7	0xE6	0xE5	0xE4	0xE3	0xE2	0xE1	0xE0
0xDF	0xDE	0xDD	0xDC	0xDB	0xDA	0xD9	0xD8	0xD7	0xD6	0xD5	0xD4	0xD3	0xD2	0xD1	0xD0
0xCF	0xCE	0xCD	0xCC	0xCB	0xCA	0xC9	0xC8	0xC7	0xC6	0xC5	0xC4	0xC3	0xC2	0xC1	0xC0
0xBF	0xBE	0xBD	0xBC	0xBB	0xBA	0xB9	0xB8	0xB7	0xB6	0xB5	0xB4	0xB3	0xB2	0xB1	0xB0
0xAF	0xAE	0xAD	0xAC	0xAB	0xAA	0xA9	0xA8	0xA7	0xA6	0xA5	0xA4	0xA3	0xA2	0xA1	0xA0
0x9F	0x9E	0x9D	0x9C	0x9B	0x9A	0x99	0x98	0x97	0x96	0x95	0x94	0x93	0x92	0x91	0x90
0x8F	0x8E	0x8D	0x8C	0x8B	0x8A	0x89	0x88	0x87	0x86	0x85	0x84	0x83	0x82	0x81	0x80
0x7F	0x7E	0x7D	0x7C	0x7B	0x7A	0x79	0x78	0x77	0x76	0x75	0x74	0x73	0x72	0x71	0x70
0x6F	0x6E	0x6D	0x6C	0x6B	0x6A	0x69	0x68	0x67	0x66	0x65	0x64	0x63	0x62	0x61	0x60
0x5F	0x5E	0x5D	0x5C	0x5B	0x5A	0x59	0x58	0x57	0x56	0x55	0x54	0x53	0x52	0x51	0x50
0x4F	0x4E	0x4D	0x4C	0x4B	0x4A	0x49	0x48	0x47	0x46	0x45	0x44	0x43	0x42	0x41	0x40
0x3F	0x3E	0x3D	0x3C	0x3B	0x3A	0x39	0x38	0x37	0x36	0x35	0x34	0x33	0x32	0x31	0x30
0x2F	0x2E	0x2D	0x2C	0x2B	0x2A	0x29	0x28	0x27	0x26	0x25	0x24	0x23	0x22	0x21	0x20
0x1F	0x1E	0x1D	0x1C	0x1B	0x1A	0x19	0x18	0x17	0x16	0x15	0x14	0x13	0x12	0x11	0x10
0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00

Con el objetivo de dejar explícita la correspondencia entre los bits digitales y su contribución analógica en una red R-2R ideal, se presenta el siguiente cuadro de pesos. Allí se diferencia el *peso digital* de la *contribución analógica* efectiva en la salida ( $V_{out}$ ), que decrece en potencias de 1/2:

Cuadro VIII: Peso digital y contribución analógica de cada bit en la red R-2R (8 bits).

Bit (PORTD)	Índice $k$	Peso digital	Contribución analógica
PD7 (MSB)	7	$2^7$	$V_{ref}/2$
PD6	6	$2^6$	$V_{ref}/4$
PD5	5	$2^5$	$V_{ref}/8$
PD4	4	$2^4$	$V_{ref}/16$
PD3	3	$2^3$	$V_{ref}/32$
PD2	2	$2^2$	$V_{ref}/64$
PD1	1	$2^1$	$V_{ref}/128$
PD0 (LSB)	0	$2^0$	$V_{ref}/256$

De manera compacta, la salida ideal puede expresarse como:

$$V_{out} = V_{ref} \left( \frac{b_7}{2} + \frac{b_6}{4} + \cdots + \frac{b_0}{256} \right) = V_{ref} \cdot \frac{D}{2^8}$$

Para controlar la frecuencia de actualización de la LUT y, en consecuencia, la frecuencia de la señal de salida, se configuró el **Timer2** en modo CTC con prescaler igual a 8. La frecuencia de muestreo  $f_s$  depende del valor cargado en el registro OCR2A:

$$f_s = \frac{f_{clk}}{\text{prescaler} \cdot (OCR2A + 1)}, \quad (3)$$

donde  $f_{clk} = 16$  MHz. Al recorrer secuencialmente las 256 muestras de la LUT, la frecuencia de salida resulta:

$$f_{out} = \frac{f_s}{256}. \quad (4)$$

En la práctica, la frecuencia se ajustó variando la constante `.equ OCR2A_VAL`, seis diferentes frecuencias (una base y cinco cambios), analizando el comportamiento de la rampa descendente en el osciloscopio.

#### V-D. Control del Plotter con ATmega328P

Para el control del trazador gráfico del laboratorio se empleó un microcontrolador ATmega328P interfaseado al PLC del plotter mediante etapas de potencia (relés o MOSFET), de modo de adaptar niveles y corriente sin cargar las entradas del controlador. El PLC gobierna los actuadores de desplazamiento en los ejes y la válvula neumática del útil de trazo (subida/bajada de solenoide). La asignación de señales desde el microcontrolador hacia las entradas del PLC se resume en el Cuadro IX.

Cuadro IX: Asignación de pines del ATmega328P hacia el PLC del plotter

Pin digital	Conexión PLC
D2	Bajar solenoide → X0
D3	Subir solenoide → X1
D4	Movimiento hacia abajo → X5
D5	Movimiento hacia arriba → X6
D6	Movimiento hacia la izquierda → X7
D7	Movimiento hacia la derecha → X10

El trabajo comenzó verificando *pin a pin* el cableado y la lógica de activación. Para ello se generaron pulsos individuales desde el ATmega328P hacia cada entrada del PLC, comprobando en forma aislada los movimientos *arriba, abajo, izquierda, derecha* y el accionamiento del solenoide *subir/bajar*. Una vez confirmada la correspondencia pin–acción, se caracterizaron los tiempos mínimos de conmutación y reposo requeridos por los relés/MOSFET y por el PLC, midiendo el retardo necesario para que cada paso de desplazamiento resultara reproducible. Con la temporización establecida se definieron las primitivas de movimiento:  $\text{PEN}_U P/$

#### VI. RESULTADOS

##### VI-A. Cinta Transportadora con Punzonadora

El montaje del modelo se logró con éxito, logrando un correcto funcionamiento en los procesos necesarios para la cinta transportadora con punzonadora. A continuación, en la siguiente Fig.8 se observa el modelo montado en físico.

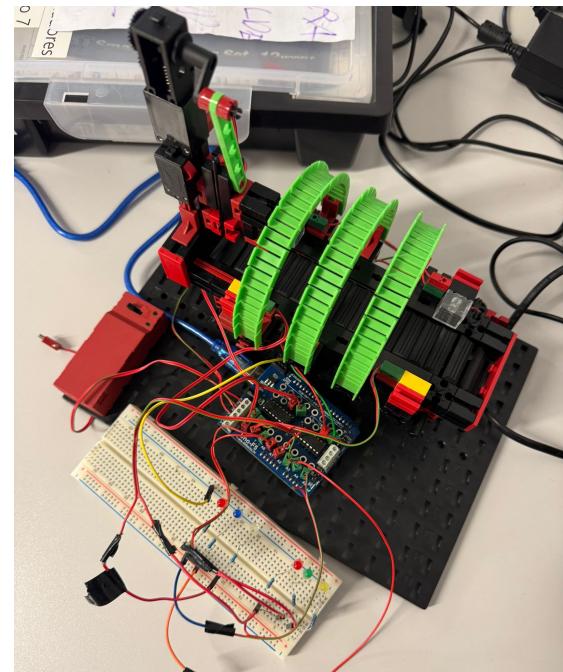


Fig. 8: Cinta transportadora con punzonadora montada en físico.

Inicialmente se planeó implementar los sensores para la detección y posicionamiento de una carga, pero no fue posible adaptar este funcionamiento para los tres tipos de carga, ya que cada tipo posee un tiempo de avance propio, por lo que si se desea que la carga llegue al sensor en diferentes tiempos, hay que ajustar la velocidad de la cinta. Esto se hace con PWM, pero después de varios intentos, no fue posible implementarlo correctamente, esto se debe a que los pines de las conexiones de los motores usan Timer distintos, por lo que al querer modificar el PWM con el Timer0, se afectaban otros pines dificultando la sincronización de los tiempos. En forma de solución, se optó por usar únicamente delays para ajustar el tiempo de avance de la cinta a los requeridos por el tipo de carga, pero surgió otra problemática, en las cargas con mayor tiempo de avance sobrepasaba la ubicación de posicionamiento, por lo que no quedó otra opción que no usar una carga física.

El monitoreo a tiempo real mediante USART se implementó correctamente, como primer paso se inicializó la USART para que funcione a 16Mhz y 9600bps, con un formato 8N1:

Listing 1: Código para inicializar USART

```
; Iniciar USART
ldi r16, high(UBRR_VAL)
sts UBRR0H, r16
ldi r16, low(UBRR_VAL)
sts UBRR0L, r16

ldi r16, (1<<RXEN0) | (1<<TXEN0) ; Habilita RX y
                                         TX
sts UCSR0B, r16

; Tamaño de la palabra a 8 bits
ldi r16, (1<<UCSZ01) | (1<<UCSZ00)
sts UCSR0C, r16
```

La transmisión TX es bloqueante, espera a que el buffer de datos esté vacío (UDRE0=1) y después escribe el byte en UDR0. La recepción RX también es bloqueante, espera a que RXC0=1 y después lee el byte desde UDR0.

Listing 2: Código para TX y RX

```
USART_TX:
ESPERAR_UDRE:
    lds r17, UCSR0A
    sbrs r17, UDRE0
    rjmp ESPERAR_UDRE
    sts UDR0, r16
    ret

USART_RX:
ESPERAR_RXC:
    lds r17, UCSR0A
    sbrs r17, RXC0
    rjmp ESPERAR_RXC
    lds r16, UDR0
    ret
```

Para la comunicación a través del monitor serial, se usa USART\_FLUSH\_RX para limpiar el buffer de recepción, para posteriormente imprimir los mensajes que están almacenados en el monitor a través de USART\_ENVIAR\_CADENA, esto se hace con un registro Z que apunta al inicio de la cadena, la cual debe terminar en 0.

Listing 3: Código imprimir mensajes en el monitor serial

```
USART_FLUSH_RX:
FLUSH_LOOP:
    lds r17, UCSR0A
    sbrs r17, RXC0      ; si RXC0=0, salta y retorna
    ret
    lds r16, UDR0      ; lee y descarta
    rjmp FLUSH_LOOP

USART_MANDAR_CADENA:
    lpm r16, Z+        ; lee byte de flash
    tst r16            ; termino en 0?
    breq FIN_CADENA
    rcall USART_TX     ; env a el byte
    rjmp USART_MANDAR_CADENA
FIN_CADENA:
    ret
```

En cada instancia del proceso, se va a hacer un rcall a USART\_MANDAR\_CADENA. A continuación, se muestra como ejemplo la parte del código donde se pide una `A` para inicializar el sistema:

Listing 4: Código ejemplo de uso para imprimir mensaje en una instancia del proceso

```
; Espera a que la persona mande una "A"
ldi ZL, low(msg_pedir_A<<1)
ldi ZH, high(msg_pedir_A<<1)
rcall USART_MANDAR_CADENA
```

De esta forma el usuario podrá visualizar en tiempo real el estado de la carga durante el proceso mediante el monitor serial. A continuación, en la Fig.9 se observa el monitor serial completo al transportar una carga.

```
En espera, enviar 'A' para inicializar
A
Carga: 1=Ligera 2=Media 3=Pesada
1
Cantidad de piezas:
1
Iniciando ciclo...
Pieza restante:
1
Alimentacion (cinta avanzando)
Pieza posicionada
Punzon: Bajando
Presionando
Punzon: Subiendo
Descarga (cinta retrocediendo)
Ciclo finalizado
En espera, enviar 'A' para inicializar
```

Fig. 9: Monitor serial durante una carga.

El timer usado fue el Timer1 CTC configurado a 1ms:

Listing 5: Código configuración Timer1 a 1ms

```
; Timer1 a 1ms
ldi r16, 0x00
sts TCCR1A, r16
ldi r16, (1<<WGM12) | (1<<CS11) | (1<<CS10)
sts TCCR1B, r16
ldi r16, low(249)
sts OCR1AL, r16
ldi r16, high(249)
sts OCR1AH, r16
```

Los delays se configuraron en ms usando OCF1A:

Listing 6: Código configuración delays a 1ms

```
DELAY_MS:
    tst r24
    brne DLY_LOOP
    tst r25
    breq DLY_FIN
DLY_LOOP:
    ldi r16, 0
    sts TCNT1H, r16
    sts TCNT1L, r16
    ldi r16, (1<<OCF1A)
    out TIFR1, r16
DLY_ESPERAR:
    in r17, TIFR1
    sbrs r17, OCF1A
    rjmp DLY_ESPERAR
    subi r24, 1
    sbci r25, 0
    brne DLY_LOOP
DLY_FIN:
    ret
```

El bucle principal para inicializar el sistema y la selección de cargas es el siguiente:

Listing 7: Bucle principal

```
PRINCIPAL:
    rcall LEDS_ESPERANDO

    ; Espera a que manden una A
    ldi ZL, low(msg_pedir_A<<1)
    ldi ZH, high(msg_pedir_A<<1)
    rcall USART_MANDAR_CADENA

ESPERA_A:
```

```

rcall USART_RX
cpi r16, 'A'
breq A_OK
cpi r16, 'a'
brne ESPERA_A
A_OK:
rcall USART_TX      ; Confirma la A y usa CRLF
ldi r16, 13
rcall USART_TX
ldi r16, 10
rcall USART_TX

rcall LEDS_FUNCIONANDO

; Seleccion de carga 1/2/3
ldi ZL, low(msg_pedir_carga<<1)
ldi ZH, high(msg_pedir_carga<<1)
rcall USART_MANDAR_CADENA

LEE_CARGA: ; LIGERA/MEDIANA/PESADA
rcall USART_RX
mov r18, r16
cpi r18, '1'
breq CARGA_LIGERA
cpi r18, '2'
breq CARGA_MEDIANA
cpi r18, '3'
breq CARGA_PESADA
rjmp LEE_CARGA

Por último, el bucle para las cargas es el siguiente:

Listing 8: Código bucle para las cargas

BUCLE_PIEZAS:
ldi ZL, low(msg_pieza<<1)
ldi ZH, high(msg_pieza<<1)
rcall USART_MANDAR_CADENA
mov r16, r26
subi r16, '-0'
rcall USART_TX
ldi r16, 13
rcall USART_TX
ldi r16, 10
rcall USART_TX

; Alimentacion
ldi ZL, low(msg_alimentacion<<1)
ldi ZH, high(msg_alimentacion<<1)
rcall USART_MANDAR_CADENA
rcall M1_ADELANTE
lds r24, t_avance
lds r25, t_avance+1
rcall DELAY_MS

rcall M1_PARAR
ldi ZL, low(msg_pausa<<1)
ldi ZH, high(msg_pausa<<1)
rcall USART_MANDAR_CADENA
lds r24, t_pausa
lds r25, t_pausa+1
rcall DELAY_MS

; Punzado
ldi ZL, low(msg_bajar<<1)
ldi ZH, high(msg_bajar<<1)
rcall USART_MANDAR_CADENA
rcall M3_BAJAR
ldi r24, low(T_M3_MEDIA)
ldi r25, high(T_M3_MEDIA)
rcall DELAY_MS
rcall M3_PARAR

ldi ZL, low(msg_mantener<<1)

ldi ZH, high(msg_mantener<<1)
rcall USART_MANDAR_CADENA
lds r24, t_mantener
lds r25, t_mantener+1
rcall DELAY_MS

ldi ZL, low(msg_subir<<1)
ldi ZH, high(msg_subir<<1)
rcall USART_MANDAR_CADENA
rcall M3_SUBIR
ldi r24, low(T_M3_MEDIA)
ldi r25, high(T_M3_MEDIA)
rcall DELAY_MS
rcall M3_PARAR

; Descarga
ldi ZL, low(msg_descarga<<1)
ldi ZH, high(msg_descarga<<1)
rcall USART_MANDAR_CADENA
rcall M1_ATRAS
lds r24, t_volver
lds r25, t_volver+1
rcall DELAY_MS
rcall M1_PARAR

sbiw r26, 1
brne BUCLE_PIEZAS

FIN_CICLO:
rcall LEDS_APAGAR_CARGAS

rcall LEDS_TERMINO
ldi ZL, low(msg_ciclo_final<<1)
ldi ZH, high(msg_ciclo_final<<1)
rcall USART_MANDAR_CADENA

ldi r24, low(T_FIN_MS)
ldi r25, high(T_FIN_MS)
rcall DELAY_MS

rjmp PRINCIPAL

```

Los videos del funcionamiento de la cinta transportadora con punzonadora y el monitoreo por USART quedan adjuntados en el Drive.

#### VI-B. Manejo de Matriz de LEDs 8x8 mediante UART

El programa desarrollado en lenguaje ensamblador para el ATmega328P permitió controlar una matriz LED de  $8 \times 8$ , mostrando tanto mensajes de texto como figuras predefinidas. El sistema incluye un menú interactivo por UART, donde el usuario selecciona qué desea visualizar en la matriz.

Uno de los resultados más relevantes es la posibilidad de modificar fácilmente tanto el mensaje a mostrar como la velocidad del desplazamiento (scroll). Para esto, se documentó el código con instrucciones claras:

#### Listing 9: Instrucciones para modificar caracteres

```

; Instrucciones para la matriz LED
; Entrar al editor: [https://xantorohara.github.io/
;   led-matrix-editor/]
; Dibujar la letra o figura que quieras.
; Copiar el patron hexadecimal que genera la pagina
;   (ej.: 0x81C3A59981818181).
; Para usarlo en la matriz, separar en bytes y
;   antepone 0x a CADA par de caracteres.
; Ejemplo: 0x81C3A59981818181     0x81, 0xC3, 0xA5,
;   0x99, 0x81, 0x81, 0x81, 0x81
; Con eso ya se puede cargar y correr en la matriz.

```

De este modo, cualquier carácter o figura diseñada en el editor puede trasladarse directamente al programa mediante tablas .db. El mensaje final se arma en la tabla MENSAJE, donde cada entrada apunta a un carácter ya definido:

#### Listing 10: Definición del mensaje en la matriz LED

```
; Tabla de punteros a caracteres para formar el
; mensaje "ME GUSTA COMER ASADO"
MENSAJE:
.dw (M<<1), (E<<1), (ESPACIO<<1), (G<<1), (U<<1), (S
<<1), (T<<1), (A<<1), (ESPACIO1<<1), \
(C<<1), (O<<1), (M1<<1), (E2<<1), (R<<1), (
    ESPACIO2<<1), (A1<<1), (S1<<1), (A2<<1), \
(D<<1), (O1<<1), (ESPACIO3<<1)
```

Asimismo, la velocidad de desplazamiento del texto está controlada por una constante en el código, lo que permite ajustar la experiencia visual sin modificar la lógica del programa:

#### Listing 11: Constante para ajustar la velocidad del scroll

```
; VELOCIDAD: numero de frames (barridos de 8 filas)
; por cada desplazamiento
; de 1 columna en el scroll. Mayor valor = scroll
; mas lento.
.equ VELOCIDAD = 10
```

Gracias a esta parametrización, es posible acelerar o ralentizar el movimiento simplemente cambiando el valor asignado a VELOCIDAD.

Además, el resto del código implementa tres funciones principales. La primera corresponde al scroll de texto, donde se combinan de manera dinámica dos caracteres consecutivos desplazando sus columnas bit a bit, lo que genera la ilusión de un mensaje en movimiento continuo. La segunda corresponde al monitor serial, que permite al usuario interactuar con el sistema: al recibir un carácter por UART se ejecuta la opción seleccionada y se imprime un mensaje de confirmación en la terminal. Finalmente, la tercera función destacada es la visualización de figuras estáticas, que pueden mostrarse indefinidamente o por un tiempo aproximado de tres segundos, gestionado mediante retardos basados en el temporizador interno.

Luego de terminado el código y documentado, se pasó a la etapa de simulación para comprobar que todo funcionara como estaba pensado. Seguidamente, en Fig 10 se muestra el circuito simulado.

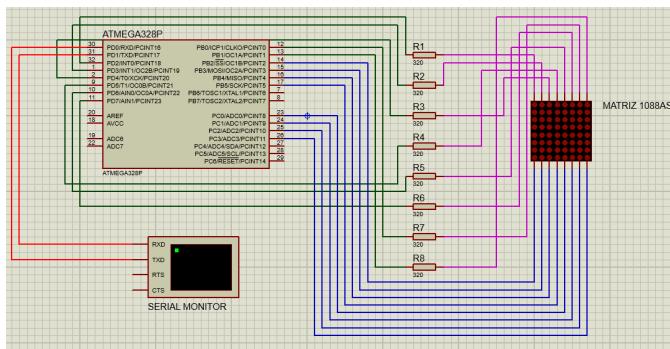


Fig. 10: Circuito para la Matriz de LEDs (Simulado)

A continuación, se procede a mostrar las imágenes capturadas de las figuras visualizadas en la matriz LED de manera simulada.

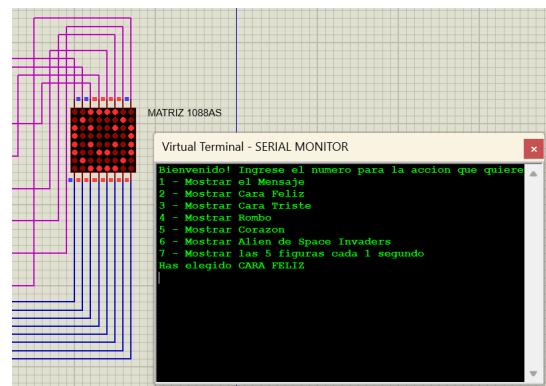


Fig. 11: Figura Cara Feliz (Simulado)

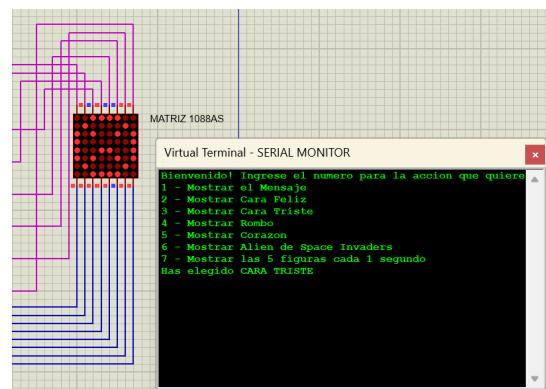


Fig. 12: Figura Cara Triste (Simulado)

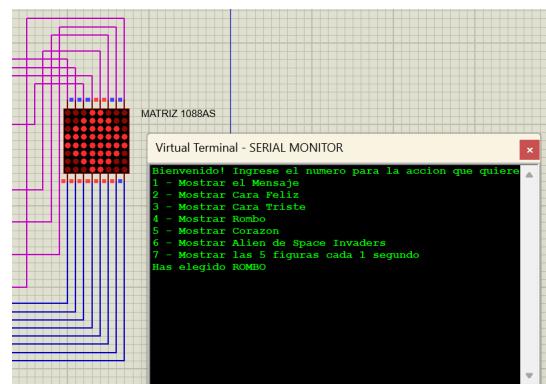


Fig. 13: Figura Rombo (simulado)

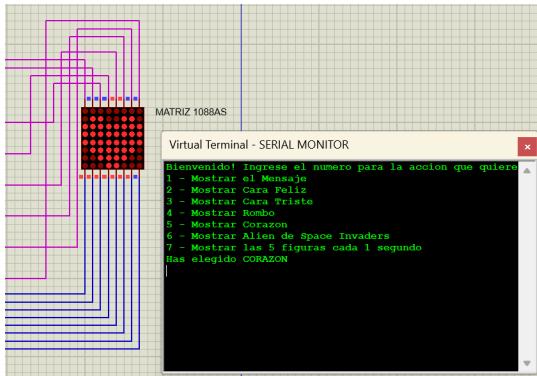


Fig. 14: Figura Corazón (simulado)

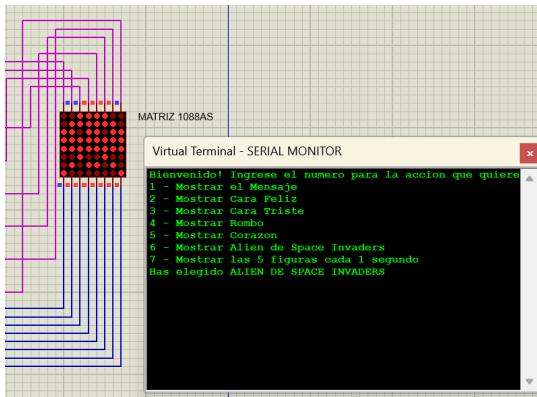


Fig. 15: Figura Alién de Alíen Space Invaders (Simulado)

Para el caso del mensaje desplazándose como para las figuras mostradas individualmente durante tres segundos se adjuntan un video en Drive para cada caso. Además, se adjunta en Drive un video del funcionamiento completo simulado, incluyendo el scroll del texto, la visualización de cada una de las caras y la secuencia animada de las figuras.

Una vez comprobado que la simulación funciona correctamente se procedió a realizar el montaje en físico como se muestra en Fig. 16

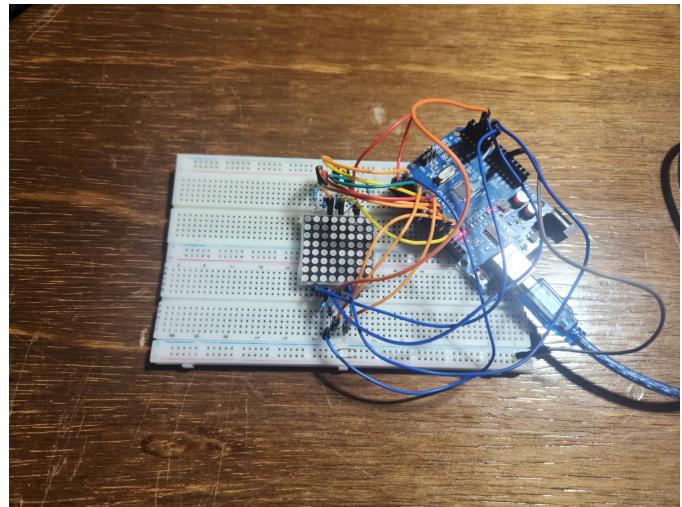


Fig. 16: Circuito para la Matriz de LEDs (Físico)

Posteriormente, se presentan las imágenes capturadas de las figuras visualizadas en la matriz LED de manera física.

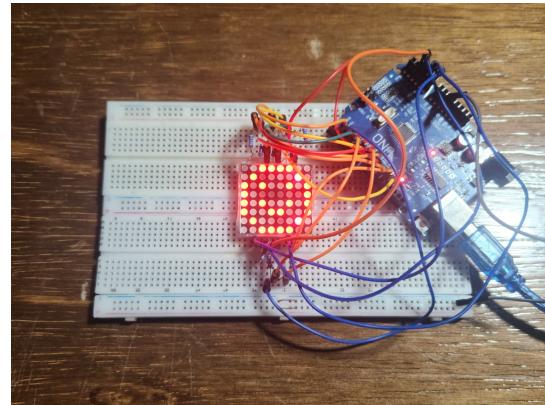


Fig. 17: Figura Cara Feliz (físico)

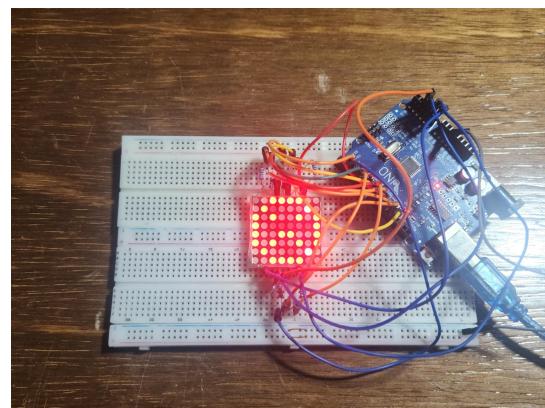


Fig. 18: Figura Cara Triste (físico)

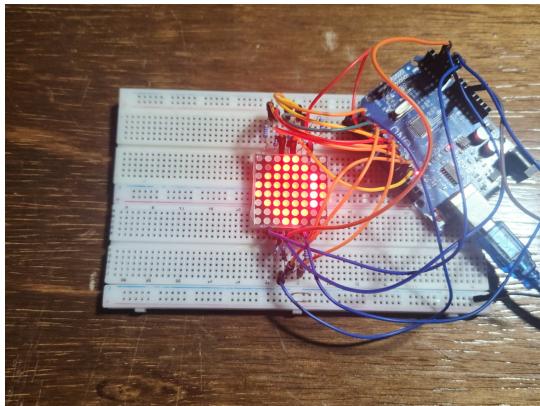


Fig. 19: Figura Rombo (físico)

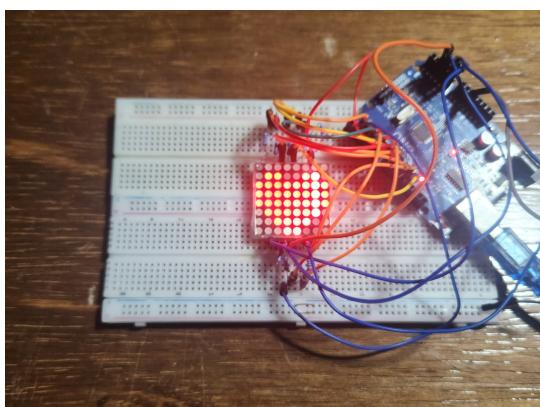


Fig. 20: Figura Corazón (físico)

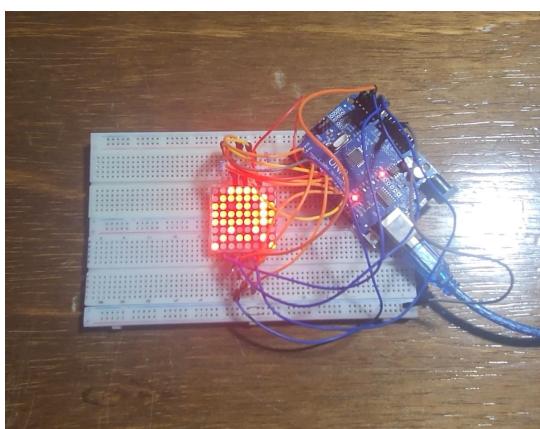


Fig. 21: Figura Alién de Alién Space Invaders (físico)

De la misma manera que en la simulación, se comparte en Drive u video del funcionamiento del montaje físico, donde se puede observar el desplazamiento del mensaje, la aparición de cada una de las figuras y la animación secuencial de las mismas.

#### VI-C. DAC R-2R con LUT

La **Señal 4** se generó a partir de una LUT de **256 muestras** con rampa descendente ( $0xFF \rightarrow 0x00$ ). El programa, escrito en ensamblador para el ATmega328P, usa el **Timer2 en CTC** como “reloj” de muestreo y, en cada pulso de comparación, envía a PORTD el **byte completo** que alimenta la red R-2R (PD7 = MSB, PD0 = LSB). La tabla está en FLASH y se recorre con Z mediante LPM Z+. El vector de interrupción empleado es el de **TIMER2\_COMPA** (OC2Aaddr, definido en `m328pdef.inc`). Al inicializar y comparar se utiliza `LUT*2/LUTFIn*2` porque la memoria de programa del AVR se direcciona por *palabras* y la LUT está en bytes.

Para que se vea con claridad qué hace el código, a continuación se muestran los extractos tal como están en el código (con `OCR2A_VAL = 30`) y luego se comenta brevemente qué aporta cada bloque.

Listing 12: Vectores, arranque y configuración de Timer2 (código original con `OCR2A_VAL=30`)

```
.cseg
.org 0x0000
    rjmp Inicio          ; Reset vector
.org OC2Aaddr
    rjmp Interrupcion   ; TIMER2_COMPA

; Registros de trabajo
.def aux1 = r16
.def aux2 = r17
.def valor = r18          ; Byte leido de LUT

.equ OCR2A_VAL = 30        ; Se varia segun la
                            Fout buscada

Inicio:
; Stack
ldi aux1, high(RAMEND)
out SPH, aux1
ldi aux1, low(RAMEND)
out SPL, aux1

; GPIO: PORTD salida (8 bits al R-2R)
ldi aux1, 0xFF
out DDRD, aux1
ldi aux1, 0x00
out PORTD, aux1

; Timer2 en CTC
ldi aux1, OCR2A_VAL
sts OCR2A, aux1          ; Valor de
                           comparacion
ldi aux1, (1<<WGM21)
sts TCCR2A, aux1          ; CTC con OCR2A
ldi aux1, (1<<OCF2A)
out TIFR2, aux1           ; Limpiar OCF2A
ldi aux1, (1<<OCIE2A)
sts TIMSK2, aux1           ; Habilitar
                           interrupcion COMPA
ldi aux1, (1<<CS20)
sts TCCR2B, aux1           ; Prescaler = 1

; Puntero de LUT en FLASH
ldi ZL, low(LUT*2)
ldi ZH, high(LUT*2)

sei                         ; Habilitar
                           interrupciones
```

En este bloque se prepara todo lo necesario antes de “leer” la tabla: se declara PORTD como salida, se pone a cero, y se deja el **Timer2** en CTC con WGM21=1. OCR2A toma 30 (por OCR2A\_VAL), se **limpia el flag** OCF2A escribiéndolo en TIFR2 (así la primera interrupción no arranca con un pendiente “viejo”), se habilita la IRQ con OCIE2A y se escoge **prescaler = 1** con CS20=1. Finalmente Z apunta al inicio de la LUT en FLASH y SEI libera las interrupciones.

Con CS20=1 (prescaler  $N = 1$ ) y OCR2A=30, la frecuencia de muestreo es

$$f_s = \frac{f_{clk}}{1 \cdot (OCR2A + 1)} = \frac{16 \text{ MHz}}{31} \approx 516.13 \text{ kHz},$$

y, como se recorren 256 muestras por período, la salida queda en

$$f_{out} = \frac{f_s}{256} \approx 2.016 \text{ kHz}.$$

Para las mediciones comparativas que se muestran más abajo también se trabajó con **prescaler  $N = 8$**  (ajuste en TCCR2B), donde

$$f_s = \frac{f_{clk}}{8(OCR2A + 1)} \quad \text{y} \quad f_{out} = \frac{f_s}{256} = \frac{7812,5}{OCR2A + 1} \text{ Hz}$$

( $f_{clk} = 16 \text{ MHz}$ ). Cambiar  $N$  sólo varía el período de la rampa; la forma y la amplitud no se alteran.

El segundo bloque es la **ISR**. Aquí está la lógica que, a cada tick, lee el próximo byte de la tabla y lo envía al DAC de golpe:

Listing 13: Rutina de servicio: lectura de LUT y salida a PORTD (código original)

```
; Interrupcion Timer2 COMPA
Interrupcion:
    lpm valor, Z+
    muestra                                ; Leer siguiente
    out  PORTD, valor                      ; Salida a 2R-R

    ; Volver al inicio si Z llego al fin de LUT
    ldi aux1, low(LUTFin*2)
    ldi aux2, high(LUTFin*2)
    cp  ZL, aux1
    cpc ZH, aux2
    brlo seguir
    ldi ZL, low(LUT*2)
    ldi ZH, high(LUT*2)
seguir:
    reti
```

La LPM Z+ toma el siguiente byte de FLASH y OUT PORTD, valor actualiza los **ocho bits** de una sola vez; así se evitan *glitches* por cambios desfasados entre líneas. El chequeo cp/cpc contra LUTFin\*2 detecta el final de la tabla y resetea el puntero para repetir el ciclo sin cortes.

En el osciloscopio se observó la rampa prevista con **256 escalones** de ancho temporal uniforme. El paso entre niveles coincide con el LSB ideal de 8 bits,

$$\Delta V = \frac{V_{ref}}{2^8},$$

que para  $V_{ref} = 5 \text{ V}$  vale  $\Delta V \approx 19.53 \text{ mV}$ . El salto de 0x00 a 0xFF al cierre del período aparece breve y definido, como

corresponde a esta LUT descendente. La ISR es corta (decenas de ciclos), por lo que no se observaron problemas de sobrecarga ni escalonado.

En la Fig. 22 se muestra la simulación en Proteus utilizada para validar el comportamiento antes del montaje.

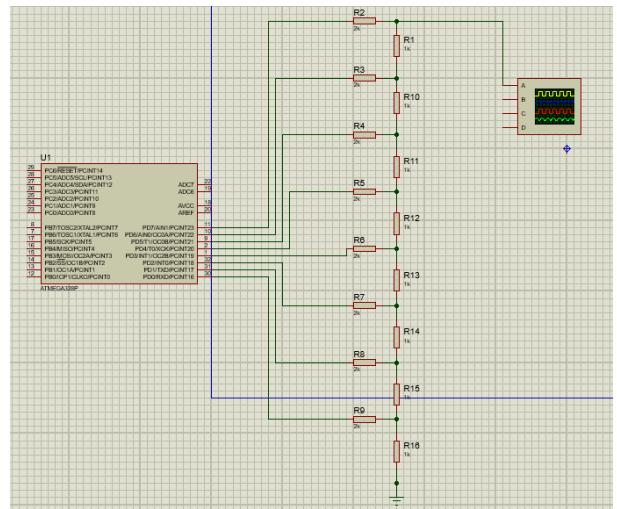


Fig. 22: Simulación del DAC R-2R en Proteus con el ATmega328P (esquema del circuito).

La Fig. 23 presenta la rampa descendente en el osciloscopio virtual de Proteus.

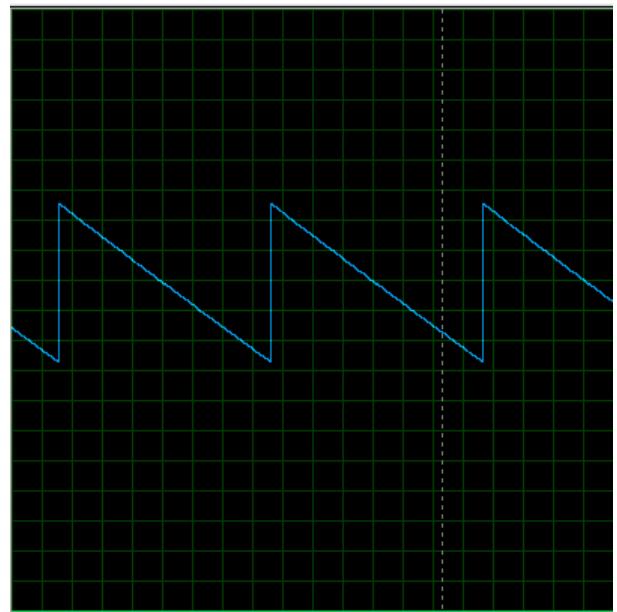


Fig. 23: Señal de rampa descendente observada en el osciloscopio virtual de Proteus.

Luego se montó el circuito en protoboard (Fig. 24) y se registró la señal en el osciloscopio real (Fig. 25), confirmando la correspondencia simulación-práctica.

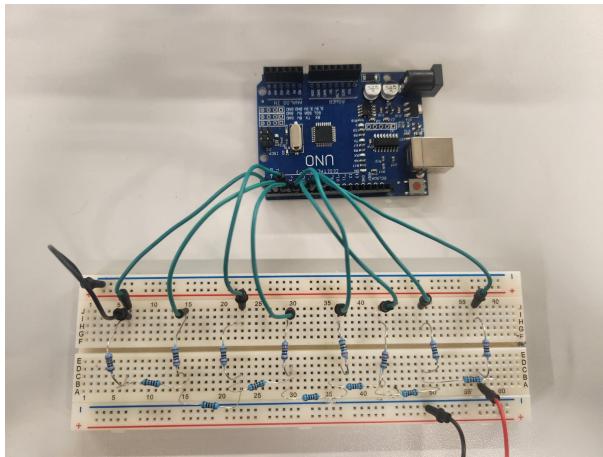


Fig. 24: Implementación física del DAC R-2R en protoboard con el ATmega328P.

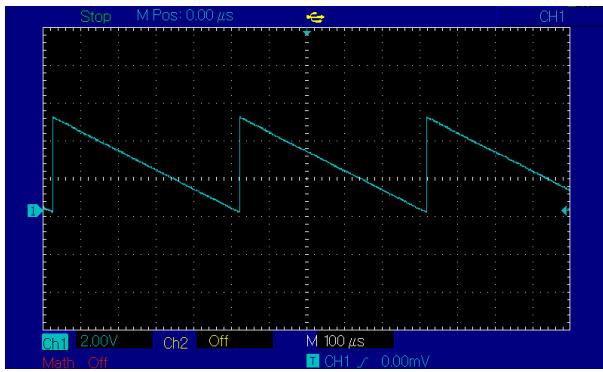


Fig. 25: Señal de rampa descendente observada en osciloscopio real.

Para analizar la variación de frecuencia se realizaron seis pruebas cambiando .equ OCR2A\_VAL en modo CTC ( $f_{clk} = 16$  MHz) con **prescaler** = 8; la base fue OCR2A=30. El Cuadro X resume OCR2A y la frecuencia teórica. En todos los casos la forma de rampa se mantuvo estable y la variación de OCR2A se reflejó únicamente en el período.

Cuadro X: Frecuencia teórica para distintas configuraciones de OCR2A ( $N = 256$ ).

Config.	OCR2A	Frecuencia teórica (Hz)
1 (base)	30	$\approx 252,02$
2	249	31,25
3	124	62,50
4	49	156,25
5	24	312,50
6	11	$\approx 651,04$

Las Figs. 26–31 muestran las capturas para cada ajuste de OCR2A; se aprecia la conservación de la forma y el cambio del período según la expresión teórica.



Fig. 26: Configuración 1: OCR2A=30 (base).



Fig. 27: Configuración 2: OCR2A=249.



Fig. 28: Configuración 3: OCR2A=124.

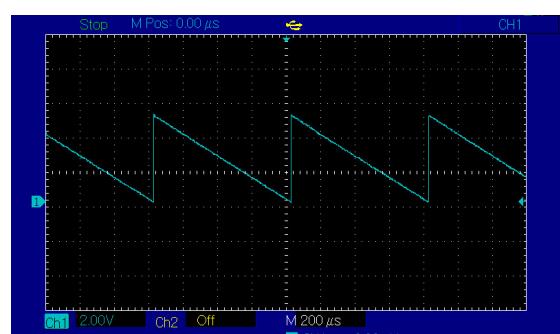


Fig. 29: Configuración 4: OCR2A=49.

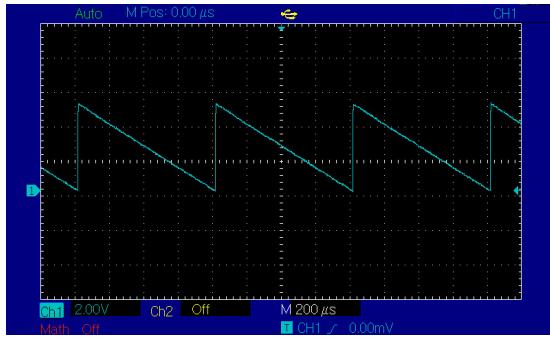


Fig. 30: Configuración 5: OCR2A=24.

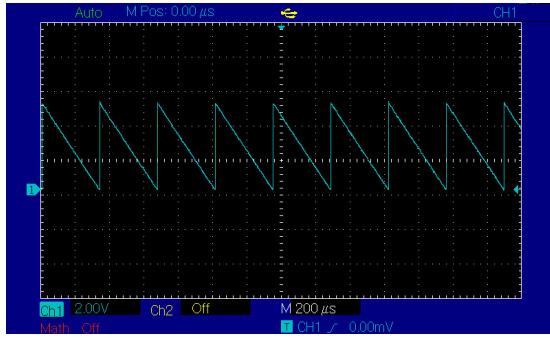


Fig. 31: Configuración 6: OCR2A=11.

En el Drive se adjuntan videos complementarios de la simulación y del montaje físico.

#### VI-D. Control del Plotter con ATmega328P

La máquina utilizada fue un plotter controlado por un PLC como se observa en Figura 32, al cual se conectó el microcontrolador ATmega328P para el envío de órdenes de desplazamiento y control del solenoide.

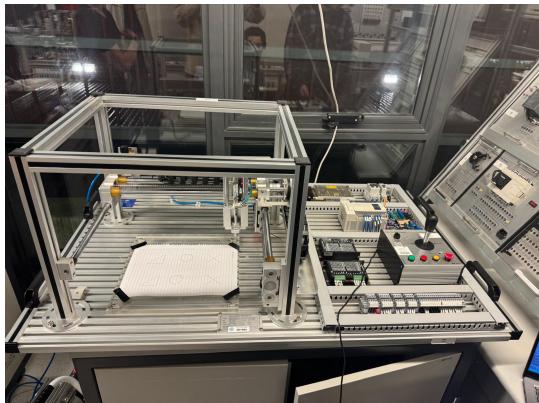


Fig. 32: Maquina del plotter

Para el triángulo, Al seleccionar la opción 1 USART, el sistema ejecuta esta rutina y dibuja el triángulo, se diseñó una rutina que primero posiciona el útil de trazo, luego baja el solenoide y ejecuta los desplazamientos hasta formar la figura, finalizando con el retorno al origen. El siguiente fragmento de código muestra la implementación:

Listing 14: Código para el dibujo de un Triangulo

```

figura_triangulo:
; POSICIONAMIENTO
    ldi      r20, (1<<PD7)
    out     PORTD, r20
    rcall   DELAY_20S_LARGO

    ldi      r20, (1<<PD7)
    out     PORTD, r20
    rcall   DELAY_10S_LARGO

    ldi      r20, (1<<PD4)
    out     PORTD, r20
    rcall   DELAY_2S_LARGO

; TRIANGULO
    ldi      r20, (1 <<PD2)
    out     PORTD, r20
    rcall   DELAY_2S_LARGO

    ldi      r20, (1<<PD4)
    out     PORTD, r20
    rcall   DELAY_5S_LARGO

    ldi      r20, (1<<PD7)
    out     PORTD, r20
    rcall   DELAY_5S_LARGO

    ldi      r20, (1<<PD6) | (1<<PD5)
    out     PORTD, r20
    rcall   DELAY_5S_LARGO

    ; VUELVE AL ORIGEN
    ldi      r20, (1<<PD3)
    out     PORTD, r20
    rcall   DELAY_1S

    ldi      r20, (1<<PD6)
    out     PORTD, r20
    rcall   DELAY_20S_LARGO

    ldi      r20, (1<<PD6)
    out     PORTD, r20
    rcall   DELAY_10S_LARGO

    ldi      r20, (1<<PD5)
    out     PORTD, r20
    rcall   DELAY_2S_LARGO

    ret

```

Se realizó un triángulo rectángulo debido a la limitación de no poder realizar un triángulo equilátero ya que los desplazamientos de la máquina se encuentran restringidos a 45 grados.

En el caso del círculo, Al ingresar el número 2 en la interfaz serial, se inicia el dibujo del círculo. El código completo no se muestra por su extensión, pero su funcionamiento se basa en el algoritmo de Bresenham mencionado anteriormente. Este algoritmo permitió aproximar el trazo a un círculo a través de micro-movimientos.

Para la cruz, el planteo consistió en realizar el posicionamiento inicial y luego bajar el solenoide para trazar la primera diagonal. Una vez finalizada, se levantó el solenoide y se efectuó un desplazamiento vertical hacia arriba en el eje Y. Posteriormente, se volvió a bajar el solenoide para realizar la segunda diagonal. Finalmente, se ejecutó el retorno al origen de

trabajo. El siguiente código corresponde a la rutina utilizada:

Listing 15: Código para el dibujo de una Cruz

```

figura_cruz:
; POSICIONAMIENTO
    ldi    r20, (1<<PD7)
    out   PORTD, r20
    rcall  DELAY_5S_LARGO

    ldi    r20, (1<<PD4)
    out   PORTD, r20
    rcall  DELAY_2S_LARGO

; CRUZ
    ldi    r20, (1<<PD2)
    out   PORTD, r20
    rcall  DELAY_2S_LARGO

    ldi    r20, (1<<PD4) | (1<<PD7)
    out   PORTD, r20
    rcall  DELAY_5S_LARGO

    ldi    r20, (1<<PD3)
    out   PORTD, r20
    rcall  DELAY_2S_LARGO

    ldi    r20, (1<<PD5)
    out   PORTD, r20
    rcall  DELAY_5S_LARGO

    ldi    r20, (1<<PD2)
    out   PORTD, r20
    rcall  DELAY_2S_LARGO

    ldi    r20, (1<<PD4) | (1<<PD6)
    out   PORTD, r20
    rcall  DELAY_5S_LARGO

; VUELVE AL ORIGEN
    ldi    r20, (1<<PD3)
    out   PORTD, r20
    rcall  DELAY_1S_LARGO

    ldi    r20, (1<<PD5)
    out   PORTD, r20
    rcall  DELAY_5S_LARGO

    ldi    r20, (1<<PD6)
    out   PORTD, r20
    rcall  DELAY_5S_LARGO

    ldi    r20, (1<<PD5)
    out   PORTD, r20
    rcall  DELAY_2S_LARGO

ret

```

Cuando se ingresa el número 3 en la interfaz serial, se ejecuta la rutina y la máquina dibuja la cruz.

Finalmente, se integraron las tres rutinas en una rutina, que al recibir la letra “t” o “T” en la interfaz serial realiza en secuencia el triángulo, el círculo y la cruz.

En el código se implementaron dos rutinas de *delay* distintas basadas en el Timer1: una versión “corta” y otra “larga”. La diferencia entre ambas está en el valor de precarga de TCNT1, lo que genera distintos tiempos de overflow. De esta manera, se obtienen pausas rápidas para los trazos y movimientos finos, y pausas más largas para los desplazamientos de posicionamiento y retorno, permitiendo adaptar el mismo temporizador a diferentes necesidades de la máquina.

En la Figura 33 se muestra el resultado final de las figuras trazadas por el plotter. En la parte superior se aprecian el triángulo, el círculo y la cruz ejecutados de forma individual, mientras que en la parte inferior se observan las tres figuras realizadas en secuencia al introducir la letra “t” o “T” en la interfaz USART.

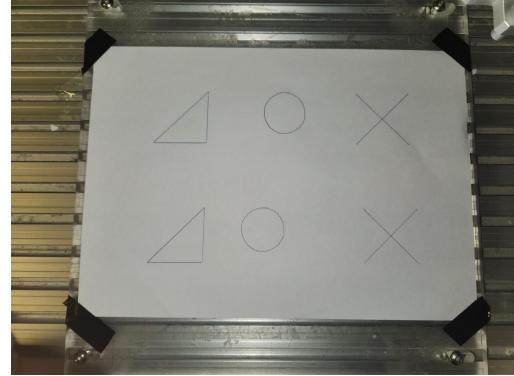


Fig. 33: Figuras realizadas por el plotter

En el Drive adjunto se muestra un video del funcionamiento.

## VII. CONCLUSIONES

La comunicación USART para controlar la cinta transportadora con punzonadora se llevó a cabo correctamente, siendo capaz el usuario de inicializar el sistema a través de esta, a la vez de modificar características de la carga, como la cantidad de cargas a enviar y el peso de esta.

Aunque tiene varias factores a mejorar, entre ellos, la implementación funcional de los sensores y un correcto uso de PWM para ajustar las velocidades a las necesidades requeridas por el tipo de carga.

La matriz de LED de  $8 \times 8$  funcionó correctamente, logrando mostrar mensajes en desplazamiento de 12 caracteres, figuras predefinidas y esas mismas figuras presentadas por intervalos de tiempo. La simulación previa y las pruebas en físico confirmaron la efectividad del control mediante el ATmega328P y USART. Además, la documentación incluida en el código facilita la modificación del mensaje y la configuración de la velocidad del desplazamiento.

Como posibles mejoras en el código se identifican el uso de interrupciones en lugar de consultar de manera continua las banderas de estado, lo que permitiría un refresco más estable y eficiente; la reducción de push/pop innecesarios para optimizar el uso de la pila.. Estas optimizaciones aportarían mayor claridad y rendimiento al programa manteniendo la misma funcionalidad.

Se implementó con éxito un conversor digital-análogo mediante una red R-2R de 8 bits controlada por el ATmega328P, generando la Señal 4 a partir de una LUT de 256 muestras con rampa descendente. La metodología incluyó validación previa en Proteus y verificación experimental en protoboard, observándose en ambos casos la forma prevista.

Las capturas de osciloscopio mostraron el *escalonado* propio de la cuantización y de la discretización por LUT, coherente

con el LSB  $\Delta = V_{ref}/2^8$  (para  $V_{ref} = 5$  V,  $\Delta \approx 19.53$  mV). En modo CTC (Timer2, prescaler 8) y manteniendo OCR2A\_VAL en {30, 249, 124, 49, 24, 11}, la frecuencia de salida teórica para  $N = 256$  fue {~252.02 Hz, 31.25 Hz, 62.50 Hz, 156.25 Hz, 312.50 Hz, ~651.04 Hz}, en concordancia con las variaciones observadas.

Como líneas de mejora, el aumento del tamaño de la LUT o la incorporación de un filtro pasa-bajos y/o un buffer en la salida podrían reducir el ripple y el efecto escalonado sin modificar la forma general de la señal. Además, el uso de resistencias con una menor tolerancia podría mejorar la uniformidad de los niveles y la repetibilidad de los resultados. Sin embargo, en este caso, no es esencial, ya que la señal cuenta con 256 muestras, lo cual es más que suficiente. En señales con menos muestras, sin embargo, estas mejoras podrían resultar más recomendables.

La experiencia con el control del plotter mediante el microcontrolador ATmega328P permitió comprobar la correcta comunicación con el PLC y la ejecución de figuras geométricas de forma estable. Se constató que las restricciones mecánicas del equipo, limitado a desplazamientos en múltiplos de 45 grados, condicionan las formas posibles, impidiendo por ejemplo un triángulo equilátero. Sin embargo, fue posible aproximar el círculo mediante el algoritmo de Bresenham y realizar trazos confiables para el triángulo y la cruz. Cabe señalar que, si bien el algoritmo resultó adecuado, una mayor exactitud en el círculo podría lograrse aumentando la cantidad de trazos de píxeles generados.

La interfaz serial demostró ser una herramienta práctica para la interacción con el usuario, ya que permitió seleccionar la figura a trazar o la secuencia completa de manera directa. Además, la calibración inicial de tiempos garantizó movimientos reproducibles y seguros en cada ejecución.

En conjunto, este apartado permitió integrar los conceptos de control digital, temporización y programación en bajo nivel. Como recomendación, se sugiere optimizar el código eliminando timers que no resultan necesarios y empleando funciones que simplifiquen las rutinas de selección de figuras, lo que contribuiría a un programa más compacto, eficiente y fácil de mantener. Asimismo, se propone unificar las rutinas de *delay* en una sola función parametrizable, evitando la duplicación entre delays “cortos” y “largos” y facilitando la calibración del temporizador.

## VIII. ANEXOS

1. Link al Repositorio de GITHUB: <https://github.com/juanferreiram-cell/Tec.Microprocesamiento>
2. Link a la Carpeta de Drive con los videos: <https://drive.google.com/drive/folders/1dfyBWJg07bqZ1Sa87PI9gOdNfCK04ClB?usp=sharing>

## IX. BIBLIOGRAFÍA

1. EduTechnik. (2025, Abril 4). Arduino UNO F5 adapter- EduTechnik. <https://edutechnik.com/producto/arduino-uno-f5-adapter/>

2. Hurtado, A. V. C. (s.f.). AOE - Abre los Ojos al Ensamblador. <https://abreojosensamblador.epizy.com/?Tarea=1&SubTarea=23&i=1>
3. Matriz LED 8x8 1088AS de 3 mm. (s.f.). ROBOTLANDIA. <https://robotlandia.es/display/728-matriz-led-88-1088as-de-3-mm.html>
4. MSMK. (2025, Febrero 26). El lenguaje Assembler. <https://msmk.university/el-lenguaje-assembler/>
5. Robotics TXT 4.0 Base Set. (s.f.). fischertechnik. <https://www.fischertechnik.de/en/products/schools/robotics/559888-robotics-txt-4-0-base-set>
6. Shinde, S. (s.f.). 2016\_Python based 3-Axis CNC Plotter. Scribd. <https://es.scribd.com/document/460138453/2016-Python-based-3-Axis-CNC-Plotter>
7. USART vs. UART: What’s the Difference? (2023, Febrero 8). Cadence. <https://resourcespcb.cadence.com/blog/usart-vs-uart-whats-the-difference>