UNIVERSIDAD DE MÁLAGA

E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD DE MÁLAGA

TRABAJO FIN DE MÁSTER

Sistema de Visión Configurable

MÁSTER EN SISTEMAS ELECTRÓNICOS PARA ENTORNOS INTELIGENTES

REALIZADO POR:

Manuel Lorente Almán

DIRIGIDO POR:

Martín González García

Departamento de Tecnología Electrónica
Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad de Málaga

Málaga, 2021

RESUMEN

El proyecto trata sobre el diseño de un sistema de captura de imágenes mediante tecnología de lógica programable (FPGA). Se utiliza el MT9V111 como sensor de imagen, una FPGA de la familia Artix-7 de Xilinx como controlador, comunicación USB mediante un conversor basado en FT245, y un software escrito en C# como interfaz con el usuario. Además, el conjunto es capaz de realizar detección de rostros posicionados frente a la cámara.

"¿Cuánto vales tú como persona? (C+H)xA.
Conocimiento, habilidad y actitud. La C y la H suman, pero la A multiplica."-Victor Kuppers

ABSTRACT

This project consists in the design of video capture system made with programmable logic technology (FPGA). The MT9V111 module is used as image sensor, a FPGA from Xilinx's Artix-7 family as system controller, USB communication through a FT245-based converter, and a software application written in C# as human-machine interface. The system has the capability to recognize faces appearing in the image acquired.

"Imagination is more important than knowledge. For knowledge is limited to all we now know and understand, while imagination embraces the entire world, and all there ever will be to know and understand."-Albert Einstein

Índice de contenido

1 ESPECIFICACIÓN DE REQUISITOS Y CASOS DE USO

1.1 Introducción

El objetivo de esta sección es capturar, analizar, y definir, las características y necesidades de alto nivel del sistema de visión configurable. Se centrará en describir las expectativas de cada una de las partes del proyecto y de los usuarios finales y por qué estas necesidades existen. Los detalles de cómo el sistema cumple estas necesidades se detallan en los casos de uso y en las especificaciones.

1.2 Objetivos

La finalidad principal de este trabajo fin de máster es el del diseño hardware de una videocámara, a partir de un sensor de imagen, y de una aplicación software para la captura de imágenes. Este software debe, además, efectuar la detección de varios rostros en tiempo real, marcando un recuadro sobre ellos para permitir su identificación.

1.3 Directivas del proyecto

1.3.1 Oportunidad de negocio

Este sistema permite al usuario estar informado del número de personas que existen frente al sensor de imagen del sistema, lo cual puede ser útil para un gran número de aplicaciones, entre las que podríamos destacar un sistema de seguridad, o incluso la identificación de personas en una zona determinada para servicios de acceso en hoteles o aeropuertos para así informar a los clientes de la ocupación del lugar.

1.3.2 Descripción del problema

El problema de	Cuantificar el número de personas en un lugar determinado.
Afecta a	Usuarios del sistema.
Lo cual tiene como impacto	Desconocer el número de personas en una localización.
Una solución satisfactoria sería	Implementar el sistema en zonas de vigilancia o control de acceso.

Tabla 1: Descripción del problema

1.3.3 Descripción del producto

Para	Cualquier hotel, aeropuerto, empresa.
Los cuales	Tengan la necesidad de detectar rostros en una zona concreta.
VICON	Es un sistema hardware-software.
Que	Detecta caras en tiempo real.
Frente a	Otras videocámaras con detección facial.
Nuestro producto	Es modular y con código abierto, por lo que la facilidad de integración es mayor.

Tabla 2: Descripción del producto

1.4 Descripción de participantes y usuarios

1.4.1 Resumen de los participantes

Nombre	Representa	Rol
Tutor del proyecto	Departamento de Ingeniería Electrónica de la UMA.	Es el participante principal. Encargado de establecer la mayoría de los requisitos del sistema. Encargado de aprobar los presupuestos. Encargado de validar el resultado del proyecto.
Alumno	Aspirante al título de máster.	Encargado de especificar, diseñar, planificar y gestionar los recursos del proyecto. Desarrolla el software y el hardware del sistema.

Tabla 3: Resumen de los participantes

1.4.2 Resumen y entorno de los usuarios

Nombre	Descripción	Participante
Tutor del proyecto	Encargado de comprobar la funcionalidad del sistema VICON.	Tutor del proyecto
Hoteles	Posibilidad de integrar el sistema en un futuro.	Usuario final

Tabla 4: Resumen de los usuarios

1.4.2.1 Entorno de los usuarios

Los usuarios del sistema serán expertos en el área, tanto el tutor del proyecto, como los miembros del tribunal de evaluación. El sistema proporciona una interfaz en ordenador mediante la cual podrá visualizarse el vídeo proveniente de la cámara conectada a la FPGA. Una sola persona será capaz de detener el vídeo en cualquier momento, y reanudarlo, así como activar y desactivar el reconocimiento de rostros en tiempo real.

Potencialmente se podría instalar el sistema en un hotel para detectar rostros que se acerquen y reconocer si son clientes o no.

La aplicación deberá ser totalmente portable y autocontenida para ser utilizada en una plataforma Windows 10 o posterior.

1.4.3 Perfiles de los participantes

1.4.3.1 Tutor del proyecto

Representante	Martín González
Tipo	Experto en el tema y usuario principal.
Responsabilidades	Revisar los requisitos y todas las funcionalidades del producto.
Criterio de Éxito	Poder visualizar una imagen en tiempo real, detectar rostros y almacenar una imagen en el PC, así como mostrar la tasa de imágenes procesadas por segundo.
Entregables	Memoria del TFM y archivos de código SW y de FPGA.
Comentarios	Será el encargado de probar la aplicación final y aprobar la finalización del proyecto.

Tabla 5: Perfil del tutor del proyecto

1.4.3.2 Alumno

Representante	Manuel Lorente
Tipo	Gestor y ejecutor del proyecto.
Responsabilidades	Encargado de especificar, diseñar, planificar y gestionar los recursos del Proyecto. Además, encargado del desarrollo, verificación y documentación del proyecto.
Criterio de Éxito	El proyecto finaliza con todos los requisitos satisfechos y verificados.
Entregables	Documento de requisitos y de arquitectura del sistema. Aplicación software portable y hardware del sistema funcional.
Comentarios	No

Tabla 6: Perfil del alumno

1.4.4 Perfiles de usuario

1.4.4.1 Tutor del proyecto

Descrito anteriormente.

1.4.4.2 Alumno

Descrito anteriormente.

1.4.4.3 Usuario final

Representante	Hoteles
Descripción	Hotel que implementa la solución.
Tipo	Utiliza el sistema.
Responsabilidades	Encargado de utilizar el sistema finalmente.
Criterio de Éxito	Puede detectar cuántas personas ocupan una zona determinada del hotel.
Comentarios	No.

Tabla 7: Perfil de usuario final

1.5 Requisitos

1.5.1 Diagrama general

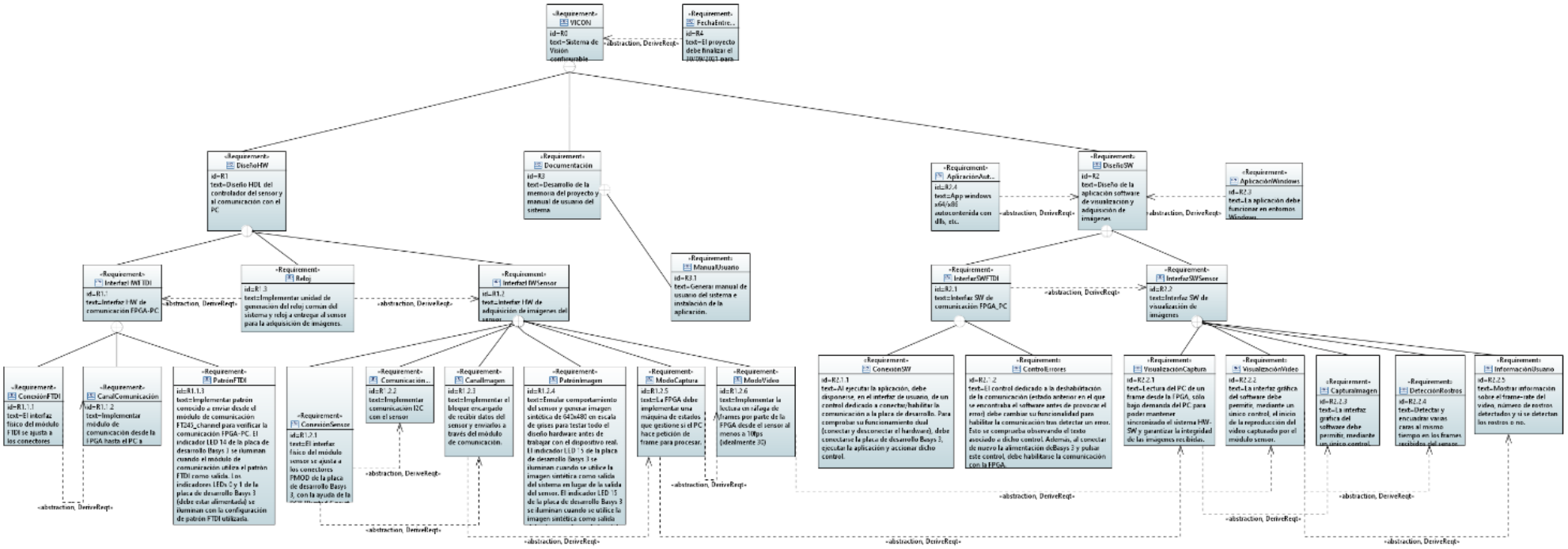


Figura 1: Diagrama de requisitios

1.5.2 Precedencia y prioridad

ID	Nombre	Prioridad	Precedencia
R1	DiseñoHW	Fundamental	
R1.1	InterfazHWFTDI	Fundamental	R1.3
R1.1.1	ConexionFTDI	Fundamental	
R1.1.2	CanalComunicacion	Fundamental	R1.1.1
R1.1.3	PatronFTDI	Fundamental	
R1.2	InterfazHWSensor	Fundamental	R1.3
R1.2.1	ConexionSensor	Fundamental	
R1.2.2	ComunicacionSensor	Opcional	R1.2.1
R1.2.3	CanalImagen	Fundamental	R1.2.1
R1.2.4	PatronImagen	Fundamental	
R1.2.5	ModoCaptura	Fundamental	R.1.2.3
R1.2.6	ModoVideo	Fundamental	R1.2.5
R1.3	Reloj	Fundamental	
R2	DiseñoSW	Fundamental	R2.3, R2.4
R2.1	InterfazSWFTDI	Fundamental	
R2.1.1	ConexionSW	Fundamental	
R2.1.2	ControlErrores	Fundamental	
R2.2	InterfazSWSensor	Fundamental	R2.1
R2.2.1	VisualizacionCaptura	Fundamental	R.1.2.5
R2.2.2	VisualizacionVideo	Fundamental	R1.2.6
R2.2.3	CapturaImagen	Fundamental	R2.2.1
R2.2.4	DeteccionRostros	Fundamental	R2.2.2
R2.2.5	InformacionUsuario	Fundamental	R2.2.2
R2.3	AplicacionWindows	Fundamental	
R2.4	AplicacionAutonoma	Fundamental	
R3	Documentacion	Fundamental	
R4	FechaEntrega	Fundamental	

Tabla 8: Requisitos del sistema

1.5.3 Requisitos funcionales

1.5.3.1 R1 DiseñoHW

Diseño RTL del controlador del sensor y la comunicación con el PC.

1.5.3.2 R1.1 InterfazHWFTDI

Interfaz HW de comunicación FPGA-PC.

1.5.3.3 R1.1.1 ConexiónFTDI

El interfaz físico del módulo FTDI se ajusta a los conectores PMOD de la placa de desarrollo BASYS 3.

1.5.3.4 R1.1.2 CanalComunicación

Implementar módulo de comunicación desde la FPGA hasta el PC a través del módulo UM232H-B.

1.5.3.5 R1.1.3 PatrónFTDI

Implementar patrón conocido a enviar desde el módulo de comunicación *FT245_CHANNEL* para verificar la comunicación FPGA-PC. El indicador LED 14 de la placa de desarrollo BASYS 3 se iluminan cuando el módulo de comunicación utiliza el patrón FTDI como salida. Los indicadores LEDs 0 y 1 de la placa de desarrollo BASYS 3 (debe estar alimentada) se iluminan con la configuración de patrón FTDI utilizada.

1.5.3.6 R1.2 InterfazHWSensor

Interfaz HW de adquisición de imágenes del sensor.

1.5.3.7 R1.2.1 ConexiónSensor

El interfaz físico del módulo sensor se ajusta a los conectores PMOD de la placa de desarrollo BASYS 3, con la ayuda de la PCB que hace de interfaz físico entre la placa y el móduo MT9V111.

1.5.3.8 R1.2.2 ComunicaciónSensor

Implementar comunicación I2C con el sensor.

1.5.3.9 R1.2.3 CanalImagen

Implementar el bloque encargado de recibir datos del sensor y enviarlos a través del módulo de comunicación.

1.5.3.10 R1.2.4 PatrónImagen

Emular comportamiento del sensor y generar imagen sintética de 640x480 en escala de grises para testar todo el diseño hardware antes de trabajar con el dispositivo real. El indicador LED 1 de la placa de desarrollo BASYS 3 se iluminan cuando se utilice la imagen sintética como salida del sistema en lugar de la salida del sensor.

1.5.3.11 R1.2.5 ModoCaptura

La FPGA debe implementar una máquina de estados que gestione si el PC hace petición de imagen para procesar.

1.5.3.12 R1.2.6 ModoVideo

Implementar la lectura en ráfaga de imágenes por parte de la FPGA desde el sensor al menos a 10 FPS (idealmente 30).

1.5.3.13 R1.3 Reloj

Implementar unidad de generación del reloj común del sistema y reloj a entregar al sensor para la adquisición de imágenes.

1.5.3.14 R2 DiseñoSW

Diseño de la aplicación software de visualización y adquisición de imágenes.

1.5.3.15 R2.1 InterfazSWFTDI

Interfaz SW de comunicación FPGA-PC.

1.5.3.16 R2.1.1 ConexiónSW

Al ejecutar la aplicación, debe disponerse, en el interfaz de usuario, de un control dedicado a conectar/habilitar la comunicación a la placa de desarrollo. Para comprobar su funcionamiento dual (conectar y desconectar el hardware), debe conectarse la placa de desarrollo, ejecutar la aplicación y accionar dicho control.

1.5.3.17 R2.2 InterfazSWSensor

Interfaz SW de visualización de imágenes.

1.5.3.18 R2.2.1 VisualizaciónCaptura

Lectura del PC de una imagen desde la FPGA, sólo bajo demanda del PC para poder mantener sincronizado el sistema HW-SW y garantizar la integridad de las imágenes recibidas.

1.5.3.19 R2.2.2 VisualizaciónVideo

La interfaz gráfica del software debe permitir, mediante un único control, el inicio de la reproducción del vídeo capturado por el módulo sensor.

1.5.3.20 R2.2.3 CapturaImagen

La interfaz gráfica del software debe permitir, mediante un único control, guardar una imagen capturada por el módulo sensor en el PC.

1.5.3.21 R2.2.4 DetecciónRostros

La interfaz gráfica del software debe permitir, mediante un único control, detectar y encuadrar varias caras al mismo tiempo en las imágenes recibidas del sensor, además del número de rostros. Además, debe existir la posibilidad de deshabilitarlo.

1.5.3.22 R2.2.5 InformaciónUsuario

La interfaz gráfica del software debe mostrar información sobre la tasa de imágenes por segundo del video, número de rostros detectados y si se detectan los rostros o no.

1.5.3.23 R2.3 AplicaciónWindows

La aplicación debe funcionar en entornos Windows 10 o posterior.

1.5.3.24 R2.4 AplicaciónAutónoma

La aplicación software debe funcionar en Windows 10 o posterior, arquitecturas x64/x86, y debe ser autocontenida con DLLs, y referencias externas.

1.5.4 Requisitos no funcionales

1.5.4.1 R2.1.2 ControlErrores

El control dedicado a la deshabilitación de la comunicación debe cambiar su funcionalidad para habilitar la comunicación tras detectar un error.

1.5.4.2 R3 Documentación

Desarrollo de la memoria del proyecto.

1.5.4.3 R4 FechaEntrega

El proyecto debe finalizar el 30/09/2021 para poder entregar a tiempo toda la documentación.

1.6 Casos de uso

1.6.1 Actores del sistema

Nombre	Descripción
Usuario	Persona que utiliza el sistema.
Sistema	Diseño HW-SW del sistema de visión configurable.

Tabla 9: Actores del sistema

1.6.2 Diagrama general

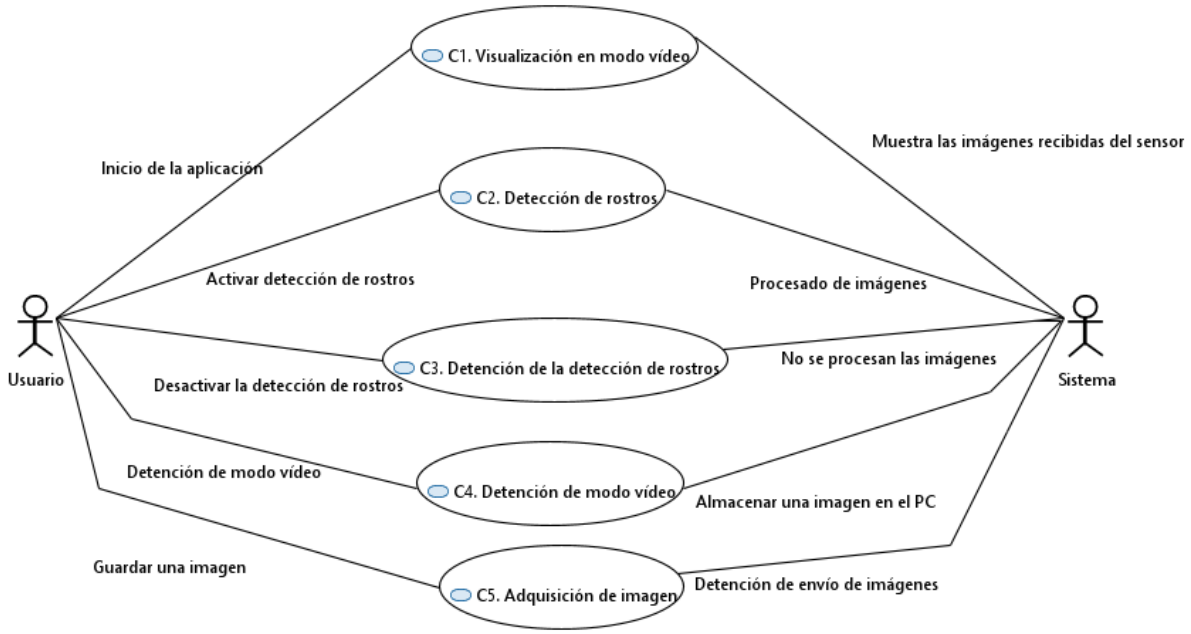


Figura 2: Diagrama de casos de uso

1.6.3 Descripción textual de los casos de uso

1.6.3.1 C1. Visualización en modo vídeo

1.6.3.1.1 Contexto de uso

Por defecto se visualiza el vídeo que proviene de la cámara en tiempo real.

1.6.3.1.2 Actor Principal

Sistema.

1.6.3.1.3 Participantes y Objetivos

Participante	Objetivo
Sistema	Adquirir imágenes y mostrarlas por pantalla.

Tabla 10: Participantes caso de uso C1

1.6.3.1.4 Pre-Condiciones

Aplicación SW Vicon instalada, sistema conectado al PC mediante el módulo UM232H-B, y FPGA encendida con el módulo sensor conectado.

1.6.3.1.5 Garantías mínimas

El vídeo se muestra al menos a 10 FPS.

1.6.3.1.6 Escenario de éxito principal

El PC solicita imágenes a la FPGA la cual inicia la adquisición y envía los datos correctamente. El PC procesa las imágenes y detecta caras en caso de estar habilitada la detección.

1.6.3.1.7 Escenario secundario 1

No se muestra imagen por pantalla.

1.6.3.1.8 Escenario secundario 2

Las imágenes llegan corruptas por problemas de transmisión.

1.6.3.2 C2. Detección de rostros

1.6.3.2.1 Contexto de uso

El usuario habilita la detección de rostros y en las imágenes comienzan a detectarse.

1.6.3.2.2 Actor Principal

Usuario.

1.6.3.2.3 Participantes y Objetivos

Participante	Objetivo
Sistema	Adquirir imágenes y mostrarlas por pantalla con los rostros encuadrados.
Usuario	Habilitar la detección de rostros.

Tabla 11: Participantes caso de uso C2

1.6.3.2.4 Pre-Condiciones

Sistema en modo vídeo funciona adecuadamente.

1.6.3.2.5 Garantías mínimas

El vídeo se muestra al menos a 10 FPS, detectando todos los rostros.

1.6.3.2.6 Escenario de éxito principal

El sistema encuadra rostros situados a menos de 2 metros de la cámara.

1.6.3.2.7 Escenario secundario 1

No se detectan rostros a pesar de estar el modo habilitado.

1.6.3.3 C3. Detención de la detección de rostros

1.6.3.3.1 Contexto de uso

El usuario detiene la detección de rostros.

1.6.3.3.2 Actor Principal

Usuario.

1.6.3.3.3 Participantes y Objetivos

Participante	Objetivo
Sistema	Dejar de procesar las imágenes entrantes
Usuario	Ordenar al sistema la deshabilitación de la detección de rostros.

Tabla 12: Participantes caso de uso C3

1.6.3.3.4 Pre-Condiciones

Sistema en modo vídeo funciona adecuadamente detectando rostros.

1.6.3.3.5 Garantías mínimas

Se detiene la detección de rostros.

1.6.3.3.6 Escenario de éxito principal

El sistema responde de manera inmediata a la solicitud de detención de detección de rostros por parte del usuario.

1.6.3.3.7 Escenario secundario 1

La detección de rostros no se detiene a pesar de solicitarlo el usuario.

1.6.3.3.8 Escenario secundario 2

La detención del reconocimiento facial se demora en significativamente una vez que el usuario lo ha solicitado.

1.6.3.4 C4. Detención de modo vídeo

1.6.3.4.1 Contexto de uso

El usuario detiene la adquisición de imágenes en tiempo real.

1.6.3.4.2 Actor Principal

Usuario.

1.6.3.4.3 Participantes y Objetivos

Participante	Objetivo
Sistema	Detener el envío de imágenes.
Usuario	Ordenar al sistema la deshabilitación del modo de vídeo.

Tabla 13: Participantes caso de uso C4

1.6.3.4.4 Pre-Condiciones

Sistema en modo vídeo funciona adecuadamente.

1.6.3.4.5 Garantías mínimas

El vídeo se detiene.

1.6.3.4.6 Escenario de éxito principal

El vídeo deja de actualizarse cuando el usuario lo solicita.

1.6.3.4.7 Escenario secundario 1

El vídeo no se detiene a pesar de solicitarlo el usuario.

1.6.3.4.8 Escenario secundario 2

La detención de vídeo se demora en significativamente una vez que el usuario lo ha solicitado.

1.6.3.5 C5. Adquisición de imagen

1.6.3.5.1 Contexto de uso

El usuario adquiere una imagen y la guarda en el PC.

1.6.3.5.2 Actor Principal

Usuario.

1.6.3.5.3 Participantes y Objetivos

Participante	Objetivo
Sistema	Enviar una única imagen y almacenarla en el PC.
Usuario	Ordenar al sistema la adquisición de una imagen.

Tabla 14: Participantes caso de uso C5

1.6.3.5.4 Pre-Condiciones

Sistema en modo vídeo funciona adecuadamente.

1.6.3.5.5 Garantías mínimas

La imagen se guarda.

1.6.3.5.6 Escenario de éxito principal

La imagen se guarda en la ruta definida por el usuario.

1.6.3.5.7 Escenario secundario 1

La imagen no se guarda.

1.6.3.5.8 Escenario secundario 2

La imagen se guarda en un formato que no puede ser representada por un visor de imágenes común.

1.7 Plan de aceptación

Requisitos	Prueba	Descripción	Salida	Errores
R2.3 R2.4 R3.1	Aplicación	Instalar la aplicación en un ordenador con Windows 10 o posterior.	La aplicación se instala con éxito y funciona correctamente.	La aplicación falla, y no es posible completar la instalación La aplicación se instala, pero no puede ejecutarse debido a la necesidad de uso de librerías y componentes externos.
R1.1 R1.2 R2.1 R2.2	Tasa de video en escala de grises	Se ejecuta la aplicación y se conecta la FPGA al PC mediante el controlador FT245.	En la aplicación se visualiza vídeo procedente de la cámara en tiempo real. Se visualiza la tasa de FPS alcanzada.	La aplicación no muestra ningún tipo de imagen por pantalla No se consigue una tasa mínima de 10 FPS. El video recibido se pixela o entrecorta. No se muestra la tasa de fotogramas por segundo.
R2.2.1 R2.2.3	Petición de imagen	Mediante una orden, el sistema debe detener o reanudar el vídeo	El sistema detiene y reanuda sin problemas el vídeo.	El sistema no es capaz de reanudar o detener el vídeo. El sistema sólo detiene o reanuda el vídeo una vez, y no vuelve a hacerlo. El sistema detiene el vídeo, pero no lo reanuda con imágenes en tiempo real.
R2.2.4 R2.2.5	Detección de rostros	Se sitúan varias personas frente a la cámara. Mediante una orden, se debe activar o desactivar la detección de caras	En la interfaz se ven todos los rostros marcados mediante un recuadro. Se muestra la tasa de fotogramas por segundos.	No se detecta ningún rostro a una distancia máxima de 2 metros, pese a que existen varios frente al sensor, y buena iluminación. El sistema no puede activar o desactivar la detección, y se queda siempre en el mismo estado. Permite ejecutar el comando una sóla vez, pero no vuelve a responder. Los rostros no se marcan con un rectángulo completamente.
R2.2.3	Almacenamiento de datos	Mediante una orden en la interfaz, se almacena la image adquirida.	El sistema genera una imagen en formato *bmp.*	El sistema genera un archivo de imagen corrupto.

Tabla 15: Plan de aceptación

1.8 Arquitectura

1.8.1 Vista general

Desde un punto de vista funcional, el sistema completo requiere una serie de implementaciones que abarca, tanto un diseño hardware como un diseño software como se puede ver en la Figura 3. Podemos decir, entonces, que tanto el hardware como el software desarrollado, debe implementar una serie de funciones que satisfagan, una vez concluida la fase de diseño e implementación, los objetivos de las especificaciones técnicas anteriormente expuestas.

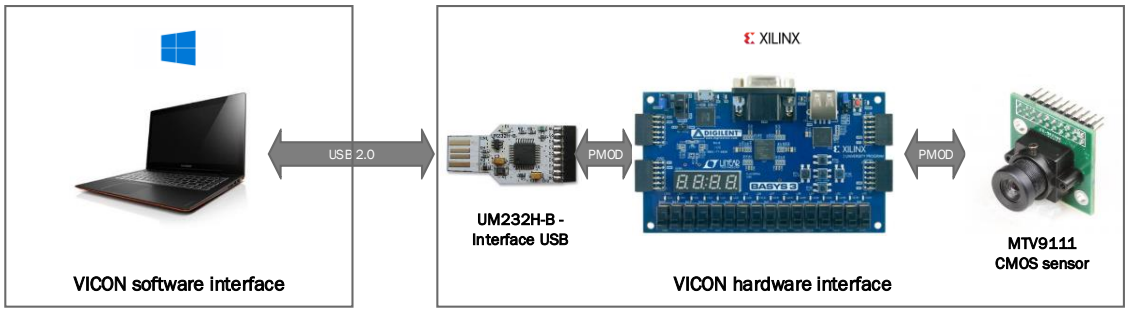


Figura 3: Vista general de la arquitectura

Los componentes empleados en el desarrollo del sistema son los siguientes:

- Módulo sensor MT9V111 de OnSemi.
- Controlador FT245 de FTDIchip para transferencia de datos.
- Placa de desarrollo BASYS 3, que integra una FPGA Artix 7 de Xilinx para la implementación del diseño hardware.
- Aplicación software VICON, diseñada para el sistema operativo Windows.
- Computador con sistema operativo Windows.

1.8.2 Funcionalidad software

El diseño software debe implementar un conjunto reducido de funcionalidades del sistema. Las funciones que debe realizar la aplicación desarrollada son:

- Gestión de un interfaz de usuario: al ejecutarse el software, debe mostrarse un interfaz, con el que el usuario pueda visualizar las imágenes de la videocámara. Las funciones que se deben incluir en este aspecto son las de responder a las acciones del usuario sobre los controles del interfaz.
- Reproducción de vídeo: a partir del diseño hardware implementado en el dispositivo lógico programable, el software debe implementar la función de lectura de imágenes de la videocámara. Esto incluye la conexión con el hardware diseñado, la lectura de datos de este y la capacidad de convertir dichos datos en una imagen, que el usuario pueda visionar en el interfaz de usuario.
- Detectar rostros en las imágenes capturadas: una vez reconstruida la imagen que se ha leído de la

FPGA, el software debe implementar la detección de rostros de la imagen, siempre que el usuario lo haya solicitado.

El diseño software consiste en una aplicación con interfaz de usuario que consta de:

- Un proceso principal, que controla los eventos producidos en los controles del interfaz de usuario.
- Un subproceso de lectura de datos y procesamiento de imágenes, activo únicamente cuando se solicita la reproducción de vídeo desde el interfaz de usuario.

1.8.2.1 Funcionalidad hardware

En este punto, se indican las funciones que debe realizar el hardware para el correcto funcionamiento de la videocámara.

- Leer datos del módulo MT9V111: esta función supone la captura de los datos que constituyen las imágenes capturadas por el sensor CMOS. Dicha captura, debe realizarse bajo petición de la aplicación software. Esta funcionalidad debe asegurar que las imágenes capturadas están completas, esto es, deben leerse todos los datos que componen una imagen, para lo que se harán uso de las señales de sincronización del módulo sensor.
- Envío de datos desde la FPGA al PC: el hardware debe implementar la funcionalidad requerida para el envío de datos de las imágenes hacia el PC (para que el diseño software pueda procesarlas) y a su vez ser capaz de gestionar la petición de imagen por parte del PC.
- Comunicación con la unidad de control del sensor: el diseño hardware debe implementar, opcionalmente, los módulos funcionales necesarios para comunicar la FPGA con la unidad de control del módulo MT9V111. De esta forma, se podrán aplicar configuraciones en el mismo. Para esto, se requiere un módulo funcional que ejerza de interfaz de comunicación con dicha unidad, ya que existe un protocolo y una temporización que cumplir.

En general, la arquitectura del sistema propuesto en este documento se basa en el módulo sensor de OnSemi y la FPGA Artix-7 integrada en la placa de desarrollo BASYS 3 que muestra la Figura 3:

- El controlador USB, permite la comunicación entre un computador y el dispositivo lógico programable, por lo que se trata de un elemento imprescindible en esta arquitectura.
- Implementación hardware, basada en un dispositivo lógico programable, que permita la captura de datos del módulo sensor CMOS y la configuración de este.

La arquitectura del diseño hardware basado en FPGA, consta de los siguientes subsistemas y módulos descritos en la Figura 4:

- Subsistema de captura de imágenes del sensor, bajo petición de la aplicación que ejecuta el computador.
- Módulo de comunicación entre el computador y la FPGA, que permita, a través del módulo USB, tanto la lectura de datos de imágenes desde el PC, como el envío de datos de configuración al módulo sensor CMOS opcionalmente.

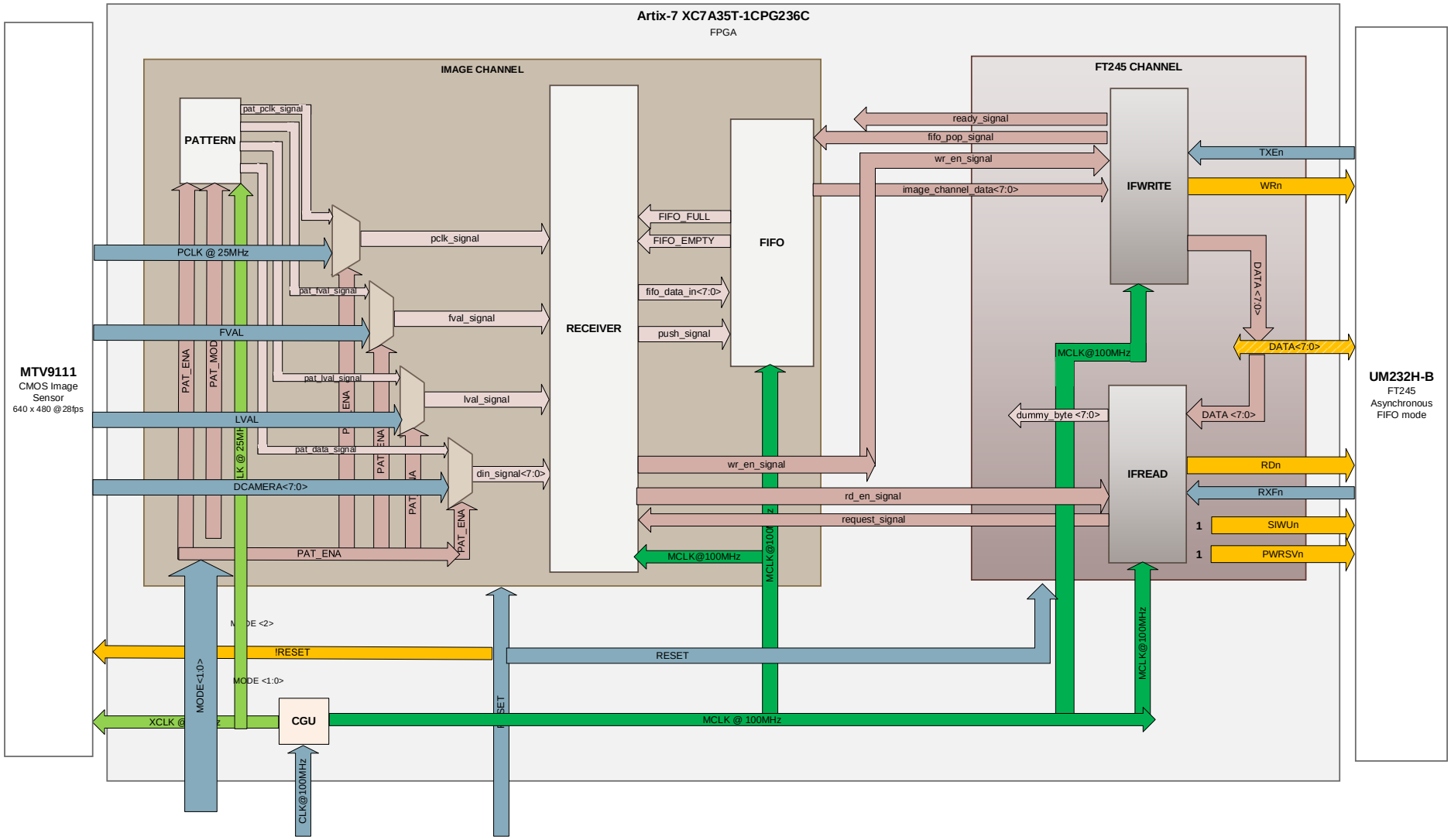


Figura 4: Arquitectura hardware

2 DISEÑO E IMPLEMENTACIÓN

2.1 Introducción

Esta sección recoge el desarrollo, tanto del diseño hardware, de la videocámara empleada para la captura de imágenes, como del diseño software, que hace uso de dicha videocámara para la representación de tales fotogramas y la detección de rostros.

Este texto describe ambos diseños siguiendo la estrategia “top-down”, ya que la arquitectura del sistema sigue una estructura jerárquica bien definida. Mediante este método, se exponen primeramente los subsistemas de niveles jerárquicos superiores, continuando con el descenso en la jerarquía, concluyendo con la descripción de los módulos (en el caso del diseño hardware) y funciones (diseño software) en los niveles inferiores.

En esta sección se presenta tanto el diseño como la implementación de este trabajo fin de máster. Ya que este proyecto abarca tanto un diseño hardware como uno software, se describirán por separado a lo largo de esta sección:

- Diseño hardware: esta fase está destinada a la descripción del sistema digital que se ha desarrollado para lograr, tanto la configuración del módulo sensor MT9V111 de OnSemi, como la captura de los fotogramas capturados por dicho módulo. Igualmente, se describe la interfaz con el dispositivo UM232-H-B que posibilita la transferencia de datos entre la placa de desarrollo BASYS 3 y un PC necesario tanto para poder realizar la lectura de los datos que conforman las distintas imágenes entregadas por el sensor CMOS, como para establecer una configuración de este.

  Para el desarrollo hardware, se empleará el lenguaje de descripción hardware VHDL.

- Diseño software: en esta sección se define el software implementado que permite establecer comunicación con la FPGA, realizar la lectura de fotogramas del sensor de imagen, configurar el módulo sensor a través de la placa de desarrollo BASYS 3 y realizar la detección de rostros en las imágenes capturadas mediante la librería OpenCV. Para la comunicación con el diseño hardware, se ha hecho uso de la API suministrada por FTDIchip. La implementación de los distintos hilos de procesamiento, así como el mecanismo de IPC, ha sido basada en el marco de trabajo .NET de Microsoft.

  El diseño software está basado, en su totalidad, en el lenguaje de programación C#.

2.1.1 FPGA

El diseño hardware está basado en la placa de desarrollo hardware BASYS 3, fabricada por Digilent. Está integra numerosos componentes siendo una FPGA Artix 7 de Xilinx el principal componente, además complementada con un chip de memoria flash QSPI, interfaces estándar como Ethernet, USB y VGA, conmutadores mecánicos, pulsadores y LED.

La alimentación para el funcionamiento de BASYS 3, es suministrada mediante el conector USB proporcionado por Digilent. Este conector USB además se emplea para configurar la FPGA y transferir datos a la placa de desarrollo. La Figura 5 muestra la placa BASYS 3.

Figura 5: Placa de evaluación BASYS 3 [2]

2.1.1.1 Conector PMOD

Debe destacarse la importancia de los conectores de expansión PMOD en este proyecto, mediante los cuales se podrá conectar el módulo sensor MT9V111. Estos son pequeños interfaces físicos de 6/12 pines que permiten ampliar la funcionalidad de los entornos de desarrollo basados en FPGA. La Figura 6 muestra el interfaz del conector PMOD.

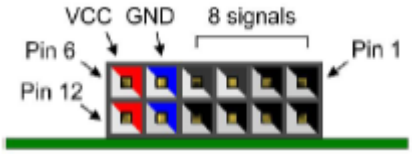


Figura 6: Conector PMOD [2]

2.1.1.2 Oscilador

Un elemento fundamental en cualquier diseño digital es la señal de reloj. En el caso de este proyecto, la placa BASYS 3 integra un oscilador de 100MHz que fijará la frecuencia de reloj a la que nuestro sistema trabajará como el de la Figura 7.

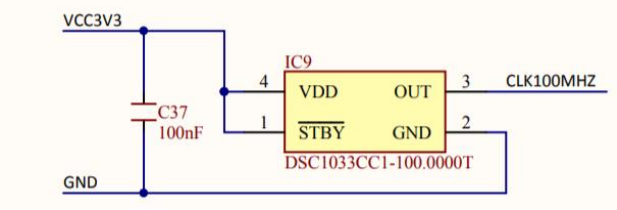


Figura 7: Generador de señal de reloj del sistema [2]

2.1.1.3 Conexión

La placa BASYS 3 permite la programación in-circuit que se realiza gracias a un conector de tipo microUSB, conectado a un controlador JTAG, que conecta directamente con la FPGA. Con esto, la conexión USB al ordenador permite la alimentación, programación y depuración del sistema al mismo tiempo. Adicionalmente, se configura el jumper JP1 en la posición QSPI, para cargar el bitstream almacenado en la memoria SPI cuando se pulse el botón de programación.

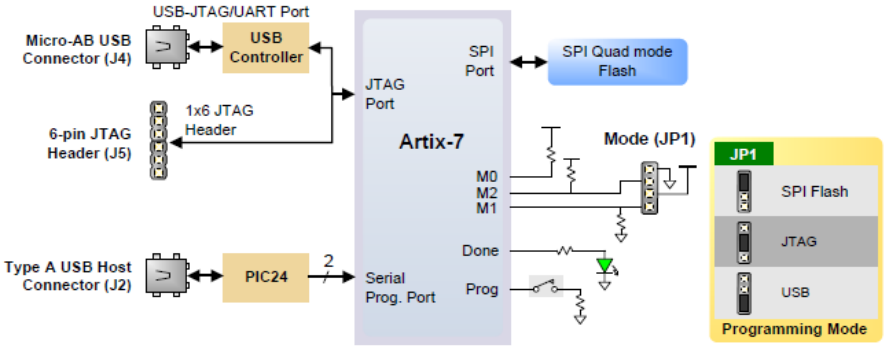


Figura 8: Esquema de conexión USB a la placa BASYS 3 [2]

2.1.2 Interfaz USB

Como interfaz entre la placa BASYS 3, y el software de la aplicación, se utilizará el módulo UM232-H-B-WE como el de la Figura 9, basado en el chip FT232H, del fabricante FTDIchip. El uso de este módulo viene motivado por su compatibilidad con los conectores PMOD de la placa, así como una salida de 5V que puede servir de alimentación a la FPGA, pudiendo hacer un sistema portable con una única entrada USB.

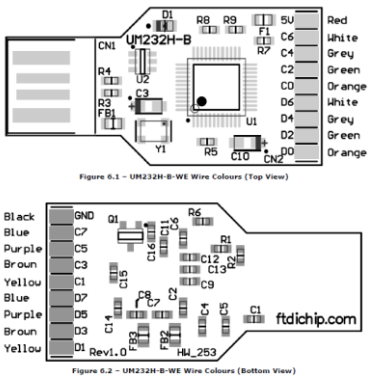


Figura 9: Esquema del módulo FTDI UM232H-B [3]

Este módulo FT232H proporciona la interfaz USB 2.0 y distintos protocolos de comunicación, además de incluir búferes de datos de transmisión y recepción de 1KB. En este proyecto se utilizará el módulo configurado como interfaz asíncrona FIFO 245 capaz de transmitir a una tasa de 8MB/s.

Para configurar el módulo en sus distintos modos de operación, el fabricante proporciona una herramienta en su página web (https://ftdichip.com/utilities/).

2.1.3 Sensor de imagen

El dispositivo principal, sobre el que se basa el diseño hardware de la videocámara, consiste en un sensor de imagen CMOS de OnSemi como el de la Figura 10, integrado sobre una placa de circuito impreso por Micron, comercialmente conocido como MT9V111. Es un SoC con diversos parámetros configurables mediante I2C, como color, resolución entre otros, siendo YUV 4:2:2 el formato de salida por defecto del módulo. Además, cuenta con una lente con distancia focal fija de 6mm, y apertura constante de F1.8.

Figura 10: Módulo sensor MT9V111

Este módulo opera en *Rolling Shutter*, y ofrece varios tipos de formato de imagen, tanto en resolución como en formato. A continuación, se listan los formatos y resoluciones disponibles:

- Resoluciones:
  - VGA 640x480 píxeles.
  - CIF 352x287.
  - QVGA 320x240.
- Formatos:
  - YCbCr
  - YUV 422
  - RGB/BGR 565
  - RGB/BGR 555
  - RGB/BGR 444

Además, la tasa de vídeo está directamente relacionada con la frecuencia de entrada del sistema, funciona a 12MHz y 12 FPS por defecto, pero si se aumenta la frecuencia de entrada a 27MHz, haciendo uso de la señal

de entrada *SCLK* de la Figura 11, podrían alcanzarse los 90 FPS con una resolución QVGA. En el caso de nuestro TFM, trabajaremos con la resolución VGA y a 25MHz.

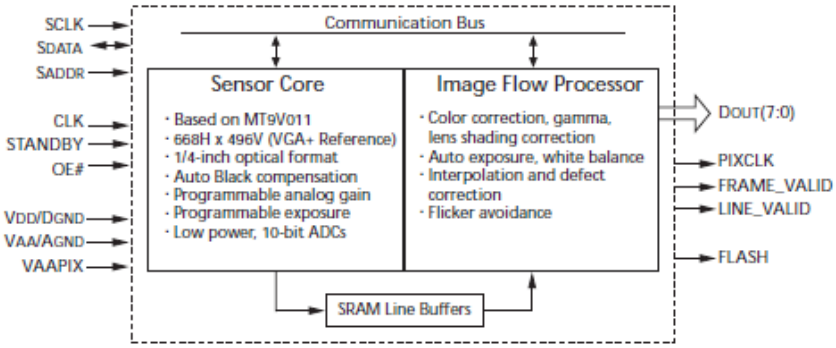


Figura 11: Arquitectura interna del módulo sensor MT9V111 [1]

Las imágenes se transmiten byte a byte en paralelo, controladas con las señales *FRAME_VALID* y *LINE_VALID* que controlan, respectivamente, el comienzo y final de cada fotograma, y el comienzo y final de cada línea que lo compone, acorde al cronograma de la Figura 12:

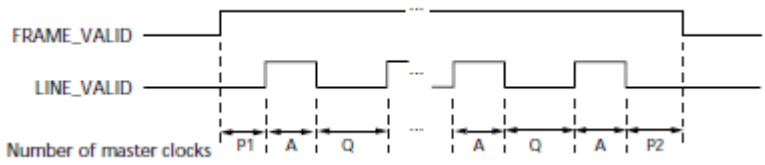


Figura 12: Cronograma de transmisión de imagen del módulo MT9V111 [1]

Con estos cronogramas, en los cuales se entrará en detalle más adelante, se desarrollarán los controladores correspondientes, para la recogida de datos válidos de imagen, objetivo principal del presente proyecto.

A pesar de que el bus de datos está compuesto de 10 líneas, sólo se emplean 8 (son las líneas de los índices comprendidos entre 9 y 2, ambas inclusive), en el desarrollo de este proyecto, pues son aquellas mediante las cual se transmiten los formatos YUV y RGB.

2.2 Implementación física

Como se comentó con anterioridad, el presente TFM se basa en un diseño conjunto hardware + software capaz de transmitir y mostrar vídeo en tiempo real desde un sensor de imagen mediante una aplicación en un ordenador personal. En concreto, el sistema debe cumplir el diagrama de bloques de la Figura 13.

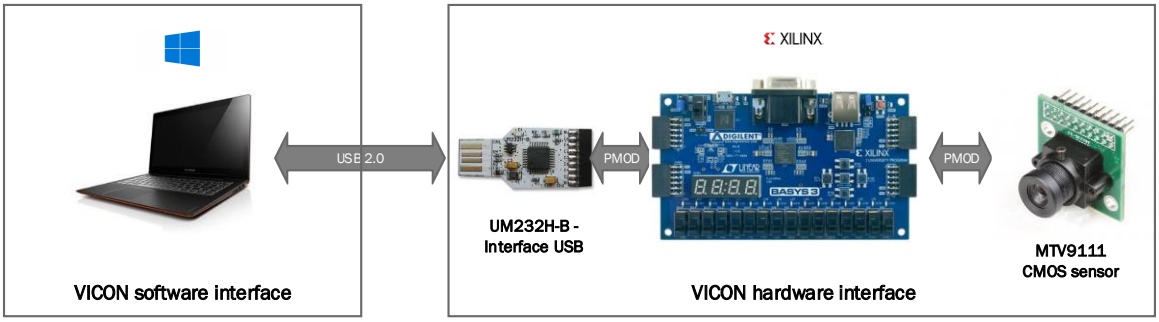


Figura 13: Implementación física del sistema

Donde se diferencian 3 partes:

- FPGA: Dispositivo central del sistema.
- VICON APP: Se trata del HMI con GUI con el que el usuario controlará algunos aspectos del sistema.
- MT9V111: Sensor de imagen CMOS. Es el periférico comercial mediante el cual se tomarán las capturas de imagen y vídeo.

2.3 Diseño Hardware

2.3.1 Introducción

En este capítulo de la descripción del diseño e implementación del proyecto VICON; se exponen todos los módulos, subsistemas y conexiones entre ellos, para el diseño de la videocámara, objeto principal de este trabajo fin de máster.

La implementación del diseño hardware, se ha realizado mediante el modelado de circuitos digitales, empleando el lenguaje de descripción hardware VHDL sobre el entorno de desarrollo Xilinx Vivado, en su versión de 2018.

2.3.2 Esquema general

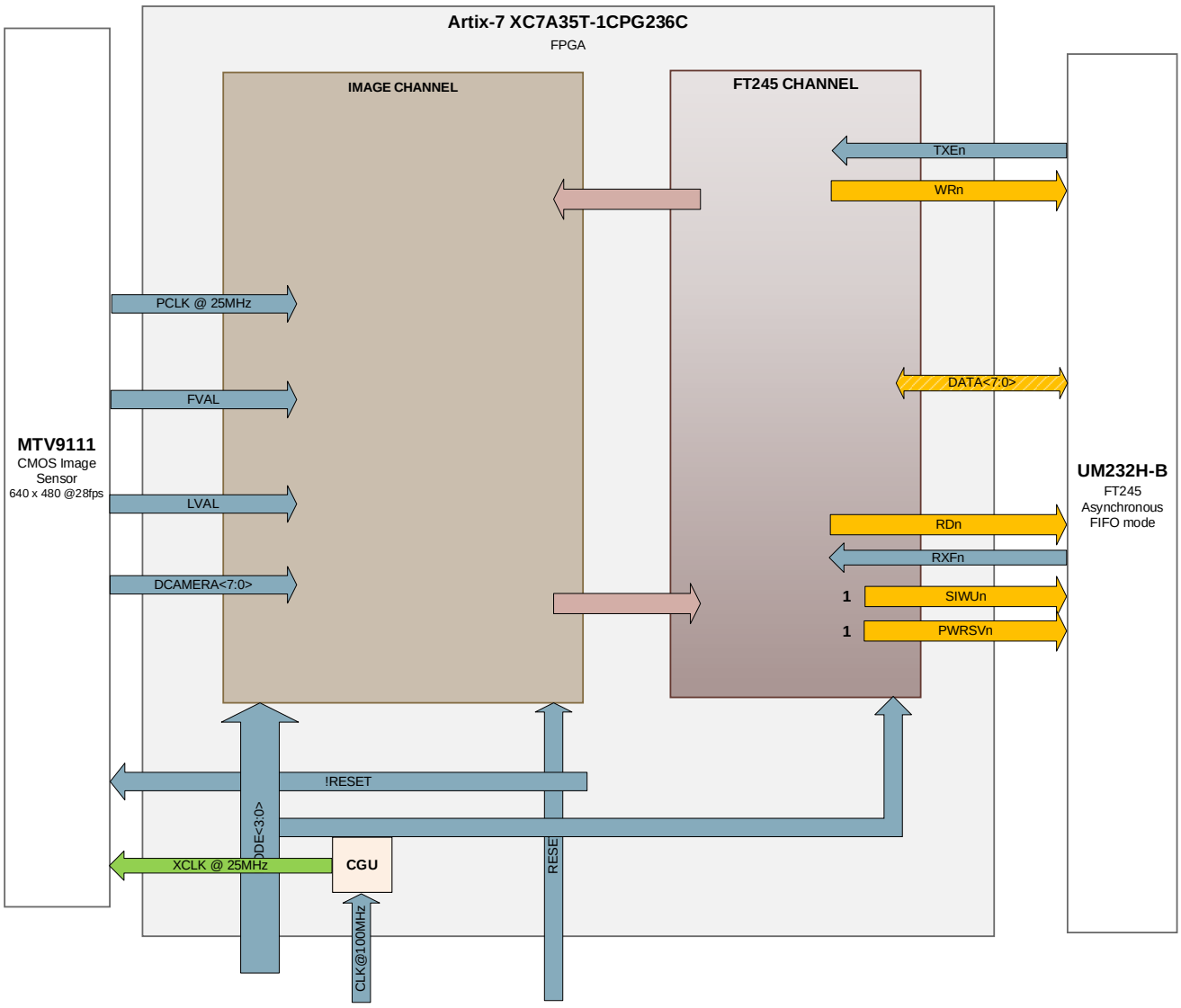


Figura 14: Arquitectura general del sistema hardware

2.3.2.1 Señales

Entrada/Salida

DATA<7:0>	Bus bidireccional de datos de 8 bits entre el PC y la FPGA. Envía imágenes al PC y recibe petición de imagen desde el PC a la FPGA

Entrada

CLK	Masterclock a 100MHz generado por la FPGA (el sistema debe ser síncrono a esta referencia)
RESET	Reset síncrono global del sistema a través de SW<15>
DCAMERA<7:0>	Datos que llegan desde el sensor de imagen
PCLK	Reloj de recuperación del sensor de imagen (copia del reloj XCLK)
FVAL	Inicio de fotograma
LVAL	Byte recibido corresponde a línea válida

SW<0>	Configuración del patrón de test	
	1: Patrón vertical	0: Patrón horizontal
SW<1>	Selector para salida del image channel	
	1: Patrón de test	0: Sensor de imagen
TXEn	Bandera que indica el estado de la FIFO TX del USB	
	1: No se pueden escribir datos	0: Se pueden escribir datos (si WR# = 0)
RXFn	Bandera que indica el estado de la FIFO RX del USB.	
	1: No se pueden leer datos	0: Se pueden leer datos (si RD#= 0)

Salida

LED<1:0>	Configuración del patrón de test	
	1: Patrón vertical	0: Patrón horizontal
LED<15>	Configuración de salida del sistema	
	1: Patrón de test	0: Sensor de imagen
XCLK	Señal de reloj a 25MHz hacia el sensor	
RDn	Bandera a la FIFO RX del USB para la lectura de datos	
	1: No leer datos	0: Leer datos
WRn	Bandera a la FIFO TX del USB para la escritura de datos	
	1: No escribir datos	0: Escribir datos
RESETN	Reset hacia el sensor (activo a nivel bajo)	
PWDn	Pin para activar el modo ahorro de energía del sensor de imagen	
	1: Operación normal	0: Modo ahorro de energía
SIWUn	Pin para despertar al host PC	
	1: Operación normal	0: Fuerza USB a modo sleep
PWRSVn	Pin para activar el modo ahorro de energía del módulo USB	
	1: Operación normal	0: Modo ahorro de energía

2.3.2.2 FT245 CHANNEL

En vista al esquema de la Figura 14, se puede ver cómo los datos de entrada del sensor pasan por el bloque *IMAGE_CHANNEL* para ser entregados a la interfaz *FT245_CHANNEL* y su posterior envío al PC. Este bloque es el encargado de gestionar la comunicación con el PC a través del dispositivo UM232-H.

Cabe destacar que en el bus de datos de entrada no interesa la información que contiene el dato, sino el envío de este. Por esto se decide obviar este dato, aunque el bus y la función de lectura esté totalmente habilitada para futuras ampliaciones o reutilizaciones del bloque.

2.3.2.3 IMAGE CHANNEL

Este bloque recibe diversas señales del resto de módulos, y gestiona tanto el *push*, como las habilitaciones de los periféricos. Además, gestiona la adquisición de imágenes, llenado de la una memoria interna para ajustar los anchos de banda de salida del sensor y escritura al PC, y permite testar el sistema con un módulo generador de imagen sintética de test.

2.3.2.4 CGU

Sencillamente, recibe el reloj *clk* del sistema, y conecta con la salida de la FPGA para proporcionar la señal de reloj al MT9V111.

2.3.3 Clock Generation Unit (CGU)

Este bloque es el encargado de generar la salida de 25MHz (Figura 15) que será entrada de reloj del módulo MT9V111. Consiste en un divisor de frecuencia mediante un contador binario de tal manera que la señal obtenida es una señal de tipo *tick*, pero que salta un ciclo de reloj gracias al contador binario.

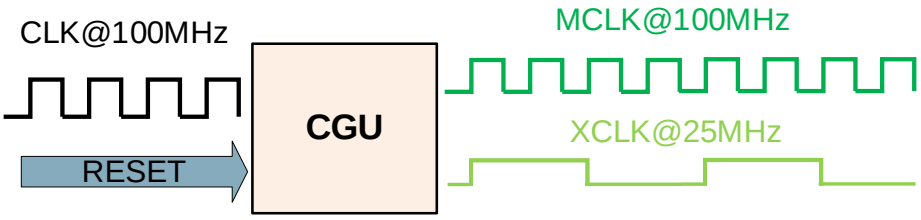


Figura 15: Diagrama de bloques *CGU*

2.3.4 FT245 channel

Es necesario implementar los controladores de lectura y escritura en el dispositivo UM232H-B siguiendo la temporización de lectura (Figura 16) y escritura en el módulo (Figura 17) indicada por el fabricante (Tabla 16):

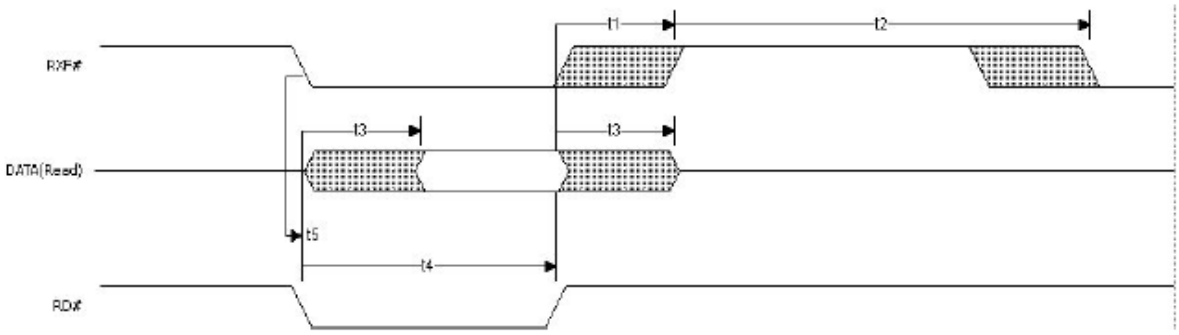


Figura 16: Cronograma lectura módulo UM232H-B en modo FT245 Async [3]

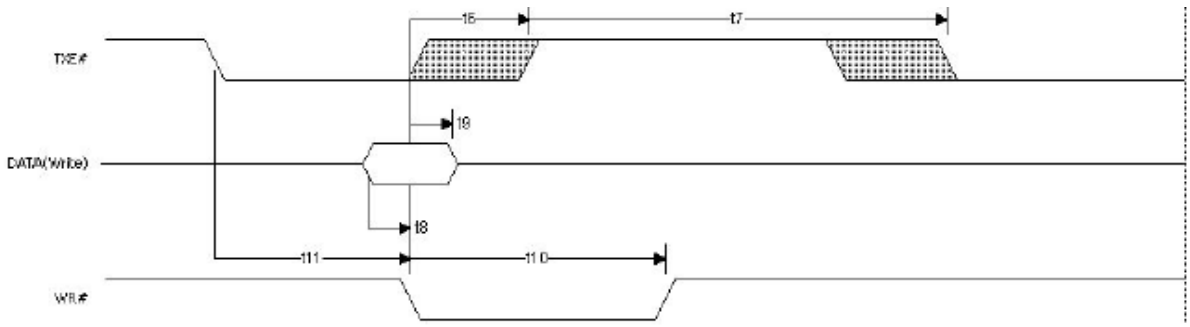


Figura 17: Cronograma de escritura módulo UM232H-B en modo FT245 Async [3]

Time	Description	Min	Max	Units
T1	RD# inactive to RXF#	1	14	ns
T2	RXF# inactive after RD# cycle	49		ns
T3	RD# to DATA	1	14	ns
T4	RD# active pulse width	30		ns
T5	RD# active after RXF#	0		ns
T6	WR# active to TXE# inactive	1	14	ns
T7	TXE# active to TXE# after WR# cycle	49		ns
T8	DATA to WR# active setup time	5		ns
T9	DATA hold time after WR# inactive	5		ns
T10	WR# active pulse width	30		ns
T11	WR# active after TXE#	0		ns

Tabla 16: Temporización lectura y escritura del UM232H-B en modo FT245 Async [3]

El diseño interno de este módulo a nivel de bloques es el indicado en la Figura 18:

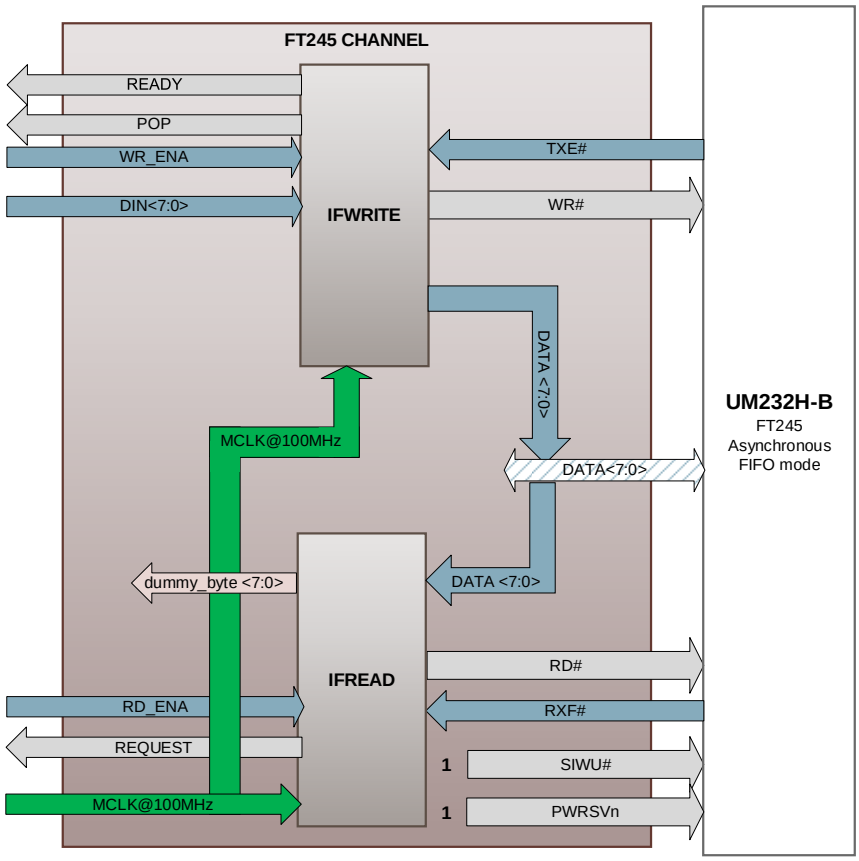


Figura 18: Diagrama de bloques *FT245_CHANNEL*

2.3.4.1 Comunicación FPGA-PC

Siendo *TXE#* la señal gobernada por el módulo FT245, siendo *WR#* la señal generada por sistema para la coordinación con el mismo, y teniendo en cuenta los datos, la secuencia seguida por el dispositivo para la transmisión de un dato desde la FPGA hacia el módulo está modelada con una FSM como la de la Figura 19 descrita a continuación:

1. En el estado inicial (*OFF*) del sistema, las señales *TXE#* y *WR#* están a nivel lógico '1'. Además, se activa la señal *READY* a nivel lógico '1' para indicar que la máquina de escritura está preparada para comenzar una transmisión. Cuando la señal de *WR_ENA* se activa a nivel lógico '1', indica que es va a iniciar una transferencia por lo que se pasa al estado de *IDLE.*
2. En estado de IDLE, las señales *TXE#* y *WR#* están a nivel lógico '1', y el sistema espera el flanco de bajada de la señal *TXE#* para escribir datos en el módulo.
   Cuando la señal *TXE#* se activa a nivel lógico '0', se pedirá un dato de entrada mediante la señal *POP*, y se establecerá la señal *WR#* a nivel '0' cuando se quiera que el dato en el bus sea escrito al dispositivo. Además, se debe garantizar que el dato está, al menos 5 ns antes del flanco de bajada de *WR#.* Se pasa al estado de *TX_DATA.*
3. Una vez se ha realizado el flanco de bajada de *WR#,* el dato debe permanecer al menos 5ns más en el bus para poder ser escrito con normalidad. En este estado se inicia un contador de 3 ciclos de sistema para garantizar ue la señal *WR#* permanece un mínimo de 30ns a nivel '0' para completar la correcta transmisión.
4. La señal *TXE#,* una vez vuelva a estado inactivo, permanecerá en este durante al menos 49ns antes de volver a estar disponible para transmitir un dato. En este punto, se volvería a *OFF* o *IDLE* dependiendo del estado de la señal de *WR_ENA* por si es necesaria una nueva transmisión.

Teniendo en cuenta el proceso de escritura de un dato con este protocolo, y que el tiempo de ciclo en la FPGA del sistema es de 10ns, se diseña una FSM tal que contiene los siguientes estados:

OFF	Estado de la máquina de escritura apagado. La salida *ready* permanece activa, indicando que el sistema no está transmitiendo, sino que está listo para ello; la salida *WRn* permanece inactiva, no se genera ningún *pop*, y el valor en el bus de datos se indica como cero.
IDLE	Cuando la máquina recibe la señal *WR_ENA*, este pasa a estado *IDLE*, en el que desactiva la salida *ready*, y espera que la señal *TXE#* del FT245 pase a estado activo para comenzar la transmisión. Se pide un dato mediante la señal *POP*, y se pone en el bus de datos el valor que haya en la salida de la FIFO en este momento. Este es el dato que se escribirá al FT245.
TX_DATA	En este punto, Se pone la señal *WR#* activa, y se desactiva la señal *pop*, de tal manera que queda de tipo *tick.* Se inicia un contador de 3 ciclos para garantizar que la señal *WR#* está los 30ns activa. Si la habilitación del sistema sigue activa, se vuelve al estado *IDLE,* a esperar otra transmisión. Si el sistema indica el apagado de la máquina de escritura, a través de la señal *WR_ENA* ésta volverá al estado *OFF.*

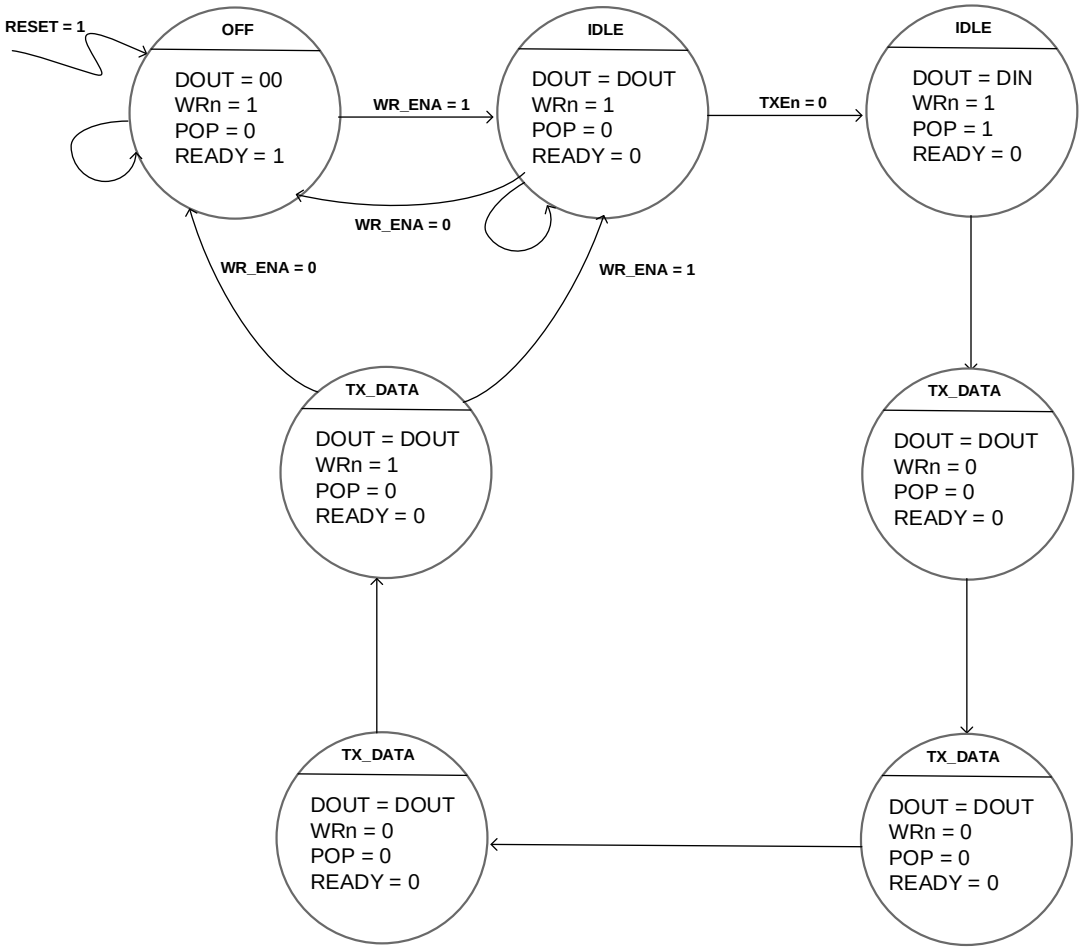


Figura 19: *FT245_WRITE* FSM

2.3.4.2 Comunicación PC-FPGA

Siendo *RXF#* la señal gobernada por el módulo FT245, siendo *RD#* la señal generada por sistema para la coordinación con el mismo, y teniendo en cuenta los datos, la secuencia seguida por el dispositivo para la transmisión de un dato desde el módulo hacia la FPGA está modelada con una FSM como la de la Figura 20 descrita a continuación:

1. En el estado inicial *OFF* del sistema, las señales *RXF#* y *RD#* están a nivel lógico '1'. Además, La señal *REQUEST* está a nivel lógico '0' para indicar que la máquina de lectura no ha recibido nada. Cuando la señal de *RD_ENA* se activa a nivel lógico '1', indica que es va a iniciar una lectura por lo que se pasa al estado de *IDLE*.
2. Cuando la señal *RXF#* se activa a nivel lógico '0', indica que es posible leer datos del módulo. En este momento se establecerá la señal *RD#* a nivel '0' cuando se quiera que el dato en el bus sea leído del dispositivo. Esta señal permanecerá al menos 30ns más en el bus para completar la correcta lectura.
3. Una vez la señal *RD#,* tras 30ns mínimo, vuelve a estado inactivo, la señal *RXF#* volverá a estado inactivo en un lapso de 1 a 14ns.
4. La señal *RD#,* una vez vuelva a estado inactivo, permanecerá en este durante al menos 49ns antes de comenzar otro ciclo de lectura.

Teniendo en cuenta el proceso de lectura de un dato con este protocolo, y que el tiempo de ciclo en la FPGA del sistema es de 10ns, se diseña una FSM tal que contiene los siguientes estados:

OFF — Estado de la máquina de lectura apagado. La salida *RD#* permanece inactiva, no se genera ninguna señal *request, y* el valor en el bus de datos se indica como cero, el dato no se recoge, pues no contiene información alguna.

IDLE — Cuando la máquina recibe la señal *RD_ENA*, este pasa a estado *IDLE*, en el que espera que la señal *RXF#* del FT245 pase a estado activo para comenzar la lectura, activando la señal *RD#.*

RX_DATA — En este paso, se activa la señal *request*, que indicará al sistema que se ha pedido el envío de un *fotograma* desde el PC. En este momento, el dato en el bus podría recogerse. No obstante, en el presente trabajo, el dato no contiene información relevante. Sólo interesa que el sistema reciba que existe un dato para leer, lo cual le indica que se ha solicitado un *fotograma.*

Se inicia un contador de 3 ciclos para garantizar que la *RD#* está los 30ns activa. Si la habilitación del sistema sigue activa, se vuelve al estado *IDLE*, a esperar otra transmisión. Si el sistema indica el apagado de la máquina de escritura, a través de la señal *RD_ENA* ésta volverá al estado *OFF*.

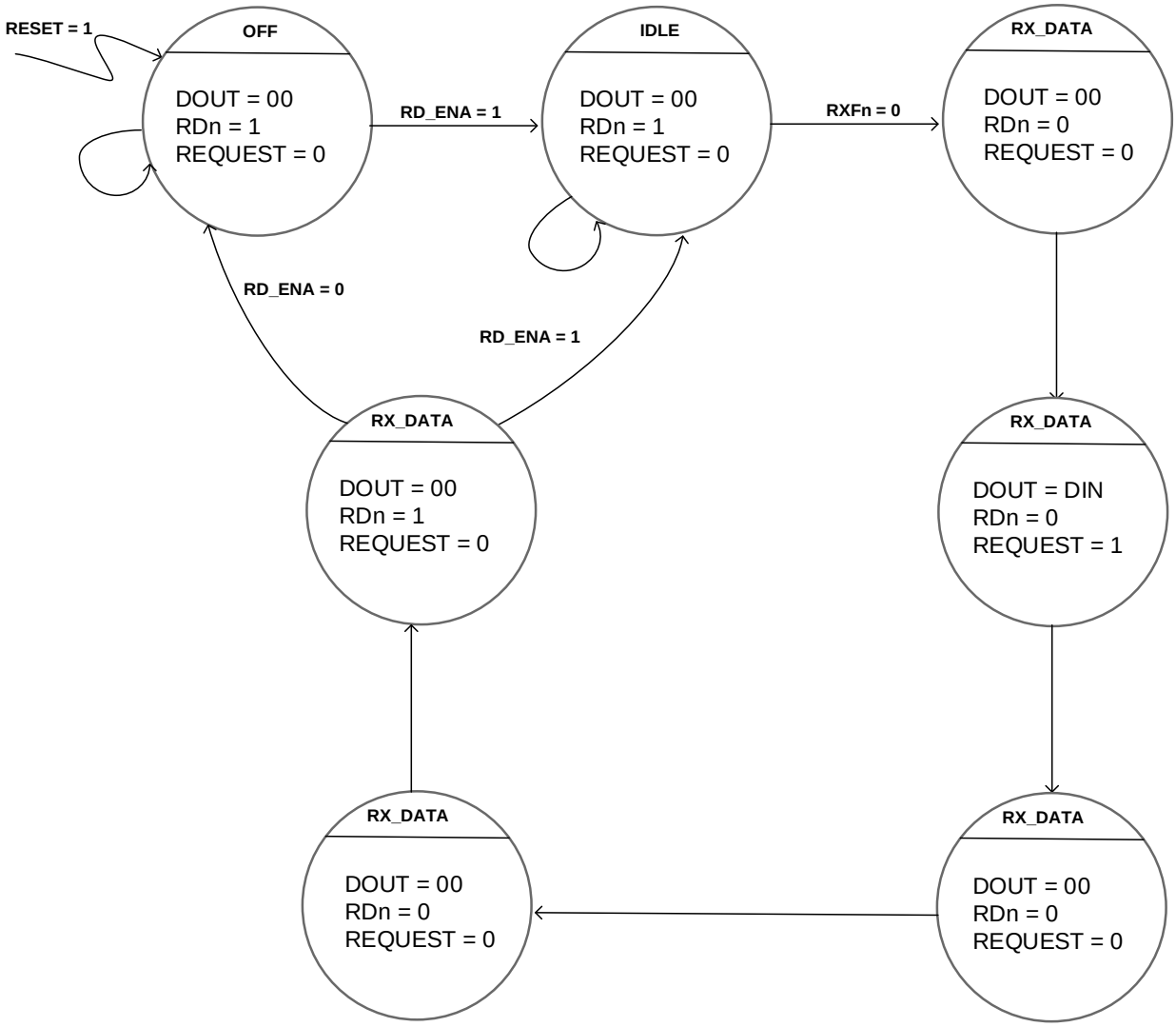


Figura 20: *FT245_READ* FSM

2.3.5 Image channel

Este bloque modelado en la Figura 21 recibe diversas señales del resto de módulos, y gestiona tanto la adquisición de la imagen, como las habilitaciones de los periféricos. Tiene como componente principal un bloque receptor que recoge los datos válidos y una FIFO para almacenar los bytes de píxeles antes de ser transmitidos de manera que no se pierda ninguno.

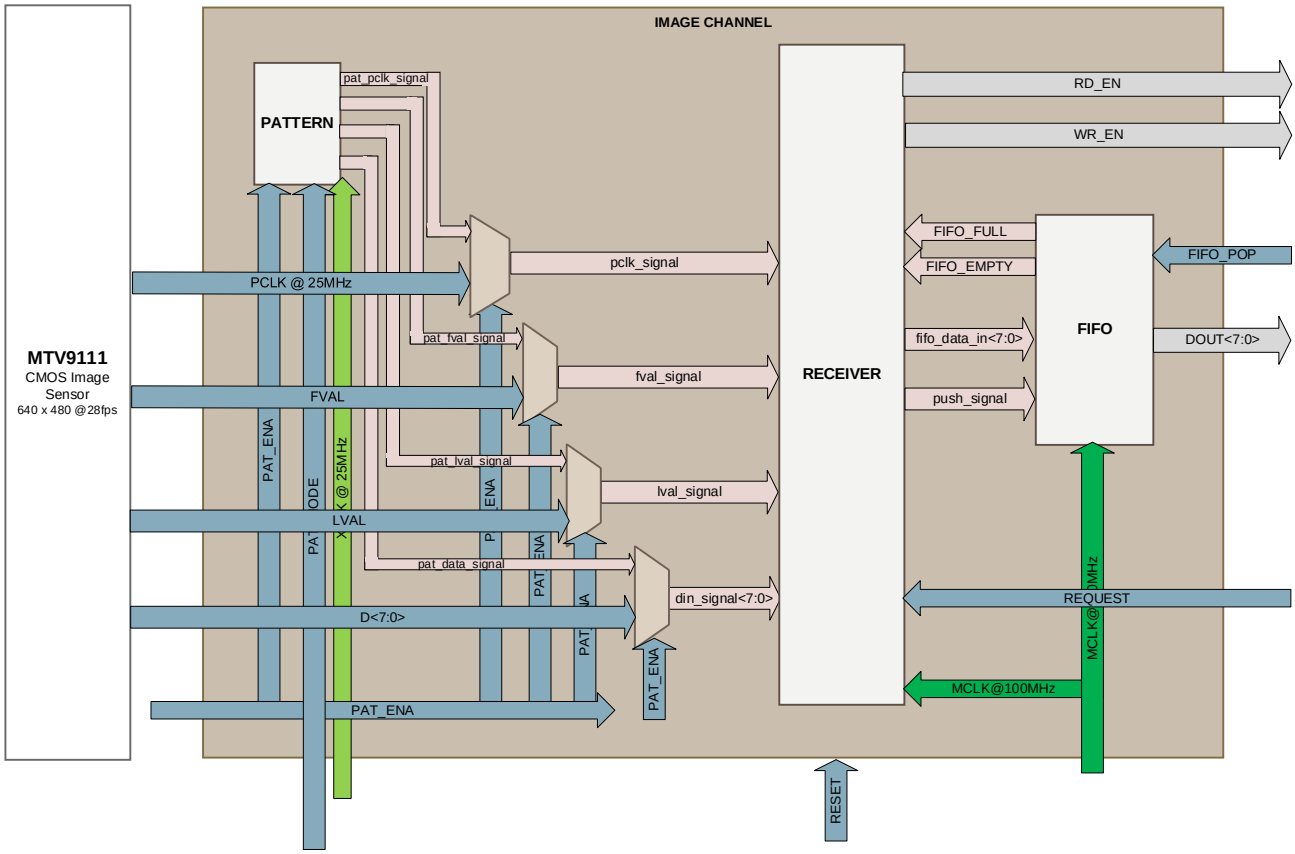


Figura 21: Diagrama de bloques *IMAGE_CHANNEL*

2.3.5.1 FIFO

Como la lectura de la imagen desde el sensor, es más rápida que la transmisión hacia el PC es necesario tener en el sistema una memoria RAM de tipo FIFO, que permita almacenar los datos provenientes del sensor de imagen, para posteriormente ser transmitidos al software sin problemas de integridad ni pérdida de información.

Se propone la utilización de una memoria basada en bloques de RAM dedicada, o BlockRAM. Este tipo de bloques están presentes físicamente en la FPGA.

El desarrollo de esta memoria se ha realizado parametrizado, es decir, es posible cambiar, tanto el ancho del bus de datos, como el de direcciones, haciendo la memoria versátil y totalmente configurable antes de ser sintetizada. En este caso, se ha determinado un ancho de bus de direcciones (B) de 16, y un ancho de bus de datos (W) de 8, teniendo con ello una capacidad total de:

2^{16} = 65536 palabras * 8 bits/palabra * 1 byte/8bits * 1kbyte/1024bytes = 64KB

Este valor ha sido elegido tras realizar una simulación de caso peor al modelo del sistema completo. En ella, se simuló un ciclo de escritura de dato desde la FPGA hacia el PC de 100ns, según los tiempos máximos que establecen los cronogramas y tiempos de espera anteriormente referenciados en

$$t_{tot} = t_6 + t_7 + t_{11} = 14ns + 49ns + 20ns = 83ns$$

Teniendo en cuenta que el sistema funciona con pasos de 10ns hay que ajustar la suma de tiempos:

$$t_{tot} = t_6 + t_7 + t_{11} = 20ns + 50ns + 20ns = 90ns$$

Aplicando un margen de seguridad del 10%:

$$t_{worst} = t_{tot} * 1.1 = 99ns \rightarrow 100ns$$

Se debe realizar una simulación completa del sistema con un tiempo de transferencia de 100ns. Se utiliza el generador de imagen sintética (*PATTERN*) para ver a través de ensayo y error con cuántos píxeles se llena la memoria sin poder ser transmitidos a tiempo al PC, provocando la consecuente pérdida de datos. Se comprueba que a nunca se almacenan 65536 palabras, por lo que se decide implementar el tamaño de memoria de 16 bits (64KB).

La lógica de control de esta memoria consiste en dos señales de entrada, *pop* y *push* que, respectivamente, indican cuando un dato debe ser liberado en el puerto de salida, y cuándo el dato en el bus de entrada debe ser almacenado. La memoria no permite la sobreescritura de datos, con el fin de garantizar la integridad de estos, y evitar pérdidas de información en condiciones límite. La lógica de estado de la FIFO (señales *empty y full*), dependen del valor del contador *count,* siendo *empty* activa cuando *count=0,* y siendo *full* activa cuando *count*=2^B.

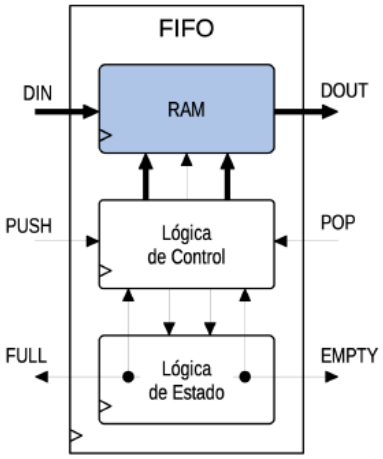


Figura 22: Diagrama de bloques memoria *FIFO*

2.3.5.2 RECEIVER

Este bloque se encarga de recibir las señales de control desde el módulo MT9V111, y proporcionar información útil al resto de módulos. El módulo MT9V111, realiza las actualizaciones de los valores en sus salidas en cada flanco de bajada del reloj *pclk.* Este reloj es una copia del reloj *xclk*, entregado por la FPGA, con un ligero retraso como se puede ver en la Figura 23:

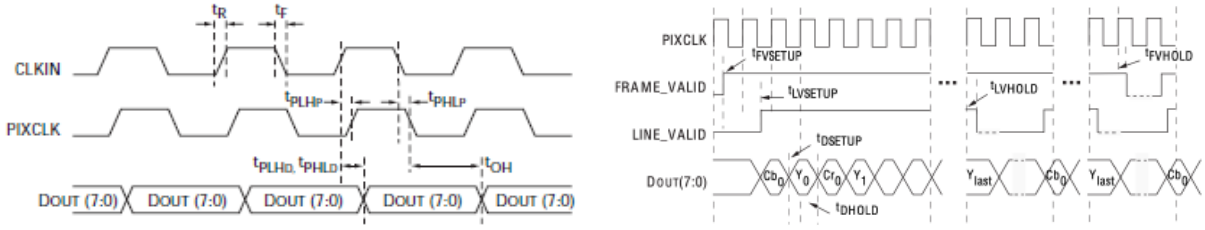


Figura 23: Cronograma de salida del sensor

La señal *PIXCLK*, en adelante, *pclk,* es la señal de reloj interna del módulo MT9V111, con la cual están sincronizados el resto de los mecanismos que intervienen en la generación de las imágenes. El módulo trabaja con actualización de datos en cada flanco de bajada de *pclk.*
Al trabajar el módulo con una señal de reloj diferente al a utilizada en la FPGA, es necesario tener esta información en el sistema. Para ello, se implementa un detector de flanco, que genera una señal de tipo *tick* cada vez que el reloj *pclk* alcanza un flanco de bajada. Esto es posible debido a que la frecuencia del reloj *pclk* es 4 veces menor que la del sistema síncrono diseñado, por lo que dicho reloj puede ser muestreado al menos 4 veces por ciclo. Esta señal será utilizada para determinar en qué momento el dato que el sensor coloca en el bus es válido y se puede trasladar a la FIFO.
Teniendo en cuenta que el sensor está configurado por defecto en modo YUV 4:2:2, y se trabajará con imágenes en escala de grises, es necesario conocer cómo se presentan estos datos.
En el caso que nos ocupa, necesitaremos tomar únicamente los datos de luminancia (Y), pues estos contienen el valor de iluminación que la imagen tiene, y corresponde con escala de grises. Para esto, se realiza en este módulo un contador binario tal que únicamente selecciona como válidos los bytes pares.

Además, este módulo controla la activación de ambas máquinas del controlador para FT245, habilita el módulo cámara, y gestiona la introducción de datos a la FIFO. Toda esta secuencia está recogida en la Figura 24 y descrita a continuación:

OFF	Se define el estado inicial del controlador cuando el PC no ha solicitado todavía la transmisión de un *fotograma*, es decir, la máquina de lectura no lee dato porque el módulo no tiene datos disponibles para ser leídos. Únicamente se habilita la máquina de lectura, dejando el sensor de imagen y la máquina de escritura inactivas. Además, no se genera ninguna señal *push.*
IDLE	Cuando se identifica una petición de dato desde la máquina de lectura, el sistema controlador pasa al siguiente estado, donde espera a que el *fotograma* comience.
BLANKING	Se llega a este estado cuando llega la señal de inicio de fotograma. Se habilita la señal de *TRIGGER* para poder capturar datos válidos cuando la señal de *LVAL* se active. Si se recibe señal de fin de fotograma, se comprueba si hay aún datos en la FIFO para transmitir al PC y si no los hay, se pasa a estado inicial a esperar otra petición del sistema.
FLUSH	En este estado, el sensor no está entregando dato válido, pero hay que vaciar la FIFO para poder transmitir la imagen completa.
DATA	En este estado, la señal *push* pasa a ser rutada con la señal *data_valid* capturada. Como se explicó en este apartado, esta señal de tipo *tick* indica cuándo el dato del bus es válido para ser leído. Con esta naturaleza, se utiliza esta señal para indicar cuándo el dato en el bus se debe almacenar en la FIFO. Además, la habilitación del controlador de escritura del FT245 se fija a la señal *full* negada, lo que significa que estará activa siempre que queden datos en la pila y no esté llena (nunca se llenará al haberse sobredimensionado).

Adicionalmente, cabe destacar que el *reset* de este módulo está modelado de manera ligeramente distinta al resto. Aquí, en lugar de depender únicamente de la señal reset *externa*, depende también de la señal de estado *full*. Esto significa que, en caso de que la pila se llene, el sistema se reiniciará al estado incial, ya que los datos del *fotograma* en progreso se perderán si se continúa, pues no se podrían seguir almacenando.

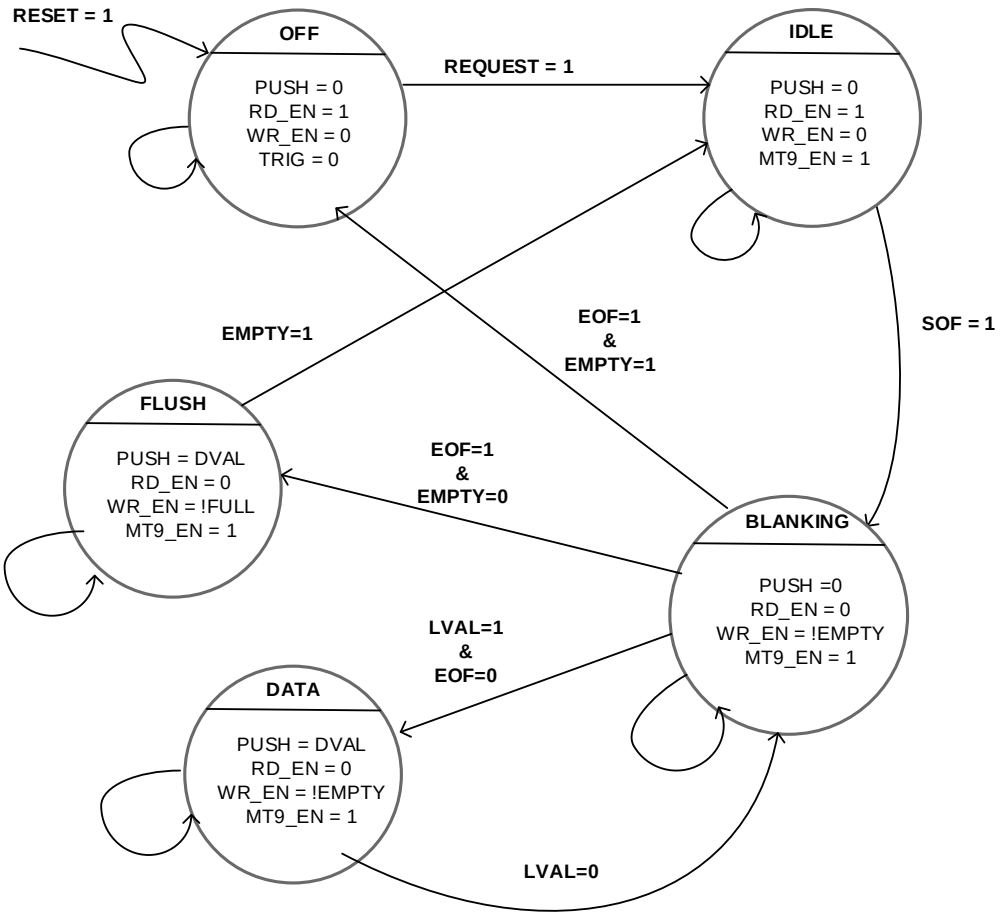


Figura 24: *RECEIVER* FSM

2.3.5.3 PATTERN

Para emular el comportamiento del módulo sensor se ha implementado una máquina de estados como la de la Figura 26 para generar la salida esperada del sensor de imagen (Figura 25 y Tabla 17). Para ello, además, se han diseñado varios contadores:

1. VERTICAL_RAMP: es un proceso que generará un patrón de 5 columnas con valor fijo cada una de ellas como imagen de test.
2. HORIZONTAL_RAMP: es un proceso que generará un patrón de 128 filas con valor fijo cada una de ellas como imagen de test.
3. PIXEL_COUNT: es un contador de píxeles generados en cada lfanco de bajada de la señal XCLK.
4. LINE_COUNT: es un contador de líneas que se incrementa con cada nueva línea, funcionamiento análogo al de la rampa horizontal.
5. HBLANK_COUNT: contador de 26.5us de blanking horizontal.
6. SOFBLANK_COUNT: contador de 25us de blanking de inicio de fotograma.
7. EOFBLANK_COUNT: contador de 1.17us de blanking de final de fotograma.
8. VBLANK_COUNT: contador de 17.3ms de blanking vertical.

La elección de patrón vertical u horizontal se realiza con la señal de control *RAMP*, conectada al SW[0].

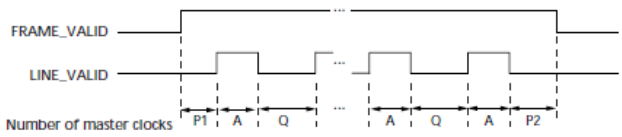


Figura 25: Cronograma señales de control del sensor [1]

Parameter	Name	Equation (Master Clocks)	Default Timing At 12 MHz
A	Active Data Time	(Reg0x04 - 7) x 2	= 1,280 pixel clocks = 1,280 master clocks = 106.7us
P1	Frame Start Blanking	(Reg0x05 + 112) x 2	= 300 pixel clocks = 300 master clocks = 25.0us
P2	Frame End Blanking	14 CLKS	= 14 pixel clocks = 14 master clocks = 1.17us
Q	Horizontal Blanking	(Reg0x05 + 121) x 2 (MIN Reg0x05 value = 9)	= 318 pixel clocks = 318 master clocks = 26.5us
A + Q	Row Time	(Reg0x04 + Reg0x05 +114) x 2	= 1,598 pixel clocks = 1,598 master clocks = 133.2us
V	Vertical Blanking	(Reg0x06 + 9) x (A + Q) + (Q - P1 - P2)	= 20, 778 pixel clocks = 20,778 master clocks = 1.73ms
Nrows x (A + Q)	Frame Valid Time	(Reg0x03 - 7) x (A + Q) - (Q - P1 - P2)	= 767,036 pixel clocks = 767,036 master clocks = 63.92ms
F	Total Frame Time	(Reg0x03 + Reg0x06 + 2) x (A + Q)	= 787,814 pixel clocks = 787,814 master clocks = 65.65ms

Tabla 17: Temporización de salida del sensor [1]

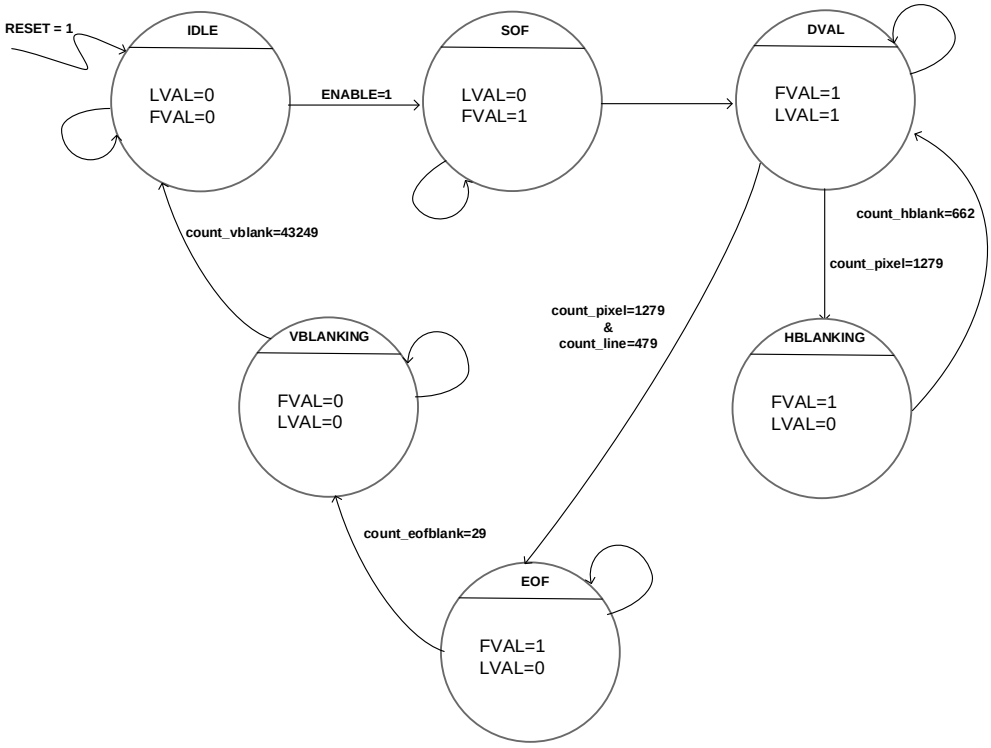


Figura 26: *IMAGE_PATTERN* FSM

2.4 Diseño Software

2.4.1 Introducción

Es necesario el desarrollo de una aplicación software que permita la interacción con el sistema, y que sea sencilla de utilizar, es por eso por lo que se desarrolla una GUI con la tecnología .NET de Microsoft en lenguaje C# para este propósito.

Esta GUI (Figura 27) hace uso tanto de librerías propias como de terceros como la DLL para control del módulo UM232H-B, y para el procesamiento de imágenes con la librería de visión OpenCV que al estar escrita en C++ precisa de un wrapper llamado EMGU para su uso.

El entorno de desarrollo ha sido el IDE Visual Studio 2017, creando una aplicación compatible con sistemas Windows versión 10 y posteriores, con el SDK .NET Framework 4.6.1. Los componentes principales se introducen a continuación.

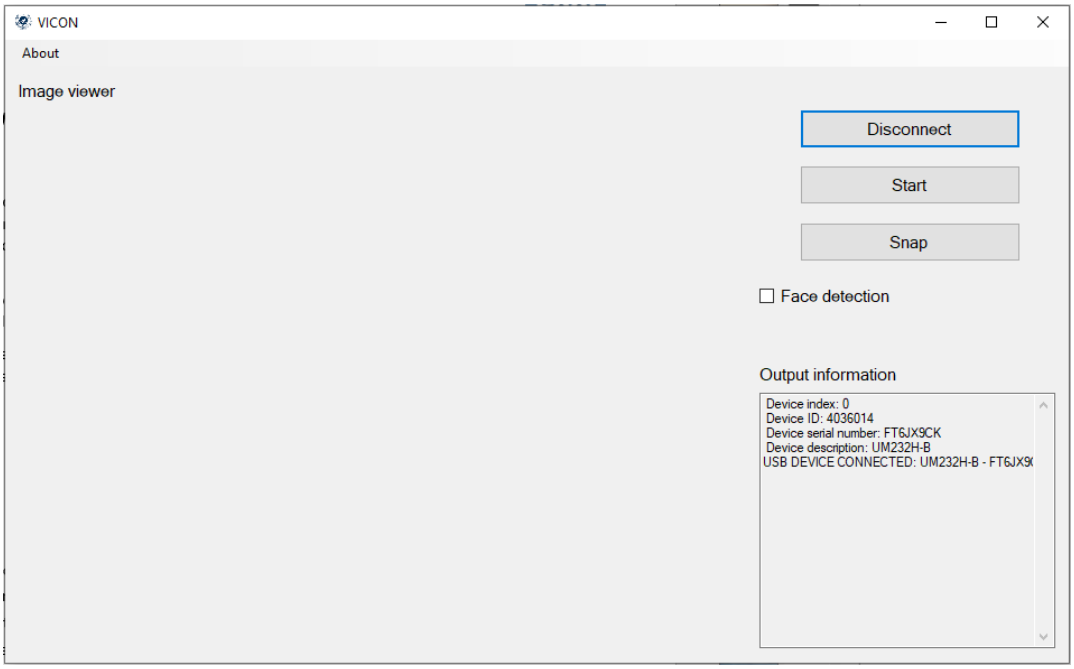


Figura 27: GUI del sistema

2.4.2 Esquema general

En este apartado se va a explicar cada una de las partes que forman la aplicación software desarrollada, y cuyo diagrama de flujo está representado en la Figura 28.

GUI

Utilizando los *forms* nativos con .NET, se realiza una interfaz gráfica de usuario que permitirá al mismo tener el control del sistema. Desde esta interfaz, podrán seleccionarse diferentes acciones: activar o desactivar la conexión, activar o desactivar vídeo, activar o desactivar reconocimiento facial, captura de imagen; así como mostrará al usuario el vídeo proveniente del sensor de imagen, y la tasa de FPS conseguida en esta transmisión. Además, ésta contiene una pequeña ventana con información de la aplicación, fecha, autor, etc.

Conexión FTDI

Estas funciones, *Connect* y *Disconnect,* se definen acorde a los métodos incluidos en la librería FTD2XX, y se definen de tipo público. Esto permite que puedan ser llamadas desde otro Form de la aplicación, lo que permite limitar la pantalla principal a la reproducción del vídeo.

Hilo de lectura de datos

Se trata de un hilo secundario definido con la función *ReadThread,* para controlar las transmisiones con el dispositivo FT232H. En él, se realizará la petición de dato hacia la FPGA, escribiendo un dato desde la aplicación hacia el conversor, y, acto seguido, procederá a escuchar los datos que llegan desde la FPGA hacia el PC.

Visualización de imágenes

GUIUpdater es la función encargada de actualizar la imagen que se muestra en la GUI. Esta imagen se recibe y almacena en el hilo secundario de lectura. La aplicación utiliza un método delegado de tal manera que lanza la ejecución dentro de la llamada desde el hilo principal. Esta función se encarga de mostrar también la información referente a la tasa de FPS *en* vivo del vídeo transmitido.

Reconocimiento de caras

No se define en una función como tal, sino que se integra en el método anterior. Se procesan las imágenes mediante la librería OpenCV. En concreto, se llama a la función *DetectMultiScale,* habiendo cargado previamente el clasificador donde se almacenan los patrones para la detección de rostros.

Captura de imagen

Se diseña una función, *SnapButton_Click,* tal que almacena el *fotograma* en el momento que se ejecuta. Esto permite capturar una instantánea del vídeo en reproducción, y guardarla en formato *bmp* para poder ser visualizada correctamente. Esta función se llama desde el control *Snap* en el interfaz principal, y abre un cuadro de diálogo donde el usuario seleccionará dónde guardar la imagen capturada.

Información de usuario

De manera similar a la visualización de imágenes, se define el método delegado *Logger* para el control del *log* de eventos implementado. Esta función también se define como pública, ya que también se pretende que sea accesible desde cualquier *form* del proyecto. Además, el hecho de estar definida con un método delegado viene de que se requiere que cualquier hilo acceda a él.

Menú de opciones

Se diseña un menú, donde se podrá acceder a la información de la aplicación.

Todo el flujo de la información de la aplicación viene recogido en diagrama de la Figura 28 y se verá detallado en las próximas secciones.

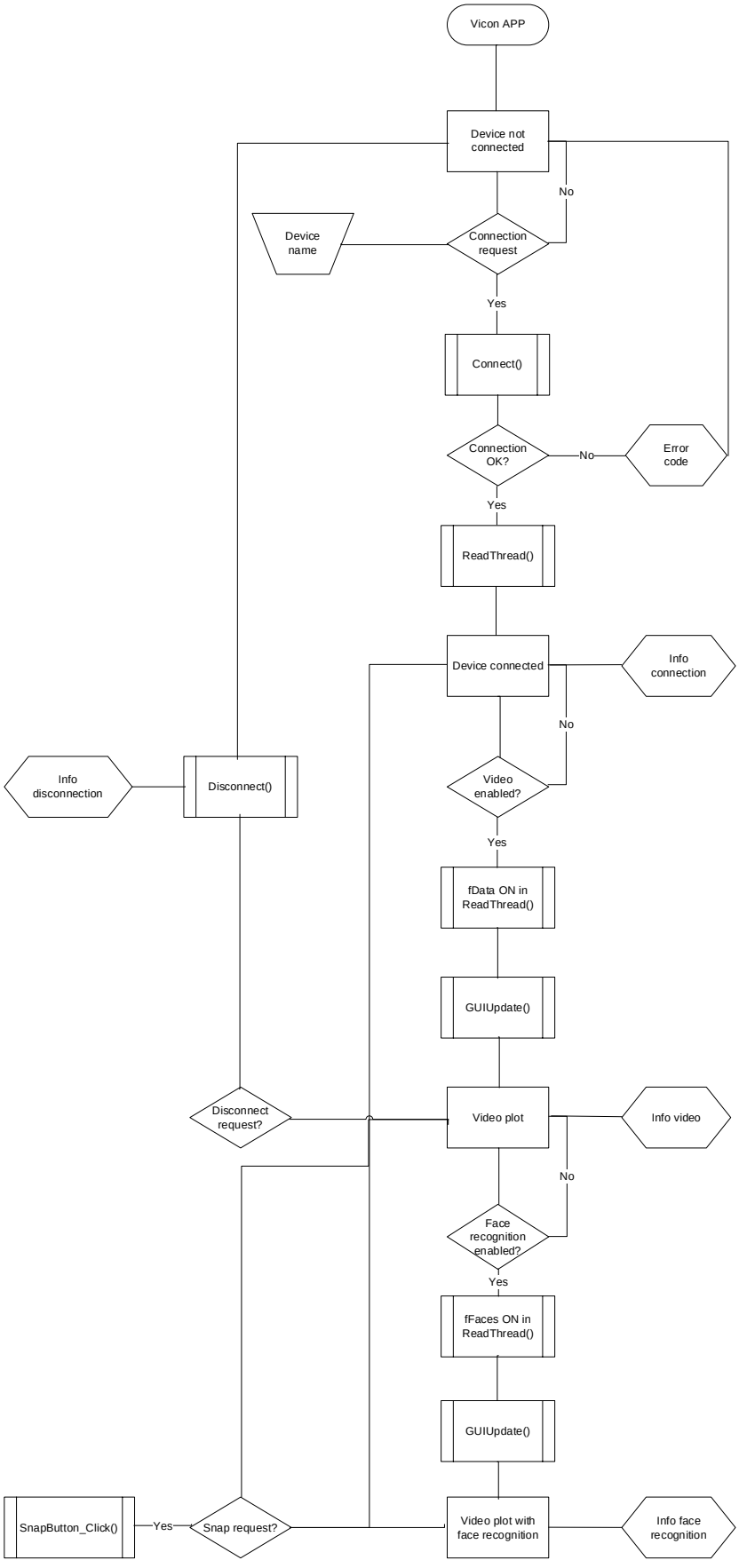


Figura 28: Diagrama de flujo principal del sistema software

2.4.3 Conexión FTDI

Estas funciones se encargan de algo más que de conectar con el dispositivo FT232H, pues busca el identificador del dispositivo UM232H-B entre los dispositivos conectados al PC.

En caso de que no haya dispositivos conectados, arrojará un error, lo mostrará en el *logger*, y terminará devolviendo un valor lógico falso. En caso de que el número no sea cero, la rutina irá dispositivo por dispositivo consultando el nombre de cada uno de ellos. En caso de que ningún nombre sea idéntico al del dispositivo UM232H-B, arrojará un error. En caso de que el dispositivo indicado sea encontrado, realizará la conexión, mediante la llamada a la función *FTDI_Open.* Si el resultado de esta función ha sido satisfactorio, se indicará en el *logger*, se inicializará el hilo de lectura, y se terminará la rutina con valor lógico verdadero. En caso de error, la función lo arrojará, y terminará con un valor falso como indica el diagrama de la Figura 29.

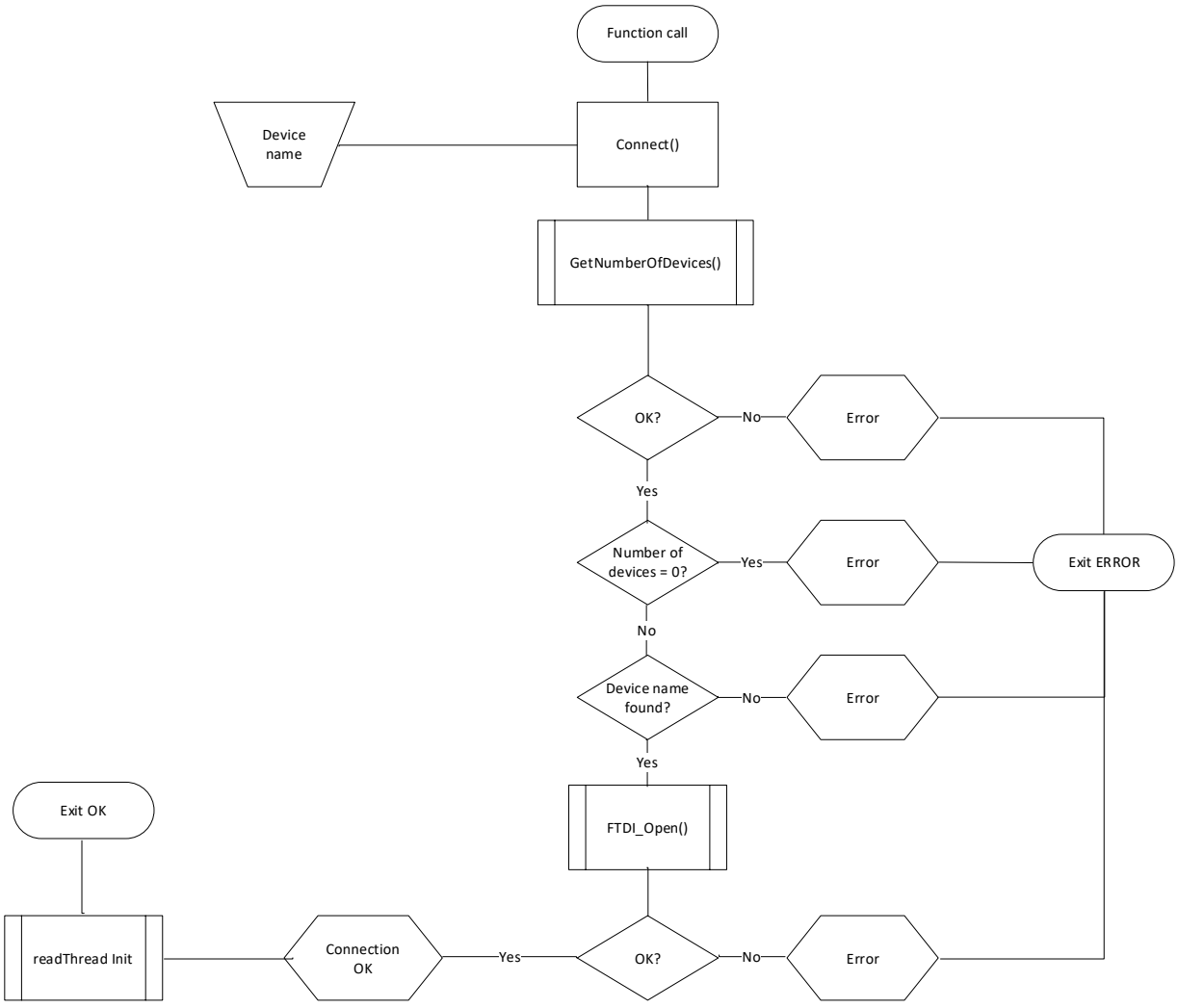


Figura 29: Diagrama de flujo función *Connect*

La gestión de la desconexión es mucho más simple como se puede ver en la Figura 30, pues no se comprueba el retorno de ninguna de las funciones, ya que no es requerido. Se oculta tanto el vídeo como el indicador de FPS en cuanto a la interfaz. En cuanto al sistema, se termina el hilo de lectura, y se cierra la conexión con el dispositivo FT232H.

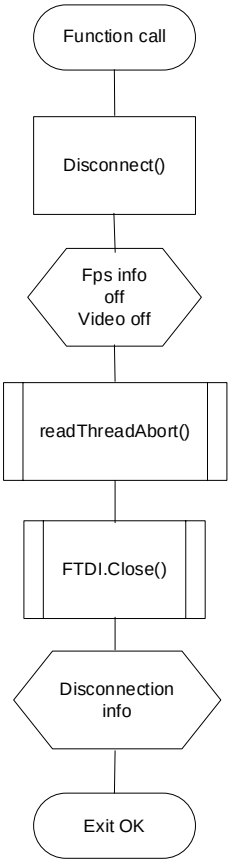


Figura 30: Diagrama de flujo función *Disconnect*

2.4.4 Lectura de datos

El hilo de lectura de datos está formado por la función *ReadThread* representada en la Figura 31, que tiene como objetivo único solicitar y recibir los datos desde el dispositivo FT232H. Esta función está continuamente esperando la activación del flag *fData*. Éste se activa a través del botón *Connect* de la GUI.

Una vez está activo el flag, el hilo entra en un bucle en el que:

1. Comprueba si se ha solicitado el fotograma a la FPGA. Esto se comprueba mediante el flag *fRead*, inicialmente con valor booleano falso, lo que significa que el dato no se ha pedido. En este caso, se envía 1 byte a la FPGA, cuyo valor no importa (dummy). En este momento, el flag *fRead* se configura a nivel booleano verdadero, indicando que el fotograma se ha solicitado. Este flag sólo volverá a ser falso cuando el fotograma haya sido correctamente procesado.
2. Cuando el fotograma se ha solicitado, lee todos los datos entrantes desde el dispositivo FT232H.

Una vez el dispositivo está leyendo, se comprueba la cantidad de datos recibidos. En una imagen VGA de las tratadas, se requieren exactamente 307200 bytes (640x480 píxeles, 1 byte por píxel). En este punto, el sistema tiene comprueba si se reciben bytes y actúa en consecuencia invocando a la función *GUIUpdate*, que se encargará de actualizar la imagen del interfaz, como se verá posteriormente.

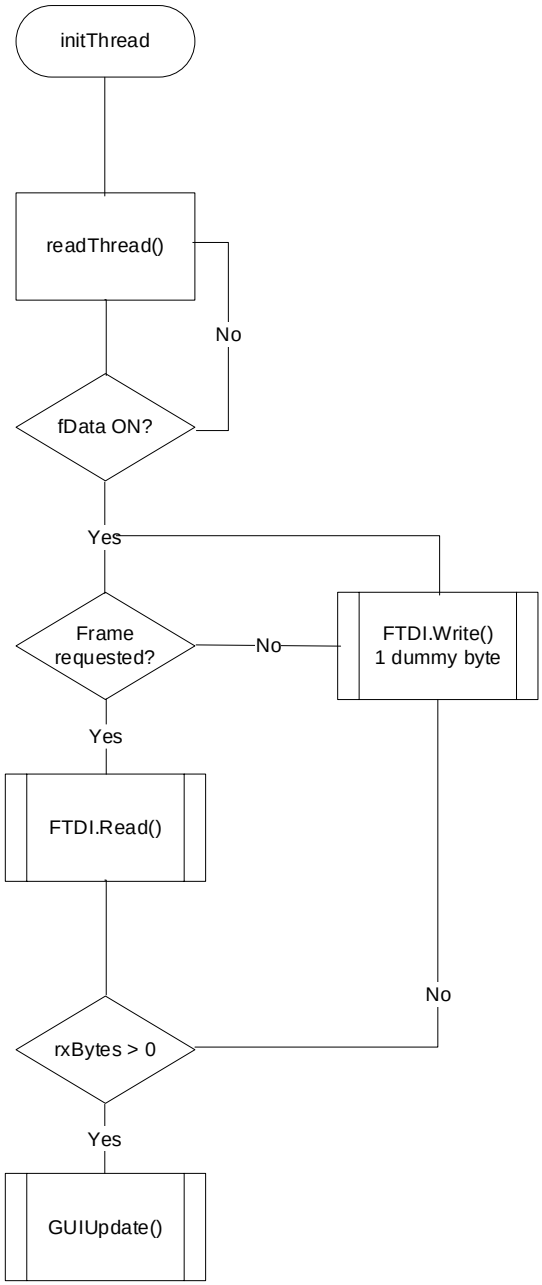


Figura 31: Diagrama de flujo función *ReadThread*

2.4.5 Visualización de imágenes y reconocimiento de caras

El reconocimiento facial se realiza en el propio *GUIUpdate.* En primer lugar, la función *GUIUpdate* (Figura 32) está desarrollada específicamente para la actualización de datos relacionados con el vídeo en la interfaz gráfica. Adicionalmente, y con el fin de dedicar el hilo de lectura, únicamente a la recepción de datos, se realiza en este apartado el procesado de imagen para la detección facial. Para éste, se utiliza el clasificador *haarcascade* incluido en la librería OpenCV.

Cuando se realiza la llamada a esta función, lo primero que se consulta es si el flag *fFaces* está activo. Este flag es el que almacena la petición del usuario desde la GUI de activar el reconocimiento facial. Por tanto, el

sistema tiene dos caminos diferenciados:

1. Reconocimiento facial no activado: Se actualiza la imagen directamente en el GUI, sin necesidad de procesamiento.
2. Reconocimiento facial activado: Se procesa la imagen en primer lugar, invocando la función *DetectMultiScale* de las librerías OpenCV. La salida de esta función almacena un fotograma donde, mediante otro bucle de procesado, se dibujan rectángulos en cada una de las caras detectadas. Este fotograma procesado se actualiza en el GUI como se hizo con el no procesado.

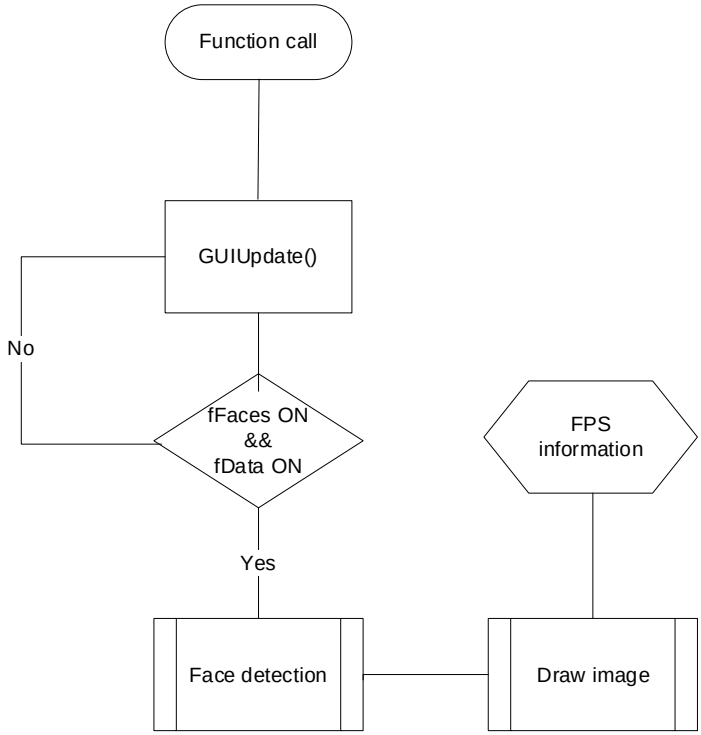


Figura 32: Diagramade flujo función *GUIUpdate*

2.4.6 Captura de imagen

Esta función, cuyo flujo está representado en la Figura 33, se define de tipo *callback*, ya que es llamada desde un botón en la GUI. Esta función coge el último fotograma almacenado en la rutina *GUIUpdate,* en la variable destinada para este fin, y la almacena con formato *tiff* mediante un cuadro *SaveFileDialog* en el directorio y con el nombre que el usuario introduzca. Cuando esta función es llamada, la función *GUIUpdate* deja de almacenar el último fotograma en la variable imagenCaptura, pues de otra manera, no se guardaría la imagen en el momento de la pulsación, sino la última antes de pulsar el botón de guardar. Una vez la imagen es guardada, se reanuda el traspaso de datos a esta variable.

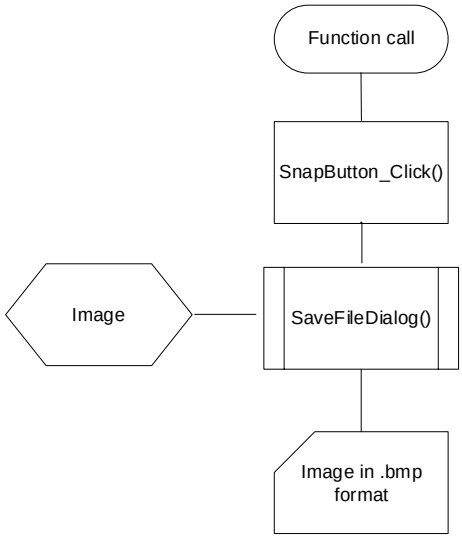


Figura 33: Diagrama de flujo función *SnapButton_Click*

2.4.7 Información de usuario

Esta rutina de la Figura 34 se encarga de escribir datos del programa como estado de la conexión, activación/deasctivación de reconocimiento de caras, etc., en el logger situado en la GUI. Ésta requiere dos entradas en su llamada, que son el tipo de mensaje a registrar, indicado mediante un tipo enumerado definido para este fin, y el texto que se quiere añadir al mensaje predeterminado acorde al tipo. El logger funciona invocando un método delegado, ya que esta función por sí sola no tiene acceso a la GUI.

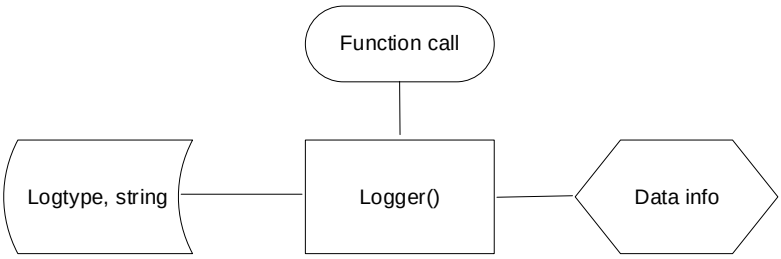


Figura 34: Diagrama de flujo función *Logger*

2.4.8 Menú de opciones

Este menú (Figura 35) simplemente abre un *DialogBox* con alguna información acerca del TFM, el autor y la universidad.

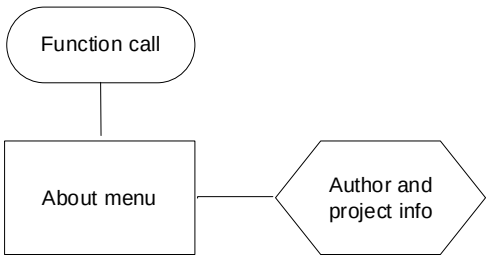


Figura 35: Diagrama de flujo función *AboutToolStripMenuItem_Click*

2.5 Implementación

Si bien en el apartado de diseño se quiso hacer más hincapié en la funcionalidad de cada bloque hardware, así como en la funcionalidad del software, se considera este apartado de implementación para detallar la manera en que los desarrollos se han llevado a cabo sobre el sistema físico. Si bien no se va a entrar en detalle para todos y cada uno de los desarrollos, sobre todo en la parte hardware ya que hablar de todos no aporta demasiada información imprescindible, sino que redunda.

2.5.1 Implementación hardware

El diseño del Sistema se ha hecho de forma modular, con tres grandes bloques independientes conectados entre sí: *IMAGE_CHANNEL*, *CGU*, y *FT245_CHANNEL*. Además, todas las FSM diseñadas en el presente TFM siguen la siguiente estructura de dos procesos:

- Lógica de estado siguiente y generación de salidas: Se tratan de procesos combinacionales donde se implementan las transiciones entre estados, las cuales dependen únicamente de las entradas, y del estado actual. De igual manera, en este apartado se calculan las salidas del sistema únicamente en función del estado actual, y, en ocasiones, de las entradas del bloque.
- Registro de estado y salidas: Con el fin de evitar *glitches* que ocasionen un funcionamiento no deseado del sistema, el estado actual se registra en cada proceso, de tal manera que, en cada flanco de reloj, se almacena el estado que la lógica de estado siguiente determine. De igual manera que con los estados, las salidas del sistema deben estar registradas, consiguiendo así que el camino de las salidas de la FSM sea el mismo, evitando estos glitches de los que se hablaba en el apartado anterior.

A continuación, se va a detallar el diseño RTL del bloque *IMAGE_CHANNEL*, representado en la Figura 36, al ser uno de los bloques principales, y más complejos, del sistema.

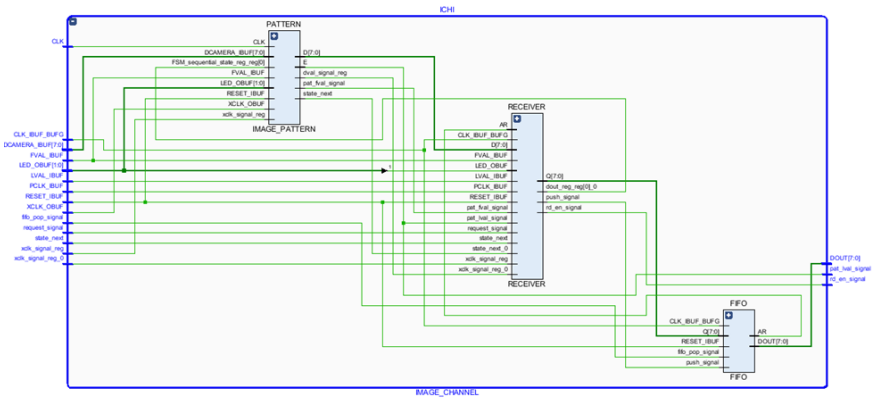


Figura 36: Esquemático del diseño *IMAGE_CHANNEL*

2.5.1.1 IMAGE_PATTERN

Este bloque recibe diversas señales del resto de módulos, y gestiona tanto el push, como las habilitaciones de los periféricos. Tiene como componente principal un bloque receptor que recoge los datos válidos y una FIFO para almacenar los bytes de píxeles antes de ser transmitidos de manera que no se pierda ninguno.

Para ello, primeramente, se diseña un generador de imágenes sintéticas basado en contadores que respeten la temporización de salida del sensor, siendo 25MHz la señal síncrona del sistema pues emula el comportamiento de sensor.

Para el desarrollo del bloque patrón, es necesario conocer todos los parámetros del sensor MT9V111 cuando está transfiriendo una imagen. En concreto, el cronograma de la aplicación aquí presente es representativo con el siguiente cronograma suministrado por el fabricante, donde se muestra cómo funcionan las señales FRAME_VALID y LINE_VALID (en adelante, FVAL y LVAL, respectivamente), y cómo los datos aparecen en el bus acorde a las mismas, siendo los datos mostrados en el cronograma de la Figura 37, en el mismo formato que en la presente aplicación (YUV 4:2:2):

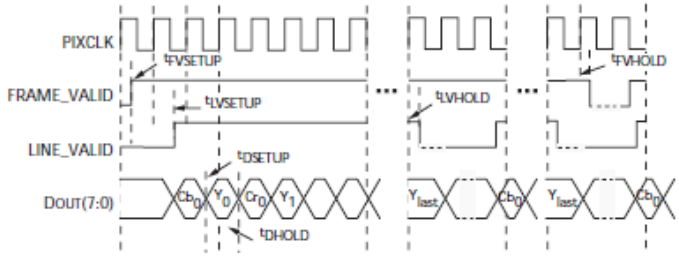


Figura 37: Cronograma MT9V111 [1]

En primer lugar, se define una FSM tal que simula el comportamiento del sensor MT9V111 acorde a los cronogramas y timings vistos en la Figura 37, en la que cada estado corresponde a uno de los parámetros de la Figura 38 (A, P1, P2, Q, V), así como los contadores para garantizar tanto los 1280 píxeles en cada línea horizontal, como las 480 líneas verticales.

Parameter	Name	Equation (Master Clocks)	Default Timing At 12 MHz
A	Active Data Time	(Reg0x04 - 7) x 2	= 1,280 pixel clocks = 1,280 master clocks = 106.7us
P1	Frame Start Blanking	(Reg0x05 + 112) x 2	= 300 pixel clocks = 300 master clocks = 25.0us
P2	Frame End Blanking	14 CLKS	= 14 pixel clocks = 14 master clocks = 1.17us
Q	Horizontal Blanking	(Reg0x05 + 121) x 2 (MIN Reg0x05 value = 9)	= 318 pixel clocks = 318 master clocks = 26.5us
A + Q	Row Time	(Reg0x04 + Reg0x05 +114) x 2	= 1,598 pixel clocks = 1,598 master clocks = 133.2us
V	Vertical Blanking	(Reg0x06 + 9) x (A + Q) + (Q - P1 - P2)	= 20, 778 pixel clocks = 20,778 master clocks = 1.73ms
Nrows x (A + Q)	Frame Valid Time	(Reg0x03 - 7) x (A + Q) - (Q - P1 - P2)	= 767,036 pixel clocks = 767,036 master clocks = 63.92ms
F	Total Frame Time	(Reg0x03 + Reg0x06 + 2) x (A + Q)	= 787,814 pixel clocks = 787,814 master clocks = 65.65ms

Figura 38: Temporización cronograma MT9V111 [1]

Con esto, se deben definir numerosas constantes y señales registradas para las cuentas de píxeles, líneas y datos no válidos o blanking.

```
-------------------------------------------------------------------
-- Counters and FSM to emulate sensor operation at 25MHz
-------------------------------------------------------------------
constant sofblank       : NATURAL   := 625;             -- 25us
constant eofblank       : NATURAL   := 30;              -- 1.17us
constant hblank         : NATURAL   := 663;             -- 26.5us
constant vblank         : NATURAL   := 432500;          -- 17.3ms
constant nlines         : NATURAL   := 480;             -- 480 lines per frame
constant npixels        : NATURAL   := 1280;            -- 1280 pixels per line. YUV:4-2-2 send (Cb_i
- Y_i - Cr_i - Y_i+1) we only need Y
type STATES is (idle, sof, dval, eof, hblanking, vblanking);
signal state_reg, state_next                        : STATES;
-------------------------------------------------------------------
-- Sensor output signals
-------------------------------------------------------------------
signal fval_reg, fval_next                          : STD_LOGIC;
signal lval_reg, lval_next                          : STD_LOGIC;
signal dout_vramp_reg, dout_vramp_next              : STD_LOGIC_VECTOR(7 downto 0);
signal dout_hramp_reg, dout_hramp_next              : STD_LOGIC_VECTOR(7 downto 0);
-------------------------------------------------------------------
-- Active data internal signals
-------------------------------------------------------------------
signal reset_pixel_reg, reset_pixel_next            : STD_LOGIC;
signal reset_line_reg, reset_line_next              : STD_LOGIC;
signal reset_data_vramp_reg, reset_data_vramp_next  : STD_LOGIC;
signal reset_data_hramp_reg, reset_data_hramp_next  : STD_LOGIC;
signal count_pixel                                  : STD_LOGIC_VECTOR(10 downto 0);
signal count_line                                   : STD_LOGIC_VECTOR(8 downto 0);
-------------------------------------------------------------------
-- Start of Frame Blanking internal signals
-------------------------------------------------------------------
signal reset_sofblank_reg, reset_sofblank_next      : STD_LOGIC;
signal count_sofblank                               : STD_LOGIC_VECTOR(9 downto 0);
-------------------------------------------------------------------
-- End of Frame blanking internal signals
-------------------------------------------------------------------
signal reset_eofblank_reg, reset_eofblank_next      : STD_LOGIC;
signal count_eofblank                               : STD_LOGIC_VECTOR(5 downto 0);
-------------------------------------------------------------------
-- Horizontal blanking internal signals
-------------------------------------------------------------------
signal reset_hblank_reg, reset_hblank_next          : STD_LOGIC;
signal count_hblank                                 : STD_LOGIC_VECTOR(9 downto 0);
-------------------------------------------------------------------
-- Vertical blanking internal signals
-------------------------------------------------------------------
signal reset_vblank_reg, reset_vblank_next          : STD_LOGIC;
signal count_vblank                                 : STD_LOGIC_VECTOR(19 downto 0);
```

Luego, es posible, a través de una entrada al sistema con uno de los interruptores de la placa BASYS 3 configurar un patrón horizontal o vertical como imagen sintética generada.

```
    ----------------------------------------------------------------------
    -- Connect output signals
    ----------------------------------------------------------------------
    PCLK <= XCLK;
    LVAL <= lval_reg;
    FVAL <= fval_reg;
    DOUT <= dout_vramp_reg when RAMP = '0' else dout_hramp_reg;
```

Y se generan los procesos síncronos a la señal de reloj correspondientes a cada contador como puede verse en el siguiente ejemplo:

```
    ----------------------------------------------------------------------
    -- Pixels counter
    ----------------------------------------------------------------------
    PIXEL_COUNT: process(XCLK, RESET, reset_pixel_reg, lval_reg)
    begin
        if RESET = '1' or reset_pixel_reg = '1' then
            count_pixel <= (others=>'0');
        elsif falling_edge(XCLK) and lval_reg = '1' then
            count_pixel <= count_pixel + 1;
        end if;
    end process;
```

Por otra parte, la lógica de generación de salidas está controlada por una máquina de estados finitos en las que el paso de un estado a otro depende del valor de los contadores para cumplir con la temporización del sensor.

```
    ----------------------------------------------------------------------
    -- Output signals logic
    ----------------------------------------------------------------------
    FSM: process (state_reg, count_sofblank, count_line, count_pixel, count_hblank, count_eofblank,
count_vblank)
        begin
            case state_reg is
                -- Start Of Frame
                when sof =>
                    if count_sofblank = sofblank-1 then
                        state_next <= dval;
                    else
                        state_next <= sof;
                    end if;
                    lval_next               <= '0';
                    fval_next               <= '1';
                    reset_hblank_next       <= '1';
                    reset_sofblank_next     <= '0';
                    reset_eofblank_next     <= '1';
                    reset_pixel_next        <= '1';
                    reset_line_next         <= '1';
                    reset_data_vramp_next   <= '1';
                    reset_data_hramp_next   <= '1';
                    reset_vblank_next       <= '1';
                -- Data valid output
                when dval =>
                    lval_next               <= '1';
                    fval_next               <= '1';
                    reset_hblank_next       <= '1';
                    reset_sofblank_next     <= '1';
                    reset_eofblank_next     <= '1';
                    reset_pixel_next        <= '0';
                    reset_line_next         <= '0';
                    reset_data_vramp_next   <= '0';
                    reset_data_hramp_next   <= '0';
                    reset_vblank_next       <= '1';
                    -- Check if end of VGA frame (640x2)x480 or end of line.
                    if count_pixel = npixels-1 and count_line = nlines-1 then
                        state_next <= eof;
                    elsif count_pixel = npixels-1 then
                        state_next <= hblanking;
```

```
                else
                    state_next <= dval;
                end if;
            -- Horizontal Blanking
            when hblanking =>
                lval_next               <= '0';
                fval next               <= '1';
                reset_hblank_next       <= '0';
                reset_sofblank_next     <= '1';
                reset_eofblank_next     <= '1';
                reset_pixel_next        <= '1';
                reset_line_next         <= '0';
                reset data vramp next   <= '0';
                reset_data_hramp_next   <= '0';
                reset_vblank_next       <= '1';
                if count_hblank = hblank-1 then
                    state_next <= dval;
                else
                    state_next <= hblanking;
                end if;
            -- End Of Frame
            when eof =>
                lval_next               <= '0';
                fval next               <= '1';
                reset_hblank_next       <= '1';
                reset_sofblank_next     <= '1';
                reset_eofblank_next     <= '0';
                reset_pixel_next        <= '1';
                reset_line_next         <= '1';
                reset data vramp next   <= '0';
                reset_data_hramp_next   <= '0';
                reset_vblank_next       <= '1';
                if count_eofblank = eofblank-1 then
                    state_next <= vblanking;
                else
                    state next <= eof;
                end if;
            -- Vertical Blanking
            when vblanking =>
                lval_next               <= '0';
                fval_next               <= '0';
                reset_hblank_next       <= '1';
                reset_sofblank_next     <= '1';
                reset_eofblank_next     <= '1';
                reset_pixel_next        <= '1';
                reset_line_next         <= '1';
                reset_data_vramp_next   <= '1';
                reset_data_hramp_next   <= '1';
                reset_vblank_next       <= '0';
                if count_vblank = vblank-1 then
                    state_next <= sof;
                else
                    state_next <= vblanking;
                end if;
            -- IDLE state
            when others =>
                lval_next               <= '0';
                fval_next               <= '0';
                reset_hblank_next       <= '1';
                reset_sofblank_next     <= '1';
                reset_eofblank_next     <= '1';
                reset_pixel_next        <= '1';
                reset_line_next         <= '1';
                reset_data_vramp_next   <= '1';
                reset_data_hramp_next   <= '1';
                reset_vblank_next       <= '1';
                state_next              <= sof;
        end case;
    end process;
```

2.5.1.2 RECEIVER

Este bloque recibirá las señales de control ya sea del patrón de prueba o del sensor de imagen y determinará que datos son válidos o no para ser capturados, además deberá de iniciar su funcionamiento únicamente cuando el PC lo solicite.

Esto se logra a través de dos detectores de flanco para las señales *FVAL* y *LVAL* generando así señales de inicio de fotograma y final de fotograma que serán utilizadas por el resto de los bloques del sistema.

```
-------------------------------------------------------------------
-- FSM to capture data
-------------------------------------------------------------------
type STATES is (off, idle, data, blanking);
signal state_reg, state_next                    : STATES;
-------------------------------------------------------------------
-- Internal signals
-------------------------------------------------------------------
signal trigger_reg, trigger_next                : STD_LOGIC;
signal wr_en_reg, wr_en_next                    : STD_LOGIC;
signal rd_en_reg, rd_en_next                    : STD_LOGIC;
signal push_reg, push_next                      : STD_LOGIC;
signal dout_reg, dout_next                      : STD_LOGIC_VECTOR(7 downto 0);
-------------------------------------------------------------------
-- Tick signals to detect PCLK edges
-------------------------------------------------------------------
signal tick_rise_pclk                           : STD_LOGIC;
signal tick_fall_pclk                           : STD_LOGIC;
-------------------------------------------------------------------
-- Signals to generate start of frame, end of frame, and data valid
-------------------------------------------------------------------
signal sof_signal                               : STD_LOGIC;
signal eof_signal                               : STD_LOGIC;
signal dval_signal                              : STD_LOGIC;
signal count                                    : STD_LOGIC;
```

Es fundamental en este bloque generar la señal de dato válido únicamente cuando sea necesario, y además sólo para los bytes pares debido al formato de salida por defecto del sensor YUV4222.

```
    -------------------------------------------------------------------
    -- DVAL generator: 1 every two input pixel data (YUV422 format)
    -------------------------------------------------------------------
    DVAL_GENERATOR: process
    begin
        wait until rising_edge(CLK);
        if RESET = '1' or sof_signal = '1' then
            dval_signal <= '0';
            count       <= '0';
        elsif tick_rise_pclk = '1' and trigger_reg = '1' and LVAL = '1' then
            dval_signal <= count;
            count       <= not count;
        else
            dval_signal <= '0';
        end if;
    end process;
```

Finalmente, para el correcto funcionamiento del sistema es necesario definir una lógica combinacional para determinar cuándo enviar datos a la salida del sistema. Esto se logra con una FSM y haciendo uso de las señales de control que llegan desde el módulo sensor, y el estado de la memoria FIFO.

```
    -------------------------------------------------------------------
    -- Combinational logic
    -------------------------------------------------------------------
    FSM: process (state_reg, FRAME_REQUEST, LVAL, sof_signal, eof_signal, dval_signal, FIFO_FULL,
FIFO_EMPTY)
        begin
            case state_reg is
                when idle =>
                    push_next           <= '0';
                    rd_en_next          <= '1';
```

```
                wr_en_next          <= '0';
                trigger_next        <= '1';
                dout_next           <= (others => '0');
                if sof_signal = '1' then
                    state_next <= blanking;
                else
                    state next <= idle;
                end if;
            when blanking =>
                push_next           <= '0';
                rd_en_next          <= '0';
                wr_en_next          <= not FIFO_EMPTY;
                trigger next        <= '1';
                dout_next           <= (others => '0');
                if eof_signal = '1' then
                    if FIFO_EMPTY = '1' then
                        state_next <= off;
                    else
                        state_next <= flush;
                    end if;
                elsif LVAL = '0' then
                    state_next <= blanking;
                else
                    state next <= data;
                end if;
            when data =>
                push_next           <= dval_signal;
                rd_en_next          <= '0';
                wr_en_next          <= not FIFO_EMPTY;
                trigger next        <= '1';
                dout_next           <= DIN;
                if LVAL = '0' then
                    state_next <= blanking;
                else
                    state_next <= data;
                end if;
            when flush =>
                push_next           <= '0';
                rd_en_next          <= '0';
                wr_en_next          <= not FIFO_EMPTY;
                trigger_next        <= '0';
                dout_next           <= DIN;
                if FIFO_EMPTY = '1' then
                    state_next <= idle;
                else
                    state_next <= flush;
                end if;
            when others =>
                push_next           <= '0';
                rd_en_next          <= '1';
                wr_en_next          <= '0';
                trigger_next        <= '0';
                dout_next           <= (others => '0');
                if FRAME_REQUEST = '1' then
                    state_next <= idle;
                else
                    state_next <= off;
                end if;
        end case;
    end process;
```

2.5.1.3 FIFO

En primer lugar, se definen las señales necesarias para su funcionamiento. Entre ellas, cabe destacar la definición de la memoria RAM, mediante la definición de un array de datos del parametrizando tanto las posiciones del array, como la longitud, teniendo así la capacidad del bloque FIFO totalmente configurable mediante únicamente dos parámetros. Además, las señales encargadas de los diferentes contadores también se parametrizan con el fin de ir acorde al almacenamiento, y de hacer el bloque más portable y configurable.

```
-------------------------------------------------------------------
-- RAM memory component internal signals
-------------------------------------------------------------------
type memory is array((2**B)-1 downto 0) of STD_LOGIC_VECTOR(W-1 DOWNTO 0);  -- RAM memory data type
signal ram          :   memory;
-- RAM memory instance
signal rptr         :   INTEGER range 0 to (2**B)-1 := 0;
-- RAM read address pointer
signal wptr         :   INTEGER range 0 to (2**B)-1 := 0;
-- RAM write address pointer
signal wr_en        :   STD_LOGIC;                                  -- RAM memory write enabl
signal rd_en        :   STD_LOGIC;                                  -- RAM memory read enable
-------------------------------------------------------------------
-- Control logic internal signals
-------------------------------------------------------------------
signal fifo_full    :   STD_LOGIC;                                  -- FIFO full flag
signal fifo_empty   :   STD_LOGIC;                                  -- FIFO empty flag
-------------------------------------------------------------------
-- Status logic internal signals
-------------------------------------------------------------------
signal words        :   INTEGER range 0 to (2**B) := 0;             -- Number of housed words
```

La RAM implementada se modela como un proceso síncrono:

```
    -------------------------------------------------------------------
    -- RAM memory block
    -------------------------------------------------------------------
    LUTRAM:process
    begin
        wait until rising_edge(CLK);
        -- Write port at 1xTclk
        if (wr_en = '1') then ram(wptr) <= DIN; end if;
        -- Read port at 1xTclk
        if (rd_en = '1') then DOUT <= ram(rptr); end if;
    end process;
```

Las señales *wr_en* y *rd_en* controlan, respectivamente, la escritura de dato en la FIFO, y la lectura de esta. En el caso de la escritura se guarda el dato en el bus de entrada en la dirección de escritura que toque, y en el caso de la lectura, se pone en el bus de salida el dato que corresponda acorde a la dirección de lectura.

```
    -------------------------------------------------------------------
    -- Read/write enable control
    -------------------------------------------------------------------
    wr_en <= PUSH and not fifo_full;
    rd_en <= POP and not fifo_empty;


    -------------------------------------------------------------------
    -- Control logic block
    -------------------------------------------------------------------
    CONTROL:process
    begin
        wait until rising_edge(CLK);
        -- Synchronous reset
```

```
        if (RESET = '1') then
            rptr <= 0;
            wptr <= 0;
        else
            -- Write/read pointers update
            if rd_en ='1' then rptr <= rptr + 1; end if;
            if wr_en ='1' then wptr <= wptr + 1; end if;
            -- Close circular buffer
            if rptr = (2**B)-1 then rptr <= 0; end if;
            if wptr = (2**B)-1 then wptr <= 0; end if;
         end if;
    end process;
```

Dichas direcciones están controladas con contadores circulares. Se activa la señal *full* cuando la cuenta alcanza el valor máximo, y se activa la señal *empty* cuando la cuenta alcanza el valor 0. Estas señales intervienen también directamente en la generación de las señales *wr_en* y *rd_en*, ya que las habilitaciones de lectura y escritura no permitirán escribir o leer datos de la FIFO si ésta está llena o vacía, respectivamente. Además, se realiza la asignación de las salidas del bloque.

```
    ------------------------------------------------------------------
     -- FIFO status update pending on number of stored words
    ------------------------------------------------------------------
    fifo_empty <= '1' when (words = 0) else '0';
    fifo_full  <= '1' when (words = (2**B)) else '0';

    ------------------------------------------------------------------
    -- Logic state block
    ------------------------------------------------------------------
    STATUS:process
    begin
        wait until rising_edge(CLK);
        -- Synchronous reset
        if (RESET = '1') then
            words <= 0;
        else
            -- UP/DOWN counter
            if (wr_en = '1') and (rd_en = '0') then
                words <= words + 1;
            elsif (rd_en = '1') and (wr_en = '0') then
                words <= words - 1;
            else
                words <= words;
            end if;
        end if;
    end process;
```

2.5.1.4 Asignación de salidas

Una vez se tiene el sistema diseñado e implementado mediante descripción VHDL, es necesario enlazar las entradas y salidas del top level con E/S físicas de la FPGA. En concreto, estas E/S están disponibles en la placa de desarrollo BASYS 3 mediante los conectores PMOD. En Vivado, se deben realizar los siguientes procesos para conseguir un archivo binario útil para la FPGA, desde la descripción VHDL:

- Síntesis: Este proceso transcribe nuestro modelo RTL escrito en VHDL, a una representación a nivel de puertas lógicas, de tal manera que se baja de nivel de abstracción.
- Implementación: Este proceso transcribe la representación conseguida en la síntesis, y genera los procesos de posicionado y rutado en los recursos de la FPGA, consiguiendo así el archivo bitstream listo para cargar en el dispositivo programable.

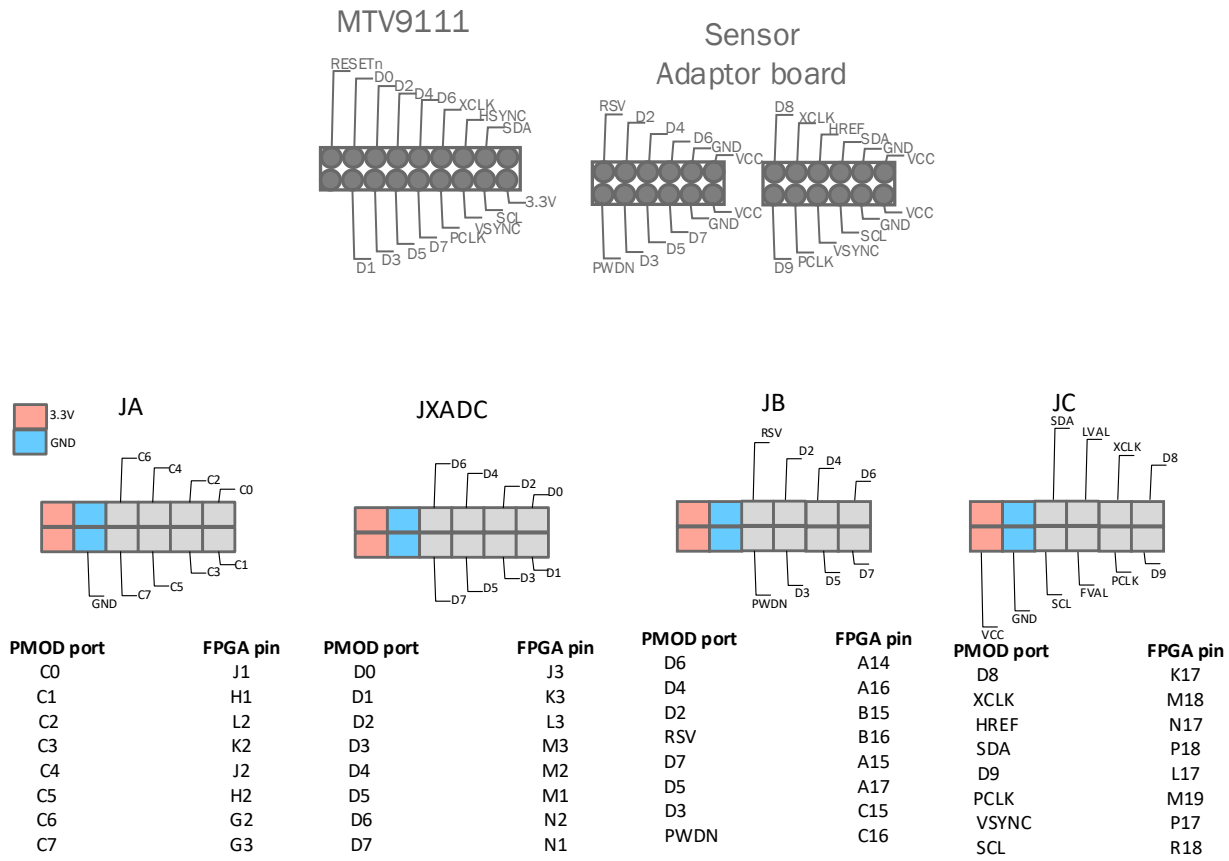
No obstante, es necesario indicar al IDE en qué pines se conectarán las entradas y salidas del diseño, de otra manera, el proceso de implementación fallará y no se obtendrá el archivo deseado. Estas indicaciones se realizan mediante el archivo constraints de la placa de desarrollo, de extensión *xdc*. En ella, se definen las entradas y salidas acorde a los pines en los que se conectarán tanto el sensor de imagen, como el módulo UM232H-B y otras señales externas como la señal de reloj, reset, leds de control o palancas de actuación.

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK}]

## Switches
set_property PACKAGE_PIN V17 [get_ports {MODE[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {MODE[0]}]
set_property PACKAGE_PIN V16 [get_ports {MODE[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {MODE[1]}]
set_property PACKAGE_PIN R2 [get_ports {RESET}]
set_property IOSTANDARD LVCMOS33 [get_ports {RESET}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
set_property PACKAGE_PIN L1 [get_ports {LED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]


##Pmod Header JA
##Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports RXFn]
set_property IOSTANDARD LVCMOS33 [get_ports RXFn]
##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports RDn]
set_property IOSTANDARD LVCMOS33 [get_ports RDn]
##Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports SIWUn]
set_property IOSTANDARD LVCMOS33 [get_ports SIWUn]
##Sch name = JA7
set_property PACKAGE_PIN H1 [get_ports TXEn]
set_property IOSTANDARD LVCMOS33 [get_ports TXEn]
##Sch name = JA8
set_property PACKAGE_PIN K2 [get_ports WRn]
set_property IOSTANDARD LVCMOS33 [get_ports WRn]
##Sch name = JA10
set_property PACKAGE_PIN G3 [get_ports PWRSVn]
set_property IOSTANDARD LVCMOS33 [get_ports PWRSVn]

##Pmod Header JB
##Sch name = JB1
set_property PACKAGE_PIN A14 [get_ports RESETn]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports RESETn]
##Sch name = JB2
set_property PACKAGE_PIN A16 [get_ports {DCAMERA[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DCAMERA[0]}]
##Sch name = JB3
set_property PACKAGE_PIN B15 [get_ports {DCAMERA[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DCAMERA[2]}]
##Sch name = JB4
set_property PACKAGE_PIN B16 [get_ports {DCAMERA[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DCAMERA[4]}]
##Sch name = JB5
set_property PACKAGE_PIN A15 [get_ports {DCAMERA[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DCAMERA[1]}]
##Sch name = JB6
set_property PACKAGE_PIN A17 [get_ports PWDn]
set_property IOSTANDARD LVCMOS33 [get_ports PWDn]
##Sch name = JB7
set_property PACKAGE_PIN C15 [get_ports {DCAMERA[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DCAMERA[3]}]
##Sch name = JB8
set_property PACKAGE_PIN C16 [get_ports {DCAMERA[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DCAMERA[5]}]

##Pmod Header JC
##Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports {DCAMERA[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DCAMERA[6]}]
##Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports XCLK]
set_property IOSTANDARD LVCMOS33 [get_ports XCLK]
##Sch name = JC3
set_property PACKAGE_PIN N17 [get_ports LVAL]
set_property IOSTANDARD LVCMOS33 [get_ports LVAL]
##Sch name = JC5
set_property PACKAGE_PIN L17 [get_ports PCLK]
set_property IOSTANDARD LVCMOS33 [get_ports PCLK]
##Sch name = JC6
set_property PACKAGE_PIN M19 [get_ports {DCAMERA[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DCAMERA[7]}]
##Sch name = JC7
set_property PACKAGE_PIN P17 [get_ports FVAL]
set_property IOSTANDARD LVCMOS33 [get_ports FVAL]

##Pmod Header JXADC
##Sch name = XA1_P
set_property PACKAGE_PIN J3 [get_ports {DATA[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA[0]}]
##Sch name = XA2_P
set_property PACKAGE_PIN L3 [get_ports {DATA[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA[2]}]
##Sch name = XA3_P
set_property PACKAGE_PIN M2 [get_ports {DATA[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA[4]}]
##Sch name = XA4_P
set_property PACKAGE_PIN N2 [get_ports {DATA[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA[6]}]
##Sch name = XA1_N
set_property PACKAGE_PIN K3 [get_ports {DATA[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA[1]}]
##Sch name = XA2_N
set_property PACKAGE_PIN M3 [get_ports {DATA[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA[3]}]
##Sch name = XA3_N
set_property PACKAGE_PIN M1 [get_ports {DATA[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA[5]}]
##Sch name = XA4_N
set_property PACKAGE_PIN N1 [get_ports {DATA[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA[7]}]

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

2.5.1.5 Prototipado

La asignación de señales entre la placa y la placa sensora teniendo en cuenta la hoja de datos del sensor sería como en la Figura 39:



PMOD port	FPGA pin
C0	J1
C1	H1
C2	L2
C3	K2
C4	J2
C5	H2
C6	G2
C7	G3

PMOD port	FPGA pin
D0	J3
D1	K3
D2	L3
D3	M3
D4	M2
D5	M1
D6	N2
D7	N1

PMOD port	FPGA pin
D6	A14
D4	A16
D2	B15
RSV	B16
D7	A15
D5	A17
D3	C15
PWDN	C16

PMOD port	FPGA pin
D8	K17
XCLK	M18
HREF	N17
SDA	P18
D9	L17
PCLK	M19
VSYNC	P17
SCL	R18

Figura 39: Asignación de señales a pines PMOD

Para finalizar, se conecta la BASYS 3 por USB al IDE Xilinx Vivado, y, haciendo uso del programador JTAG integrado en ella, programamos la FPGA con el bitstream generado con el diseño en este documento detallado. Según el archivo de configuración definido, se deben conectar los módulos como en la Figura 40 para un correcto funcionamiento.

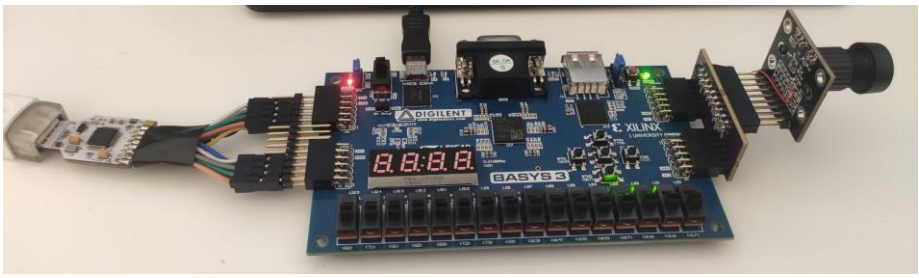


Figura 40: Montaje de prototipo

2.5.2 Implementación software

El software diseñado en el apartado anterior no necesita implementación física, como sí precisa el desarrollo hardware realizado, más allá del proceso de compilación del código en el IDE Visual Studio, que ya genera el ejecutable de la aplicación.

El proyecto del diseño software incluye las referencias externas añadidas en el mismo proyecto, de tal manera que el resultado de la compilación contiene, además del ejecutable, las dependencias necesarias. No obstante, y como es requisito del presente TFM, se distribuye la aplicación mediante un instalador (Figura 41), que abstrae al usuario de la copia de archivos a su sistema, garantizando así que siempre se copian los mismos archivos, y asegurando que todo usuario tendrá disponibles exactamente los mismos ejecutables y dependencias.

Figura 41: Instalador VICON

La creación del instalador se ha realizado con una versión de evaluación del software comercial Advanced Installer (en su versión de prueba). Éste tiene diversas configuraciones que permiten crear instaladores en formatos *.exe* o *.msi* para gran variedad de proyectos. En el caso que nos atañe, se crea un instalador *.exe* para un proyecto de Windows Forms bajo el IDE Visual Studio.

3 VALIDACIÓN

3.1 Introducción

En el presente documento se exponen las pruebas, tanto individuales de cada uno de los subsistemas que componen el diseño hardware implementado, como de la comunicación establecida entre los mismos. Dichas pruebas de verificación de funcionalidad de los distintos modelos VHDL (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language) desarrollados, son realizadas mediante simulaciones software, para lo que se emplean scripts TCL y el entorno desarrollo Xilinx Vivado.

El objetivo de este documento es validar el correcto funcionamiento del sistema desarrollado, mostrando evidencias de la validación funcional realizada por bloques, por funciones y, por ende, al sistema completo.

3.2 Plan de validación

Prueba	Responsable	Bloque	Descripción	Prerequisitos	Procedimiento	Resultado esperado
HW1	Alumno	FT245_IF_WRITE	Se realiza la simulación del bloque encargado de la transmisión de datos desde la FPGA hasta el PC.	Xilinx Vivado 2018.3 o posterior.	Se realiza realiza un script de simulación simulando el envío de datos con el generador de patrón USB desde la FPGA hacia el dispositivo UM232H-B funcionando como FT245 asíncrono. Se ejecuta el script *FT245_IF_WRITE_tb.tcl* en el entorno de simulación integrado en el Software Xilinx Vivado.	El sistema garantiza los tiempos necesarios acorde a la especificación del dispositivo UM232H-B funcionando como FT245 asíncrono. Los datos que se reciben son correctos.
HW2	Alumno	FT245_IF_READ	Se realiza la simulación del bloque encargado de la transmisión de datos desde el PC hasta la FPGA	Xilinx Vivado 2018.3 o posterior.	Se realiza un script de simulación simulando la recepción de datos desde el dispositivo UM232H-B funcionando como FT245 asíncrono hacia la FPGA. Se ejecuta el script *FT245_IF_READ_tb.tcl* en el entorno de simulación integrado en el Software Xilinx Vivado.	El sistema garantiza los tiempos necesarios acorde a la especificación del dispositivo UM232H-B funcionando como FT245 asíncrono. Se genera la señal de *request* correctamente.
HW3	Alumno	FIFO	Se realiza la simulación del bloque encargado de almacenar los datos provenientes del sensor de imagen.	Xilinx Vivado 2018.3 o posterior.	Se realiza un script de simulación en el que se introducen y extraen datos desde el módulo FIFO hacia el bus de datos de salida. Se ejecuta el script *FIFO_tb.tcl* en el entorno de simulación integrado en el Software Xilinx Vivado.	El sistema almacena los datos existentes en el bus cuando se hace push y los saca al bus de salida cuando se hace pop. Los flags *empty* y *full* se generan correctamente.

HW4	Alumno	CGU	Se realiza la simulación del bloque que genera la señal de reloj para el módulo sensor.	Xilinx Vivado 2018.3 o posterior.	Se realiza realiza un script de simulación donde configura el reloj maestro y corre un tiempo determinado. Se ejecuta el script *CGU_tb.tcl* en el entorno de simulación integrado en el Software Xilinx Vivado.	La salida del módulo genera una señal estable de 25MHz.
HW5	Alumno	IMAGE_PATTERN	Se realiza la simulación del bloque encargado de generar imágenes emulando el comportamiento del sensor.	Xilinx Vivado 2018.3 o posterior.	Se realiza un script activando la señal de reloj y señal de activación al módulo generador de imagen sintética. Se ejecuta el script *IMAGE_CHANNEL_tb.tcl* en el entorno de simulación integrado en el Software Xilinx Vivado.	Se genera una imagen sintética correcta respetando la temporización de los datos de salida definida en la documentación del módulo sensor.
HW6	Alumno	IMAGE_CHANNEL	Se realiza la simulación del módulo encargado de coordinar y controlar el resto de bloques e interfaces externos. Se utiliza el patrón de imagen.	Xilinx Vivado 2018.3 o posterior.	Se realiza un script donde se simula la lectura de un dato proveniente del PC, y se habilita el generador de imagen sintética. Se ejecuta el script *TOP_tb.tcl* en el entorno de simulación integrado en el Software Xilinx Vivado.	El sistema gestiona correctamente el inicio de adquisición de imágenes, y envía la información útil hacia el bus de datos de salida.
SW1	Alumno	Instalación	Se realiza la instalación del software.	Sistema operativo Windows 7 o superior	El usuario ejecuta el instalador y sigue los pasos.	La aplicación se instala y ejecuta correctamente.
SW2	Alumno	Conexión y desconexión.	Se establece y se cierra la conexión de la plataforma software con el diseño hardware.	Aplicación abierta, Hardware encendido, y	El usuario ejecuta la aplicación. El usuario pulsa el botón *Connect*. El usuario pulsa el botón *Disconnect*.	El sistema se conecta y se desconecta correctamente al dispositivo.

				conectado al PC.		
SW3	Alumno	Modo vídeo	Se activa y desactiva la entrada de vídeo y visualiza por pantalla tanto la imagen como la tasa de FPS.	Aplicación abierta, Hardware encendido, y conectado al PC.	El usuario ejecuta la aplicación. El usuario pulsa el botón *Connect*. El usuario pulsa el botón *Start*. El usuario pulsa el botón *Stop*.	En la aplicación aparece la entrada de vídeo en tiempo real y es capaz de detenerse. En la aplicación se muestran en tiempo real, la tasa de FPS conseguida.
SW4	Alumno	Reconocimiento facial	Se activa y detiene el reconocimiento facial sobre el vídeo durante la adquisición.	Aplicación abierta, Hardware encendido, y conectado al PC.	El usuario ejecuta la aplicación. El usuario pulsa el botón *Connect*. El usuario activa el cuadro *Face detection.* El usuario pulsa el botón *Start*. El usuario desactiva el cuadro *Face detection.*	En la imagen aparecen los rostros con un recuadro sobre ellos durante la adquisición y luego se detiene la detección.
SW5	Alumno	Captura de imagen	Se lanza una adquisición y se guarda la imagen en el PC.	Aplicación abierta, Hardware encendido, y conectado al PC.	El usuario ejecuta la aplicación. El usuario pulsa el botón *Connect*. El usuario pulsa el botón *Snap*. El usuario indica la ruta para la imagen.	Se captura una imagen y se guarda correctamente en la ruta indicada.

Tabla 18: Plan de validación

3.3 Herramientas

El plan de pruebas descrito requiere el uso de herramientas capaces de monitorizar el sistema paso por paso, permitiendo garantizar la integridad del sistema para todas las situaciones plausibles, desde ejecución normal deseada, hasta la gestión de casos no deseables. La herramienta utilizada para tal efecto es presentada a continuación.

3.3.1 Vivado Simulator

El software Vivado Simulator (Figura 42) se encuentra integrado en la suite Vivado HLx WebPack. Esta herramienta soporta tanto Verilog, como VHDL, lenguaje de descripción hardware utilizado en el presente TFM. Esta herramienta permite, dado un script de simulación, escrito en lenguaje TCL, de homónima extensión (*.tcl*), conocer el comportamiento del hardware descrito, sin necesidad de utilizar un dispositivo lógico programable físico, lo que reduce el tiempo de desarrollo y facilita la detección de errores de diseño.

La simulación, además, permite forzar el sistema a situaciones potencialmente no deseadas que fuera muy difícil conseguir en un entorno físico real, con lo quese consigue probar el sistema en escenarios adversos para conocer su funcionamiento y, así, poder editar el diseño para robustecerlo frente a eventos no deseados.

Para realizar una simulación de un proyecto de Vivado, clicaremos la opción *Run Behavioral Simulation*. La simulación de comportamiento elegida realiza la simulación a nivel RTL, permitiendo comprobar que la sintaxis del hardware descrito es correcta, y que los sistemas funcionan comose pretendía.

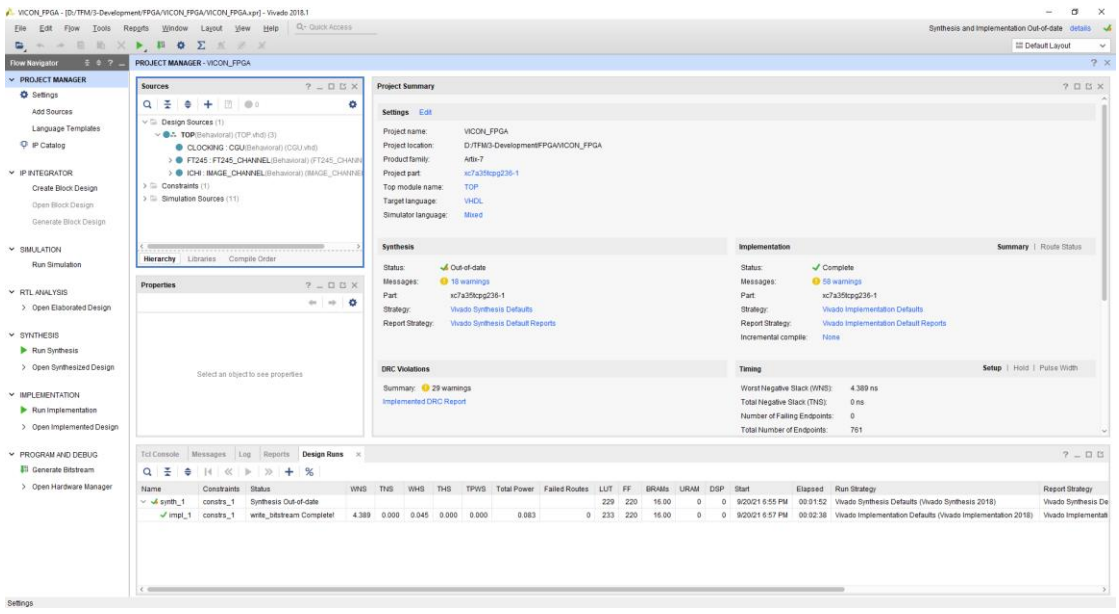


Figura 42: Vivado IDE

El resto de las simulaciones, post-síntesis y post-implementación, tienen como núcleo de simulación el *netlist* estructural, obtenido de los procesos de síntesis e implementación, y no el modelo RTL descrito en el desarrollo.

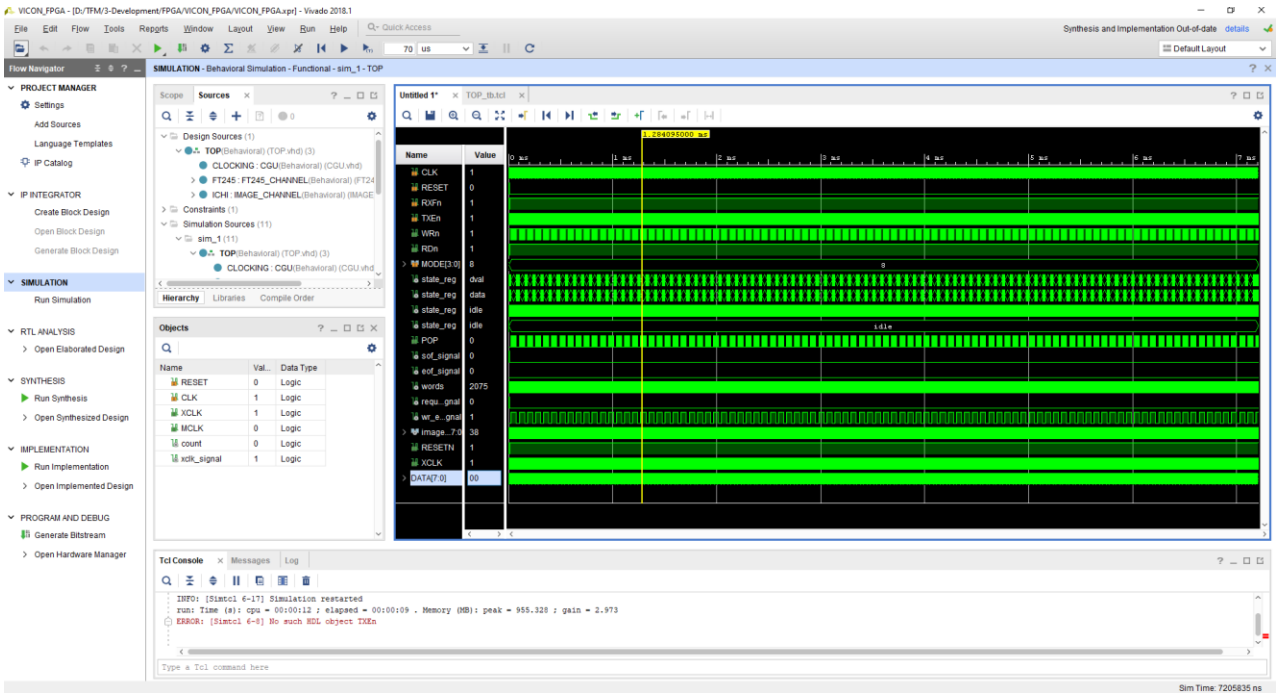


Figura 43: Simulación VIVADO

En la Figura 43, aparecen 4 ventanas cuyo significado es importante conocer:

- *Scope*. Muestra los bloques que componen el diseño, aparecerán todos los bloques que hayan sido instanciados en el top level.
- *Objects*. Aquí, aparecen las entradas, salidas y señales descritas en el módulo bajo simulación. Servirá para seleccionar qué señales se quieren mostrar en la simulación.
- *Ventana de simulación*. Aquí se arrastrarán desde la ventana *Objects* las entradas, salidas o señales cuyo comportamiento quiera ser observado.
- *Tcl Console*. En esta ventana, se introducen las instrucciones *tcl* para realizar la simulación deseada. En el presente diseño, se ha desarrollado un script *tcl* para cada simulación.

3.4 Resultados

Con el plan de pruebas descrito en la Tabla 18, se pretende confirmar la completa funcionalidad del sistema, tanto por bloques individuales, como en conjunto. Las pruebas realizadas se pueden dividir de la siguiente manera:

1. Pruebas hardware (HW1, HW2, HW3, HW4, HW5, HW6): Realizan la validación funcional del hardware descrito durante el desarrollo del presente TFM.
2. Pruebas software (SW1, SW2, SW3, SW4, SW5): Estas pruebas, una vez validado el hardware, se utilizan para validar el software desarrollado.

3.4.1 Pruebas hardware

3.4.1.1 HW1

En esta prueba se valida el correcto funcionamiento del bloque FT245_IF_WRITE. Como se vio en la sección de desarrollo, este bloque contiene una FSM encargada de generar las señales necesarias para el envío de datos desde la FPGA hacia el PC, por medio del módulo UM232-H-B, con el cual la FPGA se comunica.

En primer lugar, se ejecuta el comando *restart* para eliminar cualquier simulación previa existente en el apartado de simulación.

En segundo lugar, se establece la señal *CLK* para simular la entrada del reloj del sistema. Ésta consiste en una señal cuadrada con un duty cycle del 50%, y de frecuencia 100MHz. Acto seguido, se establecen las entradas necesarias para comprobar la entidad del bloque.

Se establece la señal de *RESET* al principio para garantizar los estados iniciales del sistema.

En concreto, se establece una señal *TXEn* periódica de 50ns y 100ns de ciclo, con el fin de comprobar el correcto funcionamiento del sistema a máxima velocidad y a la velocidad de operación esperada del sistema.

Se establecen una serie de datos aleatorios en el bus de comunicación con el FT245, así como una activación de la transmisión de datos mediante la señal *WRn*.

```
# Restart simulation
restart

# Input signals
add_wave {CLK}
add_wave {RESET}
add wave {ENABLE}
add_wave {DIN}
add_wave {TXEn}
# Internal signals
add_wave {state_reg}
# Output signals
add wave {WRn}
add_wave {READY}
add_wave {POP}
add_wave {DOUT}

# CLK at 100MHz (10ns) definition
add force {CLK} -radix bin {0 0ns} {1 5ns} -repeat every 10ns

# Init signals value
add_force {DIN} -radix hex  {00 0ns}
add_force {RESET} -radix hex {1 0ns} {0 10ns}
add_force {ENABLE} -radix bin {0 0ns} {1 10ns}
```

```
add_force {TXEn} -radix bin {1 0ns}
run 50ns

# Send data at max speed
# Data write timing:    DIN (t11=0)     0 (t8+t9=5+5=10)
# WREN timing:          0   (t8=5)      1   (t10=30)    0
# TXEN timing:          0   (t11=0)     1   (t6=14)     0 (t7=49)
add_force {DIN} -radix hex  {BB 0ns}
run 100ns
add_force {DIN} -radix hex  {5A 0ns} {1C 50ns} {53 100ns} {34 150ns} {CA 200ns}
add_force {TXEN} -radix bin {0 0ns} {1 40ns}            -repeat_every 50ns
run 300ns

# Send data at system speed
# Data write timing:    DIN (t11=0)     0 (t8+t9=5+5=10)
# WREN timing:          0   (t8=5)      1   (t10=30)    0
# TXEN timing:          0   (t11=0)     1   (t6=14)     0 (t7=49)
add_force {DIN} -radix hex  {5A 0ns} {1C 100ns} {53 200ns} {34 300ns} {CA 400ns}
add_force {TXEN} -radix bin {0 0ns} {1 10ns}            -repeat_every 100ns
run 600ns

# End of communication
add_force {ENABLE} -radix bin {0 0ns}
run 50ns
```

Tras la ejecución, el simulador muestra los resultados de la Figura 44:

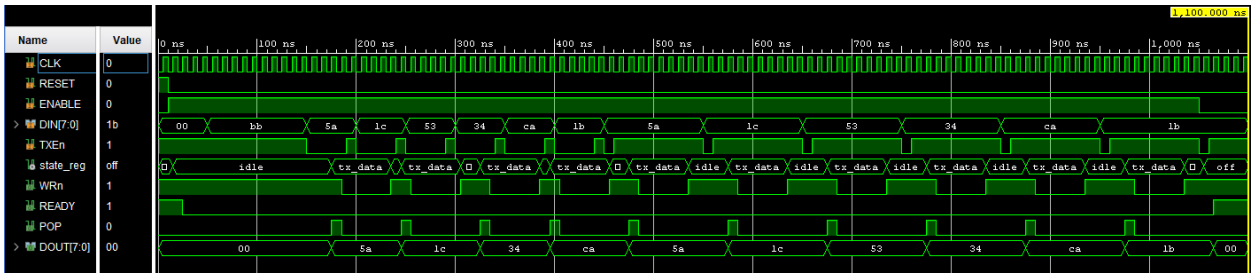


Figura 44: Resultado simulación prueba HW1

A la vista del resultado se puede concluir lo siguiente:

- La señal ready se activa cuando está activo el estado *off*.
- La FSM espera correctamente a que la señal *TXEn* alcance el valor '0' para realizar el resto de sincronismos. En caso de que *TXEn* alcance el valor '0' durante una transmisión, el sistema la ignora y espera a terminar la transferencia actual.
- Se garantiza el tiempo mínimo de 5ns de hold del dato antes del flanco de bajada de *WRn*.
- Se garantizan 40ns de *WRn* a nivel bajo, cumpliendo las especificaciones del FT245.
- En *tx_data*, la máquina genera un *pop* para extraer un dato de la FIFO al bus de datos. Además, el dato permanece en el bus, por lo que se garantiza el tiempo mínimo de setup.

Con lo descrito anteriormente, se concluye que la prueba ha sido exitosa.

3.4.1.2 HW2

En esta prueba se valida el correcto funcionamiento del bloque FT245_IF_READ. Como se vio en la sección de desarrollo, este bloque contiene una FSM encargada de generar las señales necesarias para la recepción de datos desde el PC hacia la FPGA, por medio del módulo UM232-H-B, con el cual la FPGA se comunica.

En primer lugar, se ejecuta el comando *restart* para eliminar cualquier simulación previa existente en el apartado de simulación.

En segundo lugar, se establece la señal *CLK* para simular la entrada del reloj del sistema. Ésta consiste en una señal cuadrada con un duty cycle del 50%, y de frecuencia 100MHz. Acto seguido, se establecen las entradas necesarias para comprobar la entidad del bloque.

Se establece la señal de *RESET* al principio para garantizar los estados iniciales del sistema.

En concreto, se establece una señal *RXFn* periódica de 100ns de ciclo, con el fin de simular peticiones de adquisición de imagen por parte del PC.

Se establecen una serie de datos aleatorios en el bus de comunicación con el FT245, así como una activación de la transmisión de datos mediante *RDn.*

```
# Restart simulation
restart

# Input signals
add_wave {CLK}
add_wave {RESET}
add_wave {ENABLE}
add_wave {DIN}
add_wave {RXFn}
# Internal signals
add_wave {state_reg}
# Output signals
add_wave {RDn}
add_wave {REQUEST}
add_wave {DOUT}

# CLK at 100MHz (10ns) definition
add_force {CLK} -radix bin {0 0ns} {1 5ns} -repeat_every 10ns

# Init signals value
add_force {DIN} -radix hex {00 0ns}
add_force {RESET} -radix hex {1 0ns} {0 10ns}
add_force {ENABLE} -radix bin {0 0ns} {1 10ns}
add_force {RXFn} -radix bin {1 0ns}
run 50ns

# Data read timing:    DIN (t3+t3=15+15=30ns)
# RXFn timing:          0   (t11=0)     1 (t1=14)   0 (t2=49)
add_force {DIN} -radix hex {BB 0ns}
run 1us
add_force {DIN} -radix hex {5A 0ns} {1C 1us}
add_force {RXFn} -radix bin {0 0ns} {1 50ns}
run 2us

# End of communication
add_force {ENABLE} -radix bin {0 0ns}
run 50ns
```

Tras la ejecución, el simulador muestra los resultados de la Figura 45:

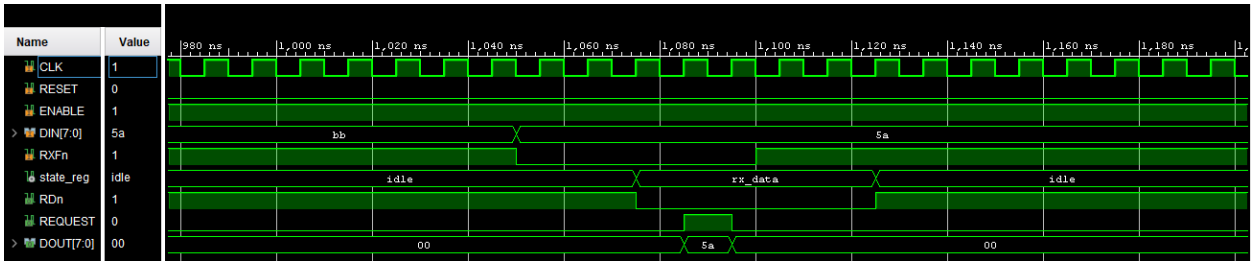


Figura 45: Resultado simulación prueba HW2

A la vista del resultado se puede concluir lo siguiente:

- La FSM espera correctamente a que la señal *RXFn* alcance el valor '0' para realizar el resto de sincronismos. En caso de que *RXFn* alcance el valor '0' durante una transmisión, el sistema la ignora y espera a terminar la transferencia actual.
- Se garantizan 40ns de *RDn* a nivel bajo, cumpliendo las especificaciones del FT245.
- En *rx_data*, la máquina genera una señal de *request* para indicar al controlador del sistema que se ha solicitado una imagen.

Con lo descrito anteriormente, se concluye que la prueba ha sido exitosa.

3.4.1.3 HW3

En esta prueba se valida el funcionamiento del módulo FIFO. Este módulo contiene la definición de una memoria FIFO RAM, de ancho de bus y de longitud de palabra configurable por diseño en el top level que, por defecto, se configura a 16 bits de dirección y 8 bits de palabra, lo que hace un total de 64KB. Si bien, con efectos de hacer la simulación más práctica y representativa, se han establecido los parámetros de la memoria modelando una memoria de 3 bits de dirección y 8 bits de palabra, pudiendo almecenar un total de 8 datos. De este modo, será posible verificar cómo la memoria crea correctamente sus flags de estado, y bloquea la lectura y escritura cuando está llena o vacía.

En el script de simulación, se ejecuta el comando *restart* para eliminar cualquier simulación previa existente en el apartado de simulación como primer paso.

En segundo lugar, se establece la señal *CLK* para simular la entrada del reloj del sistema. Ésta consiste en una señal cuadrada con un duty cycle del 50%, y de frecuencia 100MHz. Acto seguido, se establecen las entradas necesarias para comprobar la entidad del bloque. Se establece la señal de reset al principio para garantizar los estados iniciales del sistema.

Se establece una serie de señales *pop* y *push* de tal manera que la simulación permita contemplar todos los escenarios posibles en la gestión de la pila. Adicionalmente, se establecen diversos datos en el bus de entrada que posteriormente serán extraídos mediante la instrucción *pop*.

```
# Restart simulation
restart

# Input signals
add_wave {CLK}
add_wave {RESET}
add_wave {DIN}
add_wave {PUSH}
add_wave {POP}
# Internal signals
add_wave {ram}
add_wave {words}
#Output signals
add_wave {DOUT}
add_wave {FULL}
add_wave {EMPTY}

# CLK at 100MHz (10ns) definition
add_force {CLK} -radix bin {0 0ns} {1 5ns} -repeat_every 10ns

# RESET
add_force {RESET} -radix hex {1 0ns} {0 10ns}
```

```
add_force {PUSH} -radix bin {0 0ns}
add_force {POP} -radix bin {0 0ns}
run 20ns

# PUSH with FIFO - Entry one dta
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {DIN} -radix hex {00 0ns}
run 10ns

# PUSH and POP at the same time with data in FIFO
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {POP} -radix bin {1 0ns} {0 10ns}
run 10ns

# PUSH with data - readout priority. 0x03 will be lost
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {DIN} -radix hex {01 0ns}
run 10ns
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {DIN} -radix hex {02 0ns}
run 10ns
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {DIN} -radix hex {03 0ns}
run 10ns
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {DIN} -radix hex {04 0ns}
run 10ns
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {DIN} -radix hex {05 0ns}
run 10ns
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {DIN} -radix hex {06 0ns}
run 10ns
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {DIN} -radix hex {07 0ns}
run 10ns

# PUSH and POP with FIFO full
add_force {POP} -radix bin {1 0ns} {0 10ns}
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
run 10ns

# POP with FIFO full
add_force {POP} -radix bin {1 0ns} {0 20ns}
run 20ns
# FIFO is empty
add_force {POP} -radix bin {1 0ns} {0 20ns}
run 50ns

# PUSH and POP at the same time with FIFO empty
add_force {PUSH} -radix bin {1 0ns} {0 10ns}
add_force {POP} -radix bin {1 0ns} {0 10ns}
run 50ns
```

En la Figura 46 se puede observar el resultado obtenido de la simulación:

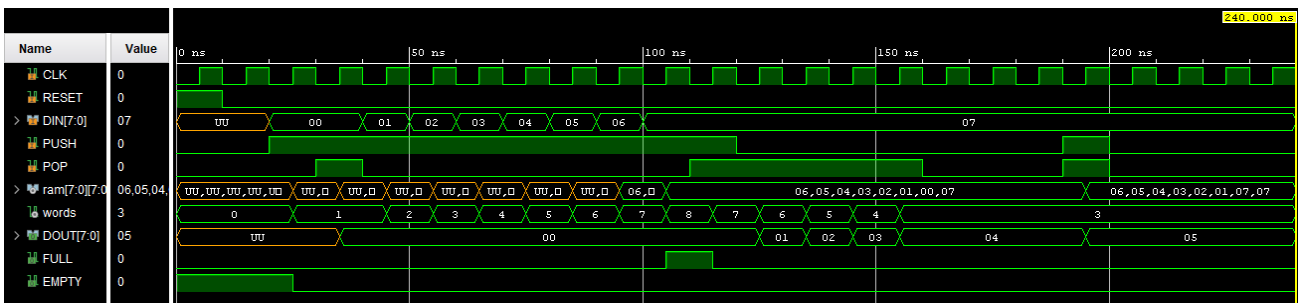


Figura 46: Resultado simulación prueba HW3

A la vista del resultado se puede concluir lo siguiente:

- La señal *empty* permanece a nivel lógico '1' siempre y cuando la cuenta indica que no hay datos. Se desactiva cuando existe al menos un dato en la pila. Cuando ésta está activa, se ignoran las peticiones de extracción de dato (*pop*) correctamente.
- En cada *pop*, el vector de lectura se incrementa, y el contador general se decrementa en una unidad.
- En cada *push*, el vector de escritura se incrementa, y el contador general también incrementa en una unidad.
- La señal *full* permanece a nivel lógico '1' cuando se ha alcanzado el límite de escritura, coincidente cuando el valor de la cuenta alcanza el valor 2^B, siendo B el ancho del bus de direcciones. Se desactiva cuando existe al menos una posición libre en la pila. Cuando ésta está activa, se ignoran las peticiones de inserción de dato (*push*) correctamente.
- Existe consistencia entre los datos escritos y leídos. El orden en el que entran a la pila (bus de datos *DIN[7:0]*) con las instrucciones *push*, es igual al orden en que salen de la misma (bus de datos *DOUT[7:0]*) con las instrucciones *pop*, cumpliendo la arquitectura FIFO.

Con lo descrito anteriormente, se concluye que la prueba ha sido exitosa.

3.4.1.4 HW4

En esta prueba se valida el funcionamiento del módulo CGU. Este módulo es el más sencillo de los que componen el proyecto, y consta de un divisor de frecuencia para generar los 25MHz que el módulo MT9V111 requiere en su entrada para su funcionamiento.

Este script de simulación es el más sencillo de todos: se ejecuta el comando *restart* para eliminar cualquier simulación previa existente en el apartado de simulación como primer paso.

Después, se establece la señal *CLK* para simular la entrada del reloj del sistema. Ésta consiste en una señal cuadrada con un *duty cycle* del 50%, y de frecuencia 100MHz. Por último, se ejecuta un *RESET* y se deja correr la simulación.

```
# Restart simulation
restart

# Input signals
add_wave {CLK}
add_wave {RESET}
# Output signals signals
add_wave {XCLK}
add_wave {MCLK}

# CLK at 100MHz (10ns) definition
add_force {CLK} -radix bin {0 0ns} {1 5ns} -repeat_every 10ns

# Init signals value
add_force {RESET} -radix hex {1 0ns} {0 10ns}
run 200ns
```

En la Figura 47 se puede observar el resultado obtenido de la simulación:

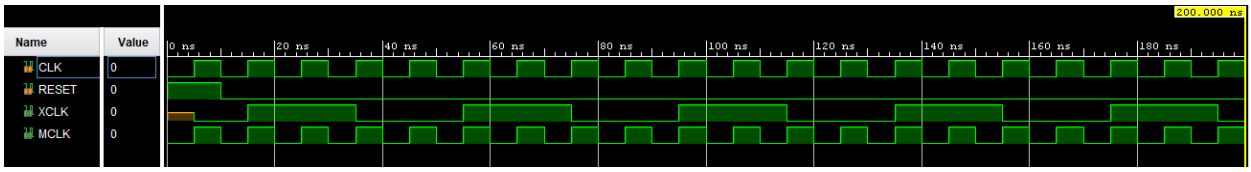


Figura 47: Resultado simulación prueba HW4

En vista a que se genera una señal periódica a 25MHz correctamente, se concluye que la prueba ha sido exitosa.

3.4.1.5 HW5

En esta prueba, se valida el funcionamiento del bloque IMAGE_PATTERN, bloque encargado de la simulación manual de las señales del módulo MT9V111.

Este módulo, que consiste en una máquina de estados con diversos contadores, de tal manera que generan dos imágenes conocidas de tamaño 640x480 píxeles, de manera exacta a cómo funciona el módulo MT9V111 real. En concreto, las imagenes generada son las mostradas en la Figura 48 y la Figura 49, que, en adelante, llamaremos imagen de test.

Figura 48: Imagen de test vertical

Figura 49: Imagen de test horizontal

En concreto, para este bloque, el script de simulación utilizado es muy sencillo, únicamente activando la entrada enable del módulo MT9V111.

```
# Restart simulation
restart

# Input signals
add_wave {XCLK}
add_wave {PCLK}
add_wave {RESET}
add_wave {RAMP}
# Internal signals
add_wave {count_line}
add_wave {count_pixel}
add_wave {state_reg}
add_wave {dout_vramp_reg}
add_wave {dout_hramp_reg}
# Output signals
add_wave {FVAL}
add_wave {LVAL}
add_wave {DOUT}

# XCLK at 25MHz (40ns) definition
add_force {XCLK} -radix bin {0 0ns} {1 20ns} -repeat_every 40ns

# Add inputs
add_force {RESET} -radix hex {1 0ns} {0 1000ns}
add_force {RAMP} -radix hex {0 0ns}
run 1us

# Emulate vertical ramp (2 frames)
add_force {RAMP} -radix hex {0 0ns}
run 70ms

# Emulate horizontal ramp (1 frame)
add_force {RAMP} -radix hex {1 0ns}
run 38ms
```

En la Figura 50 se puede observar el resultado obtenido de la simulación:

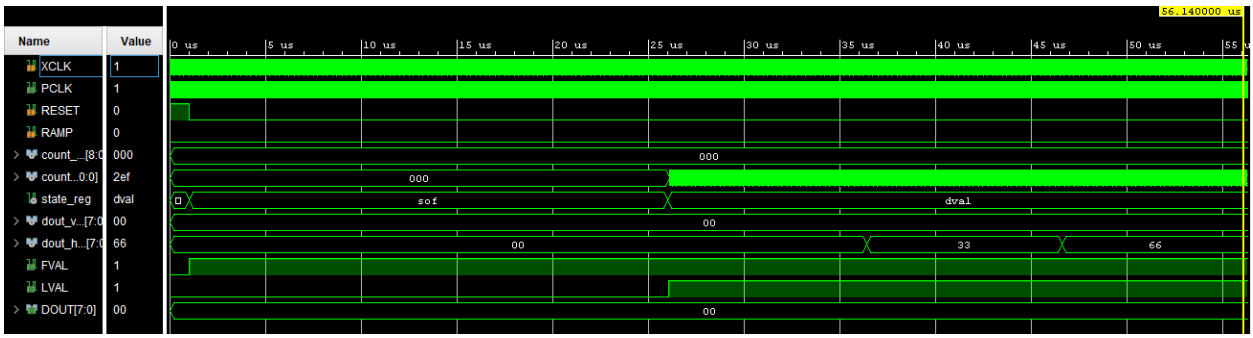


Figura 50: Resultado simulación prueba HW5 (I)

Se aprecia cómo el módulo arranca, directamente activando la señal *FVAL*, indicador de comienzo de frame. También cabe destacar que el módulo comienza la primera línea de la imagen. Tras finalizar la cuenta en el contador *sof*, la FSM cambia de estado, activa la señal *LVAL* indicando que los datos en el bus son válidos, y arranca el contador de píxeles.

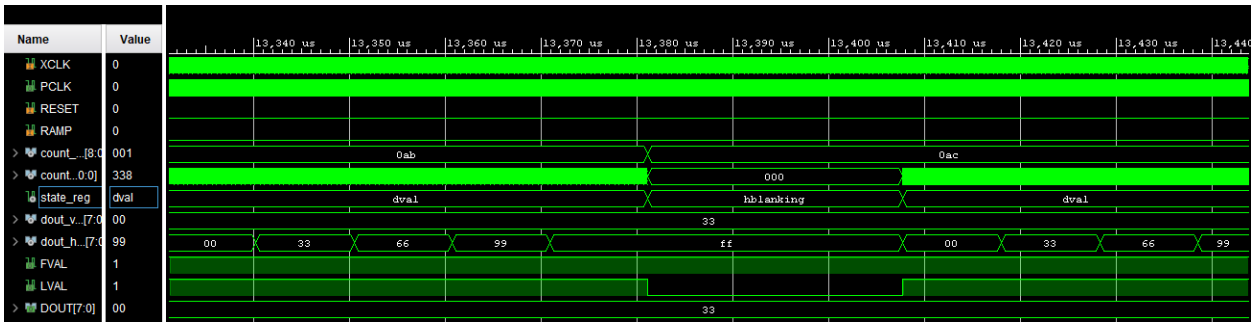


Figura 51: Resultado simulación prueba HW5 (II)

Tras el final de la línea, indicado con el nivel bajo de la señal LVAL, comienza el horizontal blanking entre líneas, controlado con el contador *hblanking*, como se detalla en la Figura 51.

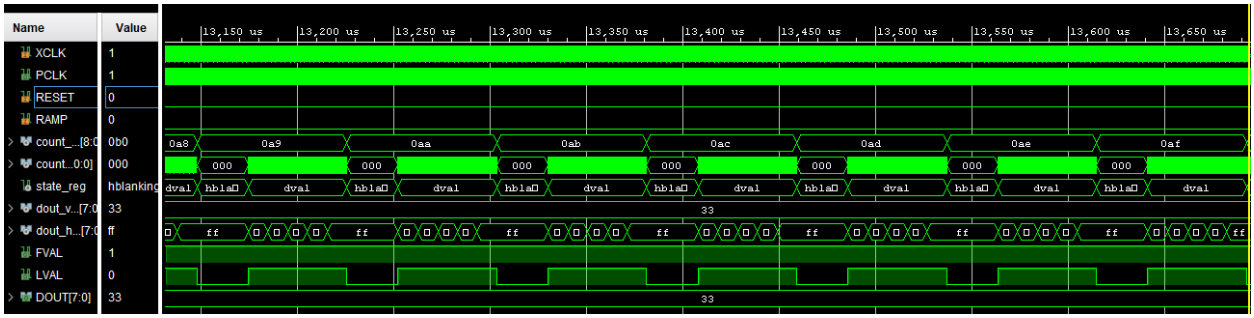


Figura 52: Resultado simulación prueba HW5 (III)

Como se ve en la Figura 52 con un zoom más lejano, el horizontal blanking es repetitivo entre líneas, y se ejecuta 479 veces.

Por último, una vez completado un frame, indicado con el nivel bajo en la señal *FVAL*, el bloque espera un tiempo determinado a emitir el siguiente. Este tiempo es el correspondiente al vertical blanking, y se corresponde con 20778 ciclos de *XCLK* como se puede ver en la Figura 53:

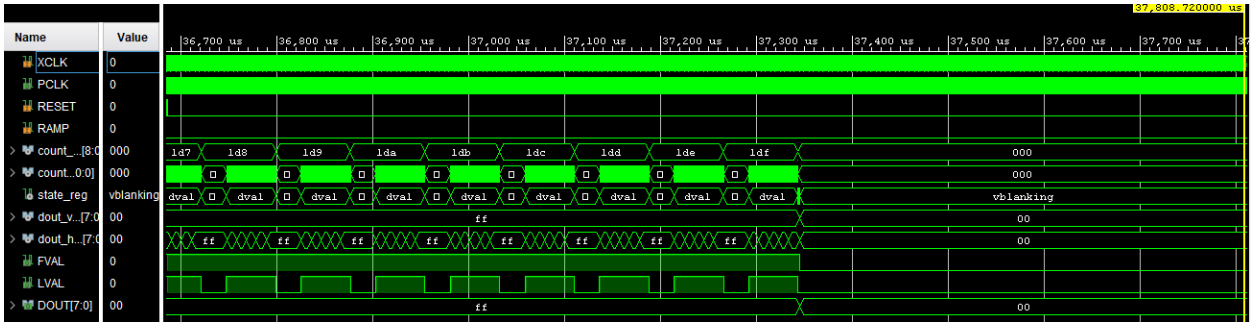


Figura 53: Resultado simulación prueba HW5 (IV)

Con estas simulaciones, se da por concluida la simulación del bloque para test con resultado satisfactorio, pues el bloque replica a la perfección todos y cada uno de los diversos estados y tiempos que el sensor MT9V111 real genera.

3.4.1.6 HW6

Una vez el bloque para simulación ha sido validado, se realiza la simulación del sistema completo, donde se pretende validar el buen funcionamiento del módulo IMAGE_CHANNEL, así como de la interconexión de todos los bloques, para dar por finalizadas las pruebas en cuanto al hardware desarrollado con el siguiente script:

```
# Restart simulation
restart

# Input signals
add_wave {CLK}
add_wave {RESET}
add_wave {RXFn}
add_wave {TXEn}
add_wave {MODE}
# Internal signals
add_wave {ICHI/PATTERN/state_reg}
add_wave {ICHI/PATTERN/dout_vramp_reg}
add_wave {ICHI/PATTERN/dout_hramp_reg}
add_wave {ICHI/RECEIVER/state_reg}
add_wave {FT245/IFWRITE/state_reg}
add_wave {FT245/POP}
add_wave {ICHI/RECEIVER/sof_signal}
add_wave {ICHI/RECEIVER/eof_signal}
add_wave {ICHI/PATTERN/DOUT}
add_wave {ICHI/FIFO/words}
add_wave {request_signal}
add_wave {wr_en_signal}
add_wave {image_channel_data}
# Output signals
add_wave {RESETN}
add_wave {XCLK}
add_wave {WRn}
add_wave {RDn}
add_wave {DATA}

# CLK at 100MHz (10ns) definition
add_force {CLK} -radix bin {0 0ns} {1 5ns} -repeat_every 10ns

# Init signals value
add_force {RESET} -radix hex {1 0ns} {0 10ns}
add_force {RXFn} -radix bin {1 0ns}
add_force {TXEn} -radix bin {1 0ns}
add_force {MODE} -radix bin {00 0ns}
run 50ns

# Request data from PC
add_force {RXFn} -radix bin {0 0ns} {1 10ns}
run 1ms

# Configure vertical ramp pattern
add_force {MODE} -radix bin {10 0ns}
add_force {TXEN} -radix bin {0 0ns} {1 10ns} {0 60ns}    -repeat_every 100ns
run 80ms

# Configure horizontal ramp pattern
add_force {MODE} -radix bin {11 0ns}
add_force {TXEN} -radix bin {0 0ns} {1 10ns} {0 60ns}    -repeat_every 100ns
run 80ms

# End of communication
add_force {TXEn} -radix bin {1 0ns}
run 60ns
```

Con el que se inicializa el sistema, se simula una petición de dato desde el PC, y se establece una señal periódica de transmisión de 100ns, siendo éste el caso peor según el datasheet suministrado por el fabricante.

Con ello, los resultados se muestran en la Figura 54, Figura 55, y Figura 56:

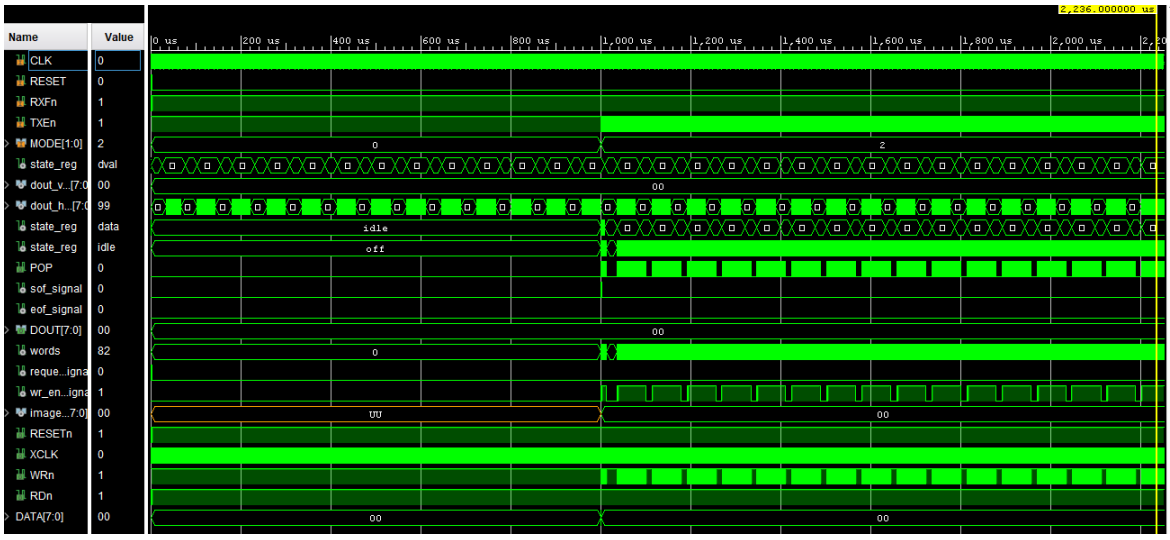


Figura 54: Resultado simulación prueba HW6 (I)

Figura 55: Resultado simulación prueba HW6 (II)

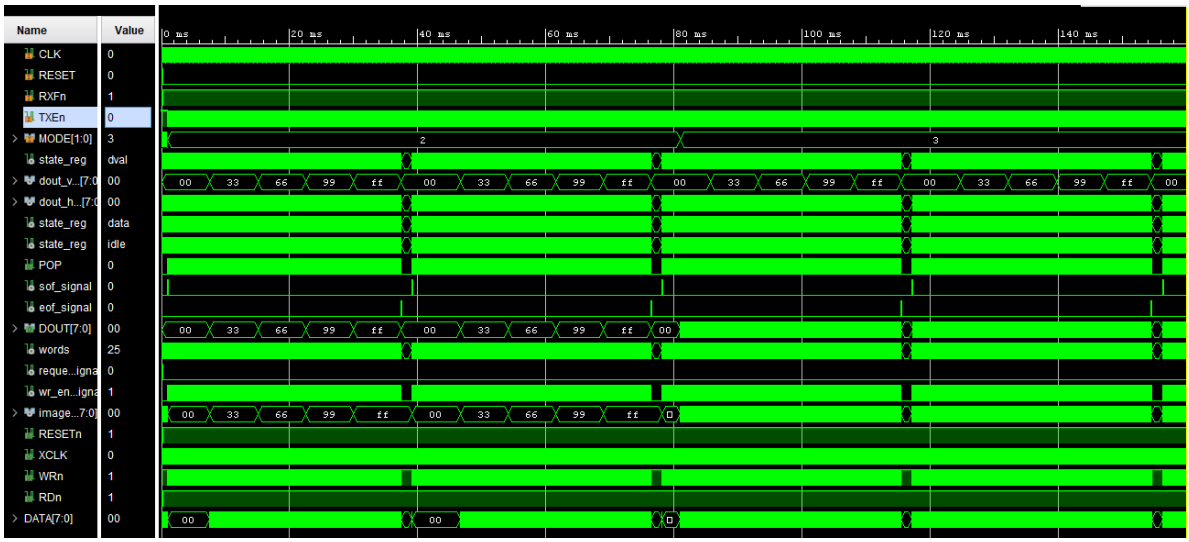


Figura 56: Resultado simulación prueba HW6 (III)

En vista de los resultados obtenidos, se obtienen las siguientes conclusiones:

- Las señales de habilitación de lectura (*RDn*) y escritura (*WRn*) se activan correctamente según la máquina de estados descrita.
- La señal *WRn* no se activa cuando la FIFO está vacía.
- La salida de datos se activa tras recibir una petición desde el PC.
- La señal *pop* se ejecuta siempre que la FIFO no está vacía.
- La señal *push* se ejecuta siempre que la FIFO no está llena, y los datos en el bus corresponden a píxeles válidos. En caso de solicitud de frame durante una transmisión del mismo desde el MT9V111 hacia la FPGA, la solicitud queda activa, a la espera de la llegada de un nuevo frame.

Con los resultados obtenidos por simulación, se puede dar el bloque global por validado, pues funciona según lo esperado.

3.4.2 Pruebas software

Una vez validado el hardware desarrollado, se realizan pruebas para comprobar el desarrollo software del presente TFM. Este software debe funcionar en concordancia con el hardware, de ahí que éste último se haya validado en primer lugar.

3.4.2.1 SW1

Con esta prueba, se comprueba que la instalación del software desarrollado es satisfactoria y no arroja problemas. Para ello, se ejecuta el instalador, y se siguen los pasos descritos en las siguientes figuras:

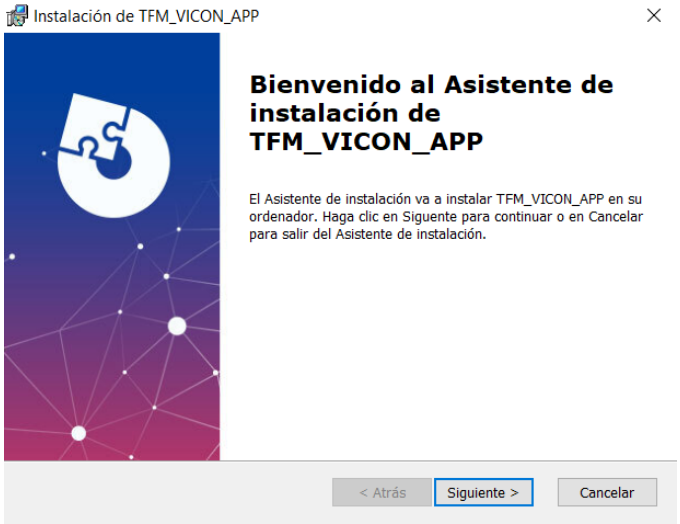


Figura 57: Asistente de instalación de la aplicación software

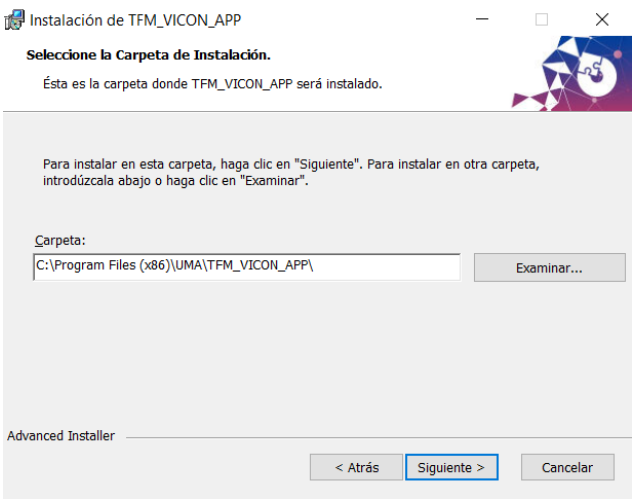


Figura 58: Selección de ruta de instalación de la aplicación software

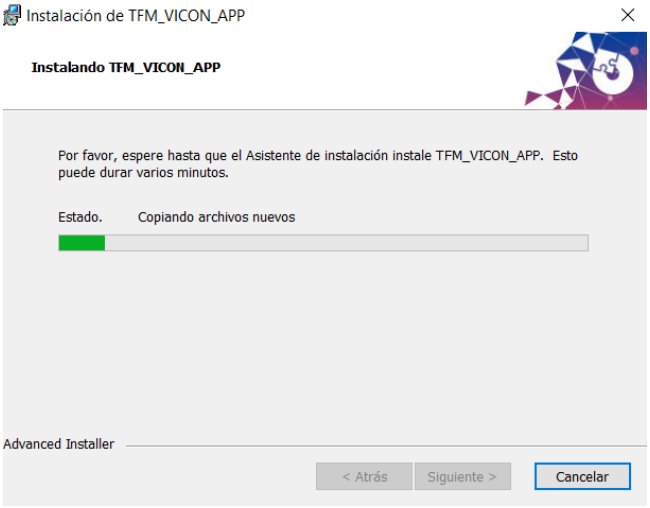


Figura 59: Instalación en progreso de la aplicación software

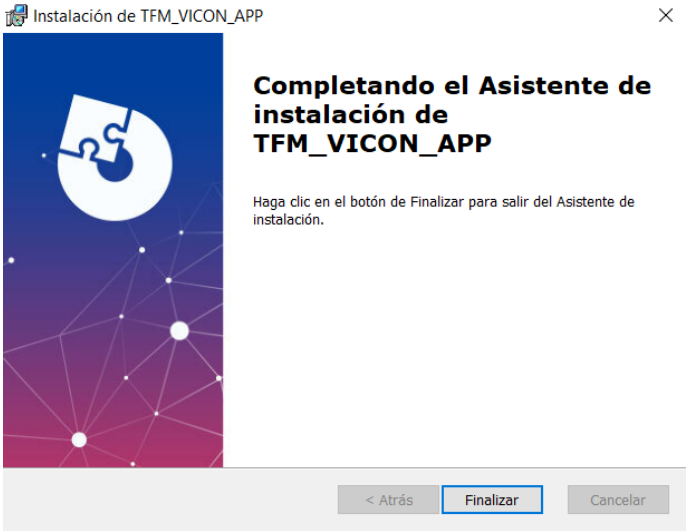


Figura 60: Instalación de la aplicación software finalizada

Figura 61: Ejecutable de la aplicación software

En vista a las figuras, la aplicación se ejecuta correctamente tras la instalación, lo que significa que ésta ha sido satisfactoria, con lo que se considera satisfactoria la prueba de instalación.

3.4.2.2 SW2

Para la prueba de conexión y desconexión, se ejecuta la aplicación y se pulsa sobre el botón *Connect.*

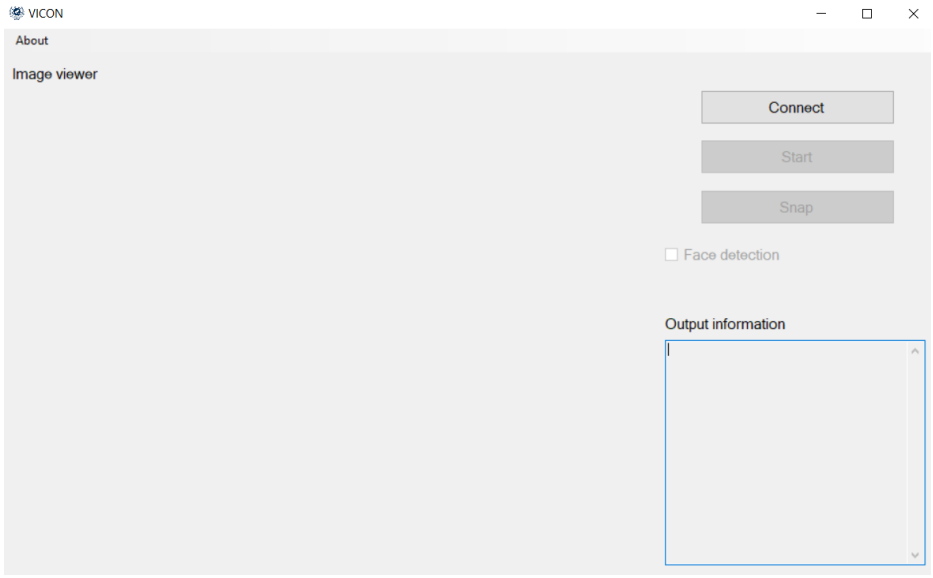


Figura 62: Aplicación TFM VICON

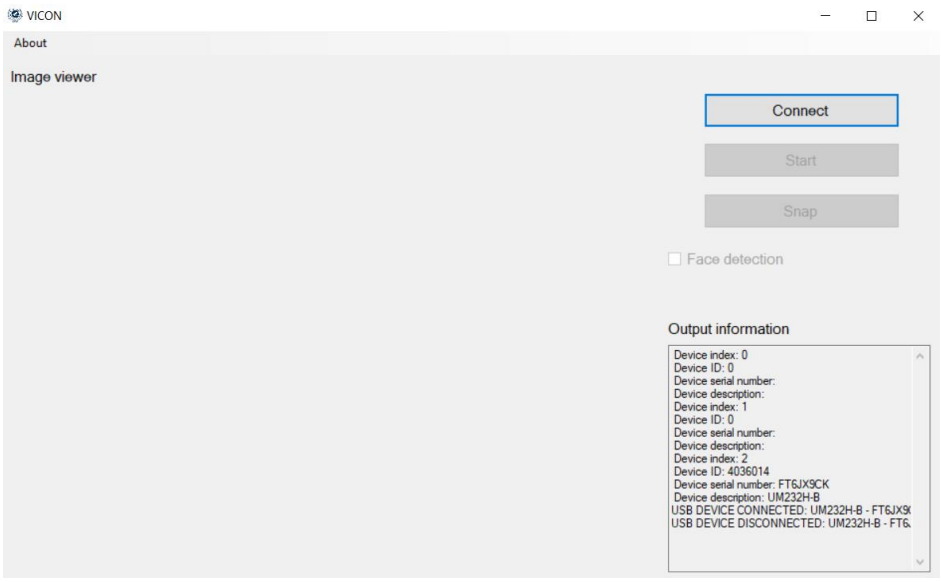


Figura 63: Resultado prueba software SW2

Visto el resultado de la Figura 63, la prueba concluye como exitosa.

3.4.2.3 SW3

El objetivo de esta prueba es que en la aplicación aparezca la entrada de vídeo en tiempo real y se pueda detener, además de mostrar la tasa de FPS en tiempo real. Para ello tras conectarse a la placa, se pulsa el botón de inicio *Start.*

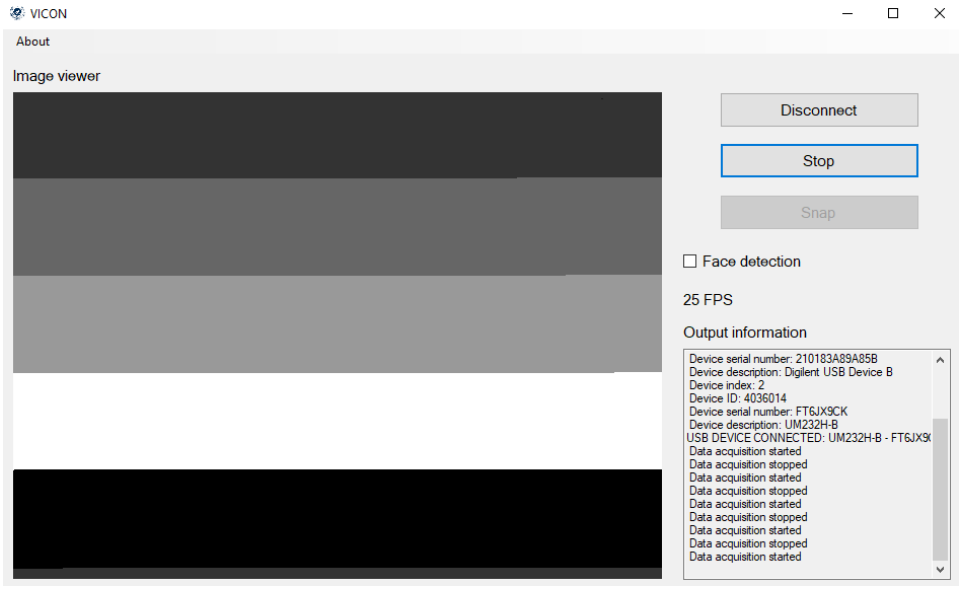


Figura 64: Resultado prueba software SW3

En vista al resultado obtenido en la Figura 64, se aprecia que la entrada de vídeo funciona correctamente, así como el indicador de tasa de FPS.

3.4.2.4 SW4

Para esta prueba, se activa el checkbox *Face detection* con el vídeo activo, y se comprueba que el sistema es capaz de detectar uno o más rostros en su campo visual.

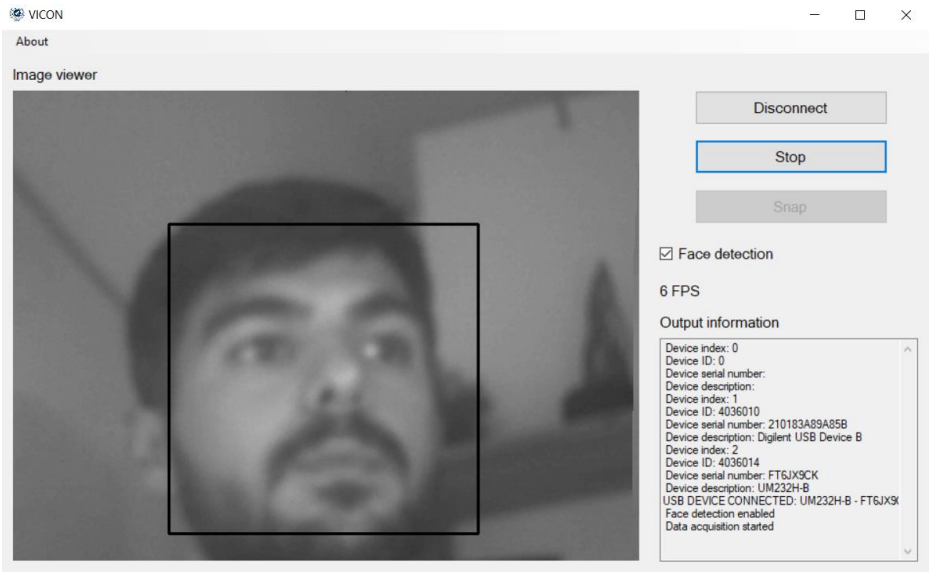


Figura 65: Resultado prueba software SW4

Los resultados de la Figura 65 muestran la capacidad del sistema para la detección de rostros anteriormente indicada por lo que la prueba se considera satisfactoria.

3.4.2.5 SW5

El objetivo de esta prueba es capturar una imagen y guardarla correctamente en la ruta indicada como se aprecia en la Figura 66.

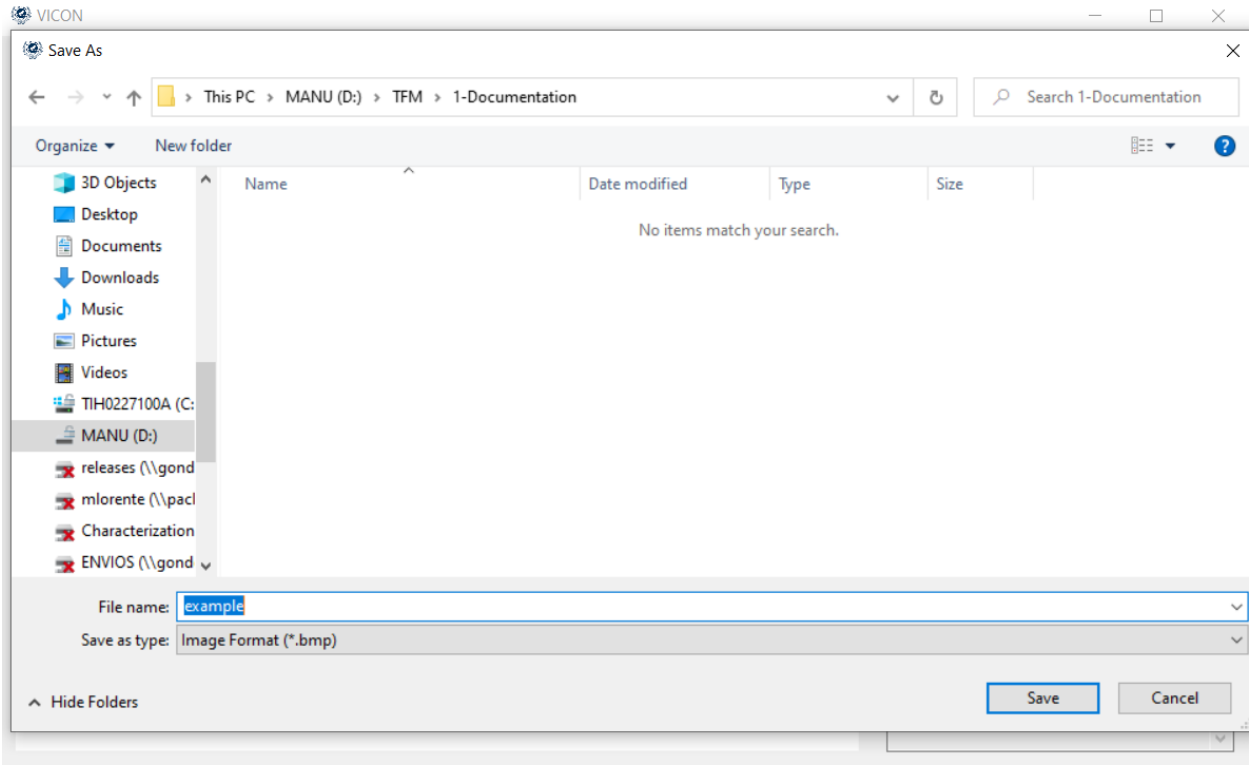


Figura 66: Resultado prueba software SW5

Figura 67: Imagen capturada en prueba software SW5

Como puede apreciarse en la Figura 67, la imagen se ha capturado y tiene el tamaño deseado de 640x480 por lo que se considera la prueba como exitosa.

4 CONCLUSIONES

4.1 Conclusiones

Finalizado el desarrollo, implementación y pruebas del presente TFM, se ha observado cómo se han cumplido todos los requisitos que no eran opcionales.

Si bien durante el desarrollo se han determinado numerosos defectos iniciales, funcionamientos no deseados, e inclusive situaciones inesperadas y no identificadas previamente, que han requerido numerosas simulaciones y pruebas, inclusive, con vaias iteraciones con el tutro del proyecto para depurar el diseño.

Gracias a la identificación de los problemas durante la fase de desarrollo, se ha conseguido un sistema robusto y funcional, capaz de cumplir con todos los requisitos especificados en el documento inicial, e incluso, entrando en detalle, reproducir vídeo a una tasa cercana a la máxima que el sensor puede proporcionar, 26 FPS, teniendo el objetivo en 10 FPS para el proyecto.

Parte de la robustez de la reproducción de vídeo proviene del mecanismo de sincronismo implementado a nivel hardware, donde se identifica cuál es el dato válido de cada transmisión desde el módulo. De haber implementado esta solución por software, las garantías de funcionamiento serían menores, ya que los datos provienen de una memoria FIFO, en la que realmente no se sabe qué datos existen almacenados.

En cuanto al software, se ha tratado en todo momento que la interfaz de usuario fuese lo más amigable para el usuario como compacta en cuanto a contenido. El vídeo, el logger, la tasa de FPS, y la detección de rostros se encuentran integradas en una misma ventana principal donde también se configura la conexión y desconexión con el sistema. Además, la funcionalidad añadida para capturar imágenes nos añade la posibilidad de tomar instantáneas sin interrumpir el vídeo en tiempo real. Estas instantáneas pueden ser utilizadas con fines demostrativos o simplemente como inmortalización del momento en el que se realiza.

En cuanto a las sensaciones más personales, se tiene la sensación de haber desarrollado un sistema muy sencillo, sin la dificultad aparente que presenta cuando se habla de él. Sin embargo, echando la vista atrás, hace no demasiados años, este desarrollo habría sido un auténtico quebradero de cabeza, modelando con puertas lógicas físicas, cableando, y con nulas posibilidades de simular y depurar los sistemas realizados. Queda muy claro y demostrado cómo la implementación de sistemas digitales con FPGA reducen enormemente los tiempos de desarrollo y, por tanto, los tiempos que tarda el producto en llegar al mercado. Además, estos sistemas permiten, en un mismo IDE, realizar simulaciones de comportamiento de lo diseñado.

También merece especial mención el uso del módulo UM232-H-B-WE. Este módulo permite realizar un diseño con conectividad USB (dentro de sus limitaciones), sin necesidad de implementar en el diseño todo el protocolo USB, sino que permite utilizar protocolos infinitamente más sencillos, como RS232, RS485, o, por ejemplo, el utilizado en el presente TFM, el protocolo basado en FIFO FT245.

En cuanto al software, se han implementado funciones de librerías escritas en C++, mediante un desarrollo en Visual C#, gracias a la amplia comunidad de desarrolladores disponible en internet, y gracias a la filosofía open-

source, que tiene sin duda un impacto enormemente positivo en el desarrollo de aplicaciones, tanto educativas, como personales o comerciales.

En líneas generales, gracias a la tecnología actual, se ha podido desarrollar una aplicación compleja, de una manera asequible, y sin necesidad de tener un laboratorio para tal fin.

4.2 Posibles mejoras y líneas de trabajo

Se plantean diferentes mejoras al presente proyecto:

- Añadir adquisición de vídeo a la aplicación software. Podría considerarse, de igual manera que se capturan imágenes, poder capturar fragmentos del vídeo entrante.
- Comunicación I2C con el módulo MT9V111. Esto permitiría tener acceso a más configuraciones del sensor, como control del contraste, brillo, color, etc.
- Procesado de imágenes a color. Si bien en el presente TFM se manejan imágenes en escala de grises por ser uno de los requisitos del proyecto, los datos de crominancia llegan a la FPGA, sólo que se desprecian. En líneas futuras, podría trabajarse el procesado de imágenes a color.
- Implementación de salida física de vídeo. Aprovechando la salida VGA que tiene la placa de desarrollo BASYS 3, podría implementarse una salida de vídeo directa desde los datos del sensor, de tal manera que, sin necesidad de un ordenador, el usuario fuese capaz de ver las imágenes captadas.

En cuanto a líneas de trabajo futuro, teniendo en cuenta alguna de las mejoras anteriormente descritas:

- Sistemas de acceso. Haciendo uso del SoC más potente, se podría evolucionar el sistema hasta uno que detecte no caras, sino identidades clasificando los rostros, teniendo de esta manera un sistema de acceso de bajo coste.
- Sensor más avanzado. Con base en el proyecto aquí desarrollado, haciendo uso de un sensor de imagen más avanzado con un interfaz de salida más convencional, y del SoC, se podría desarrollar una videocámara más avanzada con mayor resolución, y mayor velocidad, de igual manera, con coste menor que los sistemas comerciales.

ANEXO I. CÓDIGO FUENTE

Tanto el código HDL del diseño hardware, como todo el código de la aplicación software se encuentran alojados en el siguiente repositorio de GitHub: https://github.com/manulorente/VICON

ANEXO II. ÍNDICE DE FIGURAS

ANEXO III. ÍNDICE DE TABLAS

ANEXO IV. ACRÓNIMOS

API: Application Programming Interface

APP: Aplicación

BMP: Bitmap Image

CMOS: Complementary Metal Oxide Semiconductor

DLL: Dynamic-Link Library

EEPROM: Electrically Erasable Programmable Read-Only Memory

E/S: Entradas/salidas

FIFO: First In First Out

FPGA: Field Programmable Gate Array

FPS: Frames per second

FSM: Finite State Machine

GUI: Graphic User Interface

HDL: Hardware Description Language

HMI: Huma Machine Interface

HW: Hardware

IDE: Integrated Development Environment

I2C: Inter-integrated circuits

IPC: Inter-Process Communication

JTAG: Joint Test Action Group

KB: Kilobyte

LED: Light Emitting Diode

MHz: Megahertzio

PC: Personal Computer

PCB: Printed Circuit Board

PMOD: Peripheral Module Interface

QSPI: Quad Serial Peripheral Interface

QVGA: Quad Video Graphics Array

RAM: Random Access Memory

RGB: Red-Gre-Blue

RTL: Register Transfer Logic

RX: Receptor

SDK: Software Development Kit

SoC: System on Chip

SW: Software

TCL: Tool Command Language

TFM: Trabajo Fin de Máster

TIFF: Tag Image File Format

TX: Transmisor

UMA: Universidad de Málaga

USB: Universal Serial Bus

VGA: Video Graphics Array

VHDL: VHSIC (Very High-Speed Integrated Circuits) Hardware Description Language

VICON: Visión Configurable

WCFG: Waveform Configuration

YUV: Luminancia (Y) Crominancia (UV)

ANEXO V. REFERENCIAS

[1] MT9V111 – 1/4 –Inch SOC VGA Digital Image Sensor Features. Micron

[2] BASYS 3 FPGA Board Reference Manual. Digilent Inc.

[3] FT232H SINGLE CHANNEL HI-SPEED USB TO MULTIPURPOSE UART/FIFO IC Datasheet Version 2.0. FTDI

[4] Application Note AN_130. FT2232H Used in an FT245 Style Synchronous FIFO Mode. FTDI

[5] Datasheet. UM232H-B USB to Serial/Parallel Break-Out Module. FTDI

[6] Software Application Development D2XX Programmer's Guide. FTDI

[7] Vivado Design Suite User Guide. Synthesis. Xilinx Inc.

[8] Xilinx 7 Series FPGAs Embedded Memory Advantages. Xilinx Inc.

[9] OpenCV on-line documentation https://docs.opencv.org/master/

[10] Emgu CV: OpenCV in .NET (C#, VB, C++ and more) http://www.emgu.com/wiki/index.php/Main_Page

[11] Introduction to Advanced Installer https://www.advancedinstaller.com/userguide/introduction.html