

# Desarrollo de robot detector de minas en un laberinto

Universidad Nacional de Colombia  
Departamento de ingeniería Eléctrica y Electrónica

Diego Alejandro Figueroa Del Castillo      Ferdy Eduardo Larrotta Ruiz  
dfigueroa@unal.edu.co      flarrotta@unal.edu.co

Edwin Jose Medina Caceres  
ejmedinac@unal.edu.co

4 de agosto de 2021

## Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Funcionalidades Básicas</b>	<b>2</b>
<b>3. Elementos utilizados para el proyecto</b>	<b>3</b>
<b>4. Diseño del SoC</b>	<b>3</b>
4.1. Partición HW/SW . . . . .	3
4.2. Disposición general del SoC . . . . .	4
4.3. Diseño hardware . . . . .	4
4.3.1. Organización control/datapath . . . . .	4
4.3.2. Diseño de cada módulo principal . . . . .	5
4.4. Diseño software . . . . .	11
<b>5. Implementación</b>	<b>12</b>
5.1. Hardware . . . . .	12
5.1.1. Memoria RAM . . . . .	13
5.1.2. Análisis del color . . . . .	13
5.1.3. Disposición final de los módulos de la FPGA . . . . .	14
5.2. Software . . . . .	15
5.2.1. Movimiento de carro . . . . .	15
5.3. Configuraciones iniciales, cámara y bluetooth . . . . .	16
5.3.1. Configuración de la cámara OV7670 mediante esp32 [4] . . . . .	16
5.3.2. Configuración de módulo bluetooth HC-05. [6] . . . . .	17
5.4. Recepción de datos en el computador . . . . .	19
5.5. Video . . . . .	19
5.6. Mapa . . . . .	19
<b>6. Integración hardware y software</b>	<b>20</b>
<b>7. Montaje Final</b>	<b>21</b>
7.1. Robot . . . . .	21
7.2. Conexión con el computador . . . . .	22
<b>8. Resultados / validación</b>	<b>22</b>

<b>9. Guía del usuario</b>	<b>25</b>
<b>10. GitHub</b>	<b>26</b>
<b>11. Conclusiones</b>	<b>26</b>

## 1. Introducción

El presente documento detalla de forma organizada las etapas que se siguieron para el diseño e implementación de un robot explorador, cuya función es desplazarse por un laberinto e identificar el color de las minas que va encontrando a su paso. En concreto se denotan las etapas de diseño digital requeridas, partiendo de la elaboración de un diseño funcional, pasando por la distribución de tareas en hardware y software, elaboración de diagramas ASM, hasta llegar a la síntesis en lenguaje de descripción de hardware. Todas estas etapas se desarrollaron teniendo siempre presentes las limitaciones del diseño y cada uno de sus componentes.

## 2. Funcionalidades Básicas

Se seleccionaron como entradas al sistema una cámara y un sensor ultrasonido con el fin de identificar de manera eficaz el entorno que rodea al robot, que en este caso será un laberinto. Como salida del sistema se tienen todas las funciones que derivan del análisis de los datos obtenidos con los periféricos de entrada los cuales se especifican en mayor detalle a continuación:

- **Reconocer Colores:** Se plantea que cada mina propuesta tenga un color determinado (Rojo, Verde y Azul) el cual debe ser plenamente identificado por el robot.
- **Generar Imágenes:** Las imágenes identificadas por la cámara se deben poder visualizar en una pantalla en tiempo real al ser transmitidas por Bluetooth.
- **Generar Mapa:** A partir de las señales de la cámara y del sensor ultrasonido el robot debe tener la capacidad de generar un mapa del laberinto, en donde se indique la trazada del robot, la posición de las minas y el color de estas. Esta información será transmitida de manera constante por medio de un módulo Bluetooth hacia un computador.
- **Mover Carro:** Una vez el robot identifique su entorno inicial debe ser capaz de determinar el camino que puede seguir, es decir, al saber a qué distancia está cada pared que lo rodea debe tener la capacidad de decidir hacia dónde hay más espacio para poder desplazarse por el laberinto ya sea girando o yendo en linea recta.

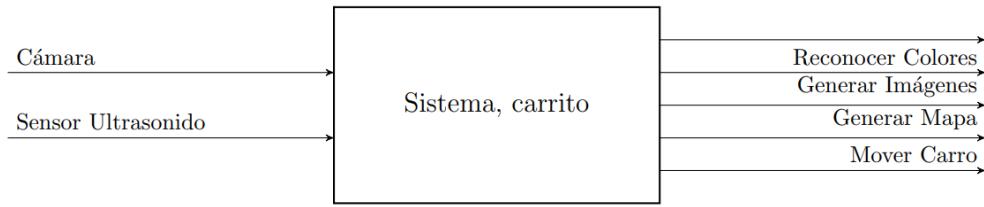


Figura N° 1: Diagrama de caja negra

### 3. Elementos utilizados para el proyecto

Para la implementación del proyecto se utilizaran los siguientes elementos:

- ESP 32
- Puente H L298N
- Módulo de cámara OV7670
- 2 módulos ultrasonido HC-SR04
- FPGA zybo Z7
- 2 Motores DC
- 2 baterías Lipo Turnigy 5 V, 1000mah
- 2 módulos bluetooth HC-05
- 1 adaptador USB para el módulo bluetooth

### 4. Diseño del SoC

#### 4.1. Partición HW/SW

Como siguiente paso en el proceso de desglosar el sistema, se procede a identificar con mayor rigurosidad que procesos harán parte del Hardware y del Software.

El Hardware estará implementado en la FPGA, en donde se llevarán a cabo las tareas de propósito específico del proyecto, en concreto se llevará a cabo la recepción de las imágenes provenientes de la cámara OV7670, y el posterior análisis de la imagen para detectar el color de las minas, esta información será enviada a la ESP 32. Además las imágenes captadas por la FPGA serán enviadas desde la tarjeta hasta el computador por medio del protocolo UART y con un módulo bluetooth.

En cuanto al software, este se implementará en un microcontrolador ESP 32, allí se llevarán a cabo las tareas de propósito general que resultan más sencillas de hacer en software y requieren poco poder computacional. En concreto se realizará el manejo de los sensores de ultra sonido, y a partir de los datos censados por ellos se tomará la decisión del movimiento del carro; por esta razón los motores también serán controlados por medio del software. Además en el microcontrolador se generará el mapa y se enviará al computador por medio de bluetooth. Por último la configuración de la cámara se realizará en una etapa de setup con el ESP 32 a través de una comunicación con el protocolo SCCB.

## 4.2. Disposición general del SoC

En términos generales el diseño del robot detector de minas se encuentra compuesto por la FPGA zybo Z7, la cual es la encargada de realizar el control de la cámara, procesar la imagen y enviar los datos al computador mediante bluetooth para su visualización, además de informar sobre las características de la imagen al ESP32 para la generación del mapa. El microcontrolador ESP32 es el encargado del movimiento y navegación del robot, este controla los motores y envía las coordenadas al computador para generar el mapa del recorrido junto a la información de las minas recibidas desde la FPGA. Finalmente el sistema está alimentado por un power bank, esto se realizó con el objetivo de alimentar adecuadamente los sistemas que demandan más potencia (motores y puente H) y con la finalidad de que el robot sea inalámbrico una vez terminado. El esquemático de las conexiones de cada elemento del proyecto se pueden observar en la figura 2.

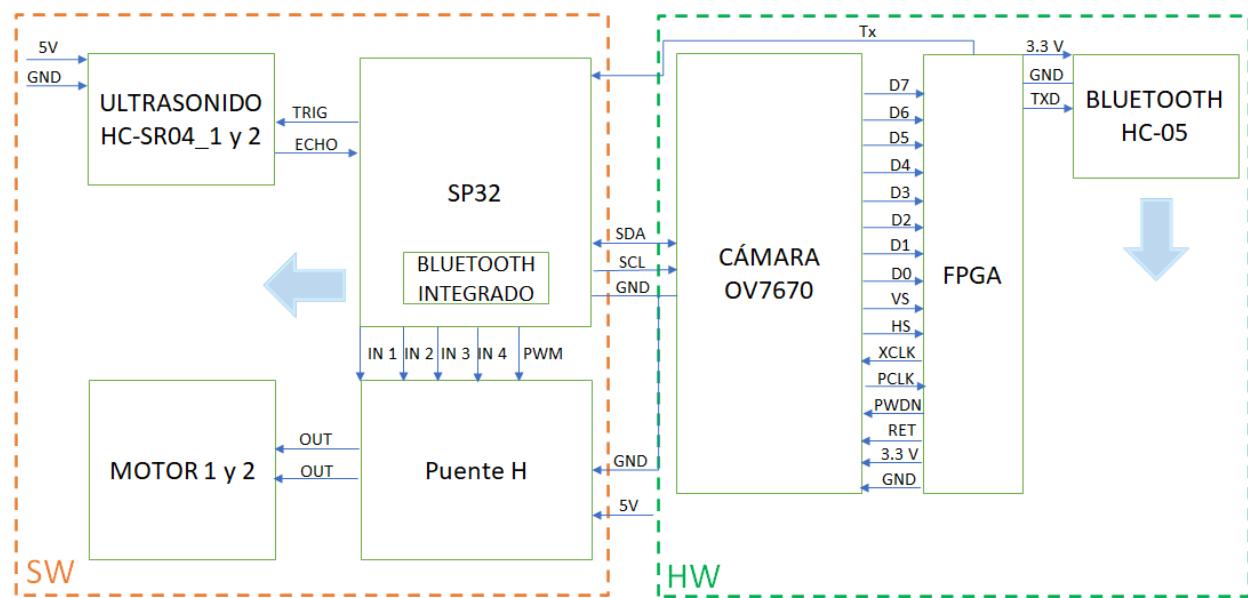


Figura N° 2: Diagrama de conexiones físicas y disposición del SoC

## 4.3. Diseño hardware

### 4.3.1. Organización control/datapath

Para el diseño del hardware se comienza por realizar una primera descripción muy general del hardware en donde se hace una separación del control y el datapath necesario, en donde cada componente del datapath corresponde a una de las grandes funcionalidades que debe tener el proyecto.

Para esto es importante notar primero que la cámara OV7670 genera imágenes constantemente, y en ese sentido el módulo encargado de recibir esa información debe trabajar permanentemente, por otra parte el envío de imágenes y el procesamiento de estas también debe ocurrir constantemente. Dadas estas condiciones se determina que en el hardware deben existir 2 procesos que corren en simultáneo, por un lado la recepción de imágenes, y por otra parte el envío y procesamiento de estas.

Además dado que la recepción de imágenes está gobernada por las señales de control que genera la cámara OV7670, se nota que este módulo no debe ser controlado por el controlador, sin embargo si debe generar ciertas señales para que el control pueda controlar correctamente el resto de componentes del datapath. Estas señales son una bandera de cuando termina/comienza a recibir una imagen, y una bandera que indique en qué parte de la memoria está guardado la imagen actual.

Finalmente el resto de componentes del datapath es controlado por el controlador y genera señales al finalizar el proceso correspondiente, con el fin de que el controlador pueda administrar correctamente el comienzo y el final de cada proceso.

Con esto en mente se genera un primer esquema de la organización del hardware, en donde se ve una separación entre el control (bloque azul), el datapath (bloques amarillos) y el módulo que recibe imágenes (bloque verde), que a pesar de ser un componente del datapath, no es controlado por el controlador y trabaja en paralelo al resto del datapath, por eso se separa un poco de los otros componentes. Además se puede ver un bloque de comunicación UART en rojo, el cual servirá para hacer la integración de hardware y software. El esquema se puede ver en la figura 3.

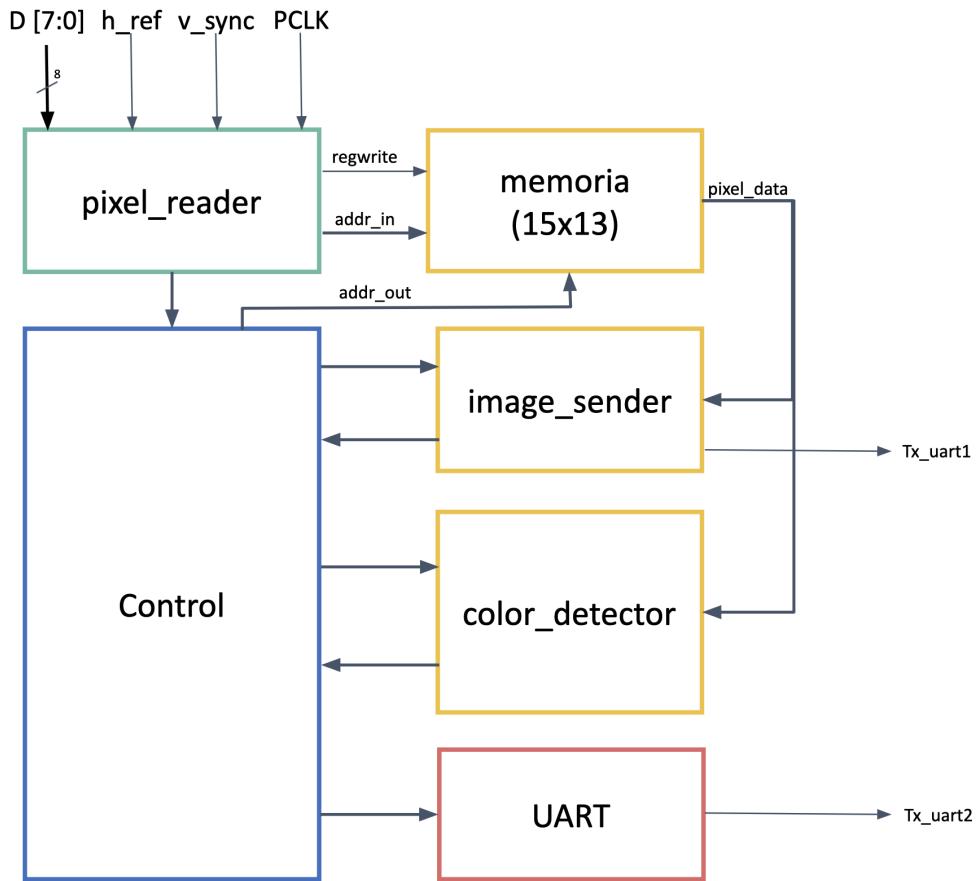


Figura N° 3: Disposición inicial del hardware, separación control y datapath

#### 4.3.2. Diseño de cada módulo principal

- **pixel\_reader:**

Para el diseño de este módulo se parte por comprender los diagramas de tiempo de la cámara, estos se pueden ver en la figura 4. De estos diagramas cabe recalcar que una señal positiva de v\_sync indica el comienzo de un nuevo frame, que la señal h.ref indica los tiempos en que hay información válida y finalmente que la señal PCLK sirve de reloj, de modo que cada flanco de subida indica información nueva en las líneas D[7 : 0].

Figure 5 Horizontal Timing

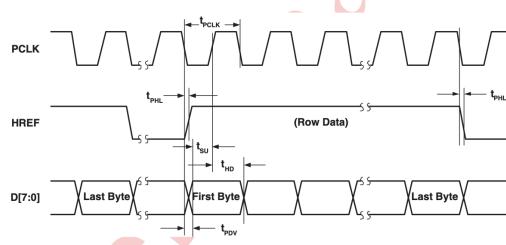


Figure 6 VGA Frame Timing

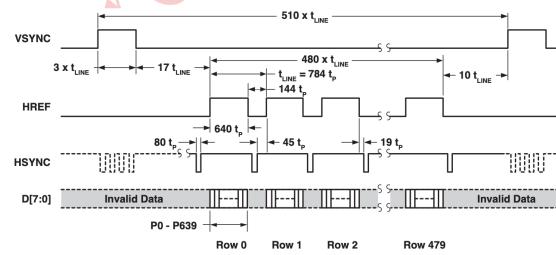


Figura N° 4: Diagrama de tiempo de las señales de la cámara OV7670. Tomado de [1]

Además es necesario fijarse en la convención con la que se recibe cada byte de información D[7 : 0] de la cámara, en donde es importante resaltar que al usar el formato RGB555 la información de cada píxel es enviada a través de dos bytes separados, según como se ve en la figura 5.

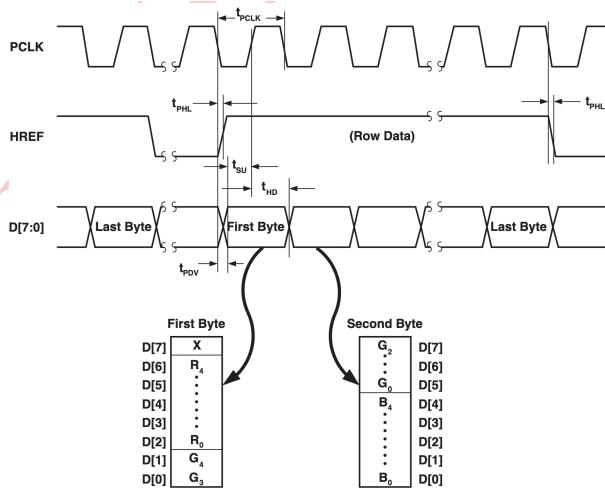


Figura N° 5: Convención de orden de cada byte y la información que lleva. Tomado de [1]

Con esto en mente se diseña el diagrama ASM que describe el módulo pixel\_reader, el cual se puede ver en la figura 6. De él se puede mencionar rápidamente que tiene un estado inicial WAIT\_VSYNC cuyo objetivo es esperar a que inicie un nuevo frame por primera vez, luego tiene un flujo normal en el que se lee el primer byte, luego el segundo byte, guarda la información del píxel en la memoria y así sucesivamente hasta guardar todo un frame en la memoria. Luego genera las señales de control de haber acabado un frame y actualiza la información de en qué bloque de memoria está guardando la información.

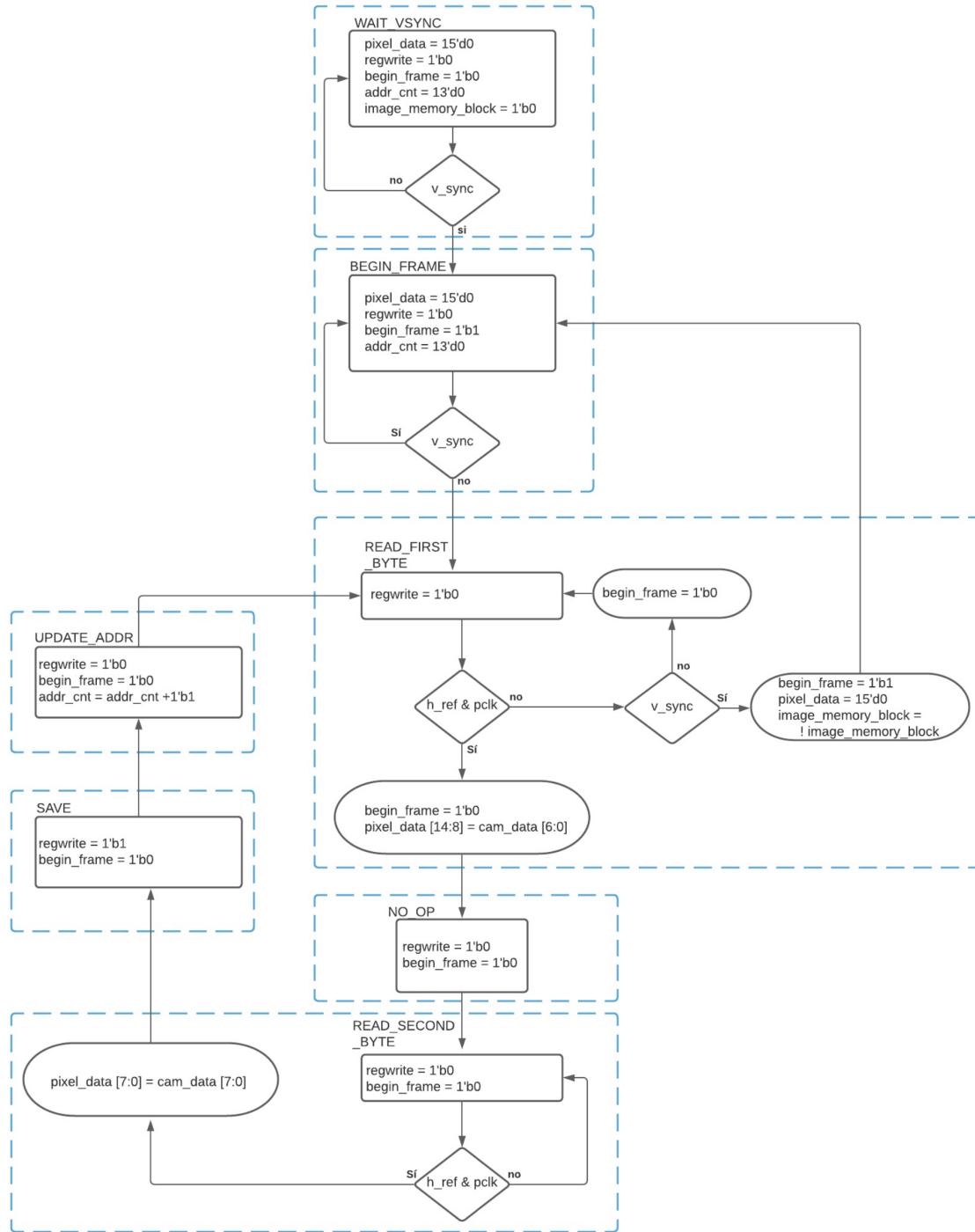


Figura N° 6: Diagrama ASM del bloque `pixel_reader`

#### ■ `image_sender`:

Este módulo tiene como única función enviar la información de cada frame por medio de UART al

computador, por ello su funcionamiento se apoya en la existencia de un submódulo UART, el cual fue tomado de [2], y tiene un funcionamiento básico en el que recibe un byte de información, y la envía al recibir un nivel alto en el puerto send\_byte, además al terminar el envío genera un pulso en la señal Tx\_done. Con eso en mente, el módulo image\_sender se concentra en recorrer la memoria hasta 3072 (dado que la resolución de la cámara usada es  $64 * 48 = 3072$ ) e ir enviando la información de cada pixel por medio de 2 bytes, para lo cual hace un correcto uso de las señales send\_byte y Tx\_done. Además al terminar el frame envía un byte especial (0xFF) que indica el final del frame. Por último vale la pena decir que el módulo comienza a operar al recibir un alto en el puerto send\_image, y genera un pulso en la señal done al terminar el proceso. El diagrama ASM del módulo se puede observar en la figura 7.

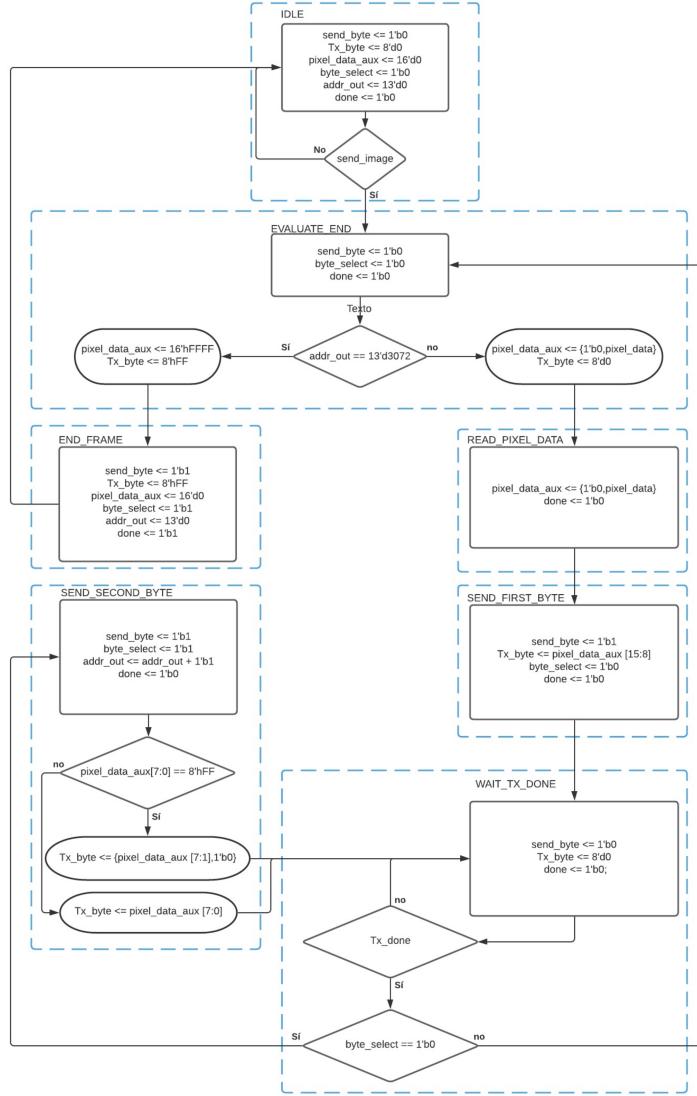


Figura N° 7: Diagrama ASM del bloque image\_sender

#### ■ color\_detector:

Como su nombre lo indica, este módulo se encarga de detectar el color de una mina, si esta existe,

para ello se basa en un submódulo llamado RGB\_pixel\_sorter, el cual tiene la función de recorrer la memoria y evaluar píxel por píxel para determinar si es un píxel rojo, azul, verde, o si por el contrario no pertenece a ninguna mina, además el módulo hace una cuenta de la cantidad de píxeles rojos, azules y verdes presentes en cada frame. El diagrama ASM de dicho módulo se puede ver en la figura 8. Otra particularidad de este módulo es que comienza a operar al recibir una señal de alto en el puerto sort\_pixels, y al terminar su función genera una señal positiva en el puerto done.

En cuanto al módulo color\_detector, este se encarga por un lado de controlar al submódulo RGB\_pixel\_sorter, y además de tomar la decisión final de si existe una mina o no, para ello usa la convención de que al haber más de 600 píxeles de un color dado se interpreta como la existencia de una mina de dicho color. Este módulo es activado por una señal de alto en el puerto detect\_color y tiene como salidas una señal para indicar el final del proceso: done, y la decisión tomada, la cual es un vector de 4 bits con la siguiente codificación: 0b0000 indica la ausencia de minas, 0b0100 es una mina roja, 0b0010 es una mina verde y 0b0001 es una mina azul. Finalmente el diagrama ASM de este módulo se puede ver en la figura9.

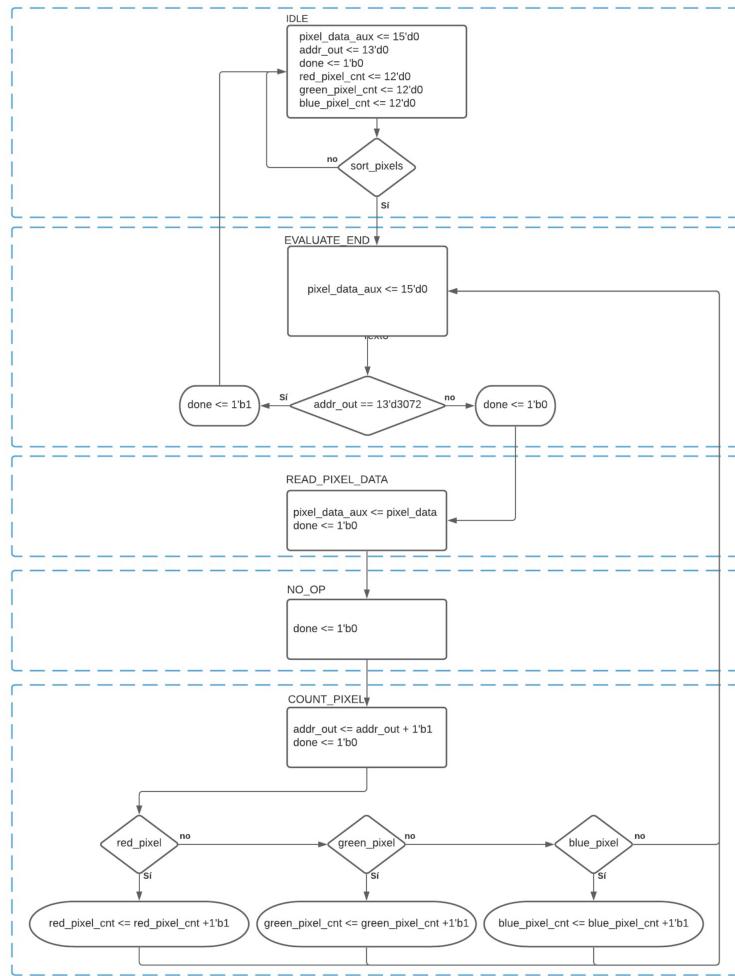


Figura N° 8: Diagrama ASM del bloque RGB\_pixel\_sorter

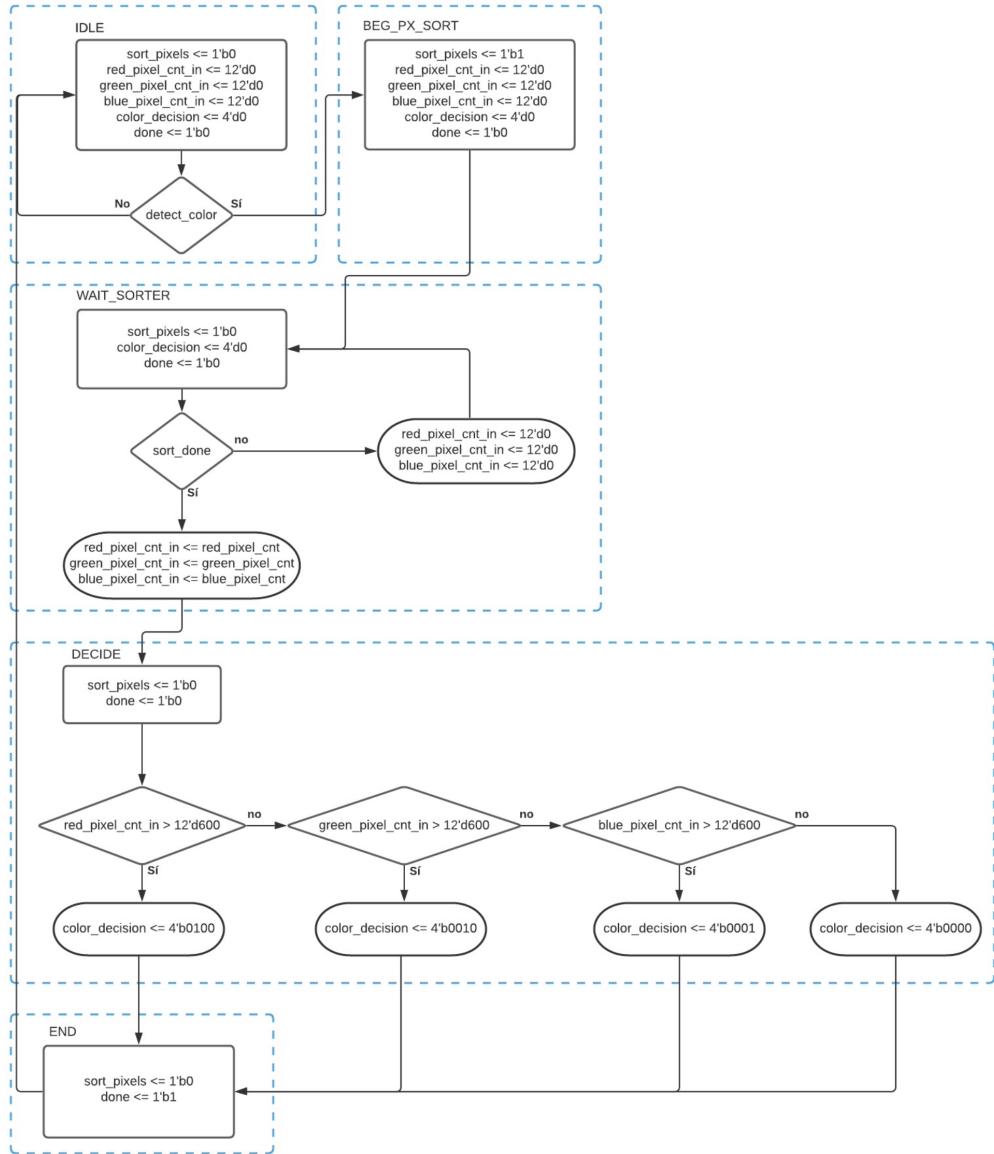


Figura N° 9: Diagrama ASM del bloque color\_detector

#### ■ control:

Como lo indica su nombre, este módulo es el encargado de controlar los componentes del datapath, a él llegan señales de estado que le indican cuando comenzar un proceso, y cuando pasar de un proceso al siguiente. También recibe el indicativo de en qué bloque de memoria se está guardando la imagen recibida, por medio de la señal image\_memory\_block, y actúa en consecuencia para que los demás componentes del datapath lean del otro bloque de la memoria, esto con el fin de prevenir errores de escritura y lectura simultánea. Por lo demás, tiene un flujo bastante natural, el cual comienza al recibir un alto en la señal begin\_frame, en donde procede a activar el módulo image\_sender y esperar a que termine, luego activa el módulo color\_detector y espera a que este termine para guardar el resultado del análisis, por último activa el módulo de UART para enviar la información de la mina a la ESP32 y vuelve al estado inicial. El diagrama

ASM del bloque se puede observar en la figura 10.

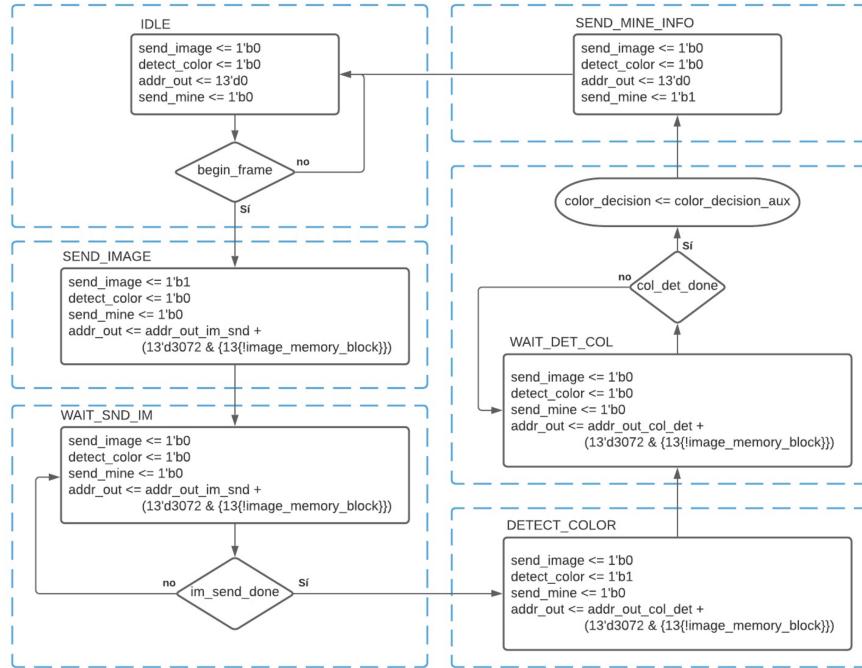


Figura N° 10: Diagrama ASM del bloque control

#### ■ memoria:

Este módulo es una memoria RAM de doble puerto, caracterizada por permitir leer y escribir en ella de forma simultánea e independiente, además de las direcciones de entrada y salida, y los datos de entrada; recibe una señal de escritura, de modo que la memoria solo escribe los datos de entrada si dicha señal está en alto. El módulo fue tomado de [3].

#### ■ UART:

Este módulo se encarga de enviar datos desde la FPGA al ESP 32, su funcionamiento es básico, en tanto que recibe el byte a enviar, una señal que le indica cuando empezar la transmisión genera una señal alta al terminar la transmisión del byte. Como ya se dijo antes, fue tomado directamente de [2].

## 4.4. Diseño software

Según lo establecido en la sección 4.1 y siguiendo con el proceso de diseño, se plantea un diagrama de flujo, presentado en la figura 11 y que será implementado posteriormente en un ESP32. En este diagrama se presentan tres etapas principales:

- Setup: este estado descrito en el diagrama con el nombre de Init es el encargado de inicializar las variables de control del movimiento de los motores y adicionalmente se encarga de realizar la configuración inicial de la cámara.
- Tratamiento de datos: Esta parte esta conformada por el estado encargado de decodificar las señales obtenidas por los sensores ultrasonidos, llamado Distance, y a partir de este resultado enviar la dirección de movimiento por medio de Bluetooth hacia el computador.

- Movimiento: Una vez se ha identificado la ubicación espacial del vehículo es necesario indicar a los motores, por medio de señales enviadas un punte H, la dirección que debe tomar, este proceso se realiza por medio de los estados Front, Left, Left\_C y Right para avanzar hacia adelante, izquierda o derecha respectivamente. Este ítem se verá a profundidad en la sección 5.2.1.

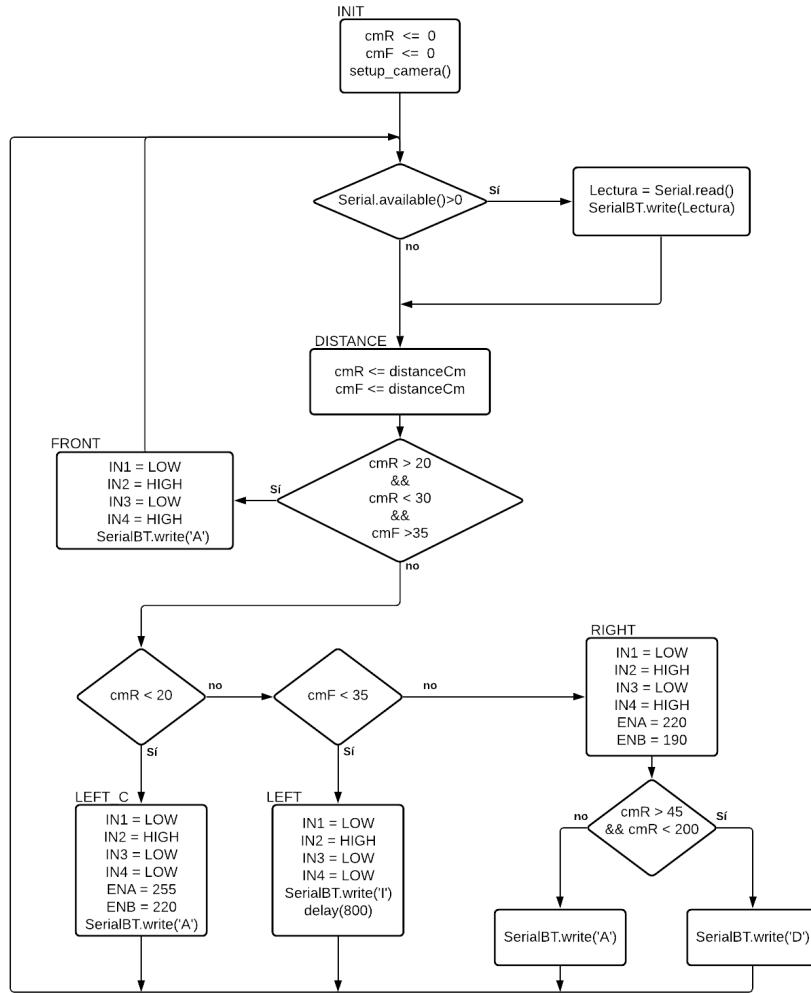


Figura N° 11: Diagrama de flujo Software

## 5. Implementación

### 5.1. Hardware

La implementación del diseño del hardware presentado se hace teniendo en cuenta la necesidad de garantizar ciertas condiciones de diseño que impone Vivado a la hora de realizar la síntesis, como por ejemplo el tamaño de memoria que es posible implementar u criterios requerimientos de timing que Vivado impone. Por otro lado la implementación de la mayoría de módulos es directa a partir de los diagramas ASM, por lo que no vale la pena entrar en detalles adicionales en esta sección, sin embargo para instanciar la memoria y para

realizar el módulo color\_detector, hizo falta tener ciertas consideraciones para realizar la implementación, dichas consideraciones se mencionan en las siguientes subsecciones.

### 5.1.1. Memoria RAM

El tamaño del buffer para la memoria RAM se encuentra limitado físicamente por la capacidad de memoria en la FPGA, adicionalmente se determinó que una calidad de imagen baja permite una mayor facilidad en la transferencia de datos, lo cual es suficiente para la detección de color y el análisis de la imagen. Por esto, la resolución de la imagen utilizada es de 64x48 píxeles y se empleo un formato de color RGB555, es decir 15 bits por cada píxel, lo cual genera un requerimiento de 46080 bits por captura de imagen.

Con el fin de evitar conflictos en la lectura y escritura de imagen se asigno el espacio en memoria para la captura de 2 imágenes, y así evitar la lectura de datos que aún no han sido escritos o que se encuentran siendo escritos.

El ancho de cada registro se encuentra definido por el formato del píxel es decir 15 bits, así pues solo falta determinar el alto del registro para definir los parámetros del buffer (Figura 12).

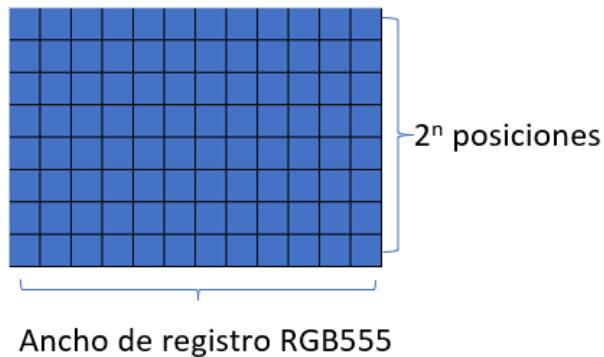


Figura N° 12: Tamaño del registro

Sabiendo que el tamaño de la imagen es de 64x48 lo que hacen 3072 píxeles empleamos el logaritmo en base 2 para obtener la altura del registro para 2 capturas:

$$\log_2(3072 \cdot 2) = 12,58 \approx 13 \quad (1)$$

Con los parámetros anteriormente obtenidos se procedió a instanciar el módulo buffer\_ram\_dp, sin perder de vista que se debe realizar la carga del archivo image.men el cual permite inicializar la memoria para su correcto funcionamiento [3]. El módulo se encuentra controlado por un bit de lectura o escritura, según sea el caso y se establece el dato de entrada y su respectiva dirección para realizar el almacenamiento o posterior lectura.

### 5.1.2. Análisis del color

Para lograr la correcta identificación y análisis del color se comienza por realizar una lectura píxel a píxel, en el cual lo que se hace es decidir si el píxel leído es Rojo, Verde, Azul o Blanco en caso de que no cumpla ninguna de las condiciones establecidas a continuación. Teniendo en cuenta que se eligió una configuración RGB555 lo que nos deja con que cada píxel estará representado por un vector de tamaño [14:0] almacenado en la variable pixel\_data\_aux en donde los píxeles rojos corresponden a los bits [14:10], los verdes a los bits [9:5] y finalmente los azules de [4:0]. Es importante aclarar que estas condiciones fueron establecidas de manera empírica a partir de encontrar un comportamiento común, cuando se mostraban diferentes colores a la cámara, analizando el valor de cada componente frente a los demás de manera que:

- Rojo:** Para clasificar un píxel como rojo es necesario que la componente roja y azul sean al menos 4 unidades mayor que la componente verde. Adicionalmente se estable como límite superior que las tres componentes sean menores a 27 unidades (en decimal).
- Verde:** Para considerar el píxel como verde es necesario que esta componente sea 2 unidades mayor a la componente roja y la azul, nuevamente todas las componentes deben tener un valor menor a 24.
- Azul:** Finalmente para que sean clasificados como azules es necesario que todas las componentes se encuentren en un rango de 8 a 19 y que la diferencia entre la componente roja y la verde sea menor a 3 unidades y que la componente azul sea mayor a las demás por 3 unidades.

Una vez están clasificados todos los píxeles que componen una imagen, se pide que hayan al menos 600 de un mismo color para decidir si la figura que se tiene al frente de la cámara es de un color u otro, lo que indica la presencia de una mina de dicho color.

### 5.1.3. Disposición final de los módulos de la FPGA

Finalmente, luego de realizar toda la implementación y tener en cuenta los detalles mencionados anteriormente se llega a la disposición final del hardware dentro de la FPGA, la cual se puede ver en la figura 13. En ella se pueden observar todas las conexiones que hay entre los módulos de forma detallada, así como las entradas y salidas de cada módulo y de la FPGA según los expuesto en la figura 2.

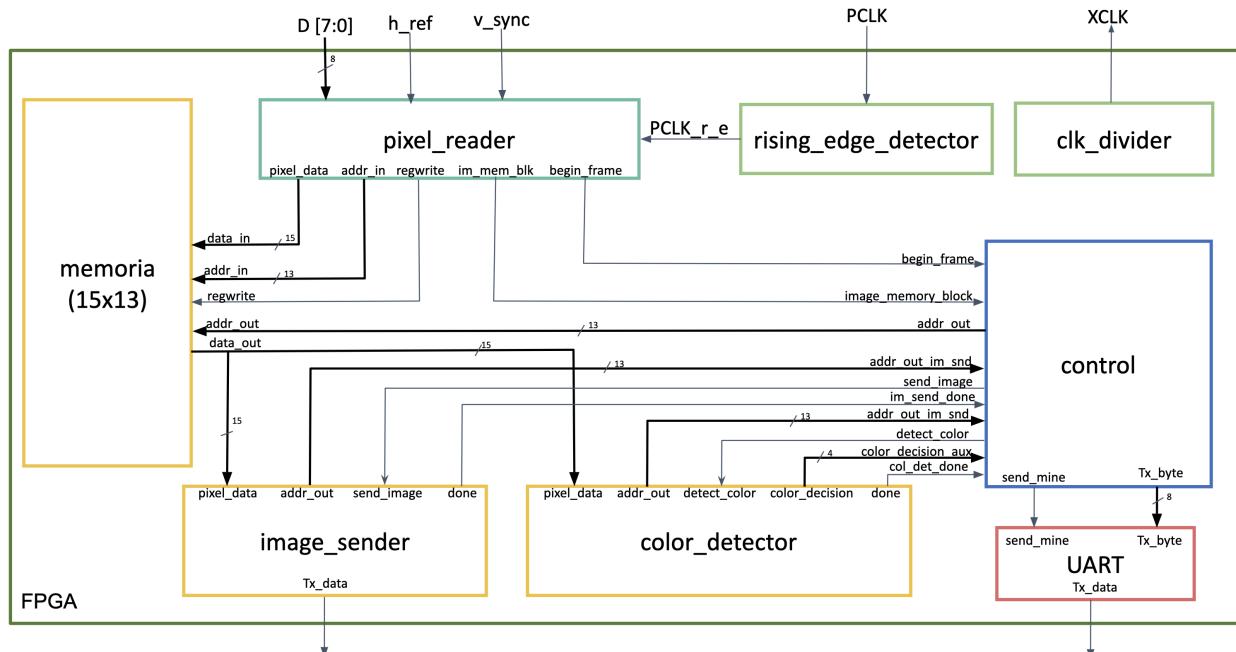


Figura N° 13: Disposición final del hardware en la FPGA

Vale la pena destacar que en el diagrama se ven 2 módulos nuevos, uno es un divisor de frecuencias llamado **clk\_divider**, el cual tiene como objetivo generar la señal de reloj para que trabaje la cámara y es independiente a los demás módulos del proyecto. El otro módulo nuevo es un detector de flanko de subida o **rising\_edge\_detector**, el cual tiene como objetivo hacer un tratamiento inicial de la señal **PCLK** proveniente de la cámara, este módulo fue necesario agregarlo para evitar errores y warnings a la hora de sintetizar el proyecto en Vivado, y tiene un funcionamiento muy básico el cual se puede ver en la figura 14.

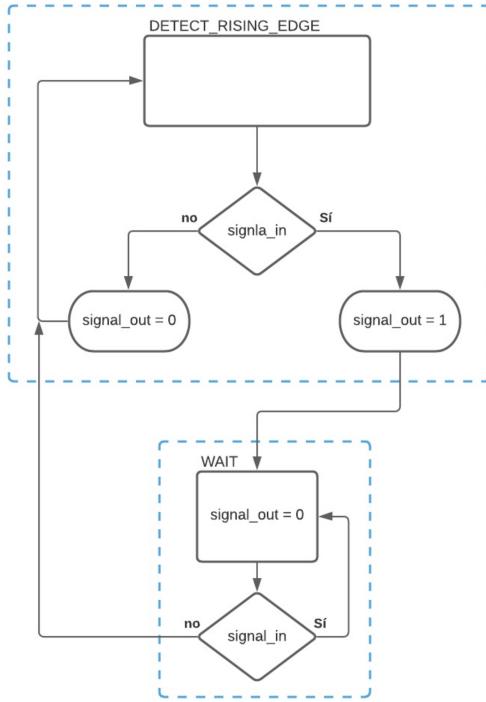


Figura N° 14: Diagrama ASM del módulo rising\_edge\_detector

## 5.2. Software

Las implementaciones referentes al apartado del software fueron realizadas utilizando el IDE de Arduino el cual permite programar el microcontrolador ESP32, teniendo esto en cuenta se va a describir a mayor profundidad la implementación para el movimiento del carro.

### 5.2.1. Movimiento de carro

Como se había enunciado previamente en la sección 4.4 y tomando como referencia el diagrama presentado en la figura 11, se empieza por recolectar el valor de las variables cmR y cmF las cuales representan la distancia a la pared derecha y la frontal respectivamente, teniendo en cuenta lo anterior hay tres posibles opciones para realizar un movimiento, las cuales se transmitirán a un puente H que a su vez esta conectado a dos motores DC tal y como se indicó en la imagen 2, las condiciones para realizar cada uno de los distintos movimientos se presentan a continuación en orden:

1. **Front:** Lo primero que se hace es comprobar si la distancia cmR se encuentra en un rango entre 20 y 30 centímetros y que la pared frontal (cmF) se encuentre a una distancia mayor o igual a 35 centímetros, si ambas condiciones se cumplen se le indica a el puente H que ambos motores deben girar en el mismo sentido por medio de las variables IN 1 a IN 4; de lo contrario es necesario evaluar la condición 2.
2. **Left\_C:** Si se ha llegado a este punto es porque la distancia a la pared derecha es menor a 20 centímetros, por lo que es necesario hacer una ligera corrección en el rumbo, por ello se baja ligeramente la velocidad del motor izquierdo para que el carro se aleje de la pared y se retome el rumbo correcto.
3. **Left:** Si se ha llegado a este punto es porque la distancia a la pared frontal es menor a 35 centímetros, por lo que es necesario hacer un giro completo a la izquierda, por ello se apaga el motor izquierdo y se permanece en este estado por 800ms para que el giro se efectue.

4. **Right:** Si por el contrario la variable cmR se desbordó por arriba, es decir que es mayor a 35 centímetros, se deberá enviar una señal para girar a la derecha, sin embargo, a diferencia del giro a la izquierda en este caso ambos motores estarán funcionando solo que se aplica un PWM de manera que el motor derecho gire considerablemente más lento que el izquierdo. Esto se realizó a modo de seguro para que en caso de perder la referencia, el robot realice un movimiento en espiral que le permita desplazarse en búsqueda de una nueva pared que pueda utilizar como guía.

### 5.3. Configuraciones iniciales, cámara y bluetooth

#### 5.3.1. Configuración de la cámara OV7670 mediante esp32 [4]

Una de las ventajas de utilizar la cámara OV7670 es que esta tiene un gran número de registros que pueden ser configurados, lo cual hace a la cámara muy versátil. Dicha configuración se hace por medio de un bus serial con el protocolo SCCB, el cual es compatible con I2C. Para el caso de esta implementación la configuración se hizo por medio de un código disponible en Github y especial para la cámara OV7670, este se puede encontrar en [5].

En cuanto a la configuración, se hicieron principalmente 2 cambios, por un lado se configuró la cámara para trabajar con una resolución de 64x48 píxeles y en formato RGB555. Por otro lado se hicieron un gran número de modificaciones para ajustar de manera adecuada los colores y el brillo de la imagen, de tal forma que facilitara la detección de colores.

##### ■ Configuración de la resolución y el formato RGB555

Como ya se mencionó antes, el formato elegido de la cámara fue el RGB555, para configurar la cámara con dicho formato fue importante la configuración de los registros COM7 y COM15. El registro COM7 fue escrito con el valor 0b00000100, de donde los 5 bits más significativos son 0 para no elegir ninguna resolución predefinida, y mediante el tercer bit más significativo y el menos significativo en valor 0b10 se selecciona el formato RGB. Luego el registro COM15 se escribe con el valor 0b10110000, de tal manera que  $\text{COM15}[5 : 4] = 0b11$  para seleccionar el formato RGB555.

En cuanto al cambio de resolución para pasar de 640x480 a una resolución de 64x48 fue necesario configurar varios registros, por un lado los registros SCALING\_XSC, SCALING\_YSC, SCALING\_DCWCTR y COM3 se editan para que la cámara haga un proceso de down sampling adecuado, y por otro parte los registros SCALING\_PCLK\_DIV y SCALING\_PCLK\_DELAY se ajustan para que la señal PCLK se acomode adecuadamente a la nueva resolución.

El registro COM3 se configura con el valor 0b00001100 de tal modo que se activa la opción del downsampling y el escalamiento por medio de COM3[3 : 2]. A Continuación el registro SCALING\_DCWCTR se configura para generar un primer proceso de downsampling en un factor de 2, 4 o 8, como en este caso se desea una reducción final de x10 se selecciona la opción de 8. Por ultimo se determina el valor de SCALING\_XSC y SCALING\_YSC mediante la formula disponible en [4]:

$$\begin{aligned} \text{SCALING\_XSC} &= 0x20 \cdot \text{image size from down sampling/new size} \\ &= 0x20 \cdot (640/8)/64 = 0x20 \cdot 80/64 = 40 \end{aligned}$$

$$\begin{aligned} \text{SCALING\_YSC} &= 0x20 \cdot \text{image size from down sampling/new size} \\ &= 0x20 \cdot (480/8)/64 = 0x20 \cdot 60/48 = 40 \end{aligned}$$

Por ultimo se ajustan los registros SCALING\_PCLK\_DIV y SCALING\_PCLK\_DELAY para generar un correcto funcionamiento de PCLK, para ello SCALING\_PCLK\_DIV[3 : 0] toma el valor de 0b0011, con lo que se divide el PCLK por 8, además se modifica SCALING\_PCLK\_DELAY de tal forma que en cada fila solo haya 48 pixeles, el valor para este registro es determinado de forma empírica llegando al valor óptimo de 0b00001001.

## ■ Configuración del color

Primero que todo se modifican los registros COM8, BLUE, RED y GGAIN, el primero desactiva la calibración automática de blancos y el control automático de ganancia, mientras que los últimos tres configuran la ganancia de cada color primario RGB. Esto se hace porque en las condiciones particulares de este proyecto, es preferible que estas funciones sean fijas y que no varíen en el tiempo, todo con el fin de que la detección de colores sea más confiable.

Por otra parte también se hace una gran configuración de los registros MTX1, MTX2, MTX3, MTX4, MTX5, MTX6, MTXS, los cuales portan el valor de los elementos de cierta matriz para ajustar los colores. Para determinar estos valores se siguen las recomendaciones hechas en la sección 5.3 de [4], en donde se utiliza como valor apropiado  $\alpha = 175^\circ$ .

Finalmente como un resumen general, se puede ver en la figura 15 el resumen de todos los registros configurados.

```
// FONDO BLANCO          //register address , data
byte config_data [] = { 0x00, B11110000,    // GAIN: gain setting
                        0x01, 0x80,        // BLUE: blue gain for white balance
                        0x02, 0x80,        // RED: red gain for white balance
                        0x03, B01000011,   // VREF: gain setting
                        0x0F, B01000001,   // COM6: no reset timing when format changes??
                        0x11, B10011100,   // CLKRC: divide clk -> 1.5 fps
                        0x12, B00000100,   // COM7: RGB
                        0x13, B10001001,   // COM8: Disable automatic white balance and automatic gain
                        0x1E, B00110000,   // MVFP: vertical and horizontal flip
                        0x24, 0xf0,        // AEW: for setting exposure time, stable upper limit
                        0x25, 0xd0,        // AEW: for setting exposure time, stable lower limit
                        0x26, 0xfa,        // AEW: for setting exposure time, control zone
                        0x3E, B00010011,   // COM14: divide PCLK by 8
                        0x40, B10110000,   // COM15: output from [01] to [FE] and RGB555
                        0x70, B00101000,   // SCALING_XSC: scaling
                        0x71, B00101000,   // SCALING_YSC: scaling
                        0x72, B00110011,   // SCALING_DCWCTR: down sampling by 8
                        0x73, B00000011,   // SCALING_PCLK_DIV: PCLK divide by 8
                        0x0C, B00001100,   // COM3: downsampling and zoom out enable
                        0xA2, B00001001,   // SCALING_PCLK_DELAY: to fit number of pixels
                        0xB1, B00000100,   // ABLC1: enable black level calibration
                        0x6A, 0xaF,        // GGAIN: green gain for white balance
                        0x6F, B10011010,   // AWBCTR0: maximum color gain x2
                        0x4F, 0x1A,        // - colour conversion matrix
                        0x50, 0x44,        // MTX2 - colour conversion matrix
                        0x51, 0x62,        // MTX3 - colMTX1our conversion matrix
                        0x52, 0x42,        // MTX4 - colour conversion matrix
                        0x53, 0x40,        // MTX5 - colour conversion matrix
                        0x54, 0x00,        // MTX6 - colour conversion matrix
                        0x58, B10001100,   // MTXS - Matrix sign and auto contrast
                        0x3D, 0xC0,        // COM13 - Turn on GAMMA and UV Auto adjust
                        0x3A, 0x04,        // TSLB Set UV ordering, do not auto-reset window
};
```

Figura N° 15: Configuración de la cámara

### 5.3.2. Configuración de módulo bluetooth HC-05. [6]

El módulo bluetooth HC-05 es empleado para transferir los datos de la cámara al computador para lograr la visualización de la imagen, para esto se debe configurar previamente la velocidad de transferencia a 230400 bps para que funcione adecuadamente.

Para realizar dicha configuración seguimos los siguientes pasos:

1. Realizamos la conexión entre el Arduino y el módulo como se observa en la imagen:

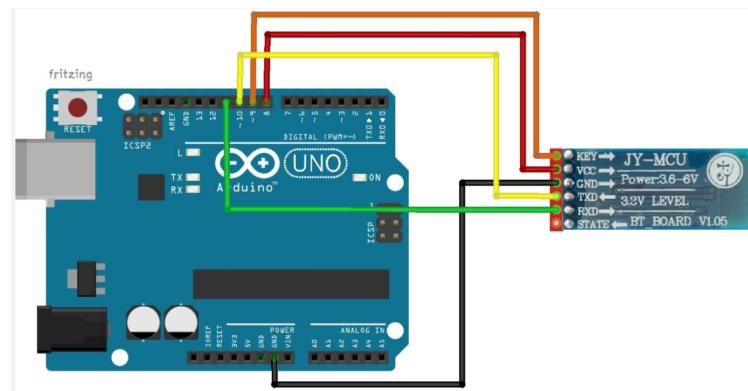


Figura N° 16: Conexión entre módulo HC-05 y arduino [6]

2. El código para programar el módulo se debe ejecutar en una consola de comandos, para el presente caso se emplea el IDE de Arduino mediante un sketch, por lo cual se abre la terminal y se ejecuta el siguiente código:

```
#include <SoftwareSerial.h>

SoftwareSerial BT1(10, 11); // RX | TX
void setup()
{
    pinMode(8, OUTPUT);      // Al poner en HIGH forzaremos el modo AT
    pinMode(9, OUTPUT);      // cuando se alimente de aqui
    digitalWrite(9, HIGH);
    delay (500);           // Espera antes de encender el modulo
    Serial.begin(9600);
    Serial.println("Levantando el modulo HC-05");
    digitalWrite (8, HIGH);   //Enciende el modulo
    Serial.println("Esperando comandos AT:");
    BT1.begin(57600);
}

void loop()
{
    if (BT1.available())
        Serial.write(BT1.read());
    if (Serial.available())
        BT1.write(Serial.read());
}
```

Figura N° 17: código de arduino

3. Posteriormente en la barra de herramientas se elije el monitor serie, el cual se debe configurar con los mismos bps que se programó el código para la comunicación, para el presente caso es de 9600 bps.
4. Una vez ejecutados los pasos anteriores se pasa a ingresar los comandos AT que nos permiten realizar la configuración del módulo HC-05.  
Los comandos empleados son los siguientes:

- AT: si el módulo tiene una correcta comunicación con el computador nos devuelve un OK, de lo contrario hay que revisar las conexiones y la configuración.
- AT+ PSWD?: devuelve la contraseña actual del dispositivo
- AT+PSWD=1234: establece la contraseña como 1234.

- AT+NAME?: devuelve el nombre actual del dispositivo.
- AT+NAME=Robot: establece el nombre del dispositivo como robot.
- AT+UART?: devuelve la velocidad de transferencia actual.
- AT+UART=230400,0,0 : configura la velocidad de transferencia a 230400 bps.
- AT+ROLE?: devuelve el modo actual de funcionamiento, 0 es slave ó 1 master.
- AT+ROLE=0: configura el módulo para ser usado como esclavo, para configurar como maestro sólo se cambia el 0 por 1.
- AT+CMODE?: devuelve la información del dispositivo, este indica 0 para conectarse a un dispositivo específico o 1 si cualquier dispositivo se puede conectar al módulo.
- AT+CMODE=1: configura el módulo para que cualquier dispositivo se pueda conectar a el.

## 5.4. Recepción de datos en el computador

### 5.5. Video

Para la recepción del vídeo en el computador, se utilizó un software libre enfocado a proyectos gráficos llamado Processing, ya que permite dibujar cosas en pantalla de una manera muy sencilla, y además posee una librería que permite trabajar con comunicación serial de forma cómoda.

El programa es en realidad bastante sencillo, por un lado se declara un puerto serial, y se configura para recibir los datos que llegan por Bluetooth al computador y generar un evento cada vez que se recibe el byte 0xFF, que corresponde a la bandera del final de un frame, esto se hace fácilmente mediante la función bufferUntil(0xFF) . Luego cada vez que se detecta el evento serial, se procede a leer todos los bytes y almacenarlos en un arreglo de bytes llamado image, para luego construir la imagen a partir del arreglo. Para esto se usa un par de ciclos for anidados que recorren toda la imagen fila por fila, y en cada iteración se interpreta la información de un pixel, compuesto por dos bytes sucesivos, y se dibuja un cuadrado del color adecuado y en la posición que corresponde al pixel, el código correspondiente a dicho algoritmo puede verse en la imagen 18.

```

for (int i=0; i<newHeight; i++) {
  for (int j=0; j<newWidth; j++) {
    FB = image[px] & 0xff;
    SB = image[px+1] & 0xff;
    pixel_Data = (FB<<8) + SB;
    R = (pixel_Data>>10) & 0x0000001F;
    R = map(R,0,31,0,255);
    G = (pixel_Data>>5) & 0x0000001F;
    G = map(G,0,31,0,255);
    B = (pixel_Data>>0) & 0x0000001F;
    B = map(B,0,31,0,255);
    fill(R,G,B);
    rect(j*width/newWidth, i*height/newHeight, dx, dy);
    px += 2;
  }
}

```

Figura N° 18: Algoritmo para mostrar la imagen a partir del arreglo de bytes

### 5.6. Mapa

La generación del mapa en el computador se hace, nuevamente, con el software de Processing por la facilidad que ofrece este para generar gráficos. El algoritmo para hacer el mapa se basa en el tipo de información que el ESP 32 envía por bluetooth al computador, esto es una serie de caracteres cuyo significado está completamente ligado al tipo de movimiento que está haciendo el carro o la mina presente en el recorrido, de modo que dependiendo del carácter recibido se hace una acción distinta:

- ‘A’: se entiende que el carro está avanzando en linea recta y sin cambiar de dirección, luego en el mapa se debe pintar un segmento recto partiendo de la posición anterior y sin cambiar la dirección.
- ‘I’: se entiende que el carro giró a la izquierda, luego en el mapa se debe realizar un giro de  $-90^\circ$  respecto a la dirección anterior y después pintar un segmento que parte de la posición anterior pero que es muy corto.
- ‘D’: se entiende que el carro giró a la derecha, luego en el mapa se debe realizar un giro de  $90^\circ$  respecto a la dirección anterior y después pintar un segmento que parte de la posición anterior pero que es muy corto.
- ‘R’ ’G’ o ‘B’: se entiende que existe una mina roja, verde o azul respectivamente, luego debe pintarse un pequeño circulo en la posición actual y del color especificado.

Ahora, pasando al código como tal implementado, se hace uso de 4 funciones propias de Processing y que son de mucha utilidad en este caso, `translate(x,y)` la cual mueve el origen del eje coordenado al punto (x,y), `rotate(angle)` la cual rota el eje coordenado según el ángulo especificado, `pushMatrix()` la cual guarda el eje coordenado actual y `popMatrix()` la cual retoma el eje coordenado guardado anteriormente.

Con estas cuatro funciones resulta particularmente fácil llevar registro de la posición actual y de la dirección actual, pues lo que se hace es siempre que se pinta un segmento, mover el origen al final del segmento, de modo que la posición actual siempre es (0,0), y además la dirección de al frente siempre será la de dirección del eje y. Con esto en mente se pueden observar las funciones para pintar un segmento en una dirección dada y para dibujar una mina en la posición actual en la figura 19, las cuales son utilizadas dependiendo del carácter que se recibe según las reglas enunciadas anteriormente.

```

void draw_segment(float len, float angle) {
    popMatrix();
    rotate(angle);
    line(0, 0, 0, -len);
    translate(0, -len);
    pushMatrix();
}

void draw_mine(color c) {
    fill(c);
    strokeWeight(1);
    circle(mine_distance, 0, mine_radius);
    strokeWeight(2);
}

```

Figura N° 19: Algoritmo para dibujar un segmento en una dirección dada y para dibujar una mina

## 6. Integración hardware y software

La integración del hardware y el software responde a la necesidad de comunicar las dos componentes principales del SoC, de forma que exista una sincronía entre los dos procesos y que el SoC funcione adecuadamente. En el caso presente, dicha Comunicación resulta bastante sencilla, pues en el diseño se eligió de una forma muy adecuada la partición de hardware y software de forma que los dos componentes poseen funciones casi independientes y en ese sentido la comunicación que deben tener es mínima.

En concreto la única comunicación que debe existir responde a la necesidad de que la FPGA comunique la información de una mina a la ESP 32 para que esta pueda enviar la información necesaria para hacer el mapa al computador.

Con esto en mente y dada la sencillez de la comunicación, se decide utilizar el protocolo UART para la

comunicación, de modo que el diagrama que expresa la conexión entre el software y el hardware se puede ver en la figura 20.

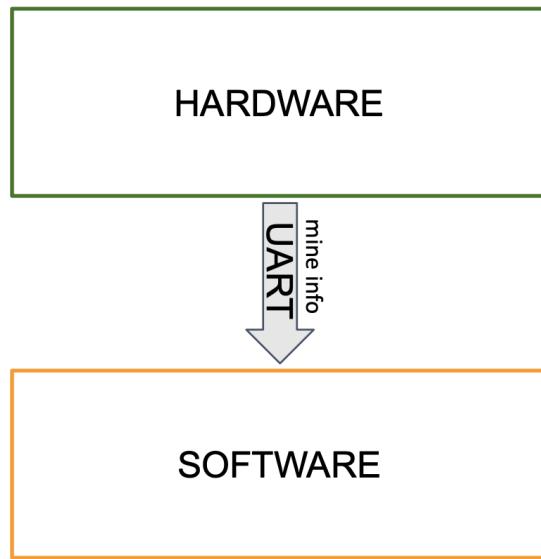


Figura N° 20: Integración hardware y software

## 7. Montaje Final

### 7.1. Robot

Para realizar el montaje final del robot fue necesario realizar ciertos ajustes y consideraciones especiales, primero que todo, y el más importante, fue la necesidad de incluir dos baterías portátiles para alimentar dispositivo, por un lado una se encarga de alimentar la FPGA, por otro lado el resto de componentes fueron alimentados con la otra batería. Esto se realizó así con el fin de que los motores tuvieran suficiente potencia, y además para garantizar una tensión uniforme y constante en la FPGA, pues esta es sensible a los picos de tensión que se pueden generar al arrancar y detener un motor.

Por otra parte fue necesario ubicar la cámara en una posición estratégica para que la detección de minas se diera correctamente, por esto se ubica sobre la FPGA para que lograse tener cierta altura y alcanzara a grabar las paredes del laberinto.

Finalmente el montaje definitivo del robot se puede observar en la figura 21 en donde se resaltan las dos baterías, una de menor tamaño sobre la FPGA que alimenta la FPGA, y otra de mayor tamaño ubicada justo debajo de la FPGA. Además se resalta la posición de la cámara.

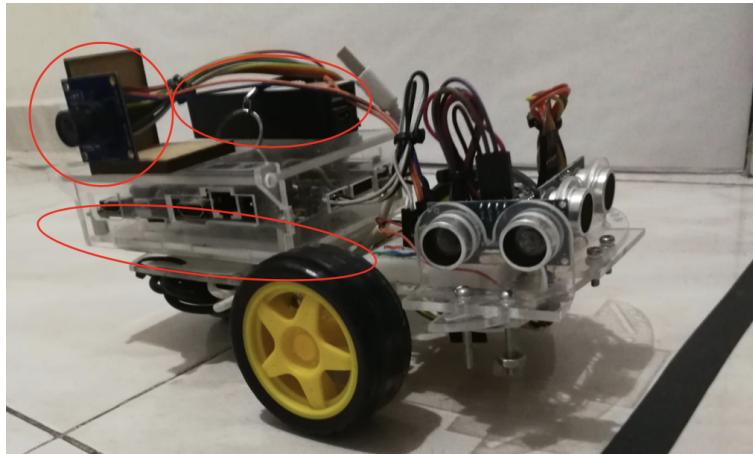


Figura N° 21: Montaje final del robot

## 7.2. Conexión con el computador

Dado que en el computador se visualiza tanto el video como el mapa generado, fue necesario hacer una conexión entre el computador y la FPGA para el envío del video y otra conexión entre el ESP 32 y el computador para el envío de la información del mapa. Ambas conexiones fueron hechas a través de bluetooth, sin embargo el computador solo posee un puerto bluetooth, por lo que fue necesario usar un modulo externo auxiliar que se conectara con la FPGA y luego enviar esa información por USB al computador, mientras que el puerto interno del computador es ocupado por el ESP 32. El esquemático de dichas conexiones se puede observar en la figura 22.

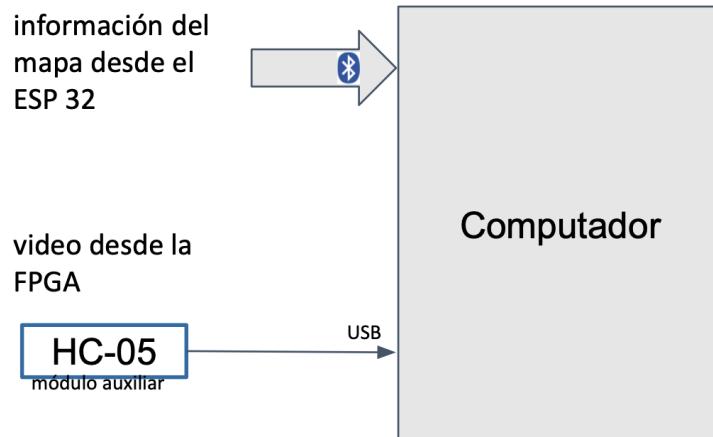


Figura N° 22: Integración hardware y software

## 8. Resultados / validación

Para realizar las pruebas del robot detector de minas se diseño un laberinto de tal manera que pudiera demostrar las funcionalidades del robot y que cumpliera con los requerimientos propuestos inicialmente (detección de color, detección de posición, recorrido Y generación de mapa).

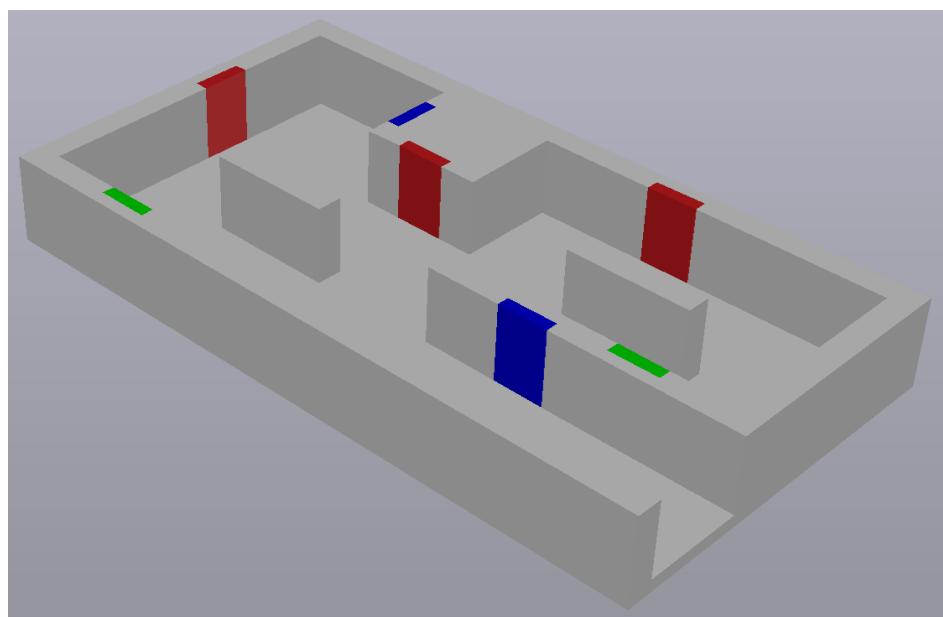


Figura N° 23: Laberinto de pruebas vista isométrica izquierda

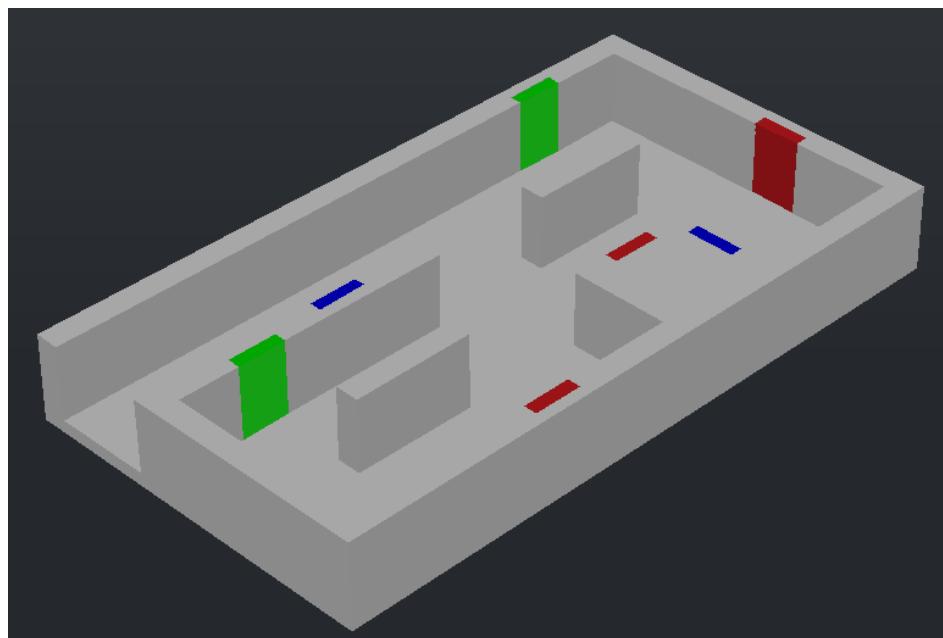


Figura N° 24: Laberinto de pruebas vista isométrica derecha

Basados en el algoritmo diseñado para el desplazamiento del robot, el desplazamiento esperado se puede observar en la Figura 25

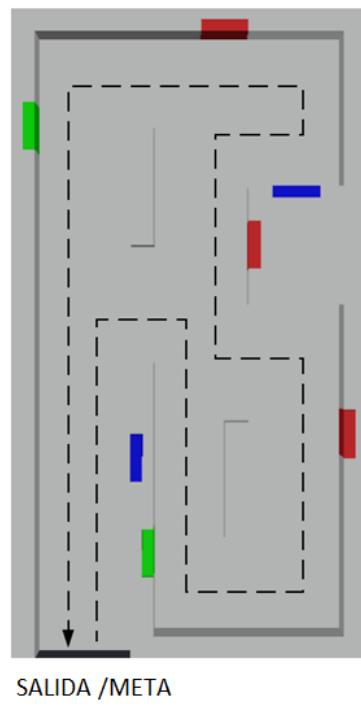


Figura N° 25: Recorrido esperado

En la figura 26 se puede observar el mapa obtenido del recorrido del robot.

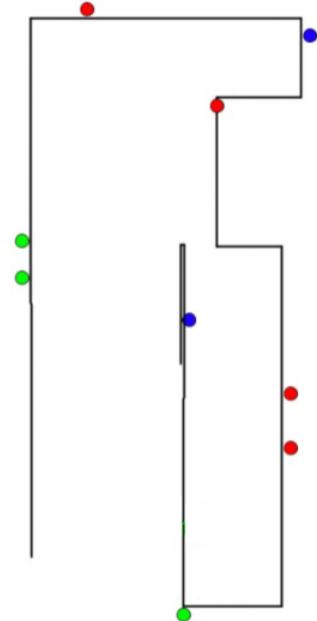


Figura N° 26: Resultado del mapa

Realizando un análisis de los resultados obtenidos podemos observar que el robot se comporta adecuadamente, el algoritmo de recorrido diseñado da prioridad a moverse hacia el frente y en caso de una bifurcación toma la decisión de desplazarse a mano derecha, adicionalmente en los puntos en que no encuentra las dos opciones anteriores realiza un giro de 180 grados para poder continuar el recorrido.

Pudimos observar que presenta un pequeño retraso en comparación con la ubicación de las minas, esto se debe al recorrido que hace el dato del color, iniciando su recepción desde la cámara, empleando el protocolo de transmisión UART hasta el microcontrolador ESP32, quien al final lo envía mediante bluetooth hasta el pc.

A pesar del retraso de la señal podemos observar que el resultado cumple con las expectativas, realiza un adecuado recorrido del laberinto, detecta el color de las minas y dibuja adecuadamente el recorrido.

## 9. Guía del usuario

Finalmente se decidió realizar una pequeña guía para que los usuarios, que pudieran llegar a utilizar el proyecto realizado, sepan al paso a paso necesario para lograr su funcionamiento por medio de una correcta inicialización y conexión tomando en cuenta ciertos parámetros ambientales. Las respectivas descripciones se realizan a continuación

### 1. Parámetros ambientales:

A continuación se describen variables externas, que afectan de manera directa en las lecturas y el desempeño del elemento y que hay que tener en cuenta para obtener un correcto funcionamiento del robot explorador.

#### a) Medidas del laberinto:

Para garantizar una correcta lectura de datos y un desplazamiento óptimo de nuestro robot explorador es necesario garantizar que nuestro laberinto tendrá el espacio necesario para que el elemento logre maniobrar de manera adecuada, para esto se estimó una medida mínima del ancho de cada corredor, la cual corresponde a 60 cm, y un radio de giro de al menos 40 cm. Las medidas fueron estimadas tomando en cuenta las dimensiones de nuestro dispositivo el cual tiene 17 cm de ancho por 23 cm de largo y los parámetros establecidos en la programación los cuales mantienen la pared derecha entre 20 cm y 30 cm y la frontal en 35 cm de distancia.

#### b) Iluminación:

Se recomienda mantener una iluminación constante en cada una de las etapas del laberinto para poder tener una buena identificación de la presencia de las minas y su respectivo color. Adicionalmente se sugiere realizar la respectiva calibración como se presento en la sección 5.3.1 antes de empezar el desplazamiento del dispositivo.

### 2. Procedimiento de encendido:

En este ítem se describen los pasos necesarios para lograr un funcionamiento completamente inalámbrico por parte del dispositivo.

#### a) Configuración FPGA

Para lograr utilizar la FPGA de manera remota es necesario conectarla tanto a la alimentación proveniente de la power bank como al computador de manera simultanea. Posteriormente subiremos el código realizado, proveniente de Vivado y cuyo repositorio se encuentra disponible en la sección 10.

Una vez se ha cargado el código lo desconectaremos solamente de la computadora, ya que mientras la FPGA se mantenga encendida, ésta se mantendrá operando de manera constante sin necesidad de volver a subir el código nuevamente, es por este motivo que se ha designado una power bank exclusiva para su funcionamiento.

**b) Configuración ESP32**

En caso de que no se hay configurado de manera previa el ESP32 se debe cargar el archivo proveniente de de arduino, disponible en la sección 10.

A diferencia de la FPGA este dispositivo almacena el código cuando se le quita la alimentación por lo cual no será necesario volver a realizar este procedimiento nuevamente.

**c) Alimentación total**

Una vez tenemos la FPGA configurada y encendida junto con el ESP32 configurado es necesario conectar la power bank restante para lo cual bastará con conectar con realizar la conexión en el puerto usb, esta realizara la alimentación a los demás periféricos utilizados en la elaboración del proyecto y nuestro explorador empezará a avanzar de manera inmediata y automática al depositarlo en el laberinto.

**3. Visualización de resultados**

Para poder visualizar las lecturas realizadas por el explorador será necesario realizar los siguientes pasos:

**▪ Configuración Bluetooth**

Es necesario que el computador tenga 2 entradas de Bluetooth, generalmente vendrá equipado con una y la otra la deberemos configurar como se vio en la sección 5.3.2 configurándolo como máster y con ayuda de un adaptador serial para el computador.

Una de estas entradas estará destinada a la visualización de la imagen y la otra, que viene desde el ESP32 servirá para visualizar el mapa.

**▪ Inicialización Processing**

Finalmente inicializamos los dos programas diseñados en processing los cuales nos permitirán la visualización del mapa y las imágenes recibidas en cámara, ambos programas están disponibles en la sección 10.

## 10. GitHub

Cada uno de los códigos realizados está disponible en el repositorio de GitHub el cual es accesible por medio del siguiente link: <https://github.com/FerdyLarrotta/Explorer>.

## 11. Conclusiones

Para lograr un correcto desarrollo es necesario seguir con un proceso de co-diseño como el propuesto en la figura 27, ya que permite comprender las partes de las que constará el proyecto por medio del desglose de cada una de sus componentes y un proceso de síntesis e integración organizada y debidamente documentada. Las características del diseño deben estar estrictamente sujetas a las limitaciones físicas o lógicas de los subcomponentes, ya que en caso de sobrepasarlos se puede ejecutar de forma incorrecta o no cumplir con los objetivos propuestos, llegando a ocasionar daños en los elementos.

Además se rescata el carácter iterativo del proceso de diseño, en donde al realizar la validación es indispensable notar las cosas que no funcionan adecuadamente y rediseñar los módulos correspondientes para un correcto funcionamiento. En este caso fue necesario rediseñar parte del software agregando el estado LEFT\_C (ver sección 5.2.1) para el correcto desplazamiento del carro.

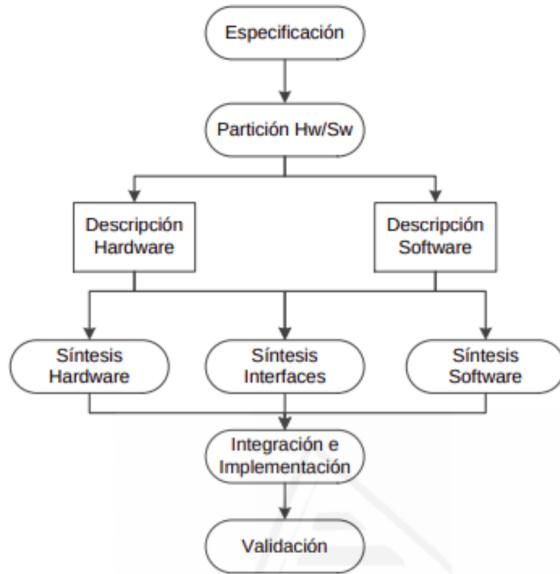


Figura N° 27: Diagrama seguido para el desarrollo del proyecto.

## Referencias

- [1] OV7670/OV7171 cmos VGA (640x480) cameraChip with OmniPixel Technology, OmniVision. Available on: <https://www.sigmaelectronica.net/wp-content/uploads/2018/04/OV7670-pdf.pdf>
- [2] UART, Serial Port, RS-232 Interface, Nandland. Available on <https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html>
- [3] Unal, proyecto final Digital i,Máxima memoria RAM. Available on <https://github.com/unal-edigital1-2019-2/work04-proyecto-final-grupo-05-1/blob/master/docs/README.md>
- [4] OV7670/OV7171 cmos VGA (640x480) cameraChip implementation guide, OmniVision. Available on: <http://www.haoyuelectronics.com/Attachment/OV7670%20+20AL422B%28FIFO%29%20Camera%20Module%28V2.0%29/OV7670%20Implementation%20Guide%20%28V1.0%29.pdf>
- [5] SCCB\_Arduino.c, muhammadyaseen. 2019. Available on <https://gist.github.com/muhammadyaseen/75490348a4644dcbc70f>
- [6] Prometec, EL MÓDULO BLUETOOTH HC-05, tutoriales. Available on <https://www.prometec.net/bt-hc05/>