

# Proyecto final: Caneca clasificadora de residuos (TrashBot)

1<sup>st</sup>

David Enrique Barón Rubio  
dbaronr@unal.edu.co

2<sup>nd</sup>

Indaliria Valentina Cardona Corredor  
incardonac@unal.edu.co

3<sup>rd</sup>

Daniel Andrés Ramírez Morales  
daramirezmor@unal.edu.co

## I. IDENTIFICACIÓN DEL PROBLEMA A RESOLVER

La problemática se centra en la inadecuada separación de residuos en la fuente, ya que en la mayoría de hogares, instituciones y espacios públicos los desechos como papel, plástico, vidrio, metales y orgánicos se depositan en una misma papelera, lo que provoca que materiales reciclables se contaminen y pierdan su valor de reacondicionamiento. Esto incrementa la cantidad de residuos enviados a rellenos sanitarios, eleva los costos de clasificación manual y genera impactos ambientales negativos. Además, las tecnologías industriales para clasificación automática suelen ser costosas y poco accesibles para comunidades o entornos educativos, lo que resalta la necesidad de soluciones económicas e innovadoras que faciliten la correcta gestión de los residuos desde su origen.

## II. DATOS QUE SOPORTEN EL PROBLEMA

Colombia genera una gran cantidad de residuos sólidos urbanos (RSU) –unos 24,8 millones de toneladas al año según el DANE– con una tasa de reciclaje muy baja (solo 11,8 % en 2019)[1]. Esto refleja que más del 60 % del material potencialmente aprovechable no se separa adecuadamente en la fuente. De hecho, la clasificación promedio de residuos en los hogares es apenas del 39,9 % [1]. En la práctica, solo alrededor de la mitad de la población declara separar sus residuos domésticos [10]. La insuficiente separación conlleva a que la mayoría de la basura vaya a disposición final (rellenos o incineración) y a un elevado impacto ambiental (emisiones de gases invernadero, contaminación del agua, presión sobre vertederos) [1] [2]. En Bogotá, por ejemplo, solo el 54 % de los hogares separa sus residuos en casa [10], lo cual es congruente con los datos nacionales de bajo reciclaje. Aunque la normativa colombiana (Ley 1259/2008, Planes PGIRS, Conpes 3874/2016, Estrategia Nacional de Economía Circular, etc.) exige la separación en la fuente, su cumplimiento sigue siendo parcial [1] [4].

## III. ANÁLISIS PESTAL DEL PROBLEMA



Figura 1. Análisis PESTAL

### *III-A. Político*

En el ámbito político hay que tener en cuenta una serie de regulaciones, campañas y políticas nacionales enfocadas al tratamiento de residuos, su recolección y su reutilización según el caso. En general en el mundo y en particular en el país se han ido desarrollando políticas de concientización y promoción del reciclaje, la separación de los diferentes tipos de basura y el planteamiento del concepto de economía circular.

Algunos ejemplos de estas políticas pueden ser:

- Las guías, planes PGIRS y documentos CONPES, todo englobado en las ideas de economía circular que promueven entre otras cosas la separación de fuentes y la responsabilidad de los distintos actores, públicos y privados. [1]
- Existen normativas sobre la responsabilidad extendida del productor que incluyen regulaciones sobre el uso y tratamiento de empaques y plásticos, por ejemplo poniendo metas de reciclaje y trazabilidad. [15]
- Hay normativas recientes sobre el uso y disposición final de residuos específicamente de tipo orgánicos. [16]
- Hay políticas colombianas que relacionan con el reconocimiento de los recicladores de oficio como parte integral del reciclaje en el país. Las tecnologías nuevas deben tener en cuenta las posibilidades de inclusión y ayuda, y su impacto social [17].

### *III-B. Económico*

Es necesario tener en cuenta el impacto económico que puede tener el reciclaje a lo largo del país, sea por generar empleo, como por abaratar costos al reutilizar cierto material, así también el dinero que se utiliza para los tratamientos y disposiciones de los distintos tipos de residuos.

Un producto tecnológico puede ayudar a automatizar y facilitar procesos, cosa que sería útil, dado que en términos generales los niveles de reciclaje en el país son bajos (sobre el 16% y en su mayoría por recicladores informales) [18].

Hay que tener en cuenta el impacto que puede tener una tecnología que pudiese implementarse, dado que puede ayudar a generar empleo e incentivos, pero a la vez requiere un costo inicial para implementarse y superar regulaciones tanto del tema de economía circular, como regulaciones para uso público. [20]

Finalmente, en el mercado existe también interés por soluciones de digitalización (IoT, trazabilidad, telemetría) que permitan cumplimiento regulatorio y optimización logística; sin embargo, la adopción depende del costo unitario, facilidad de uso y capacidad de integración con sistemas municipales [19].

### *III-C. Social*

El éxito de cualquier tecnología depende de la cultura ciudadana. Encuestas en Bogotá revelan que aunque un 54 % asegura separar residuos, solo 39,9 % de los materiales en el hogar se clasifican efectivamente [1]. Las barreras citadas incluyen falta de educación ambiental, insuficiencia de recipientes de colores y escasa cooperación familiar [10]. Para cambiar esto, entidades locales realizan intensas campañas de sensibilización. La Secretaría de Ambiente de Bogotá informa haber desarrollado cerca de 1000 actividades de educación ambiental sobre separación de residuos entre 2020–2022, con más de 48.000 participantes [3]. Similarmente, programas posconsumo nacionales (para pilas, electrónicos, aceites usados, etc.) buscan involucrar a los ciudadanos y recicladores informales. Aun así, la corresponsabilidad social es baja: por ejemplo, solo el 21 % de quienes separan sacan los aprovechables el día del reciclador en su barrio [10]. Esto indica que más que tecnología, se requiere fortalecer la educación ambiental y cultura de reciclaje en todos los estratos.

### *III-D. Tecnológico*

La innovación tecnológica en la gestión de residuos sólidos en Colombia se centra en el desarrollo de papeleras inteligentes y sistemas automatizados que integran sensores avanzados, visión artificial y mecatrónica. Por ejemplo, proyectos universitarios colombianos han prototipado “canecas inteligentes” basadas en Arduino que usan cámaras y algoritmos de inteligencia artificial para identificar plásticos, vidrios u orgánicos y redirigirlos automáticamente a contenedores específicos [7]. Estos sistemas incorporan sensores ultrasónicos e infrarrojos para medir el nivel de llenado y diferenciar materiales, junto con actuadores mecatrónicos (motores, servos) que separan físicamente los residuos. A nivel internacional la tendencia es similar: empresas como AMP Robotics en EE.UU. utilizan visión por computador y robots articulados para detectar y extraer plásticos y metales de las cintas transportadoras de reciclaje [9]. Aunque en Colombia la mayoría de estas soluciones aún está en fase de prototipo o piloto académico, muestran un nivel tecnológico alineado con las prácticas globales de gestión inteligente de residuos (por ejemplo, contenedores IoT con compactadores solares) y cierran paulatinamente la brecha con estándares internacionales. En conjunto, el estado del arte nacional evidencia avances significativos en IA y mecatrónica aplicados al reciclaje, señalando un potencial creciente acorde con las tendencias mundiales [7] [9].

### *III-E. Ambiental y legal*

En Colombia existe un marco legal que obliga la separación en la fuente. La Ley 1259 de 2008 introdujo

el comparendo ambiental para infracciones ecológicas (incluyendo mal manejo de residuos) [4] [5]. En 2019 el Ministerio de Ambiente unificó a nivel nacional el código de colores para bolsas: blanco para aprovechables (plástico, papel, vidrio, metales), negro para no aprovechables y verde para orgánicos (véase figura 2)[4]. Este estándar –vinculado a los planes PGIRS municipales– facilita que los ciudadanos y prestadores de aseo utilicen la misma codificación en todo el país. Además, Colombia asumió los Objetivos de Desarrollo Sostenible (ODS) en 2015 y promueve en el Plan Nacional de Desarrollo la Economía Circular: reducir residuos, reciclar e “involucrar a todos los sectores” para producir conservando [1] [4]. A nivel ambiental, una clasificación más eficiente reduciría emisiones de gases de efecto invernadero; por ejemplo, solo los residuos de Bogotá aportan 18 % de las emisiones de la ciudad [11]. En la práctica, la supervisión gubernamental (MinAmbiente, IDEAM, Superservicios) incluye auditorías a rellenos sanitarios y reportes estadísticos; sin embargo, la implementación queda a cargo de gobiernos locales y operadores de aseo. Así, las tecnologías emergentes deben acoplarse a este marco: deben integrarse a planes PGIRS y respetar normativas (seguros contra incendios, manejo de contaminantes, accesibilidad) para ser viables en el mercado.



Figura 2. Código de colores unificado para la separación de residuos

#### IV. ANTECEDENTES

- Uno de los proyectos, realizado para el Centro Escolar Nicolás J. Bran en El Salvador [12], se centró en la creación de un contenedor para evitar la acumulación de basura y los focos de contaminación. Este sistema utilizaba una placa Arduino UNO conectada a un sensor de ultrasonido HC-SR04 para detectar el nivel de llenado. Cuando el contenedor estaba lleno, se encendía una luz LED roja como indicador y un servomotor bloqueaba la compuerta para impedir que se introdujera más basura. El código para el Arduino fue desarrollado en su propio entorno de programación.
- Otro proyecto, desarrollado en el Instituto Tecnológico de Ciudad Juárez [13], se enfocó en la clasificación y separación automática de residuos. Este

prototipo era capaz de diferenciar entre aluminio, plástico y vidrio. Para lograrlo, utilizaron un sensor inductivo para detectar metales como el aluminio, y un sistema de diferentes tipos de LEDs (Rojo, Verde, Blanco, Azul, Ultravioleta e Infrarrojo) junto con un sensor de luz para identificar plástico y vidrio basándose en la frecuencia de la luz percibida. Un motor se encargaba de girar un disco selector para depositar cada material en uno de los cuatro compartimentos internos. Todo el sistema fue programado utilizando el software LabVIEW.

- Finalmente, un proyecto más reciente implementó un sistema de clasificación automática utilizando inteligencia artificial y visión por computadora [14]. Este prototipo fue diseñado para clasificar cuatro categorías: papel, cartón, plástico y metal, logrando tasas de eficiencia de hasta el 100 % para el metal y 94 % para el plástico. El sistema integra técnicas de aprendizaje profundo (deep learning) y cuenta con una interfaz gráfica para el monitoreo en tiempo real del proceso de clasificación.

Ahora bien, haciendo comparaciones entre factores, tenemos que:

- Papeleras o contenedores “inteligentes”: Varios proyectos universitarios han diseñado canecas automáticas que identifican y clasifican el residuo ingresado. Por ejemplo, un prototipo de la UNAL Manizales/EAFIT usa RFID e inteligencia artificial para leer el chip del empaque y distinguir orgánico, recicitable o no recicitable [2]. Asimismo, la startup académica “Momentum” (Universidad EAN) desarrolló una caneca con Arduino y sensores que divide automáticamente el material en subcontenedores, buscando minimizar la contaminación y educar a los usuarios [7]. Estos sistemas aprovechan tecnologías de IoT, microcontroladores y visión para evitar errores humanos al desechar.
- Clasificación automática con visión computacional: Varios trabajos de ingeniería han usado visión artificial y aprendizaje profundo (deep learning) para clasificar residuos en tiempo real. Una tesis de la Pontificia Javeriana Cali (2023) entrenó redes neuronales para distinguir “aprovechables” vs. “no aprovechables” en imágenes, alcanzando 93 % de precisión [5]. De manera similar, un proyecto de Uninorte (2024) integró cámaras y algoritmos de IA para clasificar plásticos, cartón, papel y metal; obtuvo eficiencias de 64–100 % según la categoría [6]. Estos sistemas de visión automatizan el reconocimiento, reduciendo el trabajo manual de clasificación en plantas de reciclaje y permitiendo la separación en la fuente de forma más rigurosa.
- Sistemas mecatrónicos y sensores: También se exploran soluciones que combinan sensores físicos

y control automático. Por ejemplo, un artículo de Ingeniare (2022) describe un sistema de voz con Raspberry Pi que abre automáticamente la tapa del contenedor correcto según el tipo de residuo indicado por comando de voz, incentivando la práctica correcta [8]. Otros prototipos incorporan sensores de peso, ultrasonido o temperatura para gestionar volúmenes de basura o detectar olores, enviando alertas de llenado o descomposición. En general, las propuestas mecatrónicas buscan modularizar las canecas tradicionales con actuadores y electrónica de bajo costo para apoyar la separación sin requerir intervención manual constante.

- Comparativa de tecnologías: Los sistemas difieren en complejidad y aplicación. Las soluciones basadas en IA/visión [5] logran alta precisión de clasificación pero requieren buena iluminación y etiquetado de datos. Las canecas RFID [2] son muy precisas para envases con chip, pero dependen de la inclusión masiva de etiquetas electrónicas en los productos. Los enfoques sencillos como control por voz o por peso [8] son menos disruptivos tecnológicamente pero más limitados a entornos controlados. En general, los prototipos nacionales (universitarios) han demostrado viabilidad en laboratorio y sugieren aumentos del reciclaje domiciliario (15 % más, según estimaciones [2]). Sin embargo, aún faltan validaciones a escala real; pocos proyectos trascienden al piloto urbano. El reto técnico común es balancear costo, robustez y usabilidad local: p.ej., una caneca inteligente debe ser económica, fácil de usar y adaptarse al comportamiento ciudadano.

## V. POSIBLES SOLUCIONES AL PROBLEMA

Una alternativa viable a nivel doméstico es un clasificador de basura automatizado, que combine sensores electrónicos y control digital para identificar el tipo de desecho en el momento de depositarlo y dirigirlo de forma automática al contenedor adecuado. Esta propuesta reduce errores de clasificación, facilita el reciclaje y promueve la gestión responsable de los residuos desde la fuente.

## VI. OBJETIVOS

### VI-A. *Objetivo principal*

Diseñar, construir y validar una papelera inteligente capaz de clasificar automáticamente residuos sólidos domiciliarios en la fuente, mediante una solución que integre sensores, control electrónico y lógica de decisión, con el fin de aumentar la recuperación de materiales reciclables y contribuir a la educación ambiental en entornos educativos y comunitarios.

### VI-B. *Objetivos específicos*

- Llevar un registro de los residuos que ingresen en la papelera según su clasificación para mayor control en un hogar.
- Implementar una interfaz (pantalla) que muestre en tiempo real el tipo de material detectado y el nivel de llenado por compartimiento.

## VII. ACTORES DE LA SOLUCIÓN Y ROLES

- **Líder de proyecto:** Coordina el trabajo, define el cronograma, supervisa el cumplimiento de objetivos y es el enlace con el docente o jurado.
- **Diseñador de hardware:** Selecciona sensores y componentes electrónicos, diseña el circuito, elige la fuente de alimentación y realiza las pruebas de funcionamiento eléctrico.
- **Programador/Desarrollador de firmware:** Implementa el código para la lectura de sensores, la lógica de clasificación y el control de los actuadores (servomotores, pantalla, etc.)
- **Diseñador mecánico:** Diseña la estructura del contenedor, los compartimentos y el mecanismo de separación, ya sea en materiales reciclables, impresión 3D o láminas.
- **Integrador y tester:** Ensambla todos los módulos (hardware, software y mecánica), realiza pruebas de sistema, depura fallos y documenta los resultados.
- **Encargado de documentación y presentación:** Redacta informes, prepara las diapositivas y lidera la sustentación ante el jurado.
- **Diseñadores de productos:** Equipo de profesionales que se encargan del diseño estético del producto. Garantizan que el producto sea atractivo, intuitivo y fácil de usar.
- **Personas del común:** Son los usuarios finales que podrían hallar útil el sistema, sea para tenerlo en el hogar como para aplicarlo en el trabajo o inclusive en la calle.
- **Servicios de mantenimiento y soporte técnico:** Si el sistema llega a presentar fallas, un equipo de soporte técnico que pueda detectarlas, corregirlas y arreglarlas se vuelve vital para el producto.
- **Tiendas de tecnología:** El producto final podría ser comercializado, sería ideal encontrarlos en tiendas especializadas en tecnología o electrodomésticos.
- **Autoridades correspondientes como el ministerio de ambiente:** El producto podría requerir cumplir ciertas regulaciones para su uso y comercialización. Además, si quisiese utilizarse en espacios públicos como una política para ayudar a reciclar y cuidar más el ambiente también se requeriría las legislaciones y contratos pertinentes.

## VIII. PRESUPUESTO

- Motor paso a paso con controlador (28BYJ-48 + ULN2003):  
COP 18,000 – COP 22,000
- Servo motor SG90:  
COP 12,000 – COP 15,000
- Filamento PLA (1 kg, 1.75 mm):  
COP 55,000 – COP 65,000
- Sensor inductivo (LJ12A3):  
COP 13,000 – COP 18,000
- Sensor de color (TCS3200/TCS230 con LEDs):  
COP 45,000 – COP 55,000
- Display LCD 16x2:  
COP 12,000 – COP 15,000
- Módulo I2C para LCD (PCF8574):  
COP 5,000 – COP 7,000
- FPGA Zybo Z7:  
COP 778,490

### Suma total (sin contar la FPGA):

Total mínimo estimado: COP 160,000

Total máximo estimado: COP 197,000

### VIII-A. Materiales y consumibles

- Estaño para soldadura (rollo pequeño):  
COP 8,000 – COP 12,000
- Silicona en barra (varias unidades):  
COP 5,000 – COP 8,000
- Cables jumper (macho-macho, macho-hembra, hembra-hembra):  
COP 10,000 – COP 15,000

### Costo estimado de consumibles:

Total mínimo: COP 23,000

Total máximo: COP 35,000

## IX. DESARROLLO FINAL DEL PROYECTO

### IX-A. Diseño de las partes

Se trabajó primeramente en el diseño de la separadora de basura Trashbot, es decir, en las partes que corresponden al recipiente y estructura que conforman el sistema.

Toda esta etapa de diseño se hizo en el programa Onshape, un programa de diseño asistido por computadora (CAD) y gestión de datos de producto (PDM) completamente en la nube. [21]

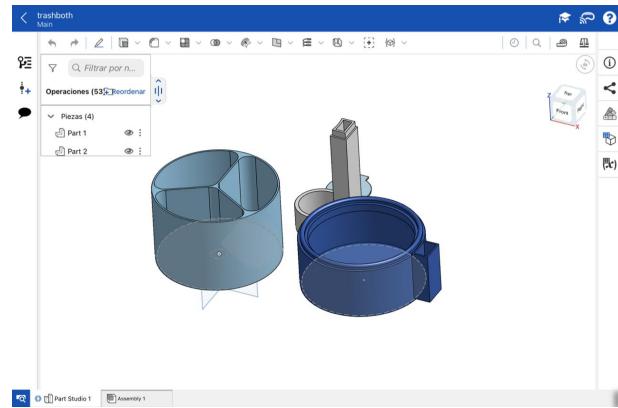


Figura 3. Entorno de diseño OnShape

La parte principal corresponde a un contenedor con tres subdivisiones. Para la respectiva separación de los residuos. Este contenedor trae consigo el espacio justo para encajar un piñón en su parte inferior, y así hacerlo rotar.

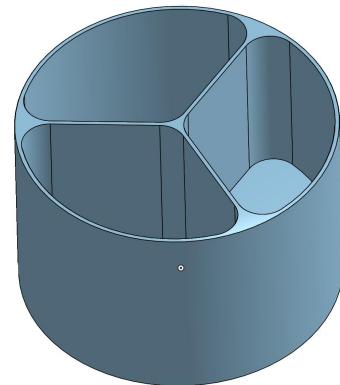


Figura 4. Contenedor principal

Esta es la imagen tanto del eje como del piñón, que se usa debajo del contenedor:



Figura 5. Eje de rotación para piñón



Figura 7. Base para la FPGA y conexiones

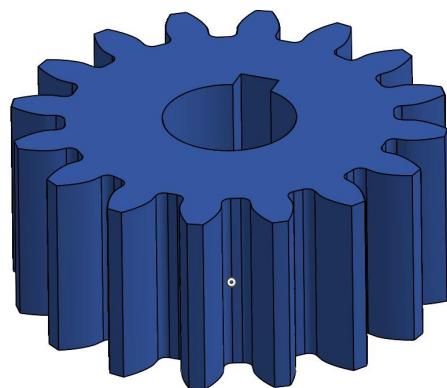


Figura 6. Piñón

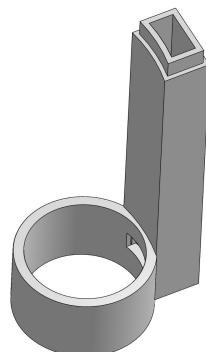


Figura 8. Base y contenedor para depositar residuos

Por otro lado, se diseñó otro contenedor específicamente para guardar la FPGA junto con todas sus conexiones.

Ese espacio rectangular adicional se ideó con el fin de encajar otra pieza, y así, por medio de estas dos, guardar o esconder los cables que se utilicen a lo largo de la caneca, esto para facilitar la organización entre conexiones, y a su vez hacer más estético el producto.

Adicional a esto se diseñó una pieza para colocar sobre la FPGA en el contenedor inferior que permitiera mantener el motor fijo y así ayudar al correcto movimiento del contenedor principal.

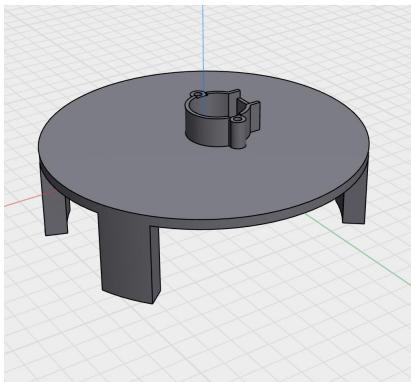
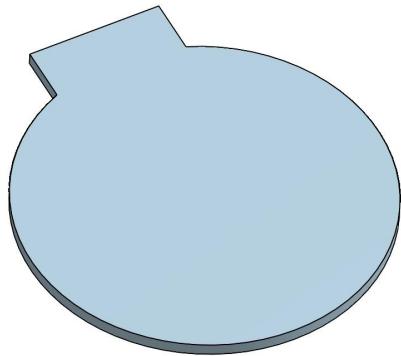


Figura 9. Pieza para mantener fijo el motor



Figura 11. Impresión contenedor principal, eje y piñón.

Finalmente, se diseñó una tapa simple para permitir el paso del resudio a la caneca.



Las demás partes se muestran a continuación:

Figura 10. Tapa para paso o cierre.

#### *IX-B. Impresión en 3D*

A continuación se evidencia el resultado de la impresión de las piezas por impresión 3D:



Figura 12. base con contenedor inferior



Figura 13. Tapa impresión 3D



Figura 15. Modelo final montado del proyecto final (Vista de frente)



Figura 14. Modelo completo impreso

### *IX-C. Montaje final*

A continuación se evidencia el modelo ya implementado del proyecto Trashbot. Este contiene la FPGA junto con todas sus conexiones en la parte inferior del modelo.



Figura 16. Modelo final montado del proyecto final (Vista desde arriba)



Figura 17. Modelo final montado del proyecto final (Separación de los residuos)

Cabe hacer la aclaración de que para la correcta integración de los sensores y la tarjeta hubo que hacer ciertas modificaciones a las piezas originales, mediante el uso de elementos como cautines, bisturíes, silicona, lija, entre otros.

Esto con el fin de poder insertar cables de alimentación para la FPGA, o mantener los sensores fijos en su posición a la vez que se les colocaba de tal forma que pudiesen medir correctamente los parámetros correspondientes.

#### IX-D. Diseño de la lógica

El sistema opera bajo una lógica secuencial basada en la lectura de sensores y la toma de decisiones según el tipo de residuo detectado. Para ello, se definió un orden de prioridad entre los sensores, de manera que cada residuo sea clasificado y depositado. De la siguiente forma

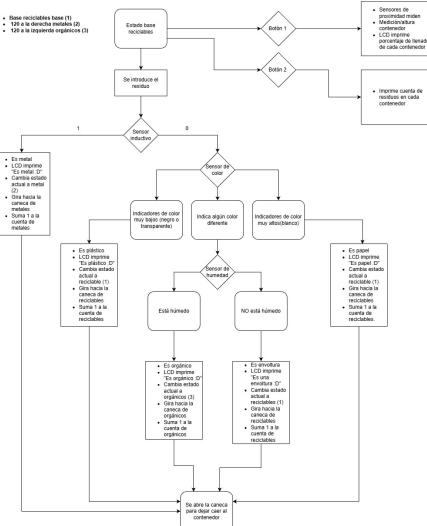


Figura 18. Diagrama de bloques del proyecto

Durante la implementación del prototipo final fue necesario realizar algunos ajustes derivados de limitaciones físicas, como el espacio disponible para ubicar los sensores, el color del material impreso en 3D y la dificultad de calibración de ciertos dispositivos. No obstante, el funcionamiento general del sistema y la jerarquía de detección establecida se mantuvieron sin cambios.

#### IX-E. Programación en Vivado y Vitis

**IX-E1. Diagrama de bloques en Vivado:** Para la implementación del sistema en la FPGA se utilizó Vivado con el fin de diseñar el diagrama de bloques del SoC basado en la tarjeta Zynq Z7. En este diagrama se configuró el procesador, los periféricos y los módulos de comunicación necesarios para el funcionamiento del proyecto.

El diagrama de bloques final se muestra a continuación:

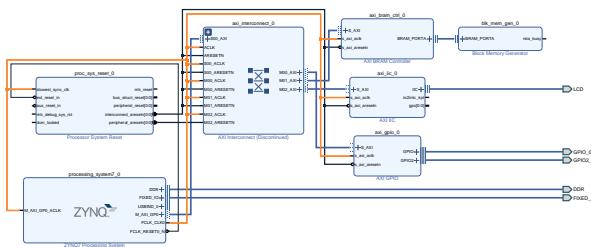


Figura 19. Diagrama de bloques generado en Vivado

**IX-E2. Descripción de los bloques del sistema:** A continuación se describe el funcionamiento de cada uno de los bloques que componen el diagrama de bloques del sistema, dentro del SoC basado en Zynq-7000.

#### ZYNQ7 Processing System (processing\_system7\_0)

**Función:** Este bloque constituye el núcleo del sistema. Integra el procesador ARM, encargado de ejecutar el software del proyecto, gestionar la lógica de control, procesar la información proveniente de los sensores y generar las señales de control hacia los actuadores. Permite implementar la capa de alto nivel del sistema embebido.

#### DDR

**Función:** Corresponde a la memoria RAM externa del sistema. Se utiliza para almacenar el programa, variables, datos temporales y estructuras requeridas durante la ejecución del software.

#### FIXED\_IO

**Función:** Agrupa las conexiones físicas dedicadas del Zynq, tales como señales de configuración, reloj, alimentación y otros pines fijos de la placa.

#### Processor System Reset (proc\_sys\_reset\_0)

**Función:** Este bloque se encarga de generar y distribuir las señales de reinicio del sistema, garantizando que el procesador, el bus AXI y los periféricos arranquen de manera sincronizada y estable.

#### AXI Interconnect (axi\_interconnect\_0)

**Función:** Permite la comunicación entre el procesador y los distintos periféricos mediante el protocolo AXI. Se encarga del enrutamiento de datos, el manejo de direcciones y el arbitraje del bus.

#### AXI GPIO (axi\_gpio\_0)

**Función:** Este bloque permite la lectura y escritura de señales digitales desde el procesador. Se utiliza para la interacción con sensores, actuadores y señales de control externas del sistema.

#### GPIO\_0

**Función:** Representa un conjunto de pines de entrada y salida digitales destinados a la conexión de sensores o señales externas utilizadas durante la clasificación de residuos.

#### GPIO2\_0

**Función:** Corresponde a un segundo conjunto de pines GPIO, utilizado para separar o agrupar distintas funciones del sistema, facilitando la organización de las señales de entrada y salida.

#### AXI IIC (axi\_iic\_0)

**Función:** Implementa el protocolo de comunicación I<sup>2</sup>C en hardware, permitiendo la interacción con dispositivos externos compatibles, como la pantalla LCD utilizada en el proyecto.

#### LCD

**Función:** Representa la pantalla LCD 16x2 con módulo I<sup>2</sup>C, empleada para mostrar información del sistema, estados de operación y mensajes asociados al proceso de clasificación.

#### AXI BRAM Controller (axi\_bram\_ctrl\_0)

**Función:** Permite que el procesador acceda a la memoria BRAM interna usando el protocolo AXI, facilitando el almacenamiento y la lectura de datos de acceso rápido durante la ejecución del sistema.

#### Block Memory Generator (blk\_mem\_gen\_0)

**Función:** Implementa la memoria BRAM interna de la FPGA, utilizada para almacenar datos temporales, buffers o variables que requieren tiempos de acceso reducidos.

La mayoría de las conexiones hacia periféricos externos se realizó mediante los bloques AXI GPIO y AXI IIC. En cada caso se definió la cantidad de bits de entrada y salida necesarios, y posteriormente se asignaron los pines físicos correspondientes en el archivo Master.xdc. El control y la lógica de cada señal enviada o recibida se implementaron en el software desarrollado en Vitis.

#### IX-F. Programación en vitis

#### IX-G. Programación en Vitis

Una vez validado el diseño del hardware, se generó el bitstream y se exportó el archivo XSA, el cual define la configuración del sistema. Con este archivo se creó un entorno de trabajo en Vitis, donde se desarrolló la aplicación encargada de controlar el funcionamiento completo del proyecto.

A diferencia de la mayor parte del curso, la programación del sistema se realizó en lenguaje C, aprovechando el procesador embebido del Zynq. El código se estructuró con varios modulos, separando el control de sensores, actuadores y periféricos en distintos archivos fuente, mientras que el archivo principal coordina la lógica general del clasificador. A continuación cada uno de los archivos fuente:

*IX-G1. paso\_a\_paso.c:* Este archivo implementa el control del motor paso a paso encargado de rotar el contenedor principal del sistema. El movimiento del motor se realiza mediante la escritura de patrones de activación en los bits correspondientes del registro de salida del AXI GPIO, recorriendo una secuencia de fases que permite el giro controlado del motor.

Se implementaron funciones independientes para el giro en sentido horario y antihorario, lo que permite posicionar el contenedor en el compartimiento correspondiente según el tipo de residuo detectado. La velocidad de rotación se controla mediante retardos por software entre cada cambio de fase.

Al finalizar cada movimiento, las bobinas del motor son desactivadas, con el objetivo de evitar consumo innecesario de energía y posibles problemas de calentamiento.

```

1 #include <stdint.h>
2
3 #define GPIO_BASE      0x41200000U
4 #define OUT_DATA      (*volatile
5          uint32_t*) (GPIO_BASE + 0x8))
6
7 #define STEPPER_MASK   0x0Fu    // bits 0-3
8 #define STEPPER_DELAY_CYCLES 600000u
9
10 static void delay_cycles(volatile uint32_t
11                         cycles) {
12     while (cycles--) {
13 }
```

```

1         __asm__ ("nop");
2     }
3
4 // Secuencia de fases del motor (28BYJ-48
5 // tpico)
6 static const uint8_t step_seq[] = {
7     0b0001,
8     0b0010,
9     0b0100,
10    0b1000
11 };
12
13 #define SEQ_LEN (sizeof(step_seq) /
14           sizeof(step_seq[0]))
15
16 void motor_pasoapaso_derecha(uint32_t pasos)
17 {
18     for (uint32_t i = 0; i < pasos; i++) {
19         uint8_t pattern = step_seq[i % SEQ_LEN];
20
21         OUT_DATA = (OUT_DATA & ~STEPPER_MASK)
22             | (pattern & STEPPER_MASK);
23
24         delay_cycles(STEPPER_DELAY_CYCLES);
25     }
26     OUT_DATA &= ~STEPPER_MASK;
27
28
29 void motor_pasoapaso_izquierda(uint32_t pasos)
30 {
31     for (uint32_t i = 0; i < pasos; i++) {
32         uint32_t idx = i % SEQ_LEN;
33         uint32_t rev_idx = (SEQ_LEN - idx) %
34             SEQ_LEN;
35
36         uint8_t pattern = step_seq[rev_idx];
37
38         OUT_DATA = (OUT_DATA & ~STEPPER_MASK)
39             | (pattern & STEPPER_MASK);
40
41         delay_cycles(STEPPER_DELAY_CYCLES);
42     }
43     OUT_DATA &= ~STEPPER_MASK;
44 }
```

Listing 1. *paso\_a\_paso.c*

*IX-G2. servo.c:* Este archivo se encarga del control del servomotor utilizado en el mecanismo de apertura y cierre para la descarga del residuo. La señal de control se genera por software mediante pulsos digitales, simulando una señal PWM sobre un pin del AXI GPIO.

El código define un periodo fijo y varía el tiempo durante el cual la señal permanece en nivel alto, lo que permite posicionar el servo en los ángulos requeridos. Durante el proceso de clasificación, el servomotor realiza un movimiento de ida hasta la posición de apertura y posteriormente retorna a su posición inicial, asegurando el correcto depósito del residuo en el compartimiento correspondiente.

```

1 #include <stdint.h>
2
3 #define GPIO_BASE      0x41200000U
4 #define OUT_DATA      (*volatile
5          uint32_t*) (GPIO_BASE + 0x8))
6
7 static void delay_cycles(volatile uint32_t
8                         cycles) {
9     while (cycles--) {
10 }
```

```

5
6 #define SERVO_BIT      4u
7 #define SERVO_MASK     (1u << SERVO_BIT)
8
9 #define SERVO_PERIOD_TICKS    1000u
10 #define SERVO_PULSE_0_TICKS   20u
11 #define SERVO_PULSE_180_TICKS 120u
12 #define SERVO_TICK_DELAY_CYCLES 2000u
13 #define DELAY_MS_FACTOR    150000u
14
15 static void delay_cycles(volatile uint32_t
16   cycles) {
17   while (cycles--) {
18     __asm__("nop");
19   }
20 }
21 static void delay_ms(uint32_t ms) {
22   for (uint32_t i = 0; i < ms; i++) {
23     delay_cycles(DELAY_MS_FACTOR);
24   }
25 }
26
27 static void servo_run(uint8_t go_180, uint32_t
28   num_periods)
29 {
30   uint32_t period_ticks = SERVO_PERIOD_TICKS;
31   uint32_t pulse_ticks = go_180 ?
32     SERVO_PULSE_180_TICKS :
33     SERVO_PULSE_0_TICKS;
34
35   for (uint32_t p = 0; p < num_periods; p++)
36   {
37     for (uint32_t cnt = 0; cnt <
38       period_ticks; cnt++) {
39
40       if (cnt < pulse_ticks) {
41         OUT_DATA |= SERVO_MASK;
42       } else {
43         OUT_DATA &= ~SERVO_MASK;
44       }
45
46       delay_cycles(SERVO_TICK_DELAY_CYCLES);
47     }
48   }
49 }
50
51 void servo_180_ida_vuelta(void)
52 {
53   servo_run(1, 100u); // ir a 180
54   delay_ms(500u);
55   servo_run(0, 100u); // volver a 0
56 }
```

Listing 2. servo.c

*IX-G3. color.c:* Este módulo implementa la lectura del sensor de color TCS3200/TCS230, el cual permite estimar las componentes de color del objeto detectado. El sensor opera seleccionando distintos filtros de color mediante líneas de control digitales y generando una señal de salida cuya frecuencia depende de la intensidad del color percibido.

El código permite alternar entre los filtros correspondientes a los canales rojo, verde, azul y claro, y mide el número de pulsos generados por el sensor dentro de una ventana de tiempo definida. A partir de estas mediciones

se obtienen valores relativos de cada componente de color, los cuales son utilizados posteriormente en la lógica de decisión del sistema para identificar residuos reciclables, como plásticos claros.

La comunicación con el sensor se realiza a través del AXI GPIO, utilizando líneas de salida para seleccionar el filtro activo y una línea de entrada para contar los pulsos generados por el sensor.

```

1 #include <stdint.h>
2 #include "xparameters.h"
3
4 #define GPIO_BASE          0x41200000U
5 #define GPIO_CH1_DATA      (*volatile
6   uint32_t*)(GPIO_BASE + 0x0))
7 #define OUT_DATA           (*volatile
8   uint32_t*)(GPIO_BASE + 0x8))
9 #define OUT_TRI            (*volatile
10  uint32_t*)(GPIO_BASE + 0xC))
11
12 #define COLOR_BIT          2u
13 #define COLOR_MASK         (1u << COLOR_BIT)
14
15 #define S2_BIT              6u
16 #define S3_BIT              7u
17 #define S2_MASK             (1u << S2_BIT)
18 #define S3_MASK             (1u << S3_BIT)
19
20 #define DELAY_US_FACTOR    150u
21
22 static void delay_cycles(volatile uint32_t
23   cycles) {
24   while (cycles--) {
25     __asm__("nop");
26   }
27 }
28
29 static void delay_us(uint32_t us) {
30   for (uint32_t i = 0; i < us; i++) {
31     delay_cycles(DELAY_US_FACTOR);
32   }
33 }
34
35 static void delay_ms(uint32_t ms) {
36   for (uint32_t i = 0; i < (ms * 1000u);
37     i++) {
38     delay_cycles(DELAY_US_FACTOR);
39   }
40 }
41
42 static uint8_t color_read_level(void)
43 {
44   return (GPIO_CH1_DATA & COLOR_MASK) ? 1u : 0u;
45 }
46
47 typedef enum {
48   COLOR_FILTER_RED = 0,
49   COLOR_FILTER_BLUE,
50   COLOR_FILTER_CLEAR,
51   COLOR_FILTER_GREEN
52 } color_filter_t;
53
54 void color_init(void)
55 {
56   OUT_DATA &= ~S2_MASK;
57   OUT_DATA |= S3_MASK;
58 }
59
60 void color_set_filter(color_filter_t f)
```

```

56 {
57     switch (f) {
58         case COLOR_FILTER_RED:
59             OUT_DATA &= ~(S2_MASK | S3_MASK);
60             break;
61
62         case COLOR_FILTER_BLUE:
63             OUT_DATA |= S2_MASK;
64             OUT_DATA &= ~S3_MASK;
65             break;
66
67         case COLOR_FILTER_CLEAR:
68             OUT_DATA &= ~S2_MASK;
69             OUT_DATA |= S3_MASK;
70             break;
71
72         case COLOR_FILTER_GREEN:
73         default:
74             OUT_DATA |= (S2_MASK | S3_MASK);
75             break;
76     }
77 }
78
79 uint32_t color_measure_pulses(uint32_t ventana_ms)
80 {
81     uint32_t pulses = 0;
82     uint32_t total_us = ventana_ms * 1000u;
83
84     uint8_t prev = color_read_level();
85
86     while (total_us--) {
87         uint8_t now = color_read_level();
88         if (now && !prev) {
89             pulses++;
90         }
91         prev = now;
92         delay_us(1u);
93     }
94
95     return pulses;
96 }
97
98 int sensor_color_leer(void)
99 {
100     return color_read_level() ? 1 : 0;
101 }
102
103 void color_medir_CRGc(uint32_t *C, uint32_t *R, uint32_t *G, uint32_t *B)
104 {
105     color_set_filter(COLOR_FILTER_CLEAR);
106     delay_ms(2);
107     *C = color_measure_pulses(10);
108
109     color_set_filter(COLOR_FILTER_RED);
110     delay_ms(2);
111     *R = color_measure_pulses(10);
112
113     color_set_filter(COLOR_FILTER_GREEN);
114     delay_ms(2);
115     *G = color_measure_pulses(10);
116
117     color_set_filter(COLOR_FILTER_BLUE);
118     delay_ms(2);
119     *B = color_measure_pulses(10);
120
121     color_set_filter(COLOR_FILTER_CLEAR);
122 }

```

Listing 3. color.c

*IX-G4. humedad.c:* Este archivo implementa la lectura de una señal digital asociada a la detección de humedad en los residuos. Debido a la complejidad del manejo directo del sensor de humedad, se utilizó una placa Arduino Uno como etapa intermedia del sistema.

El Arduino se encarga de realizar la medición analógica del sensor, compararla con un umbral previamente definido y enviar a la FPGA una señal digital que indica si el residuo supera o no dicho umbral. La FPGA lee esta señal a través del AXI GPIO y la utiliza como criterio para clasificar el residuo como orgánico durante el proceso de decisión.

```

1 #include <stdint.h>
2 #define GPIO_BASE      0x41200000U
3 #define GPIO_CH1_DATA  (*(volatile
4                                uint32_t*)(GPIO_BASE + 0x0))
5
6 #define DHT_BIT    1u
7 #define DHT_MASK   (1u << DHT_BIT)
8
9 int sensor_humedad_digital(void)
10 {
11     return ( (GPIO_CH1_DATA & DHT_MASK) ? 1 :
12             0 );
13 }

```

Listing 4. humedad.c

*IX-G5. humedad\_arduino.ino:* Con el fin de simplificar la integración del sensor DHT22 con el SoC, se utilizó un Arduino como etapa auxiliar. En esta parte, el microcontrolador realiza la lectura de humedad y temperatura mediante la librería del DHT, valida que la medición sea correcta y, a partir de un umbral de humedad, genera una salida digital que posteriormente es leída por la Zybo Z7 como una entrada GPIO.

La salida digital se mantiene en nivel bajo por defecto y solo se activa cuando la humedad supera el umbral definido, lo que permite representar la detección de residuos orgánicos mediante una señal binaria. En caso de falla de lectura, el programa fuerza la salida a nivel bajo por seguridad y reporta el error por el monitor serial.

```

1 #include <DHT.h>
2
3 #define DHTPIN    2 // Pin donde
4           // conectaste DATA del DHT22
5 #define DHTTYPE   DHT22
6
7 #define HUM_OUT_PIN 3 // Pin que va a la
8           // Zybo (a traves del divisor)
9
10 const float HUM_THRESHOLD = 60.0; // % de
11           // humedad para disparar
12
13 DHT dht(DHTPIN, DHTTYPE);
14
15 void setup() {
16     Serial.begin(9600);
17     dht.begin();
18
19     pinMode(HUM_OUT_PIN, OUTPUT);
20     digitalWrite(HUM_OUT_PIN, LOW); // Por
21           // defecto, no hay organico
22 }

```

```

18 }
19
20 void loop() {
21     float h = dht.readHumidity();
22     float t = dht.readTemperature();
23
24     if (isnan(h) || isnan(t)) {
25         Serial.println("Error leyendo DHT");
26         digitalWrite(HUM_OUT_PIN, LOW); // Por
27         seguridad
28         delay(2000);
29         return;
30     }
31
32     Serial.print("Humedad: ");
33     Serial.print(h);
34     Serial.print(" % Temp: ");
35     Serial.println(t);
36
37     // Si supera el umbral, ponemos la salida en
38     1
39     if (h > HUM_THRESHOLD) {
40         digitalWrite(HUM_OUT_PIN, HIGH);
41     } else {
42         digitalWrite(HUM_OUT_PIN, LOW);
43     }
44
45     delay(2000); // Lee cada 2 s
46 }
```

Listing 5. humedad\_arduino.ino

*IX-G6. inductivo.c:* Este módulo permite la lectura del sensor inductivo utilizado para la detección de residuos metálicos. El sensor entrega una señal digital que indica la presencia de un objeto metálico dentro de su rango de detección.

La señal generada por el sensor es leída directamente desde el AXI GPIO y utilizada por el sistema como uno de los criterios principales de clasificación, asignando los residuos metálicos a su compartimiento correspondiente.

```

1 #include <stdint.h>
2
3 #define GPIO_BASE 0x41200000U
4 #define GPIO_CH1_DATA (*(volatile
5     uint32_t*) (GPIO_BASE + 0x0))
6
7 #define INDUCTIVE_BIT 0u
8 #define INDUCTIVE_MASK (1u << INDUCTIVE_BIT)
9
10 int sensor_inductivo_leer(void)
11 {
12     return (GPIO_CH1_DATA & INDUCTIVE_MASK) ?
13         1 : 0;
14 }
```

Listing 6. inductivo.c

*IX-G7. lcd.c:* Este archivo gestiona la comunicación con la pantalla LCD 16x2 mediante un adaptador I<sup>2</sup>C basado en el integrado PCF8574. El código inicializa el controlador I<sup>2</sup>C del sistema y envía los comandos necesarios para operar la pantalla en modo de 4 bits.

Se implementaron funciones para inicializar la pantalla, limpiar su contenido y escribir texto en cada una de sus líneas. La pantalla se utiliza para mostrar

información relevante al usuario, como el tipo de residuo detectado y el estado general del sistema durante su operación.

```

1 #include <stdint.h>
2 #include "xparameters.h"
3 #include "xiic.h"
4 #include "xil_printf.h"
5
6 int lcd_init(void);
7 void lcd_clear(void);
8 void lcd_print_line(uint8_t line, const char
9                     *s);
10
11 #define IIC_DEVICE_ID 0
12 #define LCD_ADDR_7B 0x27
13
14 #define LCD_RS (1u<<0)
15 #define LCD_RW (1u<<1)
16 #define LCD_EN (1u<<2)
17 #define LCD_BL (1u<<3)
18
19 static XIic Iic;
20
21 static void delay_cycles(volatile uint32_t c) {
22     while(c--) __asm__ ("nop");
23 }
24 static void delay_us(uint32_t us) { while(us--)
25     delay_cycles(150);
26 }
27 static void delay_ms(uint32_t ms) {
28     while(ms--) {
29         for(int i=0;i<1000;i++) delay_us(1);
30     }
31 }
32
33 static int pcf_write(uint8_t data)
34 {
35     int sent = XIic_Send(Iic.BaseAddress,
36                          LCD_ADDR_7B, &data, 1, XIIC_STOP);
37     return (sent == 1) ? 0 : -1;
38 }
39
40 static void lcd_pulse(uint8_t data)
41 {
42     pcf_write(data | LCD_EN);
43     delay_us(2);
44     pcf_write((uint8_t)(data &
45                 (uint8_t)~LCD_EN));
46     delay_us(50);
47 }
48
49 static void lcd_write4(uint8_t nibble, uint8_t rs)
50 {
51     uint8_t data = 0;
52     data |= LCD_BL;
53     if (rs) data |= LCD_RS;
54     data |= (uint8_t)((nibble & 0x0Fu) << 4);
55
56     pcf_write(data);
57     lcd_pulse(data);
58 }
59
60 static void lcd_send(uint8_t byte, uint8_t rs)
61 {
62     lcd_write4((uint8_t)(byte >> 4), rs);
63     lcd_write4((uint8_t)(byte & 0x0F), rs);
64 }
65
66 static void lcd_cmd(uint8_t cmd)
67 {
```

```

62     lcd_send(cmd, 0);
63     if (cmd == 0x01 || cmd == 0x02)
64         delay_ms(2);
65
66 static void lcd_data(uint8_t d)
67 {
68     lcd_send(d, 1);
69 }
70
71 int lcd_init(void)
72 {
73     XIic_Config *cfg =
74         XIic_LookupConfig(IIC_DEVICE_ID);
75     if (!cfg) {
76         xil_printf("ERROR:
77             XIic_LookupConfig(%d) failed\r\n",
78             IIC_DEVICE_ID);
79         return -1;
80     }
81
82     int st = XIic_CfgInitialize(&Iic, cfg,
83         cfg->BaseAddress);
84     if (st != XST_SUCCESS) {
85         xil_printf("ERROR: XIic_CfgInitialize
86             failed\r\n");
87         return -1;
88     }
89
90     XIic_Start(&Iic);
91     delay_ms(50);
92
93     lcd_write4(0x03, 0); delay_ms(5);
94     lcd_write4(0x03, 0); delay_us(150);
95     lcd_write4(0x03, 0); delay_us(150);
96     lcd_write4(0x02, 0); delay_us(150);
97
98     lcd_cmd(0x28);
99     lcd_cmd(0x0C);
100    lcd_cmd(0x06);
101    lcd_cmd(0x01);
102
103    return 0;
104 }
105
106 static void lcd_set_cursor(uint8_t row,
107     uint8_t col)
108 {
109     uint8_t addr = (row == 0) ? 0x00 : 0x40;
110     addr += col;
111     lcd_cmd((uint8_t)(0x80 | addr));
112
113 void lcd_print_line(uint8_t line, const char
114     *s)
115 {
116     uint8_t row = (line <= 1) ? 0 : 1;
117     lcd_set_cursor(row, 0);
118
119     for (int i = 0; i < 16; i++) {
120         char c = s[i];
121         if (c == '\0') {
122             for (; i < 16; i++) lcd_data(' ');
123             break;
124         }
125         lcd_data((uint8_t)c);
126     }
127 }

```

```

125     }
126 }

```

Listing 7. lcd.c

*IX-G8. main.c:* El archivo principal coordina el funcionamiento general del sistema. En él se configuran los puertos de entrada y salida del AXI GPIO, se inicializan los sensores y la pantalla LCD, y se establece el ciclo continuo de operación del clasificador.

Durante cada iteración del ciclo principal se realiza la lectura de los sensores, se detectan eventos mediante flancos de subida y se ejecuta la acción de clasificación correspondiente. Dependiendo del tipo de residuo identificado, el sistema acciona el motor paso a paso y el servomotor, y actualiza la información mostrada en la pantalla LCD.

#### Resumen de funcionamiento por secciones:

- *Mapeo de registros GPIO:* Se acceden registros del AXI GPIO por memoria mapeada (base 0x41200000) para configurar dirección y escribir salidas.
- *Delays:* Se implementan retardos por bucles de `nop` para temporización básica de actuadores y muestreo.
- *Integración modular:* Declara prototipos de funciones que están en otros .c (inductivo, humedad, color, stepper, servo, LCD).
- *Clasificación por color:* Define `es_color_plastico()` con rangos sobre C, R, G, B para detectar “plástico blanco” y evitar falsos positivos exigiendo que B sea mayor que R y G por un margen.
- *Detección de eventos:* Usa “flanco de subida” (`rising_...`) para que la acción ocurra solo cuando aparece un residuo nuevo y no se repita mientras el sensor siga activo.
- *Acciones:*
  - Metal → mover 120° (derecha), accionar servo, volver 120° (izquierda).
  - Orgánico → mover 120° (izquierda), accionar servo, volver 120° (derecha).
  - Plástico → accionar servo

```

1 #include <stdint.h>
2 #include "xparameters.h"
3 #include "xil_printf.h"
4
5 // -----
6 // Direcciones GPIO (AXI GPIO en
7 // 0x41200000)
8 // -----
9 #define GPIO_BASE      0x41200000U
10 // Canal 1: entradas (inductivo, humedad
11 // digital, color OUT, infrarrojo si lo
12 // usas)

```

```

11 //define GPIO_CH1_TRI    (*volatile
12     uint32_t*) (GPIO_BASE + 0x4))
13 // Canal 2: salidas (motor, servo, LCD,
14     S2, S3, etc.)
14 #define OUT_TRI      (*volatile
15     uint32_t*) (GPIO_BASE + 0x0C))
16 // Bits de canal 1 (entradas)
17 #define INDUCTIVE_BIT 0u // 
18     GPIO_0_tri_i[0] -> inductivo (JC1 V15)
18 #define HUM_BIT      1u // 
19     GPIO_0_tri_i[1] -> humedad (JC2 W15)
19 #define COLOR_BIT    2u // 
20     GPIO_0_tri_i[2] -> OUT TCS3200 (JC3
21     T11)
20 // #define IR_BIT      3u // 
21     GPIO_0_tri_i[3] -> infrarrojo (JC?
22     U12) si lo vuelves a usar
22 //
23 // Delays locales
24 //
25 #define DELAY_MS_FACTOR 150000u
26
27 static void delay_cycles(volatile uint32_t
28     cycles) {
29     while (cycles--) {
30         __asm__ ("nop");
31     }
32
33 static void delay_ms(uint32_t ms) {
34     for (uint32_t i = 0; i < ms; i++) {
35         delay_cycles(DELAY_MS_FACTOR);
36     }
37 }
38
39 //
40 // PROTOTIPOS (otros .c)
41 //
42 //
43 // Sensores
44 int sensor_inductivo_leer(void); // 
45 int sensor_humedad_digital(void); // 
46
47 // COLOR (TCS3200) - color.c
48 void color_init(void);
49 void color_medir_CRGc(uint32_t *C,
50     uint32_t *R, uint32_t *G, uint32_t *B);
51
52 // Actuadores
53 void motor_pasoapaso_derecha(uint32_t
54     pasos); // paso_a_paso.c
53 void motor_pasoapaso_izquierda(uint32_t
54     pasos); // paso_a_paso.c
54 void servo_180_ida_vuelta(void);
55     // servo.c
56
57 // LCD (lcd.c autocontenido)
58 int lcd_init(void);
59 void lcd_clear(void);
59 void lcd_print_line(uint8_t line, const
60     char *s);
61
61 // Paso para 120
62 #define STEPS_PER_REV      2048u
63 #define DEG_PER_MOVE        120u
64 #define STEPS_FOR_120_DEG   ((STEPS_PER_REV
65     * DEG_PER_MOVE + 180u) / 360u)
66 //
67 // Clasificador: BLANCO (bolsa plastica)
68 //
69 static int es_color_plastico(uint32_t C,
70     uint32_t R,
71     uint32_t G,
72     uint32_t B)
73 {
74     if (C < 6u || C > 10u) return 0;
75     if (R < 5u || R > 10u) return 0;
76     if (G < 5u || G > 9u) return 0;
77     if (B < 18u || B > 26u) return 0;
78
79     if (B <= R + 8u) return 0;
80     if (B <= G + 8u) return 0;
81
82     return 1;
83 }
84
85 //
86 // MAIN
87 //
88 int main(void)
89 {
90     xil_printf("Sistema iniciado...\r\n");
91
92     GPIO_CH1_TRI |= ( (1u <<
93         INDUCTIVE_BIT) |
94             (1u << HUM_BIT)
95             |
96                 (1u << COLOR_BIT) );
97
98     OUT_TRI = 0x00000000u; // todo salida
99
100    color_init();
101
102    if (lcd_init() == 0) {
103        lcd_clear();
104        lcd_print_line(1, "TRASHBOT :D");
105        lcd_print_line(2, "INICIADO <3");
106    } else {
107        xil_printf("OJO: lcd_init()
108        fallo\r\n");
109    }
110
111    delay_ms(800);
112
113    uint8_t prev_ind =
114        (uint8_t)sensor_inductivo_leer();
115    uint8_t prev_hum =
116        (uint8_t)sensor_humedad_digital();
117    uint8_t prev_plast = 0u;
118
119    uint32_t dbg = 0;
120
121    while (1) {
122
123        int ind_now =
124            sensor_inductivo_leer();
125        int hum_now =
126            sensor_humedad_digital();

```

```

120     uint32_t C=0, R=0, G=0, B=0;
121     color_medir_CRGC(&C, &R, &G, &B);
122
123     int plast_now =
124     es_color_plastico(C, R, G, B);
125
126     uint8_t rising_ind    = (ind_now
127     && !prev_ind);
127     uint8_t rising_hum   = (hum_now
128     && !prev_hum);
129     uint8_t rising_plast = (plast_now
130     && !prev_plast);
131
132     if (rising_ind) {
133         xil_printf("EVENTO:
134 METAL\r\n");
135
136         lcd_clear();
137         lcd_print_line(1, "RESIDUO:");
138         lcd_print_line(2, "METAL");
139
140         motor_pasoapaso_derecha(STEPS_FOR_120_DEG);
141         delay_ms(500);
142         servo_180_ida_vuelta();
143
144         motor_pasoapaso_izquierda(STEPS_FOR_120_DEG);
145     }
146     else if (rising_hum) {
147         xil_printf("EVENTO:
148 ORGANICO\r\n");
149
150         lcd_clear();
151         lcd_print_line(1, "RESIDUO:");
152         lcd_print_line(2, "ORGANICO
153 ~");
154
155         motor_pasoapaso_izquierda(STEPS_FOR_120_DEG);
156         delay_ms(500);
157         servo_180_ida_vuelta();
158
159         motor_pasoapaso_derecha(STEPS_FOR_120_DEG);
160     }
161     else if (rising_plast) {
162         xil_printf("EVENTO:
163 PLASTICO(BLANCO) \r\n");
164         xil_printf("RGB -> C=%u R=%u
165 G=%u B=%u\r\n",
166         (unsigned)C, (unsigned)R, (unsigned)G, (unsigned)B);
167
168         lcd_clear();
169         lcd_print_line(1, "RESIDUO:");
170         lcd_print_line(2,
171 "RECICLABLE");
172
173         servo_180_ida_vuelta();
174     }
175
176     prev_ind    = (uint8_t)ind_now;
177     prev_hum   = (uint8_t)hum_now;
178     prev_plast = (uint8_t)plast_now;
179
180     dbg++;
181     if (dbg >= 5) {
182         dbg = 0;
183         xil_printf("DBG COLOR -> C=%u
184 R=%u G=%u B=%u plast=%d\r\n",
185

```

```
175     (unsigned)C, (unsigned)R, (unsigned)G, (unsigned)B,
176     plast_now);
177     delay_ms(200);
178 }
179
180 return 0;
181 }
```

### Listing 8. main.c

## *IX-H. Video del funcionamiento*

Para finalizar, se realizó y grabó una prueba de funcionamiento de nuestro clasificador de basura, al cual se puede acceder mediante cualquiera de los siguientes links:

- Video de funcionamiento (YouTube): [https://youtube.com/shorts/hHHT8F48\\_9s](https://youtube.com/shorts/hHHT8F48_9s)
- Archivo de evidencia (Google Drive): <https://drive.google.com/file/d/1NhY5ENlGxqjsos3I4fv3G7QFApT3IJ9X/view?usp=sharing>

## REFERENCIAS

- [1] DNP/MinAmbiente, “Guía Nacional para la adecuada separación de residuos sólidos”, DNP, 2022 (cifras DANE 2019).
- [2] Ministerio de Agricultura (Agrocadenas), “Contenedor inteligente mejoraría separación de residuos”, Agronet (Bogotá), 16 Sep 2019.
- [3] Secretaría Distrital de Ambiente (Bogotá), “Secretaría de Ambiente recuerda importancia de separar residuos”, Observatorio Ambiental Bogotá, 18 May 2021.
- [4] MinAmbiente, “Gobierno unifica el código de colores para la separación de residuos...”, Comunicado de prensa, 27 Dic. 2019.
- [5] J.E.Torres Tamayo y J.A.Rosero Mora, “Clasificación automática de residuos aprovechables y no aprovechables... Deep Learning”, Tesis Ing. Electrónica, Univ. Pont. Javeriana Cali, 2023.
- [6] V.I.Oliveros Corredor y M.J.Meléndez Villadiego, “Sistema de clasificación automática de residuos sólidos basado en visión por computadora”, Proyecto Ing. Electrónica, Univ. del Norte, 2024.
- [7] N.I. Zamudio Muñoz, “Diseño prototípico de caneca de basura inteligente capaz de clasificar los residuos de forma adecuada automáticamente”, Tesis de Ingeniería, Univ. EAN, Bogotá D.C., 2022.
- [8] Y.Palacios M. y J.A.Román F., “Sistema automático para un uso adecuado de los botes de basura controlados por voz”, Ingeniare, vol.33, pp.61–66, 2022.
- [9] Forbes Staff, “Así trabaja el robot de 300.000 dólares que automatiza el reciclado de basura”, Forbes Colombia, 13 Nov. 2020.
- [10] Bogotá: Encuesta Ambiental revela cuántos hogares separan los residuos | Bogota.go.co. Disponible en: <https://bogota.gov.co/mi-ciudad/cultura-recreacion-y-deporte/bogota-encuesta-ambiental-revela-cuantos-hogares-separan-los-residuos>
- [11] «El problema de los residuos», Greenpeace Colombia. Accedido: 16 de septiembre de 2025. [En línea]. Disponible en: <https://www.greenpeace.org/colombia/el-problema-de-los-residuos/>
- [12] C. O. Mazi Melara y W. A. Lopez Guardado, “Diseñar un basurero inteligente para la modernización y erradicación de la basura en Centro Escolar Nicolás J. Bran,” Trabajo de Graduación, Facultad de Informática y Ciencias Aplicadas, Universidad Tecnológica de El Salvador, San Salvador, 2018.
- [13] A. Valles Chávez, J. R. Alemán Cuellar, y R. Alcantar Olguin, Innovación de un contenedor de basura inteligente”, Innovación y Sustentabilidad Tecnológica, vol. 1.
- [14] V. I. Oliveros Corredor y M. J. Meléndez Villadiego, “Sistema de clasificación automática de residuos sólidos basado en visión por computadora,”[Proyecto de grado], Departamento de Ingeniería Electrónica, 2024. Disponible en: <http://hdl.handle.net/10584/12974>
- [15] Packaging School, “EPR for Packaging in South America,” Packaging School, 2022. Disponible en: <https://packagingschool.com/lessons/epr-for-packaging-in-south-america>
- [16] Alcaldía Mayor de Bogotá, “Decreto 594 de 2023 – Por el cual se adopta el Plan de Gestión Integral de Residuos Sólidos para Bogotá,” Normograma Alcaldía de Bogotá. Disponible en: <https://www.alcaldiabogota.gov.co/sisjur/normas/Normal.jsp?i=1-80888>
- [17] A. Ruiz-Restrepo, “Policy Environment and Informal Workers in Waste Management in Latin America,” WIEGO, 2019. Disponible en: [https://www.wiego.org/wp-content/uploads/2019/09/Ruiz-Restrepo\\_Policy\\_Environment\\_Informal\\_Workers.pdf](https://www.wiego.org/wp-content/uploads/2019/09/Ruiz-Restrepo_Policy_Environment_Informal_Workers.pdf)
- [18] E. L. R. S.A.S, «En Colombia se logra reciclar al año 16 de residuos gracias a recicladores de oficio», Diario La República. Accedido: 17 de septiembre de 2025. [En línea]. Disponible en: <https://www.larepublica.co/especiales/economia-circular-empresarial/en-colombia-se-logra-reciclar-16-de-los-residuos-4074012>
- [19] M. C. Sierra Puentes, E. M. Puerto-Rojas, S. N. Correa-Galindo, y J. A. Aristizábal Cuellar, «Using Community-Based Social Marketing to Promote Pro-Environmental Behavior in Municipal Solid Waste Management: Evidence from Norte de Santander, Colombia», Environments, vol. 12, n.º 8, p. 262, jul. 2025, doi: 10.3390/environments12080262.
- [20] GIZ, Economía Circular en Colombia — Oportunidades para sistemas de envases reutilizables y participación de las mujeres, Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ), 2024. [En línea]. Disponible: <https://www.giz.de/de/downloads/giz2024-es-colombia-circular-economy.pdf>
- [21] “Why Onshape?” Onshape | Product Development Platform. Accedido el 14 de noviembre de 2025. [En línea]. Disponible: <https://www.onshape.com/en/why-onshape>
- [22] H. S. L. y S. P. Harris, *Digital Design and Computer Architecture*, 2nd ed. San Francisco, CA: Morgan Kaufmann, 2013.
- [23] D. D. Gajski, *Principles of Digital Design*. Upper Saddle River, NJ: Prentice Hall, 1997.
- [24] Digilent, “Zybo Z7 Reference Manual,” Digilent Inc., 2023. [En línea]. Disponible en: <https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual>. Accedido: 15 oct. 2025.
- [25] AMD (Xilinx), “Vivado Design Suite User and Reference Guides.” 2024. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug949-vivado-design-methodology/Vivado-Design-Suite-User-and-Reference-Guides>. Accedido: 15 oct. 2025.
- [26] AMD (Xilinx), “Vitis Unified Software Platform Documentation,” 2024. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1400-vitis-embedded>. Accedido: 15 oct. 2025.
- [27] Digilent, “Zynq-7000 SoC Architecture Overview,” Digilent Inc., 2023. [En línea]. Disponible en: <https://digilent.com/reference/programmable-logic/zynq/start>. Accedido: 15 oct. 2025.
- [28] Arduino, “Arduino Documentation,” Arduino S.r.l., 2024. [En línea]. Disponible en: <https://docs.arduino.cc/>. Accedido: 15 oct. 2025.
- [29] Adafruit Industries, “DHT Sensor Library,” GitHub repository, 2024. [En línea]. Disponible en: <https://github.com/adafruit/DHT-sensor-library>. Accedido: 15 oct. 2025.
- [30] YouTube, “Lista de reproducción – Implementación del sistema,” 2025. [En línea]. Disponible en: <https://www.youtube.com/playlist?list=PL01D33s9LDzaNn2FQfO3uFua1hIZ8JVF4>.
- [31] Google Drive, “Repositorio de evidencias de implementación,” 2025. [En línea]. Disponible en: <https://drive.google.com/drive/folders/109AuJ9VsYDqh0NAeJvt85cxnVNlal5TP>.