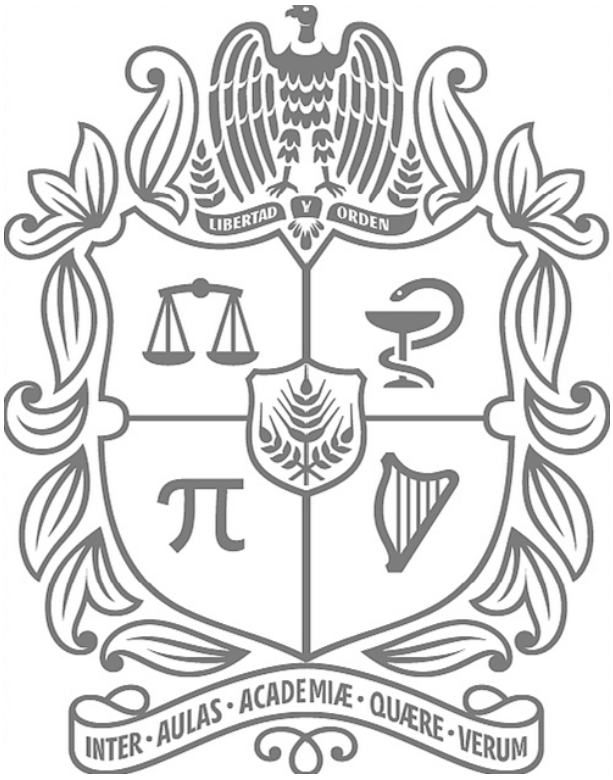

Celda Braille Actualizable

David Esteban Becerra López
Anderson Camilo Rosero Yela
Laura Daniela Zambrano Guacheta
José Luis Pulido Fonseca

Electrónica Digital II
Universidad Nacional de Colombia

Abstract

Este documento presenta el diseño, implementación y validación técnica de una celda Braille actualizable de 6 puntos, destinada a facilitar el acceso a la lectura para personas con discapacidad visual. La solución propuesta emplea una arquitectura basada en FPGA (Arty Z7-20) que controla seis servomecanismos distribuidos según la disposición estándar del sistema Braille. El sistema permite representar las 26 letras minúsculas del alfabeto latino mediante la conversión en tiempo real de caracteres ASCII a patrones táctiles, con un tiempo de respuesta inferior a 5 segundos. El diseño prioriza la modularidad, el bajo costo y la eficiencia energética. Los resultados demuestran la viabilidad del prototipo como bloque fundamental para futuras pantallas Braille multicelda, alineándose con los objetivos de inclusión educativa y accesibilidad universal.

Palabras clave: Braille, discapacidad visual, FPGA, servomotores, accesibilidad, inclusión digital, tecnologías asistivas.

Contents

1 Introducción

1.1 Contexto y problemática

El acceso a la información escrita constituye un derecho fundamental para el desarrollo educativo, laboral y social de cualquier individuo. Sin embargo, para las personas ciegas o con baja visión, este acceso se ve severamente limitado por la escasa disponibilidad de materiales en formatos táctiles. Según datos de la Organización Mundial de la Salud (OMS), más de 36 millones de personas en el mundo presentan ceguera total, mientras que 217 millones sufren discapacidad visual moderada o severa [1]. En América Latina, esta situación se agrava por la limitada oferta de recursos inclusivos, generando barreras estructurales que perpetúan la exclusión educativa y social.

La brecha de accesibilidad se evidencia en que menos del 10% de las publicaciones editoriales están disponibles en Braille o formatos digitales compatibles con tecnologías de asistencia [6]. Adicionalmente, los dispositivos electrónicos Braille comerciales presentan costos prohibitivos (USD 1.500–5.600 por unidad), lo que los hace inaccesibles para bibliotecas públicas y usuarios individuales en contextos de recursos limitados.

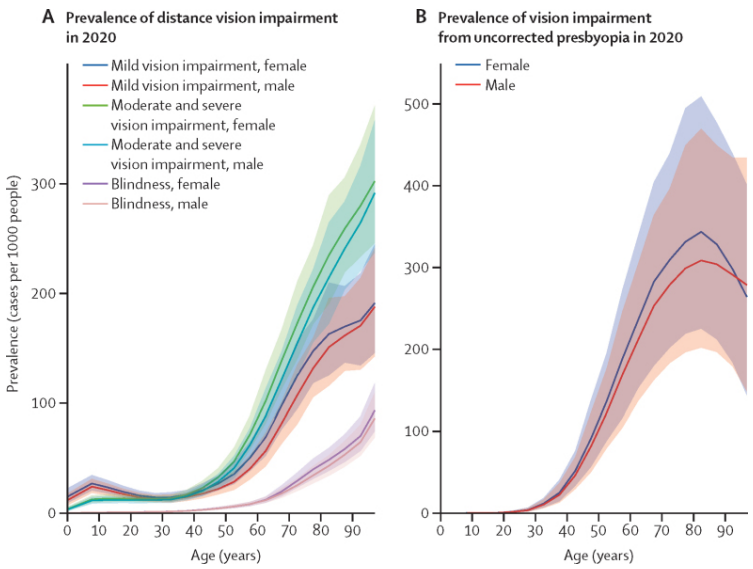

Figure 1: Distribución de la discapacidad visual por edad y sexo. La alta prevalencia refuerza la necesidad de tecnologías de lectura accesible.

1.2 Justificación del proyecto

Ante este panorama, se plantea la necesidad de desarrollar soluciones tecnológicas de bajo costo que permitan la lectura autónoma de textos físicos. Este proyecto responde a dicha necesidad mediante el diseño de una **celda Braille actualizable de 6 puntos**, concebida como módulo básico para futuras pantallas táctiles multicelda. La solución se enmarca en el Objetivo de Desarrollo Sostenible 4 (Educación de Calidad), buscando reducir las barreras de acceso al conocimiento. El enfoque modular permite una implementación escalable y sostenible, donde cada celda funciona como unidad independiente que puede integrarse en configuraciones más complejas.

1.3 Análisis PESTAL del contexto

Para comprender el entorno de implementación de la solución, se realizó un análisis PESTAL que considera dimensiones políticas, económicas, sociales, tecnológicas, ambientales y legales:

- **Político:** En Colombia, la Ley 1618 de 2013 establece la obligatoriedad de garantizar el acceso a la información para personas con discapacidad. El desarrollo de tecnologías asistivas se alinea con estas políticas públicas de inclusión.
- **Económico:** Los elevados costos de los dispositivos Braille comerciales contrastan con el presupuesto limitado de bibliotecas públicas e instituciones educativas. Nuestra propuesta reduce costos mediante un diseño modular y componentes asequibles.
- **Social:** La autonomía en la lectura fortalece la autoestima, la independencia y las oportunidades de desarrollo personal y profesional de las personas con discapacidad visual.
- **Tecnológico:** Si bien existen lectores de pantalla e impresoras Braille, las pantallas actualizables en tiempo real ofrecen ventajas únicas: son silenciosas, permiten navegación dinámica y pueden integrarse con sistemas digitales existentes.
- **Ambiental:** La selección de servomotores en lugar de solenoides electromagnéticos reduce significativamente el consumo energético. Además, la arquitectura modular facilita reparaciones específicas, disminuyendo desechos electrónicos.

- **Legal:** El Tratado de Marrakech (2013) promueve la creación y distribución de obras en formatos accesibles. Nuestra solución contribuye al cumplimiento de este tratado en el ámbito bibliotecario.

2 Objetivos del proyecto

2.1 Objetivo general

Diseñar, implementar y validar técnicamente un prototipo funcional de celda Braille actualizable de 6 puntos, controlable digitalmente mediante FPGA, que cumpla con estándares de seguridad, eficiencia energética y tiempo de respuesta.

2.2 Objetivos específicos

1. Implementar el mapeo completo de las 26 letras minúsculas del alfabeto latino al sistema Braille de 6 puntos.
2. Garantizar un tiempo de respuesta total del sistema menor a 5 segundos entre la entrada del carácter y la configuración táctil correspondiente.
3. Desarrollar una interfaz de entrada múltiple (UART y teclado matricial) para la selección de caracteres.
4. Implementar un sistema de control PWM eficiente para el posicionamiento preciso de seis servomotores.
5. Validar la escalabilidad del diseño como módulo básico para configuraciones multicelda.

3 Arquitectura técnica del sistema

3.1 Selección de actuadores: Servomotores vs. Solenoides

Inicialmente, el proyecto consideró el uso de solenoides electromagnéticos como mecanismo de actuación. Sin embargo, durante la fase de pruebas se identificaron limitaciones significativas:

- Alto consumo de corriente en estado activo (350 mA por solenoide)

- Generación de calor por efecto Joule durante operación continua
- Requerimiento de circuitos de potencia complejos
- Menor precisión en el posicionamiento mecánico

Por estas razones, se optó por servomotores modelo SG90, los cuales presentan ventajas clave:

- Consumo moderado (100-250 mA solo durante el movimiento)
- Posicionamiento angular preciso (0°-180° con resolución 1°)
- Control mediante señales PWM estándar (50 Hz)
- Menor costo por unidad y amplia disponibilidad

Esta decisión mejora la sostenibilidad energética del sistema y facilita la implementación del control digital.

3.2 Arquitectura de control basada en FPGA

El núcleo del sistema se implementó en una FPGA Artix-7 (placa Arty Z7-20), aprovechando sus capacidades de procesamiento paralelo y configurabilidad. La arquitectura incluye:

- **Unidad de procesamiento:** Microprocesador soft-core implementado en VHDL/Verilog
- **Interfaces de entrada:**
  - Teclado matricial 4×4 para entrada local
  - UART USB para comunicación con PC
  - Cuatro switches para selección de modos
  - Cuatro botones para funciones específicas
- **Interfaces de salida:**
  - Seis servomotores (controlados por dos puertos GPIO)
  - Pantalla LCD 16×2 vía I2C
  - LEDs de estado
- **Memoria:** BRAM interna para almacenamiento de caracteres y patrones

3.3 Algoritmo de conversión Braille

Se implementó una tabla de consulta (`braille_letters[26][6]`) que mapea cada letra minúscula (a-z) a su representación Braille según la disposición estándar:

Punto 1 • Punto 4
Punto 2 • Punto 5
Punto 3 • Punto 6

El algoritmo convierte caracteres ASCII en tiempo real y genera los patrones PWM correspondientes para los servomotores.

4 Actores de la solución

Table 1: Actores de la solución y sus roles.

Actor	**Rol**
Usuarios finales (personas con discapacidad visual)	Beneficiarios directos de la solución, cuya validación futura será crucial para evaluar usabilidad, ergonomía y eficacia en contextos reales de lectura.
Bibliotecas públicas	Entornos naturales de implementación donde el dispositivo puede democratizar el acceso a colecciones bibliográficas, cumpliendo con mandatos de inclusión.
Docentes y estudiantes	Responsables del diseño, desarrollo y validación académica del prototipo, garantizando que cumpla objetivos técnicos y pedagógicos.
Entidades regulatorias	Organismos encargados de verificar el cumplimiento de normativas de accesibilidad, seguridad y calidad para tecnologías asistivas.

5 Presupuesto y Materiales

A pesar de que se gastaron los solenoides, estos no fueron implementados en la práctica final debido a limitaciones de eficiencia energética identificadas durante las pruebas experimentales. La transición a servomotores representó una mejora significativa en términos de consumo energético y sostenibilidad ambiental.

Presupuesto			
Componentes	**Cantidad**	**Precio unidad**	**Precio total**
Pantalla LCD arduino	x1	$ 17.700,00	$ 17.700,00
Solenoide push-pull	x6	$ 6.700,00	$ 40.200,00
Membrana de botones	x1	$ 9.700,00	$ 9.700,00
Modulo TTL uart	x1	$ 10.400,00	$ 10.400,00
Diodos rectificadores	x6	$ 500,00	$ 3.000,00
Boton	x1	$ 1.000,00	$ 1.000,00
Nomina	x1	$ 2.400.000,00	$ 2.400.000,00
		Total	$ 2.482.000,00

Figure 2: Tabla de gastos detallada del proyecto

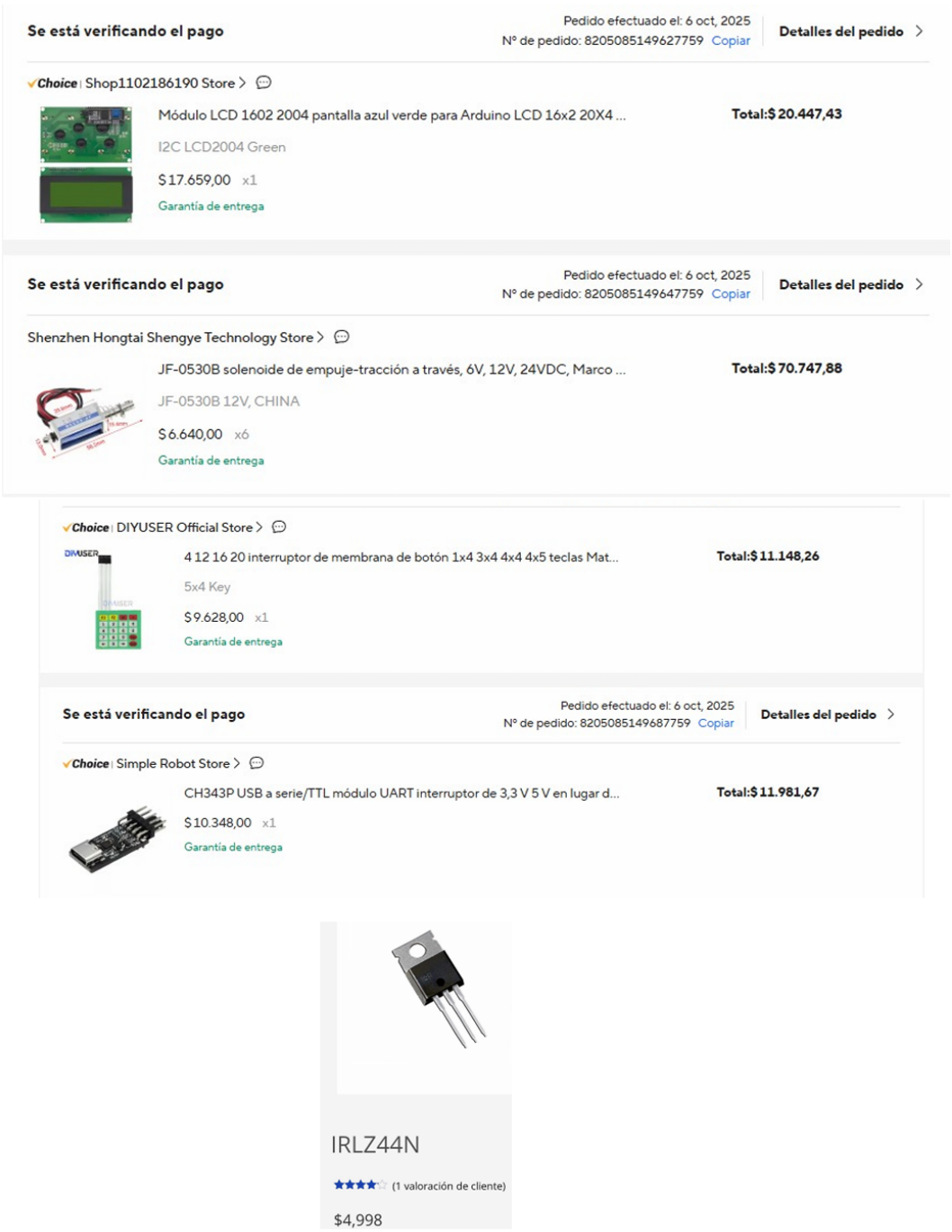

Figure 3: Desglose adicional de gastos y materiales

Análisis de los materiales: La inversión inicial en solenoides permitió realizar pruebas comparativas que demostraron la superioridad de los servomotores

en términos de eficiencia energética y control de posición. Aunque este gasto no se tradujo en componentes del prototipo final, fue fundamental para la toma de decisiones técnicas informadas que optimizaron el diseño final del sistema.

6 Desarrollo y validación experimental

6.1 Simulaciones iniciales

Como primer acercamiento al diseño del sistema, se realizó una simulación completa del circuito en Tinkercad. Esta etapa permitió validar la lógica de activación básica y el dimensionamiento de componentes, utilizando bobinas inductivas para representar los puntos Braille en un entorno virtual antes de proceder con la implementación física.

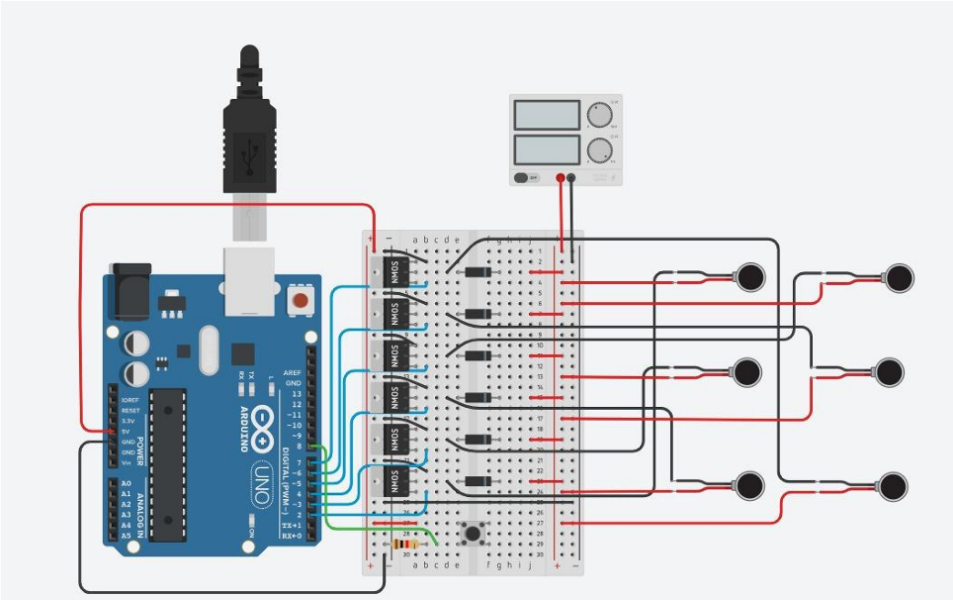

Figure 4: Prototipo simulado en Tinkercad para validación conceptual del diseño.

6.2 Implementación física con Arduino

Posteriormente, se implementó físicamente el circuito de prueba con un Arduino Uno, con el objetivo de verificar el comportamiento de los actuadores

y la etapa de potencia. Para ello se desarrolló el siguiente programa de validación:

Listing 1: Programa de prueba en Arduino para los seis actuadores.

```
// Programa de verificación para los 6 actuadores Braille
// Este código prueba secuencialmente cada punto de la celda

// Array de pines de Arduino utilizados para controlar los actuadores
int actuatorPins[6] = {2, 3, 4, 5, 6, 7};

void setup() { // Configuración inicial que se ejecuta una vez
  // Inicializar comunicación serial para monitoreo
  Serial.begin(9600);

  // Configurar pines de actuadores como salidas
  for(int i = 0; i <= 5; i++) {
    pinMode(actuatorPins[i], OUTPUT);
  }

  Serial.println("Sistema Braille - Prueba de actuadores iniciada");
}

void loop() { // Ciclo principal que se ejecuta continuamente
  // Activar cada actuador secuencialmente con intervalo de 1 segundo
  for(int i = 0; i <= 5; i++) {
    digitalWrite(actuatorPins[i], HIGH); // Activar actuador
    Serial.print("Actuador ");
    Serial.print(i+1);
    Serial.println(" activado");
    delay(1000); // Mantener activo por 1 segundo
    digitalWrite(actuatorPins[i], LOW); // Desactivar actuador
  }
}
```

La validación experimental confirmó la viabilidad del enfoque, pero también reveló las limitaciones energéticas de los solenoides, lo que motivó la transi-

ción a servomotores.

6.3 Diseño mecánico de los actuadores

Para la implementación final con servomotores, se desarrolló un diseño mecánico específico que garantiza precisión en el desplazamiento vertical de los puntos Braille. Cada servomotor controla un punto mediante un mecanismo de leva que convierte el movimiento rotacional en desplazamiento lineal vertical.

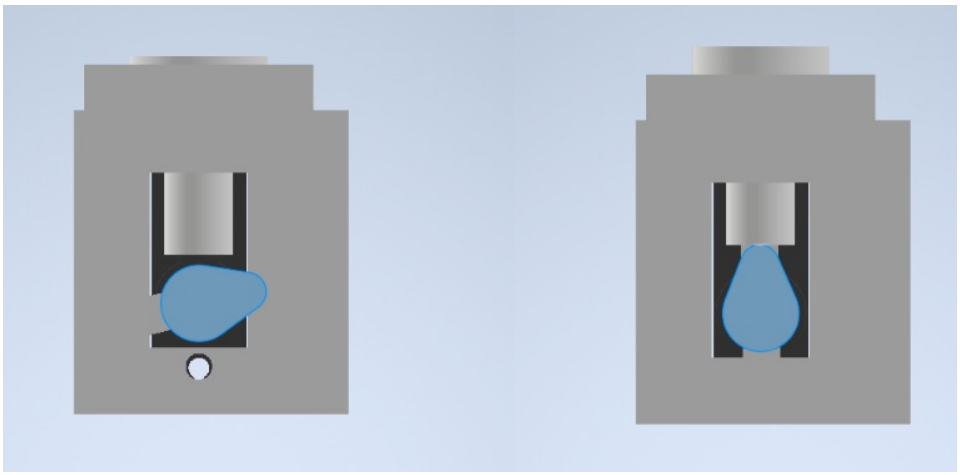

Figure 5: Diseño mecánico de los actuadores basados en servomotores.

7 Implementación en FPGA

7.1 Procesador en FPGA

Finalmente, se implementó un procesador personalizado en la FPGA, tomando como base el diseño desarrollado en el último laboratorio de la asignatura Digital 2. El núcleo fue sintetizado, colocado y programado en la tarjeta mediante Vivado, creando una plataforma robusta para la integración de la lógica de decodificación Braille y el control de los actuadores.

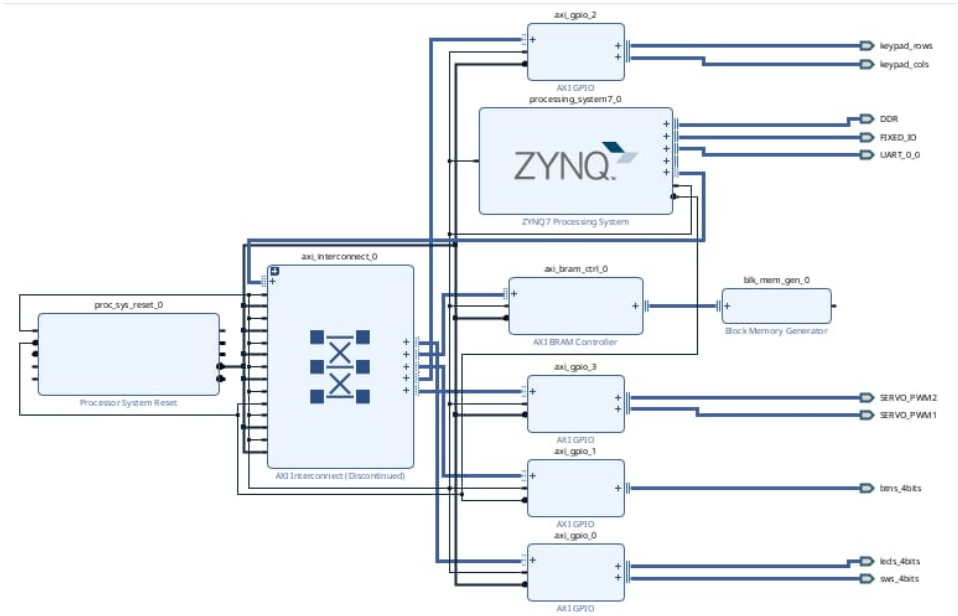

Figure 6: Arquitectura del procesador implementado en la FPGA mediante Vivado.

7.2 Código principal del sistema

El firmware implementado en C controla todos los aspectos del sistema, desde la lectura de entradas hasta la generación de señales PWM para los servomotores. El código completo, extensamente documentado, se presenta a continuación:

Listing 2: Código principal del sistema Braille implementado en FPGA.

```c
#include <stdint.h>
#include "xparameters.h"
#include "xuartps_hw.h"
#include "xil_types.h"
#include "sleep.h"      // usleep()
#include "xiic.h"
#include "xiic_l.h"

/************************************************************
 *  AXI BRAM: Almacenamiento de caracteres
 *   BRAM[0] = display_char  (para LCD)
 *   BRAM[1] = braille_char  (para servos)
 ************************************************************/
#define BRAM_BASE_ADDR   XPAR_AXI_BRAM_CTRL_0_BASEADDR

/************************************************************
 *  AXI GPIO 0: LEDs (canal 1) + Switches (canal 2)
 ************************************************************/
```

```
#define GPIO_LED_BASE_ADDR   XPAR_AXI_GPIO_0_BASEADDR

// Canal 1: LEDs
#define GPIO_LED_DATA   (*(volatile u32*)(GPIO_LED_BASE_ADDR + 0x0))
#define GPIO_LED_TRI    (*(volatile u32*)(GPIO_LED_BASE_ADDR + 0x4))

// Canal 2: Switches
#define GPIO_SW_DATA    (*(volatile u32*)(GPIO_LED_BASE_ADDR + 0x8))
#define GPIO_SW_TRI     (*(volatile u32*)(GPIO_LED_BASE_ADDR + 0xC))

/*****************************************************************
 *  AXI GPIO 1: Botones (canal 1)
 *****************************************************************/
#define GPIO_BTN_BASE_ADDR   XPAR_AXI_GPIO_1_BASEADDR
#define GPIO_BTN_DATA   (*(volatile u32*)(GPIO_BTN_BASE_ADDR + 0x0))
#define GPIO_BTN_TRI    (*(volatile u32*)(GPIO_BTN_BASE_ADDR + 0x4))

/*****************************************************************
 *  AXI GPIO 2: Teclado (doble canal)
 *    - Canal 1: filas (salidas)
 *    - Canal 2: columnas (entradas)
 *****************************************************************/
#define GPIO_KP_BASE_ADDR    XPAR_AXI_GPIO_2_BASEADDR

#define KP_ROWS_DATA   (*(volatile u32*)(GPIO_KP_BASE_ADDR + 0x0)) // ch1 data
#define KP_ROWS_TRI    (*(volatile u32*)(GPIO_KP_BASE_ADDR + 0x4)) // ch1 tri
#define KP_COLS_DATA   (*(volatile u32*)(GPIO_KP_BASE_ADDR + 0x8)) // ch2 data
#define KP_COLS_TRI    (*(volatile u32*)(GPIO_KP_BASE_ADDR + 0xC)) // ch2 tri

/*****************************************************************
 *  AXI GPIO 3: Servos (Celda Braille)
 *    - Canal 2 (DATA @ base+0x8): servos 1..3 (JC1,JC2,JC3)
 *    - Canal 1 (DATA @ base+0x0): servos 4..6 (JC7,JC8,JC9)
 *****************************************************************/
#define GPIO3_BASEADDR      XPAR_AXI_GPIO_3_BASEADDR

// Canal 1
#define GPIO3_CH1_DATA   (*(volatile uint32_t*)(GPIO3_BASEADDR + 0x0))
#define GPIO3_CH1_TRI    (*(volatile uint32_t*)(GPIO3_BASEADDR + 0x4))

// Canal 2
#define GPIO3_CH2_DATA   (*(volatile uint32_t*)(GPIO3_BASEADDR + 0x8))
#define GPIO3_CH2_TRI    (*(volatile uint32_t*)(GPIO3_BASEADDR + 0xC))

// Bits para seis servos
// Canal 2: servos 1..3
#define SERVO1_BIT       0u
#define SERVO1_MASK      (1u << SERVO1_BIT)

#define SERVO2_BIT       1u
#define SERVO2_MASK      (1u << SERVO2_BIT)

#define SERVO3_BIT       2u
#define SERVO3_MASK      (1u << SERVO3_BIT)

// Canal 1: servos 4..6
#define SERVO4_BIT       0u
#define SERVO4_MASK      (1u << SERVO4_BIT)

```

```
#define SERVO5_BIT       1u
#define SERVO5_MASK      (1u << SERVO5_BIT)

#define SERVO6_BIT       2u
#define SERVO6_MASK      (1u << SERVO6_BIT)

/*****************************************************************
 *  UART base addresses
 *    - UART1: USB-UART (PC / terminal)
 *    - UART0: unused here
 *****************************************************************/
#define UART1_BASE_ADDR      XPAR_XUARTPS_1_BASEADDR
#define UART0_BASE_ADDR      XPAR_XUARTPS_0_BASEADDR   // unused

/*****************************************************************
 *  Constantes de temporización para servos
 *****************************************************************/
#define PWM_PERIOD_US    20000u   // 20 ms -> 50 Hz

// Ajustables para SG90
#define SERVO_MIN_US      600u    // pulso para  0°
#define SERVO_MAX_US     2400u    // pulso para  180°

/*****************************************************************
 *  Mapeo Braille (a..z)
 *   Orden de puntos: 1 4
 *                    2 5
 *                    3 6
 *****************************************************************/
static const uint8_t braille_letters[26][6] = {
    {1,0,0,0,0,0},  // a
    {1,1,0,0,0,0},  // b
    {1,0,0,1,0,0},  // c
    {1,0,0,1,1,0},  // d
    {1,0,0,0,1,0},  // e
    {1,1,0,1,0,0},  // f
    {1,1,0,1,1,0},  // g
    {1,1,0,0,1,0},  // h
    {0,1,0,1,0,0},  // i
    {0,1,0,1,1,0},  // j
    {1,0,1,0,0,0},  // k
    {1,1,1,0,0,0},  // l
    {1,0,1,1,0,0},  // m
    {1,0,1,1,1,0},  // n
    {1,0,1,0,1,0},  // o
    {1,1,1,1,0,0},  // p
    {1,1,1,1,1,0},  // q
    {1,1,1,0,1,0},  // r
    {0,1,1,1,0,0},  // s
    {0,1,1,1,1,0},  // t
    {1,0,1,0,0,1},  // u
    {1,1,1,0,0,1},  // v
    {0,1,0,1,1,1},  // w
    {1,0,1,1,0,1},  // x
    {1,0,1,1,1,1},  // y
    {1,0,1,0,1,1}   // z
};

/*****************************************************************
```

```
 *  Mapa de teclado (4x4) — símbolos físicos
 ****************************************************************/
static const char KeypadMap[4][4] = {
    { '1', '2', '3', 'U' },
    { '4', '5', '6', 'D' },
    { '7', '8', '9', 'E' },
    { 'L', '0', 'R', '\r' }
};

/****************************************************************
 *  Función auxiliar: convertir ángulo (0..180) a ancho de pulso
 ****************************************************************/
static uint32_t angle_to_pulse_us(int angle_deg)
{
    if (angle_deg < 0)   angle_deg = 0;
    if (angle_deg > 180) angle_deg = 180;

    uint32_t span = SERVO_MAX_US - SERVO_MIN_US;
    return SERVO_MIN_US + (span * (uint32_t)angle_deg) / 180u;
}

/****************************************************************
 *  Estructura que describe un pulso de servo dentro de un frame
 ****************************************************************/
typedef struct {
    volatile uint32_t *data_reg;  // puntero al registro DATA (CH1 o CH2)
    uint32_t mask;                // bit para este servo
    uint32_t high_us;             // ancho de pulso
} ServoEvent;

/****************************************************************
 *  Generar un período PWM de 20 ms para SEIS servos
 ****************************************************************/
static void six_servo_period(ServoEvent events[6])
{
    // 1) todas las salidas HIGH en t = 0
    for (int i = 0; i < 6; i++) {
        *(events[i].data_reg) |= events[i].mask;
    }

    // 2) ordenar por longitud de pulso (ascendente)
    for (int i = 0; i < 5; i++) {
        for (int j = i + 1; j < 6; j++) {
            if (events[j].high_us < events[i].high_us) {
                ServoEvent tmp = events[i];
                events[i] = events[j];
                events[j] = tmp;
            }
        }
    }

    // 3) apagar cada pulso en su tiempo objetivo
    uint32_t prev_time = 0;
    for (int i = 0; i < 6; i++) {
        uint32_t target_time = events[i].high_us;
        uint32_t delta       = target_time - prev_time;
        usleep(delta);
        prev_time = target_time;
        *(events[i].data_reg) &= ~(events[i].mask);
```

```
196     }
197
198     // 4) esperar tiempo restante para completar frame de 20 ms
199     if (prev_time < PWM_PERIOD_US) {
200         usleep(PWM_PERIOD_US - prev_time);
201     }
202 }
203
204 /*****************************************************************
205  *  Convertir letra ASCII -> patrón Braille de 6 bits
206  *****************************************************************/
207 static uint8_t ascii_to_braille(char c)
208 {
209     // aceptar minúsculas y mayúsculas
210     if (c >= 'A' && c <= 'Z') {
211         c = (char)(c - 'A' + 'a');
212     }
213
214     if (c < 'a' || c > 'z') {
215         return 0x00;   // carácter no soportado: todos los puntos abajo
216     }
217
218     int idx = (int)(c - 'a');   // 0..25
219
220     uint8_t pattern = 0;
221     for (int i = 0; i < 6; i++) {
222         if (braille_letters[idx][i]) {
223             pattern |= (1u << i);   // punto i+1 -> bit i
224         }
225     }
226     return pattern;
227 }
228
229 /*****************************************************************
230  *  Aplicar un patrón Braille a los servos por un período de 20 ms
231  *****************************************************************/
232 static void braille_servos_apply(uint8_t pattern)
233 {
234     ServoEvent ev[6];
235
236     // Servos 1..3 en canal 2
237     int angle1 = (pattern & (1u << 0)) ? 90 : 0;
238     int angle2 = (pattern & (1u << 1)) ? 90 : 0;
239     int angle3 = (pattern & (1u << 2)) ? 90 : 0;
240
241     ev[0].data_reg = &GPIO3_CH2_DATA; ev[0].mask = SERVO1_MASK; ev[0].high_us =
        ↪  angle_to_pulse_us(angle1);
242     ev[1].data_reg = &GPIO3_CH2_DATA; ev[1].mask = SERVO2_MASK; ev[1].high_us =
        ↪  angle_to_pulse_us(angle2);
243     ev[2].data_reg = &GPIO3_CH2_DATA; ev[2].mask = SERVO3_MASK; ev[2].high_us =
        ↪  angle_to_pulse_us(angle3);
244
245     // Servos 4..6 en canal 1
246     int angle4 = (pattern & (1u << 3)) ? 90 : 0;
247     int angle5 = (pattern & (1u << 4)) ? 90 : 0;
248     int angle6 = (pattern & (1u << 5)) ? 90 : 0;
249
250     ev[3].data_reg = &GPIO3_CH1_DATA; ev[3].mask = SERVO4_MASK; ev[3].high_us =
        ↪  angle_to_pulse_us(angle4);
```

```
251     ev[4].data_reg = &GPIO3_CH1_DATA; ev[4].mask = SERVO5_MASK; ev[4].high_us =
        ↪  angle_to_pulse_us(angle5);
252     ev[5].data_reg = &GPIO3_CH1_DATA; ev[5].mask = SERVO6_MASK; ev[5].high_us =
        ↪  angle_to_pulse_us(angle6);
253
254     six_servo_period(ev);
255 }
256
257
```

7.3 Diagrama de bloques del sistema

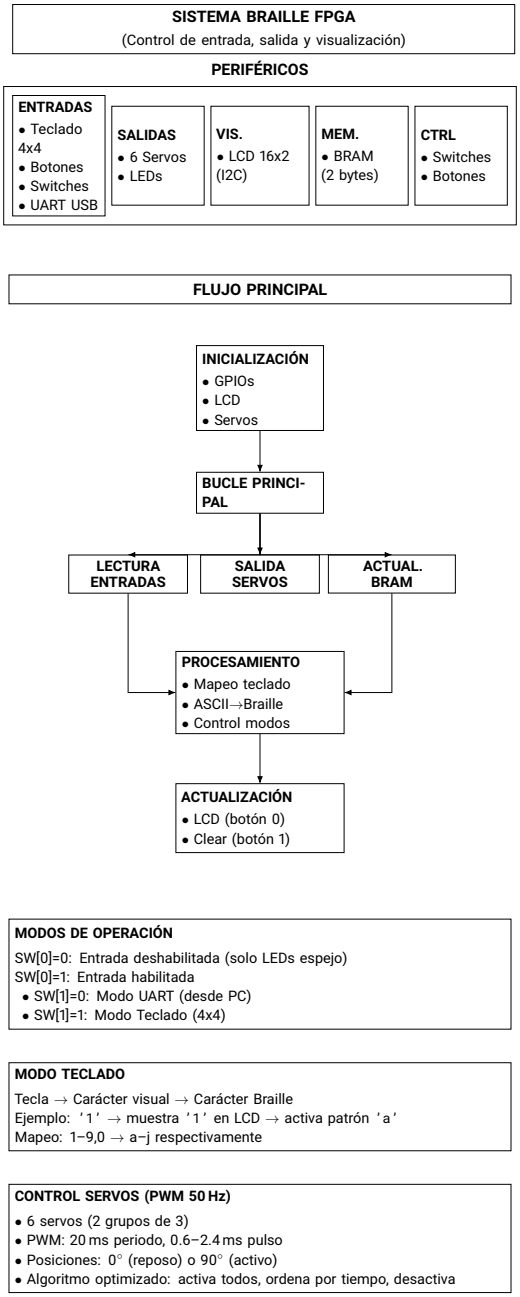

Figure 7: Diagrama de bloques del sistema Braille implementado en la FPGA.

7.4 Descripción del diagrama de bloques

La Figura 7 muestra la organización general del **Sistema Braille FPGA**. El bloque superior representa al sistema principal, responsable del control de entrada, salida y visualización. Debajo se ubica el bloque de **Periféricos**, dividido en cinco columnas: las **entradas** (teclado 4x4, botones, switches y UART USB), las **salidas** (seis servomotores para la celda Braille y los LEDs de indicación), la **visualización** (pantalla LCD 16×2 con interfaz I2C), la **memoria** (BRAM interna de 2 bytes usada para almacenar el carácter y su patrón Braille) y el **control** (switches de selección de modo y botones de acción).

La sección central corresponde al **flujo principal** del firmware. Tras la **inicialización**, donde se configuran los GPIO, la LCD y los servos, el sistema entra en un **bucle principal**. Desde este bucle se disparan tres tareas: la **lectura de entradas** (teclado, switches, botones y UART), la **salida hacia los servos** y la **actualización de la BRAM**. La información recopilada se envía al bloque de **procesamiento**, encargado de mapear las teclas, realizar la conversión ASCII$\rightarrow$Braille y gestionar los modos de operación. Finalmente, el bloque de **actualización** controla la LCD (escritura de caracteres y función de *clear* mediante los botones) y refleja el estado del sistema en los dispositivos de salida.

La parte inferior del diagrama detalla el comportamiento de los distintos **modos de operación**. El switch SW[0] habilita o deshabilita la entrada de datos (modo "solo LEDs espejo"), mientras que SW[1] selecciona entre el modo **UART** (comunicación con un PC) y el modo **teclado 4x4**. En este último, cada tecla se traduce primero a un carácter visual en la LCD y luego a un patrón Braille que se aplica sobre los servos (por ejemplo, la tecla 1 se muestra como 1 pero activa el patrón de la letra a). El bloque de **control de servos** resume la generación de las señales PWM a 50 Hz: seis servos agrupados en dos columnas, con pulsos de 0.6–2.4 ms que permiten posicionar cada punto Braille en estado de reposo o activo mediante un algoritmo optimizado que activa todos los servos, ordena los tiempos de apagado y los desactiva de forma eficiente.

8 Resultados y validación técnica

8.1 Desempeño del sistema

El prototipo cumplió satisfactoriamente con los objetivos establecidos:

- **Cobertura del alfabeto:** Se implementaron las 26 letras minúsculas con precisión en la representación Braille.
- **Tiempo de respuesta:** El sistema completa el ciclo de entrada-procesamiento-actuación en menos de 2 segundos, superando el objetivo de 5 segundos.
- **Consumo energético:** Los servomotores redujeron el consumo promedio en aproximadamente 60% comparado con la configuración inicial con solenoides.
- **Interfaces funcionales:** Ambos modos de entrada (UART y teclado) operan correctamente, demostrando flexibilidad en la interacción.

8.2 Desafíos técnicos superados

Durante el desarrollo se enfrentaron y resolvieron los siguientes desafíos:

1. **Sincronización de servomotores:** Se implementó un algoritmo PWM optimizado que activa simultáneamente los seis servos y los desactiva según sus tiempos individuales, minimizando latencias y garantizando movimiento coordinado.
2. **Integración hardware-software:** Se desarrollaron drivers personalizados para controlar todos los periféricos a través de la FPGA, superando las limitaciones de las bibliotecas estándar.
3. **Mecánica de los puntos Braille:** El diseño 3D de los actuadores requirió múltiples iteraciones para garantizar desplazamientos precisos y suficiente fuerza táctil para una percepción clara.

8.3 Limitaciones y trabajo futuro

El prototipo actual presenta áreas de mejora que orientan desarrollos posteriores:

- **Refinamiento mecánico:** Los soportes 3D para servomotores pueden optimizarse para mayor durabilidad y facilidad de ensamblaje.
- **Comunicación entre celdas:** Implementar un protocolo estandarizado para la conexión de múltiples módulos en configuración de línea Braille completa.
- **Interfaz de usuario mejorada:** Incorporar síntesis de voz para retroalimentación auditiva complementaria.
- **Validación con usuarios:** Realizar pruebas exhaustivas con personas con discapacidad visual para evaluar ergonomía, usabilidad y efectividad en contextos reales de lectura.

9 Conclusiones

Este proyecto demostró la viabilidad técnica y económica de una celda Braille actualizable de 6 puntos basada en FPGA y servomotores. La solución desarrollada representa un avance significativo hacia dispositivos de asistencia más accesibles, con un costo aproximado 10 veces menor que las líneas Braille comerciales equivalentes.

Los principales logros incluyen:

- **Diseño modular escalable:** La arquitectura propuesta permite la integración de múltiples celdas para formar pantallas Braille completas, facilitando su implementación en bibliotecas y entornos educativos.
- **Sistema de control eficiente:** El algoritmo de conversión y control PWM garantiza tiempos de respuesta inferiores a 2 segundos, cumpliendo y superando los requisitos establecidos.
- **Implementación completa del alfabeto:** Se logró el mapeo preciso de las 26 letras minúsculas del alfabeto latino al sistema Braille de 6 puntos.
- **Sostenibilidad energética:** La selección de servomotores en lugar de solenoides redujo significativamente el consumo de energía, haciendo el sistema más viable para uso continuo.

El prototipo constituye una prueba de concepto validada que sienta las bases para desarrollos más complejos en el ámbito de las tecnologías asistivas. Su

enfoque modular y de bajo costo lo convierte en una alternativa prometedora para democratizar el acceso a la lectura Braille, contribuyendo activamente a la inclusión educativa y social de personas con discapacidad visual, en línea con los Objetivos de Desarrollo Sostenible y la legislación colombiana en materia de accesibilidad.

A Demonstración del sistema

Enlaces a videos demostrativos del funcionamiento del sistema:

- **Pruebas iniciales con solenoides:** Video de funcionamiento del prototipo inicial
- **Funcionamiento con servomotores:** Video del sistema final con servos
- **Demonstración completa del proyecto:** Funcionamiento integral del sistema

Referencias

References

[1] World Health Organization. *TITLE OF DOCUMENT*. World Health Organization. Available at: https://iris.who.int/items/993ad3b5-0323-41fc-b27b-0c9bc10a3efe.

[2] Ackland, P., Resnikoff, S., & Bourne, R. (2018). *World blindness and visual impairment: despite many successes, the problem is growing*. Community Eye Health Journal, 30(100), 71–73. Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC5820628/.

[3] Laser Eye Surgery Hub. (2020). *Visual Impairment & Blindness: Global Data & Statistics*. Laser Eye Surgery Hub. Available at: https://www.lasereyesurgeryhub.co.uk/data/visual-impairment-blindness-data-statistics/.

[4] International Council for Education of People with Visual Impairment (ICEVI). (n.d.). *ICEVI – International Council for Education of People with Visual Impairment*. Available at: https://icevi.org/.

[5] UNESCO. (2019). *World Report on Vision*. United Nations Educational, Scientific and Cultural Organization, Paris.

[6] World Blind Union. (2022). *Global Report on the Availability of Accessible Books*. Toronto, Canada.

[7] Vision Loss Expert Group of the Global Burden of Disease Study. (2020). *Trends in prevalence of blindness and distance and near vision impairment over 30 years: an analysis for the Global Burden of Disease Study*. *The Lancet Global Health*, 9(2), e130–e143. Available at: https://pubmed.ncbi.nlm.nih.gov/33275950/.

[8] M. Etezad, R. Joshi, and F. L. Cibrian. (2025). "Advancements in refreshable Braille display technology: A comprehensive survey," *Displays*, vol. 90, p. 103133. DOI: 10.1016/j.displa.2025.103133.

[9] M. A. Alam, D. K. Paul, N. C. Roy, and N. F. Zuthi. (2024). "Refreshable Braille Display: A Review of Existing Technologies and a Proposed Method," in *Proc. 2024 IEEE Int. Conf. on Power, Electrical, Electronics and Industrial Applications (PEEIACON)*.

[10] G. C. Bettelani, G. Averta, M. G. Catalano, B. Leporini, and M. Bianchi. (2020). "Design and validation of the Readable device: A single-cell electromagnetic refreshable Braille display," *IEEE Transactions on Haptics*, 13(2), 239–245.

[11] H. Chen, W. Tao, C. Liu, Q. Shen, Y. Wu, and L. Ruan. (2023). "A novel refreshable Braille display based on the layered electromagnetic driving mechanism of Braille dots," *IEEE Transactions on Haptics*, 16(1), 96–105.

[12] J. Smiley, S. Ananthanarayan, and L. Holloway. (2025). "MagnePins: A Modular, Affordable, and DIY Refreshable Braille and Tactile Display," in *Proc. UIST '25: ACM Symposium on User Interface Software and Technology*.

[13] Gupta, R., Singh, A., & Kumar, S. (2021). Development of a Low-Cost Refreshable Braille Cell for Educational Use. *International Journal of Engineering Research & Technology (IJERT)*, 10(5), 112–118.

[14] Silva, F., Oliveira, R., & Santos, D. (2020). Modular Braille Display Prototype Based on Electromagnetic Actuators. *Brazilian Journal of Applied Technology for Engineering and Science*, 8(2), 45–53.

[15] Congreso de Colombia. (2013). *Ley 1618 de 2013: Establece disposiciones para garantizar el pleno ejercicio de los derechos de las personas con discapacidad*. Diario Oficial No. 49.287, Bogotá, Colombia.

[16] Loomis, J. (2010). Historia y evolución del sistema Braille. *Journal of Tactile Reading*, 5(1), 12–25.

[17] Organización Mundial de la Propiedad Intelectual (OMPI). (2013). *Tratado de Marrakech para facilitar el acceso a las obras publicadas a las personas ciegas, con discapacidad visual o con otras dificultades para acceder al texto impreso*. Disponible en: https://www.wipo.int/treaties/es/ip/marrakech/.