UNIVERSIDAD
NACIONAL
DE COLOMBIA

Electrónica Digital II

Departamento de Ingeniería Eléctrica y Electrónica

Proyecto Final: Tercera Entrega

Estudiante: Tomás Esteban Cuartas Feliciano
Salomón Velasco Rueda

Docente: Diego Alexander Tibaduiza Burgos

Fecha: 12 de diciembre de 2025

Período Académico 2025-II

Índice general

Resumen

En este informe se presenta el avance del proyecto *Guía Adaptativa de Puertas para Reducir el Dwell Time en TransMilenio*, cuyo objetivo es diseñar e implementar en FPGA un sistema de señalización que recomiende en tiempo real la puerta con menor ocupación en estaciones del sistema BRT de Bogotá. Con ello se busca reducir el tiempo de parada (*dwell time*) de los buses articulados, disminuir aglomeraciones en la plataforma y mejorar la velocidad comercial del corredor.

La presente entrega recopila los principales elementos de la formulación inicial del proyecto (definición del problema, contexto, análisis PESTAL, antecedentes y objetivos) y documenta los avances técnicos alcanzados hasta la fecha. En particular, se describen la arquitectura general del prototipo de mesa, la selección de sensores y actuadores, la revisión de hojas de datos y la verificación funcional de los componentes utilizando una plataforma Arduino como entorno de pruebas.

A partir de los resultados preliminares se decidió sustituir los sensores infrarrojos inicialmente propuestos por sensores ultrasónicos de distancia para la detección de ocupación por puerta, con el fin de reducir la dependencia frente a la iluminación y el color de la vestimenta de los usuarios. Aunque aún no se dispone de mediciones cuantitativas de reducción de *dwell time*, se deja establecida la base electrónica necesaria para la implementación posterior en FPGA y para la evaluación de indicadores de desempeño en el prototipo.

TransMilenio, tiempo de parada, dwell time, FPGA, sensores ultrasónicos, transporte masivo, señalización adaptativa.

1 Marco teórico

El sistema TransMilenio de Bogotá es un ejemplo de transporte masivo tipo Bus Rapid Transit (BRT), caracterizado por la operación de buses articulados y biarticulados en carriles exclusivos, con estaciones de embarque a nivel y múltiples puertas por vehículo. El desempeño del sistema depende en gran medida del tiempo de parada o *dwell time*, es decir, el intervalo en que el bus permanece detenido en la estación para permitir el intercambio de pasajeros.

Diversos estudios han mostrado que el *dwell time* crece con el número de pasajeros que suben y bajan y con la forma como se distribuyen a lo largo de las puertas de embarque. Cuando los usuarios se concentran en una o dos puertas (por lo general la central), se generan cuellos de botella que incrementan el tiempo de servicio por puerta, reducen la velocidad comercial y empeoran la experiencia de viaje.

En varias estaciones de la red troncal se observa empíricamente una mala distribución de pasajeros en plataforma: se forman filas largas frente a ciertas puertas, mientras otras permanecen subutilizadas. Esta situación sugiere la necesidad de mecanismos de guía que orienten al usuario hacia la puerta con menor ocupación.

1.1 Formulación del problema

El problema que aborda este proyecto puede resumirse así: la falta de información local y en tiempo real sobre la ocupación de cada puerta hace que los pasajeros no se distribuyan de manera eficiente en la estación. Sin una guía visible y confiable, la elección de puerta se basa en costumbre, imitación o en la posición actual del usuario, lo que tiende a concentrar la demanda en pocas puertas.

La consecuencia directa es un aumento del *dwell time* promedio por parada, mayores aglomeraciones en la plataforma y una utilización ineficiente de la infraestructura de puertas disponible. Reducir estos tiempos, incluso en algunos segundos por parada, puede traducirse en mejoras relevantes en capacidad efectiva y tiempos de viaje, sin necesidad de adicionar flota.

1.2 Análisis PESTAL

El análisis PESTAL realizado en la primera entrega se resume a continuación:

- **Político:** el proyecto se alinea con los objetivos de mejora del servicio de transporte público en Bogotá. No implica modificaciones tarifarias ni obras civiles, sino un prototipo de bajo costo orientado a la gestión de estaciones.
- **Económico:** una reducción del *dwell time* contribuye a incrementar la capacidad efectiva por hora y a disminuir el tiempo de viaje sin adquirir buses adicionales. Se trata de una estrategia de mejora operacional con inversión moderada.
- **Social:** una mejor distribución de pasajeros reduce aglomeraciones en la plataforma, mejora la percepción de seguridad y confort, y puede disminuir situaciones de conflicto por acceso a las puertas.
- **Tecnológico:** el proyecto explora el uso de una FPGA para implementar lógica de control determinista, lectura en paralelo de sensores y señalización mediante LEDs. Se aprovechan buenas prácticas de diseño BRT en la interfaz estación–vehículo.
- **Ambiental:** menores tiempos de parada implican menos tiempo de los buses al ralentí en la estación, con potencial reducción de emisiones por ciclo de operación.
- **Legal:** el prototipo de mesa no captura imágenes ni datos personales. Un eventual piloto en campo requeriría autorizaciones de TransMilenio S. A. y el cumplimiento de normas de seguridad en estaciones, pero el concepto es no intrusivo y anónimo.

1.3 Antecedentes y soluciones relacionadas

TransMilenio es una referencia internacional en sistemas BRT de alta demanda, y su desempeño ha sido estudiado en términos de tiempos entre estaciones y tiempos de parada. Los trabajos previos muestran que el *dwell time* depende tanto de los volúmenes de pasajeros que suben y bajan como de la organización del embarque.

En otras experiencias BRT se han utilizado estrategias como señalización en puertas, guías en piso, información dinámica y organización de filas para acelerar el intercambio de pasajeros. Estas soluciones refuerzan la idea de que pequeñas intervenciones en la interfaz estación–vehículo pueden generar mejoras apreciables en el servicio.

El prototipo propuesto se enmarca en esta línea, pero con la particularidad de utilizar una FPGA y sensores discretos (sin cámaras) para implementar un sistema de guía adaptativa que recomiende la puerta con menor ocupación.

1.4 Objetivos del proyecto

El **objetivo general** del proyecto es:

> Diseñar e implementar en FPGA un señalizador adaptativo de puertas que reduzca el *dwell time* estimado frente a un escenario base (sin guía) en un prototipo de mesa.

A partir de este objetivo se plantean los siguientes objetivos específicos:

- Desarrollar al menos dos modos de operación para la guía de puertas: uno basado en ocupación instantánea y otro basado en conteo de pasajeros.
- Implementar en FPGA la lógica digital necesaria (máquinas de estados, contadores, comparadores, temporizadores, antirrebote e histéresis) para procesar las señales de los sensores y actualizar la recomendación por puerta.
- Medir indicadores de desempeño en el prototipo, tales como el *dwell time* estimado por llegada de bus, el balance de uso entre puertas y el porcentaje de ciclos con sobrecarga.

2 *Diseños*

2.1 *Arquitectura general del sistema*

El sistema implementado corresponde a un esquema de control y señalización para dos puertas (A y B) de un bus, basado en mediciones de proximidad mediante sensores ultrasónicos. La plataforma de cómputo usada fue una Zybo Z7, con un diseño en Vivado donde el procesador (PS) ejecuta la aplicación en Vitis y se comunica con la lógica programable (PL) a través de un periférico de propósito general (AXI GPIO).

La implementación final priorizó el correcto funcionamiento de:

- Sensor de proximidad del bus (habilitación del sistema).
- Sensores ultrasónicos asociados a cada puerta (A y B).
- Dos buzzers (uno por puerta).
- Dos indicadores LED RGB (flechas), usando únicamente canales rojo y verde para obtener los colores: verde, rojo y amarillo (rojo+verde).

Inicialmente se planeó incluir tres sensores de presión (velostats) por puerta como entradas adicionales, pero debido a dificultades de integración física y/o de restricciones de pines, se decidió simplificar el comportamiento final para cumplir con un flujo funcional verificable antes de la entrega.

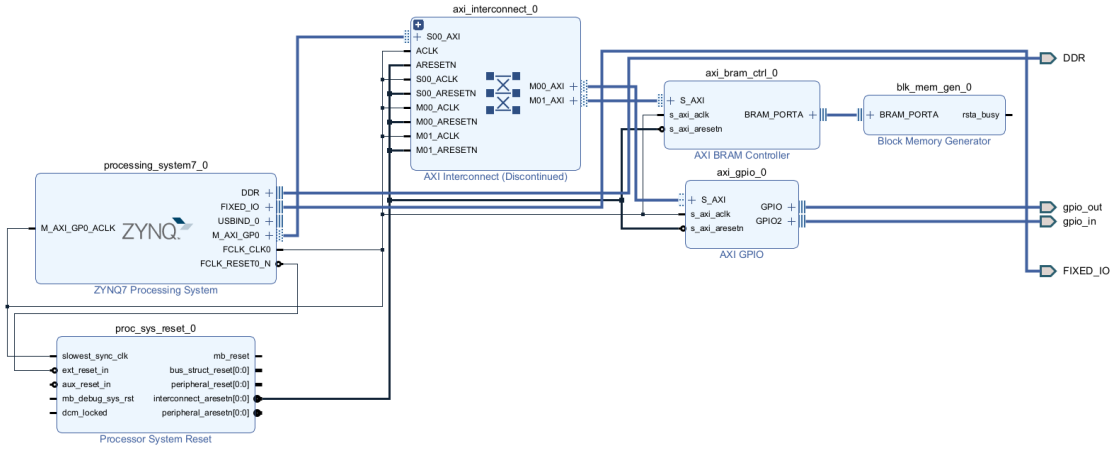

Figura 1: Block Design en vivado

2.2 *Diseño de hardware*

2.2.1 *Sensores ultrasónicos y adecuación de niveles*

Se emplearon sensores ultrasónicos alimentados a 5 V. Dado que los pines de la FPGA operan a 3.3 V, la señal *ECHO* de cada sensor se acondicionó mediante un divisor resistivo de tensión para reducir su nivel lógico y evitar sobrevoltaje en las entradas de la Zybo Z7.

Adicionalmente, para la alimentación a 5 V de los sensores se utilizó una fuente externa basada en un Arduino, actuando como fuente de 5 V para los módulos ultrasónicos. La FPGA generó los pulsos *TRIG* (lógica a 3.3 V) y recibió los *ECHO* ya reducidos a 3.3 V.

2.2.2 *Actuadores: buzzers y LEDs*

El sistema incluye dos buzzers (A y B) controlados desde salidas digitales. La señalización visual se realizó mediante dos LEDs RGB (uno por puerta) en forma de flecha, usando:

- Verde: indicación de redirección hacia la otra puerta.
- Amarillo (R+G): advertencia de ocupación media en la puerta opuesta.
- Rojo: indicación de ocupación alta (puerta opuesta llena).

2.3 *Diseño en Vivado (Block Design)*

A nivel de arquitectura hardware en Vivado, se integró un sistema mínimo con un AXI GPIO accesible por el procesador para:

- **Canal de salida**: control de buzzers, LEDs (R/G) y señales *TRIG*.
- **Canal de entrada**: lectura de señales *ECHO* provenientes de los sensores.

Figura 2: Diagrama de bloques (Block Design) implementado en Vivado para la comunicación PS–PL mediante AXI GPIO.

2.4 *Asignación lógica de señales*

La comunicación con el periférico AXI GPIO se realizó por registros mapeados en memoria, diferenciando canal de salida y canal de entrada. A nivel lógico, se definieron máscaras de bits para:

- Buzzers: puerta A y puerta B.
- LEDs: (A_R, A_G) y (B_R, B_G).
- TRIG: (A, B y BUS).
- ECHO: (A, B y BUS), leídas por el canal de entrada.

Esta separación permitió reutilizar posiciones de bits equivalentes para *TRIG* (salida) y *ECHO* (entrada) sin conflicto, al estar en canales distintos del GPIO.

3 *Desarrollos finales*

3.1 *Implementación del control en Vitis (aplicación en C)*

El desarrollo final se implementó en `main.c` mediante acceso directo a los registros del AXI GPIO por medio de la dirección base provista por `xparameters.h`. La aplicación sigue un ciclo infinito de:

1. Medición del sensor de proximidad del bus (habilita o deshabilita todo el sistema).
2. Medición de los sensores asociados a cada puerta (A y B).
3. Clasificación de ocupación por puerta en tres niveles: libre, medio y lleno.
4. Aplicación de reglas de decisión para activar buzzers y establecer el color de los LEDs.

3.1.1 *Medición ultrasónica (TRIG/ECHO)*

La medición de distancia se realizó generando un pulso *TRIG* y midiendo la duración del pulso *ECHO* en unidades de *ticks* (conteo por software). El procedimiento implementado fue:

- Forzar *TRIG* a 0 y esperar un retardo corto.
- Activar *TRIG* durante un intervalo fijo y volver a 0.
- Esperar a que *ECHO* suba (con *timeout*).
- Contar ciclos mientras *ECHO* permanezca alto (con límite máximo).

Si no se detecta un pulso válido (*timeout*), la lectura se considera cero.

3.1.2 *Habilitación por proximidad del bus*

El sensor asociado al bus se trató como un habilitador binario: si la lectura está por debajo de un umbral (`TICKS_UMBRAL_BUS`), el bus se considera "cerca" y el sistema se activa. En caso contrario, las salidas permanecen apagadas.

3.1.3 *Clasificación de ocupación por puerta*

Para cada puerta se definió una función de clasificación por rangos de *ticks*:

- **Nivel 0 (libre)**: sin eco o lectura mayor al umbral de "medio".
- **Nivel 1 (medio)**: lectura menor a `TICKS_PUERTA_MED`.
- **Nivel 2 (lleno)**: lectura menor a `TICKS_PUERTA_FULL`.

Estos umbrales se dejaron como constantes ajustables para calibración en campo, ya que dependen de la geometría del montaje y del rango real de medición.

3.1.4 *Reglas finales de señalización (buzzers y LEDs)*

Con los niveles `la` (puerta A) y `lb` (puerta B), se aplicaron las siguientes reglas:

- **Ambas libres (la=0, lb=0):** se activan ambos buzzers para atraer usuarios hacia ambas puertas.
- **Una puerta más libre que la otra:**
  - Se enciende en **verde** la flecha de la puerta que debe redirigir hacia la puerta más libre.
  - En la puerta más libre se muestra el estado de la otra puerta: **amarillo** si está en "medio" y **rojo** si está "llena".
  - El buzzer se activa únicamente en la puerta que esté clasificada como **libre**, para evitar atraer usuarios hacia una puerta congestionada.
- **Iguales:**
  - **Ambas medio (la=1, lb=1):** se muestra **amarillo** en ambos indicadores.
  - **Ambas llenas (la=2, lb=2):** se mantienen las salidas apagadas (no se redirige ni se atrae hacia ninguna).

3.2 *Validación experimental y evidencias*

La validación se realizó comprobando el encendido de buzzers y el cambio de color de los LEDs al variar la distancia medida por los sensores ultrasónicos, y verificando que el sensor del bus actuara como condición de habilitación global del sistema.

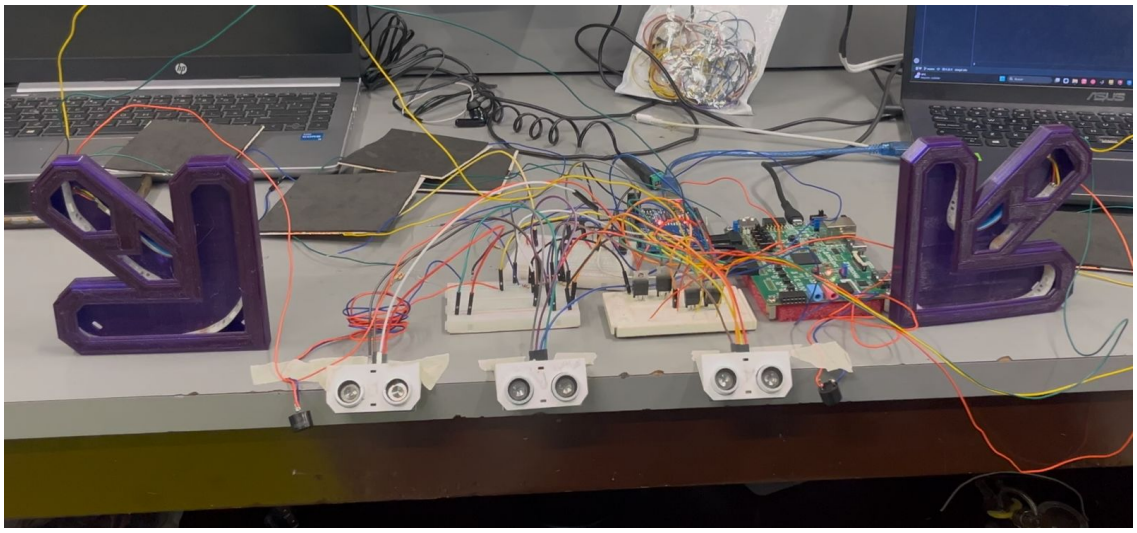

Figura 3: Montaje final del sistema: sensores ultrasónicos alimentados a 5 V con fuente externa, divisores de tensión para 3.3 V en ECHO, y conexión a Zybo Z7 para control de LEDs y buzzers.

3.3 *Limitaciones y trabajo no integrado*

Aunque el diseño original contemplaba tres velostats por puerta para estimar la cantidad de usuarios esperando, su integración completa no se logró para la entrega final debido a problemas durante la fase de conexión y/o definición de pines en el archivo de restricciones. Dado que la inclusión de seis entradas adicionales aumentaba significativamente los casos posibles, se optó por consolidar la lógica usando únicamente los tres sensores ultrasónicos (bus, puerta A y puerta B), garantizando un comportamiento estable y demostrable en la implementación final.

4 *Adjunto:Código de main.c*

```
#include "xparameters.h"
#include <stdint.h>


#define GPIO_BASE XPAR_AXI_GPIO_0_BASEADDR

#define GPIO_OUT_REG (*(volatile uint32_t *)(GPIO_BASE + 0x0))
#define GPIO_IN_REG (*(volatile uint32_t *)(GPIO_BASE + 0x8))


#define BUZ_A (1u << 0)
#define BUZ_B (1u << 1)
#define LED_A_R (1u << 2)
#define LED_A_G (1u << 3)
#define LED_B_R (1u << 4)
#define LED_B_G (1u << 5)
```

```
#define TRIG_A (1u << 6)
#define TRIG_B (1u << 7)
#define TRIG_BUS (1u << 8)


#define ECHO_A (1u << 6)
#define ECHO_B (1u << 7)
#define ECHO_BUS (1u << 8)


#define TICKS_UMBRAL_BUS 800u

#define TICKS_PUERTA_FULL 2000u
#define TICKS_PUERTA_MED 6000u


static void delay_cycles(volatile uint32_t n)
{

 while (n--) {
  __asm__("nop");
 }
}


static uint32_t ultrasonic_ticks(uint32_t trig_mask, uint32_t echo_mask)
{
 GPIO_OUT_REG &= ~trig_mask;
 delay_cycles(1000);

 GPIO_OUT_REG |= trig_mask;
 delay_cycles(2000);
 GPIO_OUT_REG &= ~trig_mask;

 uint32_t timeout = 200000;
 while (!(GPIO_IN_REG & echo_mask) && timeout--) {
  __asm__("nop");
 }

 if (timeout == 0) {
  return 0;
 }

 uint32_t ticks = 0;
 while ((GPIO_IN_REG & echo_mask) && (ticks < 200000)) {
```

```
  ticks++;
  __asm__("nop");
 }

 return ticks;

}


static int ultrasonic_near(uint32_t trig_mask, uint32_t echo_mask, uint32_t ticks_thresh
{
 uint32_t ticks = ultrasonic_ticks(trig_mask, echo_mask);

 if (ticks > 0 && ticks < ticks_threshold) return 1;

 return 0;
}


static int puerta_level(uint32_t ticks)
{
 if (ticks == 0) return 0;

 if (ticks < TICKS_PUERTA_FULL) return 2;
 if (ticks < TICKS_PUERTA_MED) return 1;

 return 0;
}


int main(void)
{
 GPIO_OUT_REG = 0;


 while (1) {

  int bus_near = ultrasonic_near(TRIG_BUS, ECHO_BUS, TICKS_UMBRAL_BUS);

  uint32_t out = 0;

  if (bus_near) {

   uint32_t ta = ultrasonic_ticks(TRIG_A, ECHO_A);
   uint32_t tb = ultrasonic_ticks(TRIG_B, ECHO_B);
```

```
  int la = puerta_level(ta);
  int lb = puerta_level(tb);


  if (la == 0 && lb == 0) {
   out |= BUZ_A | BUZ_B;
  }

  else if (la < lb) {
   if (la == 0) out |= BUZ_A;
   out |= LED_B_G;

   if (lb == 2) {
    out |= LED_A_R;
   } else if (lb == 1) {
    out |= (LED_A_R | LED_A_G);
   }
  }

  else if (lb < la) {
   if (lb == 0) out |= BUZ_B;
   out |= LED_A_G;

   if (la == 2) {
    out |= LED_B_R;
   } else if (la == 1) {
    out |= (LED_B_R | LED_B_G);
   }
  }

  else {
   if (la == 1 && lb == 1) {
    out |= (LED_A_R | LED_A_G | LED_B_R | LED_B_G);
   } else {

   }
  }
 }

 GPIO_OUT_REG = out;
 delay_cycles(100000);
}

return 0;
```

```
}
```

Referencias

[1] TransMilenio S. A., "Los más importantes logros del transporte público de Bogotá durante 2024," 2024. [En línea]. Disponible: https://www.transmilenio.gov.co/publicaciones/154412/los-mas-importantes-logros-del-transporte-publico-de-bogota-durante-2024/. Accedido: 14 nov. 2025.

[2] Alcaldía Mayor de Bogotá, "TransMilenio en Bogotá: conoce los datos e información del sistema." [En línea]. Disponible: https://bogota.gov.co/mi-ciudad/movilidad/transmilenio-en-bogota-conoce-los-datos-e-informacion-del-sistema. Accedido: 14 nov. 2025.

[3] H. M. Scorcia Tenjo, "Obtención de la función tiempo de parada para el sistema TransMilenio," Tesis de maestría, Univ. de los Andes, Bogotá, 2003.

[4] C. Sandoval Ardila, "Modelo estimativo del tiempo entre estaciones en el sistema TransMilenio," Tesis de maestría, Univ. de los Andes, Bogotá, 2017.

[5] Institute for Transportation and Development Policy (ITDP), *BRT Planning Guide*, 4th ed. [En línea]. Disponible: https://brtguide.itdp.org/. Accedido: 14 nov. 2025.