ESCUELA SUPERIOR DE INGENIERÍA

INGENIERA TÉCNICO INDUSTRIAL

ELECTRÓNICA INDUSTRIAL

DISEÑO DE UNA INTERFAZ DE COMUNICACIÓN BASADA EN EL PROTOCOLO DE COMUNICACIÓN AER PARA SENSORES DE VISIÓN BASADOS EN EVENTOS

Mª del Mar Rodríguez Azor

Cádiz, Diciembre 2017

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERA TÉCNICO INDUSTRIAL

ELECTRÓNICA INDUSTRIAL

DISEÑO DE UNA INTERFAZ DE COMUNICACIÓN BASADA EN EL PROTOCOLO DE COMUNICACIÓN AER PARA SENSORES DE VISIÓN BASADOS EN EVENTOS

DEPARTAMENTO: INGENIERÍA EN AUTOMÁTICA, ELECTRÓNICA, ARQUITECTURA Y REDES DE ORDENADORES

ÁREA: ELECTRÓNICA

DIRECTOR DEL PROYECTO: JUAN ANTONIO LEÑERO

AUTOR DEL PROYECTO: Mª DEL MAR RODRÍGUEZ AZOR

Cádiz, Diciembre 2017

A mi padre

Agradecimientos

En primer lugar, le agradezco a mi padre que me haya guiado para escoger esta carrera, sin su ayuda ahora estaría seguramente en otro sector preguntándome si elegí bien.

También quiero agradecerle a mi madre su eterna preocupación para cerrar esta etapa para la obtención de un título universitario.

A mi hermano Carlos y mi hermana Begoña por su apoyo y compresión.

A mi mejor amiga de infancia Fátima y mis compañeros de trabajo, también por su apoyo y paciencia en mis momentos tensos.

Y por último pero no menos importante, a mi tutor Juan Antonio Leñero, ya que sin él no habría sido capaz de realizar este proyecto.

ÍNDICE

DEFINICIONES Y ABREVIATURAS 12

INTRODUCCIÓN 16

1.1. Descripción 16

1.2. Antecedentes 17

1.3. Alcance 18

1.4. Motivación 18

1.5. Estructura del proyecto 19

INFORMACIÓN PREVIA 21

2.1. Protocolo de comunicación AER 21

2.2. Memoria RAM 22

2.2.1. DRAM 23

2.2.2. SRAM 24

2.3. FPGA: Opal Kelly XEM7360 26

2.4. Sensor de visión basado en eventos 27

2.4.1. APS 27

2.4.2. Sensor de imagen HDR o basado en eventos 28

INTERFAZ 33

3.1. Memoria RAM 33

3.2. FSM 36

3.3. Reset 38

3.4 Módulo final 39

SIMULACIONES 42

4.1. Simulación Memoria RAM 42

4.2. Simulación Reset 43

4.3. Simulación FSM 45

4.4. Pixel 46

CONCLUSIONES 49

PRESUPUESTO 52

BIBLIOGRAFÍA 54

ANEXOS 57

Índice de Figuras

Figura 1.1. A) Configuración experimental formada por una PCB personalizada, una lente y una FPGA. La PCB también dispone de un conector IDC de 40 pines compatible con la placa USBAERmini. B) Detalles de la interconexión opcional entre el sistema PCB y la placa USBAERmini. _____ *17*

Figura 1.2. Interfaz personalizada programada para representar en tiempo real los datos de salida del sensor. _____ *18*

Figura 2.1. Modelo del sistema AER formado por un emisor (S), un receptor (C), las señales de control request (R), acknowledgement (A) y el canal para la transmisión de la información (Data). _____ *21*

Figura 2.2. Esquema de la comunicación AER punto-a-punto entre un emisor y un receptor. _____ *22*

Figura 2.3. Esquema de una memoria DRAM A) para la lectura de datos y B) para la escritura/almacenamiento. _____ *24*

Figura 2.4. Esquema SRAM de 6 transistores MOSFET. El biestable lo forman los transistores M1-M4. Los transistores de acceso M5 y M6 controlan la conexión a los buses BL y BL. El bus de control WL es el que controla a los transistores de acceso. *25*

Figura 2.5. FPGA XEM7360 de Opal Kelly. _____ *26*

Figura 2.6. Diagrama de bloque de la FPGA XEM7360 de Opal Kelly. _____ *26*

*Figura 2.7. Voltajes transitorios de píxeles en la capacitancia de integración. Cada vez que se alcanza ese nivel de voltaje V_{bot}, se envía un evento fuera del chip.*_____ *29*

Figura 2.8. Cronograma de las señales involucradas en la operación del píxel. La detección comienza después de un reinicio global. El píxel se mantiene estable hasta que se toma una nueva medición. El intervalo de tiempo entre mediciones consecutivas se denota como T_{scan}. _____ *29*

Figura 2.9. Esquemas de los píxeles. El circuito de la izquierda se encarga de la conversión de luz a frecuencia. El de la derecha, para manejar la lógica de la comunicación AER. Los tamaños de los transistores son (W/L, µm/µm): M_{p1}=1/1. M_{p2}=3/1. M_{p3}=0.5/1. M_{n1}=0.5/0.7. M_{n2}=0.7/0.7. M_{n3}=1/0.7. M_{n4}=M_{n5}=0.5/0.7. C_{int} = C_1 =40fF, C_{ph} =5fF. Bias voltages: V_{bot} =1V, V_{bias_comp} =4.3V. _____ *30*

Figura 2.10. Diagrama de los bloques principales del sensor. La placa externa Opal Kelly con un FPGA está conectada al sensor para la interfaz. El FPGA puede enviar datos a una PC para la depuración y el monitoreo. _____ *31*

Figura 3.1. A) Tabla con las entradas y salidas de la memoria RAM diseñada. B) Diagrama de las entradas y salidas de la memoria. _____ *34*

Figura 3.2. Tabla de estados de la salida dout respecto a las entradas ram_enable y write_enable _____ *35*

Figura 3.3. Esquema del funcionamiento de la memoria en modo lectura y escritura 35

Figura 3.4. Diagrama de la FSM: entradas, salidas y relación con la RAM. _____ *36*

Figura 3.5. A) Diagrama de estados de la FSM y B) Tabla con las acciones de cada estado _____ *37*

Figura 3.6. Tabla de estados de la FSM _____ *38*

*Figura 3.7. Diagrama del módulo final: Pixel*_____ *40*

Figura 4.1. Ejemplo 1del funcionamiento de la memoria. Se señala el momento en el que comienza el modo escritura t=200 ns y el instante en el que se almacena t=250 ns. _____ *43*

Figura 4.2. Ejemplo 2 del funcionamiento de la memoria. Modo lectura. _____ *43*

Figura 4.3. A) Inicio de la simulación de la función Reset. En el primer flanco positivo se activan ram_enable y write_enable, mientras que addr no cambia hasta pasado un flanco positivo más. B) Funcionamiento global de la función Reset. _____ *44*

Figura 4.4. Ejemplo final de la función Reset en t=4μs y comienzo de la FSM en t=4,3μs _____ *44*

Figura 4.5. Ejemplo fallo en la FSM al cambiar addr cuando no está preparada para un nuevo evento. _____ *45*

*Figura 4.6. Ejemplo correcto funcionamiento de la FSM.*_____ *46*

Figura 4.7. Ejemplo simulación módulo completo Pixel. _____ *47*

DEFINICIONES Y ABREVIATURAS

Verilog HDL	*Verilog Hardware Description Language*. Lenguaje de descripción de hardware que puede describir hardware no sólo a nivel de puertas y transferencia de registros, sino también a nivel de algoritmos. Esto permite que la traducción de un diseño descrito en Verilog a puertas lógicas sea un proceso sencillo.
Protocolo AER	Protocolo de comunicación *Address Event Representation.*
(SD)RAM	*(Synchronous Dynamic) Random Access Memory* o Memoria Síncrona Dinámica de Acceso Aleatorio.
DDR	*Double Data Rate*.
FPGA	*Field Progammable Gate Array*. Dispositivo lógico programable que permite describir un circuito digital usando un lenguaje específico (VHDL o Verilog entre otros) de manera que al integrarlo en el chip las interconexiones programables proporcionan la interconexión de los bloques lógicos y la conexión de las entradas y salidas.
Datalogger	*Datalogger* o registrador de datos es un dispositivo electrónico que registra datos en el tiempo o en relación a la ubicación por medio de instrumentos y sensores propios o conectados externamente.
PCB	*Printed Circuit Board* o Placa de Circuito Impreso. Tarjeta o placa en la cual se emplazan los distintos elementos que conforman un circuito y las interconexiones electrónicas entre ellos.
(H)DR	*(High) Dynamic Range*.
APS	*Active Pixel Sensor*.
TM	*Tone Mapping* o Mapeo de Tonos.

FR	*Frame Rate* o velocidad de fotograma.
Conector Mezzanine	Conector tarjeta a tarjeta que permite la expansión de diseños electrónicos y trabaja a alta velocidad.
FSM	*Finite State* Machine o Máquina de Estados Finitos, también conocida como Autómata Finito.
Testbench	Modelo escrito en lenguaje Verilog que aplicas estímulos, compara las respuestas de las salidas e informa de cualquier desajuste funcional.

MEMORIA

CAPÍTULO 1

INTRODUCCIÓN

1.1. Descripción

Este documento es la memoria del proyecto final de carrera que describe el proceso seguido para la creación de una interfaz de comunicación empleada entre sensores basados en eventos que operan a alta velocidad, con muy bajo consumo, y una FPGA comercial.

Dicha interfaz gestiona las señales de control de la cámara según el protocolo de comunicación Address Event Representation (AER), permitiendo, junto con otra interfaz ya creada, la representación de la información que transmite la cámara en tiempo real en el monitor de un PC.

El lenguaje elegido para creación del módulo es Verilog HDL, el cual permite que el traspaso del diseño creado a puertas lógicas se haga de manera sencilla y fácil.

Para la realización del módulo en lenguaje Verilog, se emplea el software Xilinx - ISE Design Suit 14.7. Los motivos de la elección de este software son principalmente tres:

i. Está diseñado específicamente para los lenguajes de descripción Verilog y VHDL, así como trabajar con FPGAs.

ii. El módulo de integración[1] utilizado en el montaje experimental del que se parte (véase siguiente apartado) está basado en una FPGA de Xilinx.

iii. Ofrece tanto licencia gratuita como de pago. La gratuita permite realizar el módulo que se desee casi sin ninguna limitación. La versión de pago está actualmente disponible en los laboratorios de la Escuela Superior de Ingeniería de la Universidad de Cádiz.

[1] *De ahora en adelante se hablará de FPGA en lugar de módulo de integración.*

1.2. Antecedentes

Se parte de un diseño que consta de un sensor de visión, un módulo de integración (Opal Kelly) y un datalogger (USBAERmini2). Dicho montaje se puede observar en la Figura 1.1.

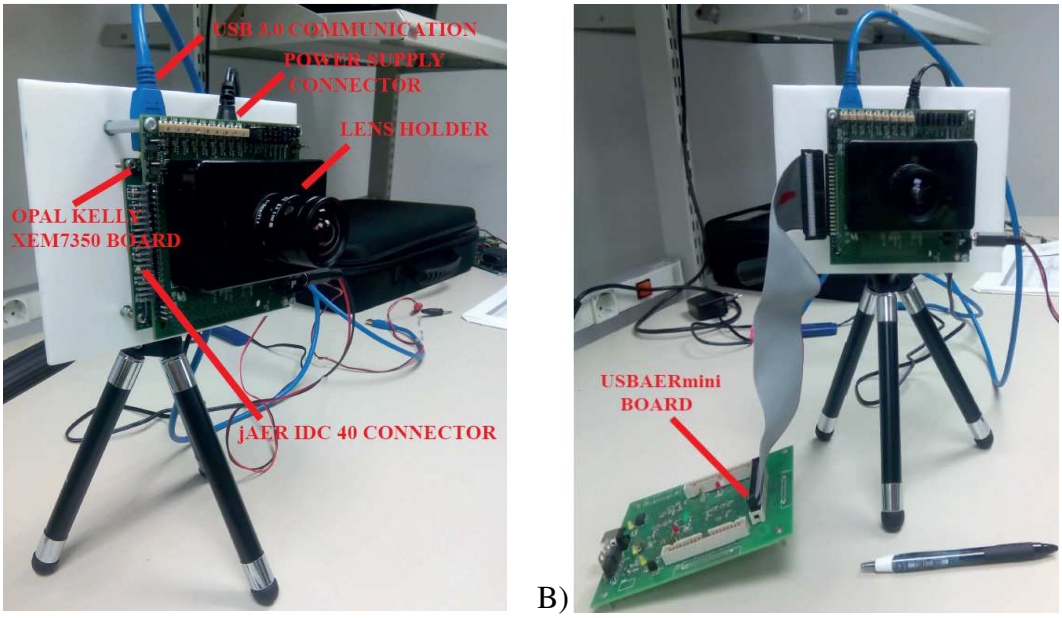


Figura 1.1. A) Configuración experimental formada por una PCB personalizada, una lente y una FPGA. La PCB también dispone de un conector IDC de 40 pines compatible con la placa USBAERmini. B) Detalles de la interconexión opcional entre el sistema PCB y la placa USBAERmini.

El datalogger, entre otras funciones, se encarga de la gestión de la comunicación AER y el almacenamiento de las salidas en una memoria. Para ello, está acoplado al sensor y envía la información a una interfaz del PC, pudiendo obtener imágenes[2] como la mostrada en la Figura 1.2.

[2] *Vídeo ejemplo funcionamiento: https://folk.uio.no/juanle/Videos/HDRImageSensor2016.mp4*

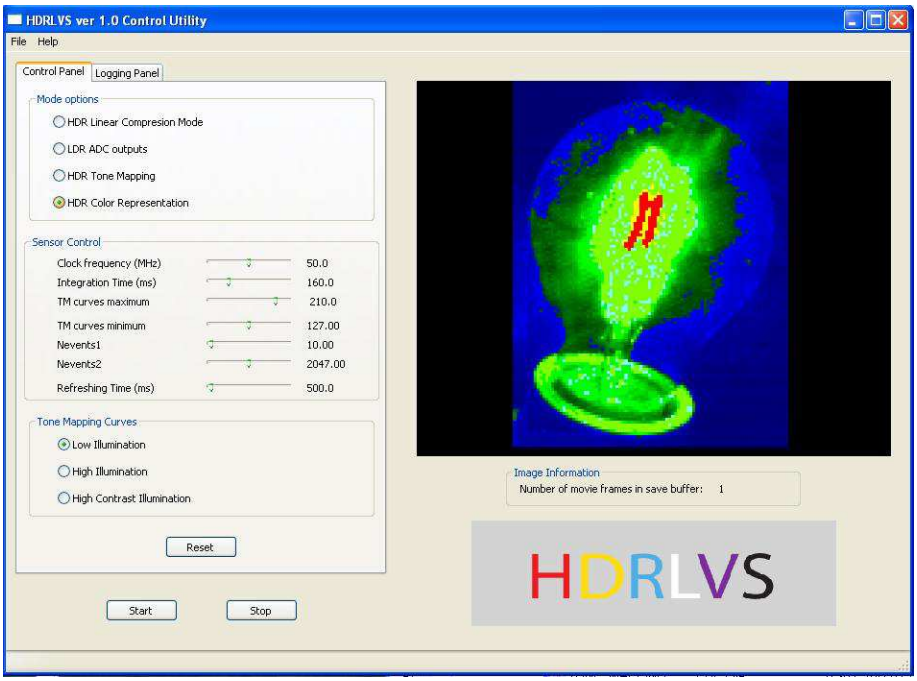


Figura 1.2. Interfaz personalizada programada para representar en tiempo real los datos de salida del sensor.

1.3. Alcance

Con este proyecto se pretende crear un módulo que sirva como interfaz entre uno sensor de visión y una FPGA, permitiendo una comunicación más rápida y eficaz y haciendo prescindible el datalogger mencionado en el punto anterior.

Para implementar dicho módulo, sería necesario definir unos pines de la FPGA como entrada/salida para las señales relativas a la comunicación AER y a la memoria: Request, Acknowledgement y Address (véase punto 2.1 para más información sobre la comunicación AER y las entradas y salidas empleadas). Esta implementación se propone como trabajo a futuro.

1.4. Motivación

La principal razón de la elección de este proyecto es el uso de la programación para fines de electrónica, pudiendo no sólo simular el resultado sino también probarlo con dispositivos físicos.

El empleo del lenguaje Verilog me ha supuesto un reto ya que desconocía este lenguaje por completo, pero a su vez incrementa la motivación de realizar este proyecto

puesto que por una parte el superar retos es algo que de por sí motiva y, por otro lado, el ampliar la visión a otros lenguajes de programación/descripción enriquece.

1.5. Estructura del proyecto

El proyecto se divide en 4 grandes bloques: Definiciones y abreviaturas, Memoria, Bibliografía y Anexos. A su vez, la memoria está dividida en 6 capítulos:

- **Introducción**: contiene la descripción, objetivos, alcance, antecedentes y motivación del proyecto.
- **Información previa**: se introduce el protocolo de comunicación AER, descripción de las memorias RAM, de la FPGA y de los sensores de visión del montaje original sobre el que se implementaría la interfaz.
- **Interfaz**: proceso seguido para la conclusión del alcance de este proyecto.
- **Simulaciones**: resultados obtenidos tras la simulación el módulo.
- **Conclusiones**.
- **Presupuesto**.

CAPÍTULO 2

INFORMACIÓN PREVIA

2.1. Protocolo de comunicación AER

El protocolo de comunicación AER (Address Event Representation) [1-3] se basa en la comunicación unidireccional, asíncrona y punto-a-punto entre un Emisor/Sender "S" y un Receptor/Receiver "C", tal como se muestra en la Figura 2.1. La conexión entre el Emisor y el Receptor consiste en dos tipos de cable, cables de control y cables de información.

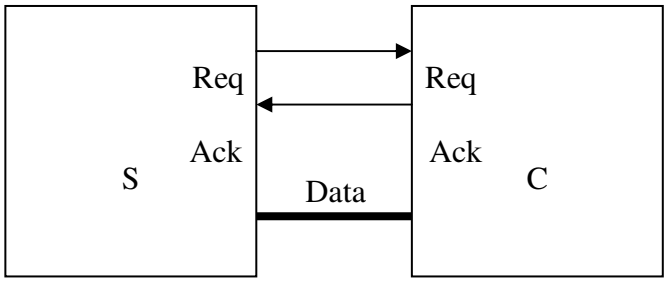


Figura 2.1. Modelo del sistema AER formado por un emisor (S), un receptor (C), las señales de control request *(R),* acknowledgement *(A) y el canal para la transmisión de la información (Data).*

El cable de información es exclusivamente dirigido por el Emisor y sentido por el Receptor. Mientras que en el caso de los cables de control, Req es exclusivamente dirigido por el Emisor y sentido por el Receptor, y Ack es dirigido por el Receptor y sentido por el Emisor. Tanto Req como Ack son señales binarias, tomando como valor de activación 1 y 0 para la inactividad.

El emisor activa la señal de control Req (request) indicando que tiene información que transmitir. Acto seguido, la información es trasmitida con un cierto retardo para asegurarse el correcto envío a través del bus de comunicación de sentido único (del emisor al receptor). Una vez captada dicha información, el receptor activa la señal de control Ack (acknowledgement) para indicar que ya ha sido recibida. Esto último provoca que automáticamente la señal de control Req se desactive, poniéndose a 0.

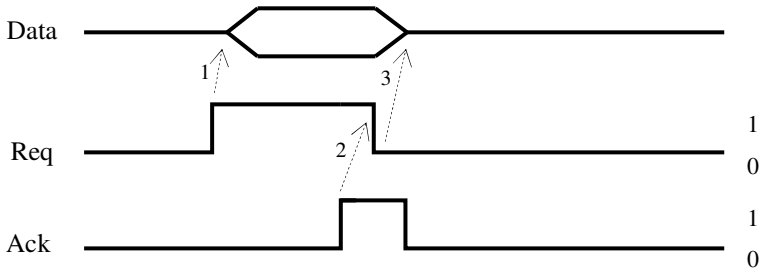


Figura 2.2. Esquema de la comunicación AER punto-a-punto entre un emisor y un receptor.

En el caso que nos concierne, el emisor es un sensor de visión que alerta los cambio sufridos por los píxeles; la información enviada es la dirección de dichos píxeles que cambian; y el receptor no es más que una máquina de estados finita digital o digital FSM (Finite State Machine), que almacena la dirección de los píxeles en una memoria.

Las principales ventajas de emplear este protocolo son:

i. Atiende a la necesidad de transmitir asíncronamente la dirección de cada píxel que haya sufrido un cambio, sin emplear un contador que almacene el número de pulsos o veces que un píxel cambia, mejorando el tiempo de transmisión entre ambas partes (emisor/receptor).

ii. Evita colisiones de información a la hora de transmitir las direcciones de los píxeles que hayan sufrido algún cambio.

2.2. Memoria RAM

La memoria RAM o memoria de acceso aleatorio es un tipo de memoria volátil, ya que al desconectarla de la fuente de alimentación pierde toda la información que almacenaba. Este tipo de memoria se utiliza memoria de trabajo para el sistema operativo, los programas y la mayoría del software. Es allí donde se cargan todas las instrucciones que ejecutan el procesador y otras unidades de cómputo. Se denominan "de acceso aleatorio" porque se puede leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición, no siendo necesario seguir un orden para acceder a la información de la manera más rápida posible.

Sin embargo, otros tipos de memoria también presentan esa característica, como por ejemplo la memoria ROM o la memoria Flash. Por tanto, la expresión memoria RAM se utiliza frecuentemente para describir a los módulos de memoria utilizados en los ordenadores y servidores.

Existen principalmente dos variedades de memoria RAM: la DRAM (Dynamic RAM) y la SRAM (Static RAM) [4-5].

2.2.1. DRAM

La DRAM o Memoria Dinámica de Acceso Aleatorio es un tipo de memoria que para mantener almacenado un dato, necesita revisar dicho dato y volver a cargarlo cada cierto periodo, en un ciclo de refresco. De ahí que se la denomine dinámica. La ventaja que ofrece esta memoria es la posibilidad de construir memorias con una gran densidad de posiciones y que todavía funcionen a una velocidad alta. En la actualidad, es una de las memorias más usadas, ya que se fabrican integrados con millones de posiciones y velocidades de acceso medidos en millones de bit por segundo.

El funcionamiento es relativamente sencillo. La celda de memoria es la unidad básica de cualquier memoria, capaz de almacenar un Bit en los sistemas digitales. La construcción de la celda define el funcionamiento de la misma, en el caso de la DRAM moderna, consiste en un transistor de efecto de campo y un condensador. El principio de funcionamiento básico, es sencillo: una carga se almacena en el condensador significando un 1 y sin carga un 0. El transistor funciona como un interruptor que conecta y desconecta al condensador. Este mecanismo puede implementarse con dispositivos discretos y de hecho muchas memorias anteriores a la época de los semiconductores, se basaban en arreglos de celdas transistor-condensador.

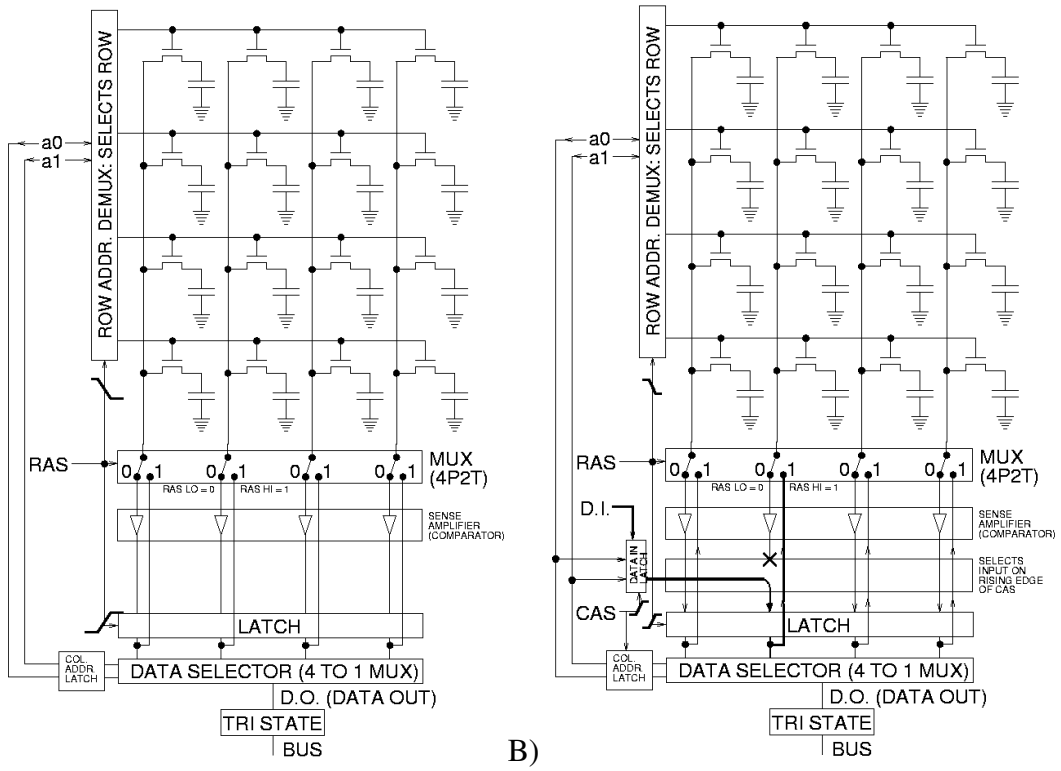


Figura 2.3. Esquema de una memoria DRAM A) para la lectura de datos y B) para la escritura/almacenamiento.

Dentro de las DRAM se encuentran a su vez otros dos tipos de memoria: DRAM Asíncrona (tradicional) y SDRAM o DRAM Síncrona. La diferencia entre ambas es que el cambio de estado de la memoria en la DRAM Asíncrona se efectúa cada cierto tempo dependiendo de sus características desde que cambian sus entradas, mientras que en la SDRAM el cambio se realiza cuando una señal de reloj se lo indica.

Dentro de las SDRAM podemos encontrar la gama DDR (Double Data Rate). Las más conocidas son la DDR3 y DDR4 (cuarta y quinta generación respectivamente de memoria SDRAM, siendo la primera la SDR SDRAM).

2.2.2. SRAM

La SRAM o Memoria Estática de Acceso Aleatorio es un tipo de memoria basada en semiconductores que, a diferencia de la DRAM, no necesita refrescar sus datos para mantenerlos. Sin embargo sigue siendo volátil, en cuanto se interrumpe la fuente de alimentación, se pierden los datos.

Una SRAM típica emplea seis transistores MOSFET para almacenar cada bit, cuatro se emplean para almacenar propiamente el bit, formando un biestable (estados 0 y 1), y los otros dos para controlar el acceso al biestable durante las operaciones de lectura y escritura. Aunque presentan dos buses de datos disponibles tanto para escritura como para lectura (en la Figura 2.4 representados como BL y $\overline{BL}$), no es obligatorio el uso de ambos. No obstante, se suelen implementar para mejorar los márgenes de ruido.

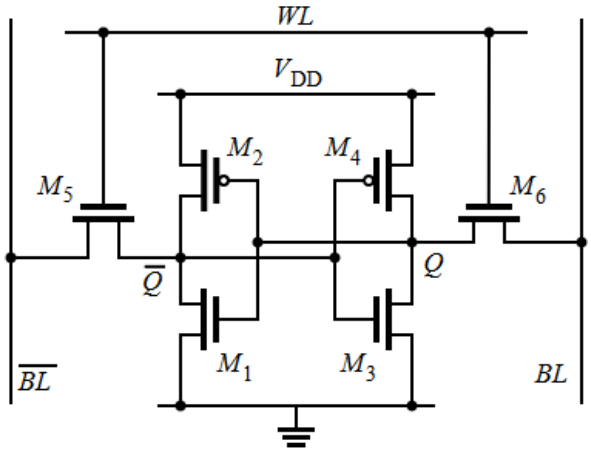


Figura 2.4. Esquema SRAM de 6 transistores MOSFET. El biestable lo forman los transistores M1-M4. Los transistores de acceso M5 y M6 controlan la conexión a los buses BL y $\overline{BL}$. El bus de control WL es el que controla a los transistores de acceso.

Las SRAM tienen tres estados de operación: Reposo, Lectura y Escritura. Durante el reposo, el bus de control está desactivado y los transistores de acceso desconectan la celda de los buses de datos, quedando el dato almacenado invariable. Para la lectura, los valores que estén almacenados (en Q y $\overline{Q}$) pasan a los buses de datos (BL y $\overline{BL}$ respectivamente), dado que se activa el bus de control (WL) y los transistores de acceso conectan la celda con los buses de datos. En la escritura, el valor a almacenar viene a través de los buses de datos, se activa el bus de control y se permite el acceso a la celda, pudiéndose almacenar dicho valor.

Las principales ventajas que presenta respecto a la DRAM es su menor consumo y su rapidez. Sin embargo, el consumo varía en función de la frecuencia, a mayor frecuencia, mayor consumo, pudiéndose igualar a la DRAM en altas frecuencias. Como desventajas principales: son más caras y no se emplean para almacenar una gran cantidad de datos como se podrían necesitar en un ordenador personal.

2.3. FPGA: Opal Kelly XEM7360

Como se ha mencionado anteriormente, se parte de un diseño en el que se incluye un módulo de integración, en concreto el Opal Kelly XEM7360. Este módulo de integración está basado en la FPGA Xilinx Kintex-7 [6].

Figura 2.5. FPGA XEM7360 de Opal Kelly.

Las características que presenta y, por tanto, los motivos por los cuales se ha elegido este modelo son:

- ✓ Conexión USB 3.0. Permite una transferencia de 340 MiB/s hacia/desde el PC.
- ✓ 2-GiByte de memoria RAM
- ✓ 8 transceptores gigabit de alta velocidad

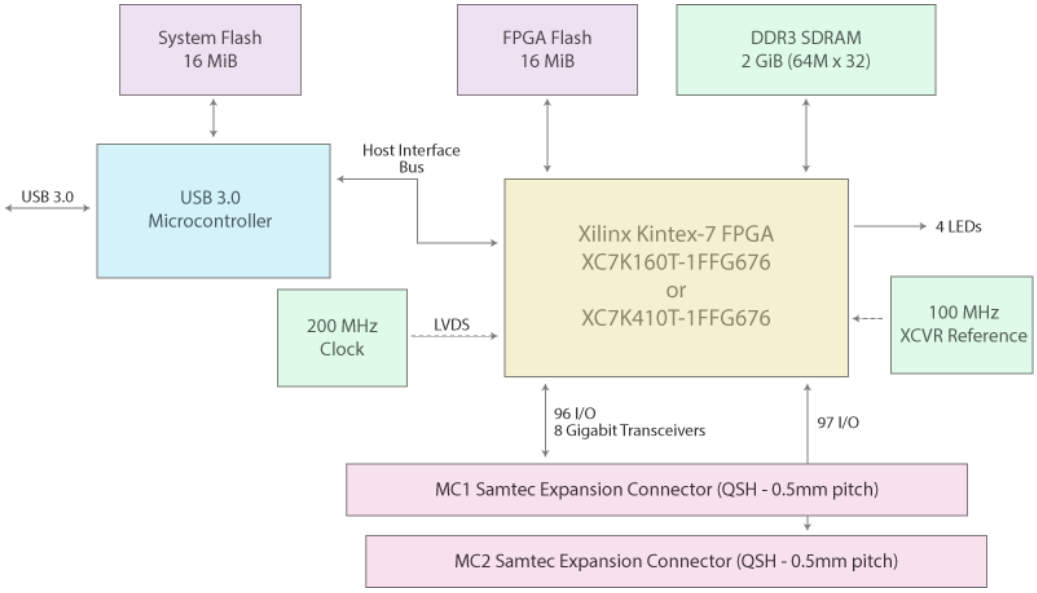


Figura 2.6. Diagrama de bloque de la FPGA XEM7360 de Opal Kelly.

Además, entre sus múltiples aplicaciones, está la de captura y procesamiento de imágenes o vídeos, objetivo final del montaje.

Se incluyen diagramas e información sobre los pines en el bloque Anexos.

2.4. Sensor de visión basado en eventos

Para la elección de un sensor de visión se plantean dos propuestas: el Active Pixel Sensor (APS) y el sensor de imagen HDR (High Dymanic Range).

2.4.1 APS

El APS presenta un rango dinámico limitado [7]. Para un tiempo de integración fijo, la menor fotocorriente que puede ser captada está limitada por el ruido de cuantificación o el ruido de lectura de un convertidor analógico-a-digital. La mayor fotocorriente está también limitada por la capacidad de llenado. Variando el tiempo de integración, se observan ciertas mejoras pero también inconvenientes:

a) Si se disminuye el tiempo de integración, se puede trabajar con mayor iluminación, pero las zonas menos iluminadas perderán precisión.

b) Al incrementar el tiempo de integración, se puede lograr captar fotocorrientes menores. Por contrapartida, la mayor fotocorriente que puede ser sentida también será menor.

Esto demuestra que la limitación viene dada por el sensor, siendo poco efectivo el cambio del tiempo de integración. En general, el APS tiene un DR (Rango dinámico) por debajo de los 70 dB, mientras que escenas naturales pueden llegar a los 120 dB. Comparando el APS con el ojo humano, la sensibilidad es peor. Para llegar a tal grado de percepción, se necesita operar en HDR.

Hay varias formas de ampliar el DR, siendo las más conocidas capturar imágenes con múltiples tiempos de integración y luego combinarlas o usar algoritmos de mapeo de tonos (TM).

La primera opción requiere costes altos dedicados a la carga computacional, el hardware y el consumo de energía. Se precisa de programar algoritmos dedicados para la combinación de los fotogramas capturados y su final representación. los distintos fotogramas se capturan individualmente y son almacenados en una memoria. Esto provoca una disminución en la velocidad de fotograma o FR y consumo mayor de energía. Otra desventaja es que si los tiempos de integración de las distintas capturas se desalinean, pueden generar bordes inexistentes y distorsionar la interpretación de la escena real.

Los algoritmos TM están basados en los histogramas de las imágenes. A los valores que son más frecuentes en la escena que se desea representar se les asigna con mayor precisión los niveles de gris. Por tanto, la relación entre la fotocorriente de los píxeles y el nivel de gris es no lineal. La imagen HDR resultante tiene un bajo número de bits para codificar la iluminación. Implementando el algoritmo TM en chips se consiguen imágenes HDR de calidad, pero todos los niveles de iluminación no son codificados con la misma precisión, perdiéndose información.

2.4.2 Sensor de imagen HDR o basado en eventos

El sensor de visión que se describe a continuación emplea el protocolo AER para mandar la información de los píxeles de manera asíncrona y está planeado para operar con HDR [8].

Los píxeles se van disparando asíncronamente. Cada vez que esto ocurre, el sensor requiere acceso al bus compartido AER. Cuando el bus está listo, se envían las coordenadas del píxel que se ha disparado a través del bus. Es posible que varios píxeles intenten acceder al bus al mismo tiempo. En ese caso, los píxeles tienen que esperar hasta que se les garantice el acceso para mandar la información.

Como se ha mencionado anteriormente, se describe únicamente el sensor AER, en el cual los píxeles generan eventos con una frecuencia proporcional a la iluminación, como se muestra en la Figura 2.7, y en concreto operando con HDR.

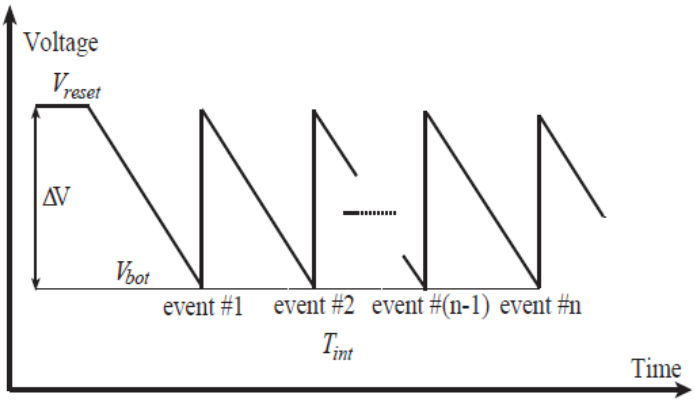


Figura 2.7. Voltajes transitorios de píxeles en la capacitancia de integración. Cada vez que se alcanza ese nivel de voltaje V_{bot}, se envía un evento fuera del chip.

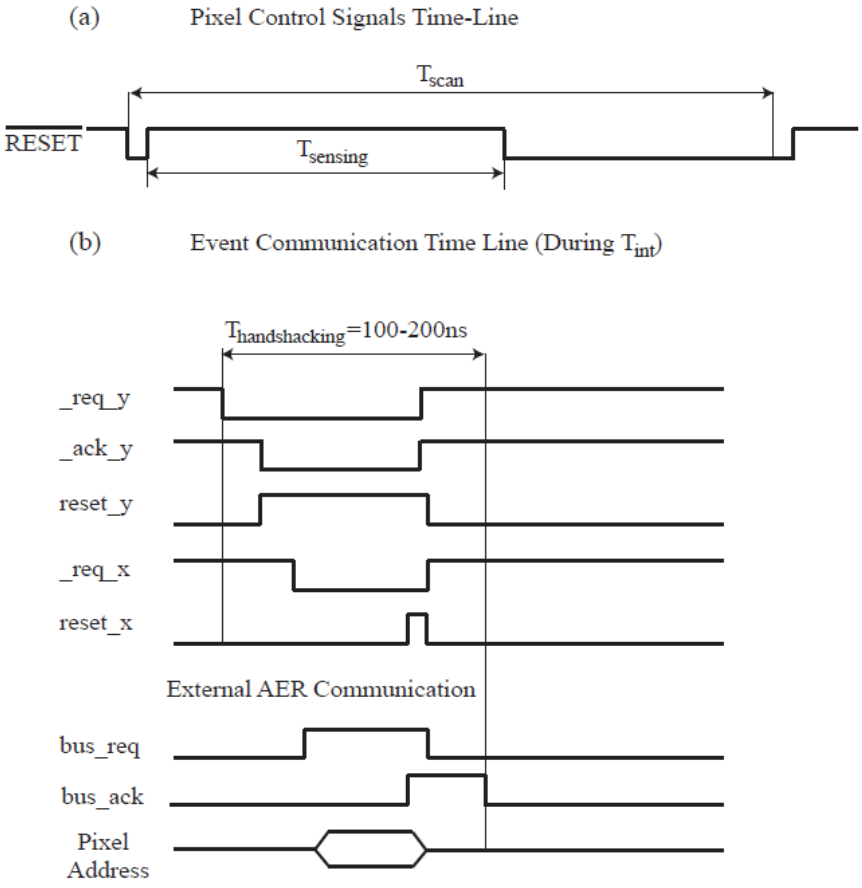


Figura 2.8. Cronograma de las señales involucradas en la operación del píxel. La detección comienza después de un reinicio global. El píxel se mantiene estable hasta que se toma una nueva medición. El intervalo de tiempo entre mediciones consecutivas se denota como T_{scan}.

El modo que se emplea para los fines deseados es la conversión de la luz a frecuencia. La Figura 2.9 muestra el circuito implicado en dicha conversión. El funcionamiento de los píxeles se inicia con un reseteo global ($\overline{RES}$). Tras esto, los condensadores se descargan con una velocidad proporcional a la iluminación local. En cuanto el voltaje del píxel alcanza el valor V_{bot}, se dispara el comparador y se restablece el condensador del píxel. A continuación, la carga del píxel comienza a integrarse nuevamente. Mientras tanto, una memoria analógica almacena en C_1 un valor que indica que se ha disparado un píxel, siendo necesario enviar un evento. La lógica de la comunicación AER maneja la comunicación para transmitir las coordenadas (x, y) del píxel disparado.

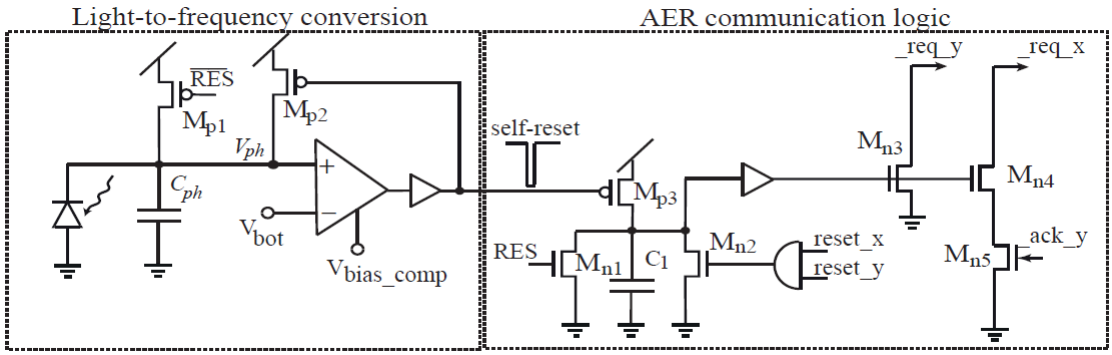


Figura 2.9. Esquemas de los píxeles. El circuito de la izquierda se encarga de la conversión de luz a frecuencia. El de la derecha, para manejar la lógica de la comunicación AER. Los tamaños de los transistores son (W/L, µm/µm): M_{p1}=1/1. M_{p2}=3/1. M_{p3}=0.5/1. M_{n1}=0.5/0.7. M_{n2}=0.7/0.7. M_{n3}=1/0.7. M_{n4}=M_{n5}=0.5/0.7. C_{int} = C_1 =40fF, C_{ph} =5fF. Bias voltages: V_{bot} =1V, V_{bias_comp} =4.3V.

Dado que el bloque de conversión luz-frecuencia es un oscilador estable, se puede calcular el periodo de oscilación mostrado en la Figura 2.7 en función de los parámetros del circuito en la Figura 2.9, siendo su valor aproximado:

$$f_{osc} \approx \frac{I_{ph}}{C_{ph} \cdot (V_{DD} - V_{bot})} = \frac{I_{ph}}{C_{ph} \cdot \Delta V} \qquad (2.1)$$

Como se puede observar, el periodo de oscilación depende del incremento de voltaje del píxel $\Delta V = V_{DD} - V_{bot}$, la capacitancia de integración del píxel C_{ph}, y de la fotocorriente I_{ph}. Dada esta relación, se puede ajustar el valor de ΔV para que la

frecuencia de disparo de los píxeles sea del orden de KHz, cumpliendo el propósito perseguido para la implementación del sensor. Al ser posible que el sensor esté expuesto a altos niveles de iluminación, los valores de la capacitancia de integración C_{ph} y ΔV deberían ser lo suficientemente altos para garantizar el periodo deseado. El valor de la fuente de alimentación se puede configurar a $V_{DD} = 5V$.

La Figura 2.10 muestra un diagrama de bloques del sensor. Se muestran los diferentes módulos de circuitos implicados en la operación basada en eventos de píxeles. Hay un núcleo de matriz de píxeles y un circuito asíncrono periférico para gestionar la comunicación AER. La placa externa Opal Kelly XEM7360, descrita en el apartado 2.2, está unida al sensor a través de un conector Mezzanine para la interfaz.

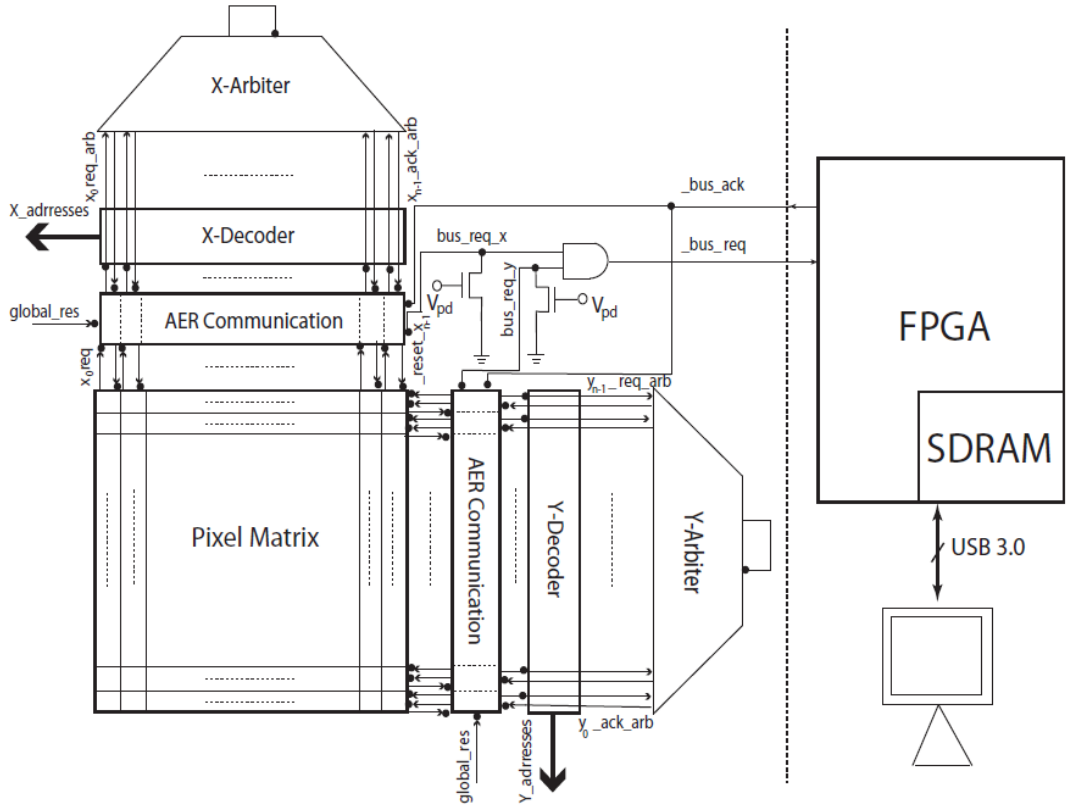


Figura 2.10. Diagrama de los bloques principales del sensor. La placa externa Opal Kelly con un FPGA está conectada al sensor para la interfaz. El FPGA puede enviar datos a una PC para la depuración y el monitoreo.

CAPÍTULO 3

INTERFAZ

Partiendo del manejo del software Xilinx - ISE Design Suit y del uso del lenguaje de descripción Verilog, la interfaz entre el sensor de visión y la FPGA puede dividirse en tres bloques:

- Memoria RAM
- FSM
- Reset

El bloque de FSM y Reset se presentan dentro del módulo principal, mientras que la memoria RAM es un módulo independiente al cual accede el módulo principal. El motivo de escoger este diseño es que la memoria es genérica y puede utilizarse para todo tipo de proyectos que requieran de una memoria RAM, mientras que la máquina de estados y la función reset están ligados a las necesidades de este proyecto en concreto. Por otra parte, la memoria también se emplea para probar por separado tanto la máquina de estados como el reseteo, tal y como se explica en los siguientes apartados.

3.1. Memoria RAM

La memoria RAM o memoria inmediata que se emplea en esta ocasión es una SDRAM, como se puede apreciar en las Figuras 2.6 y 2.10. Para el diseño de un bloque de memoria en lenguaje Verilog y su posterior implementación en la FPGA supone dos cosas:

I. Se dispone de una gran capacidad de almacenamiento de datos (si fuese una SRAM habría que prestar atención a la limitación de capacidad).

II. Los tiempos de respuesta pueden no ser tan rápido como se desea, teniendo que poner especial atención a los retardos. De otra forma, podría no almacenarse correctamente la información dado que la memoria no sería capaz de atender las órdenes de escritura/lectura que se le pide.

Partiendo de estos conocimientos y, como se ha explicado en el punto anterior, la memoria se utiliza a través del módulo principal y para comprobar el correcto funcionamiento de los otros bloques, el primer paso para el diseño del módulo propuesto es comenzar diseñando el módulo de la memoria.

La memoria cuenta con cinco entradas o inputs y una salida o output. Dichas entradas y salidas se definen en la Figura 3.1. Por otra parte, se utilizan parámetros y registros internos para definir las dimensiones de la memoria (número de posiciones y anchura de las palabras). Este paso es importante, ya que si se define una memoria con una anchura menor al valor que deseamos almacenar, no será posible; al igual que si se desea almacenar un dato en una posición que no existe. Para este caso, se define una memoria de 2^{16} posiciones con una anchura de palabra de 20 bits.

A)

Inputs	
Addr	Contiene la dirección del píxel que se ha disparado
Din	Valor a almacenar
Clk	Señal de reloj
ram_enable	Habilita/deshabilita el acceso a la RAM
write_enable	Habilita/deshabilita la escritura de la RAM
Output	
dout	Valor almacenado

B)

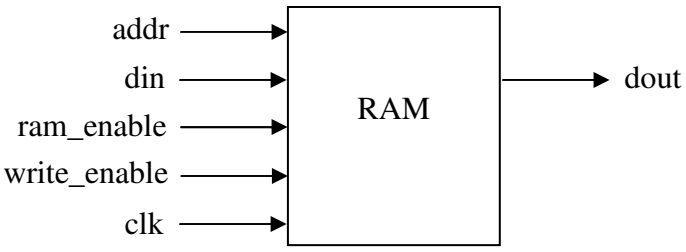


Figura 3.1. A) Tabla con las entradas y salidas de la memoria RAM diseñada. B) Diagrama de las entradas y salidas de la memoria.

El proceso de lectura/escritura es sencillo: cada vez que hay un flanco positivo de la **señal de reloj** (también se puede definir con un flanco negativo), se comprueba si **ram_enable** está activado o no, es decir si vale 1 ó 0. En el caso de que su vaor fuese 0, no se podría ni leer ni escribir. Si vale 1, comprueba si la escritura está activada o no

(**write_enable**). Si lo está, accede a la posición de memoria indicada por **addr** e introduce el valor que indique **din**. De otra forma, si la escritura está desactivada, estará en modo lectura, por tanto se accede a la posición de memoria que indique **addr** y se obtendrá el dato almacenado mediante **dout**.

ram_enable	write_enable	dout
0	X	X
1	0	Valor almacenado en posición indicada addr
1	1	din

Figura 3.2. Tabla de estados de la salida dout respecto a las entradas ram_enable y write_enable

En cualquiera de los dos casos (lectura o escritura) se ha establecido que **dout** se actualiza después de un ciclo de reloj respecto al cambio de **ram_enable** y **write_enable**, como se puede apreciar en la Figura 3.3. De esta forma se garantiza su correcto funcionamiento, evitando cualquier posible solape que lleve a un error.

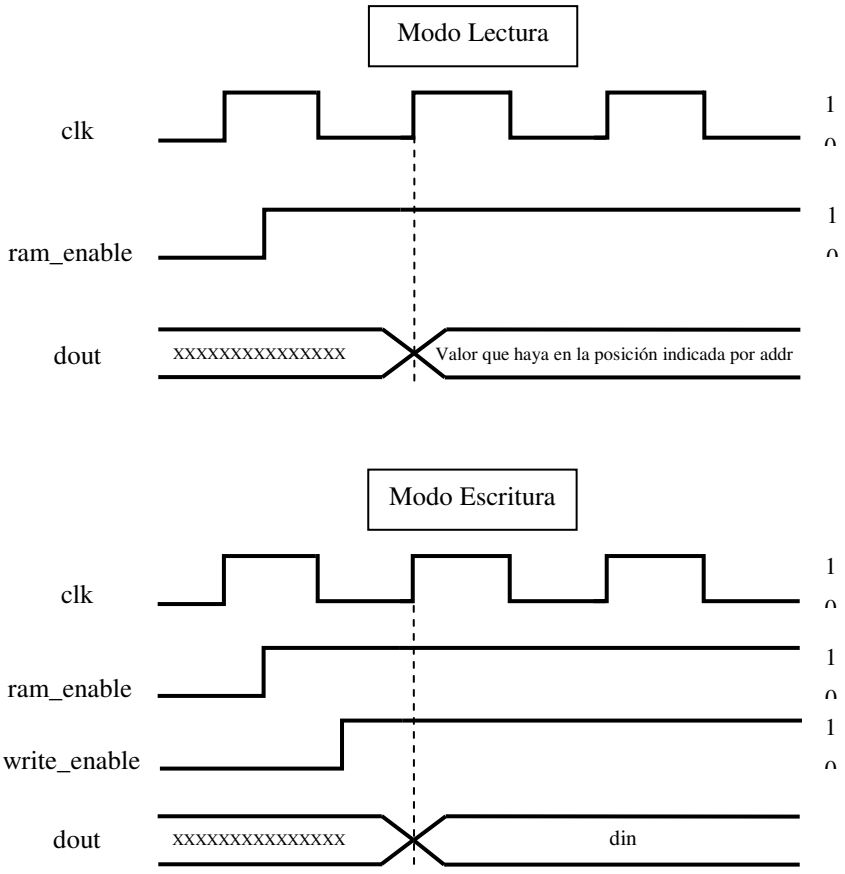


Figura 3.3. Esquema del funcionamiento de la memoria en modo lectura y escritura

3.2. FSM

Definida la memoria, se procede con la máquina de estados finitos o FSM, la cual se encarga de manejar la información precedente del sensor y transmitirla a la memoria. Está diseñada de forma que cumple con el protocolo de comunicación AER.

Una FSM o autómata finito, es un modelo computacional. La máquina "abstracta" está compuesta por un número finito de estados. Para cambiar de un estado a otro, depende de las entradas y de las condiciones de transición. Por ese motivo, para definir una FSM, es necesario proporcionar una lista de estados, su estado inicial y las condiciones para que se produzcan las transiciones. Para alcanzar el objetivo de esta FSM, su definición es:

- Nº de estados: 4.
- Estado inicial: estado 0.
- Condiciones para las transiciones: va a depender de las entrada **req** y la salida **ack**, tal como se muestran en la Figura 3.5.

Además de la entrada **req**, la FSM va a tener una entrada para la señal de reloj **clk** y otra para la dirección **addr** del píxel disparado. Internamente, va a controlar las entradas de la memoria **ram_enable**, **write_enable**, **addr** y **din**. La señal **clk** siempre va a ser la misma para todas los bloques o módulos, ya que su función es sincronizar.

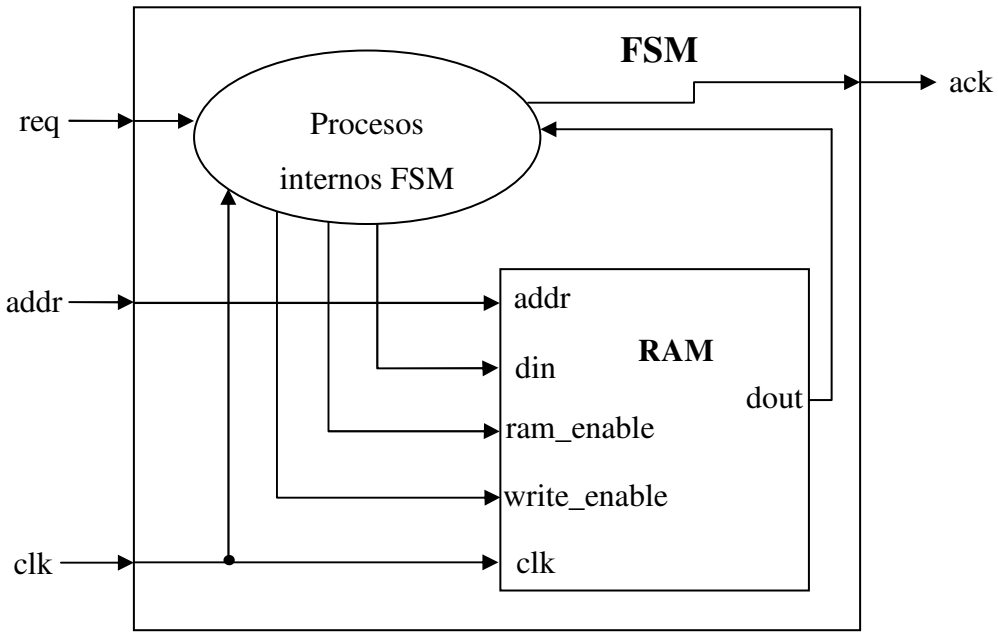
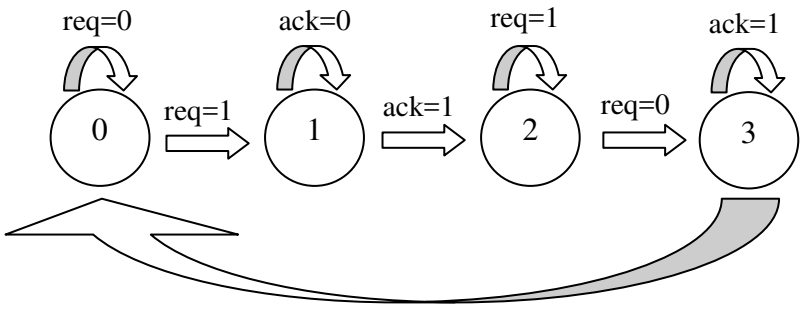


Figura 3.4. Diagrama de la FSM: entradas, salidas y relación con la RAM.

El funcionamiento de la FSM es relativamente sencillo. Como se ha descrito en puntos anteriores, cuando se dispara un píxel, el sensor activa una señal (**req**) para indicar que tiene información que mandar (**addr**). Es entonces la FSM cambia de estado y comienza la fase de recepción de información y transmisión a la memoria. En dicho estado, por tanto, se accede a la posición de memoria indicada por **addr** y se incrementa el valor que hubiese en 1 (**din**), que es la forma de indicar que ese píxel ha sufrido un cambio. Una vez recibida la información y almacenada, se activa **ack** como señal de que la información ha sido correctamente recibida, cambiando al siguiente estado. En este nuevo estado, se asegura que el sensor desactive **req**, evitando así que se tome como otro requerimiento aparte. Una vez desactivado **req**, pasa al siguiente y último estado, en el cuál se desactiva **ack** y se deshabilita el acceso a la RAM tanto para escritura como para lectura, de manera que para el siguiente requerimiento del sensor, éste no tome el acuse de información recibida incluso antes de mandarla.

A)



B)

Estado	Acción
0	Habilita el acceso a la RAM y espera a que **req**=1.
1	Lee el valor que hubiese en la dirección que indica **addr**, incrementa su valor en 1 y lo graba en esa misma posición. Activa **ack**=1 y pasa al estado 2.
2	Espera a que **req**=0 y entonces pasa al estado 3.
3	**ack**=0, se deshabilita la RAM y pasa al estado inicial 0.

Figura 3.5. A) Diagrama de estados de la FSM y B) Tabla con las acciones de cada estado

La FSM está diseñada de tal manera que no pueda saltarse ningún estado, ya que al depender de de una salida (**ack**) es posible controlar la transición por todos los estados. Es por ello que hay transiciones que no dependen del valor de la entrada **req** (véase Figura 3.6). No obstante, al no poderse controlar la entrada, lo que puede ocurrir es que pierda cierta información o se falsee la misma. Estos aspectos se detallan en el siguiente capítulo (*Simulaciones*). También se ha programado para que en caso de fallo o no saber a qué estado ir, vuelva a estado inicial 0.

Estado actual	req	ack	Estado siguiente
0	0	0	0
0	1	0	1
1	x	0	1
1	x	1	2
2	1	1	2
2	0	1	3
3	x	1	3
3	x	0	0

Figura 3.6. Tabla de estados de la FSM

3.3. Reset

Una función muy importante y que a veces no se presta mucha atención es la función Reset o reseteo. Esta función se encarga de poner a 0 cualquier dato que se desee. Para este diseño en concreto, se encarga de poner todas las posiciones de la memoria RAM a 0. Aunque se pueda pensar que no es necesario implementar en esta función en una memoria volátil que al desconectarla de la fuente de alimentación va a perder todos los datos, tiene un papel importante por dos motivos:

i. Asegura que se incremente correctamente el valor que había en 1. Al inicializar por primera vez la RAM, no hay nada cargado, lo cual es en parte beneficioso y en parte no lo es. Es beneficioso en cuanto a que nos aseguramos de que no se va a falsear la información que queremos grabar, pero por otro lado afecta a la hora de retroalimentar la información. Dado que el valor que s graba es el que había más 1, al

intentar obtener el valor previo a la incrementación, puede dar fallo, como se detallará en el capítulo *Simulaciones*.

ii. Si por cualquier motivo ha trabajado anteriormente con la memoria y se requiere de nuevo su uso, se puede restablecer sus posiciones a 0 sin necesidad de cortar la fuente de alimentación. Si se están realizando ensayos o simulaciones, es muy útil.

La función reset tiene simplemente dos entradas: **reset** y **clk**. Como se ha mencionado antes, todos los bloques trabajan con el mismo ciclo de reloj para garantizar la sincronía entre ellos. Generalmente, a la hora de iniciar un programa o módulo, todas las variables se suelen inicializar en 0. Por ese motivo, se he diseñado el módulo reset de manera que si vale 0, inicia a resetear la memoria, evitando olvidar iniciar el modo reset.

El reseteo comienza por la posición de memoria **addr** = 0, grabando en dicha posición el valor 0 (**din** = 0). A continuación se incrementa en 1 la dirección de memoria, es decir, **addr** = 1 y nuevamente se graba en esa posición el valor 0. Esta operación se repite hasta llegar a la última posición de memoria en la cual **addr** = n-1, donde *n* es el número total de posiciones de memoria. Esta operación necesita bastante tiempo para ejecutarse por completo debido a que son muchas posiciones de memoria, como en este caso, 2^{16} = 65.536 posiciones de memoria.

3.4. Módulo final

Diseñados los distintos bloques (memoria RAM, la máquina finita de estados y reset), el último paso es conexionarlos de manera que no haya interferencias entre las operaciones que realice cada uno y que se trabaje con una única memoria RAM.

Por tanto, se propone unir en un único módulo las funciones FSM y Reset, las cuales acceden a la misma memoria RAM. El módulo que unifica todas estas funciones queda como el que se muestra en la Figura 3.7 (Pixel), siendo sus entradas y salidas:

- **addr** como la dirección del píxel se dispara (input).
- **req** como indicador de que un píxel ha cambiado (input).
- **reset** para controlar el restablecimiento de la memoria (input).

- **clk** para la señal de reloj (input).
- **ack** para notificar que ya se ha recibido la información relativa al píxel (output).
- **dout** para poder acceder a la información almacenada en la memoria RAM (output).

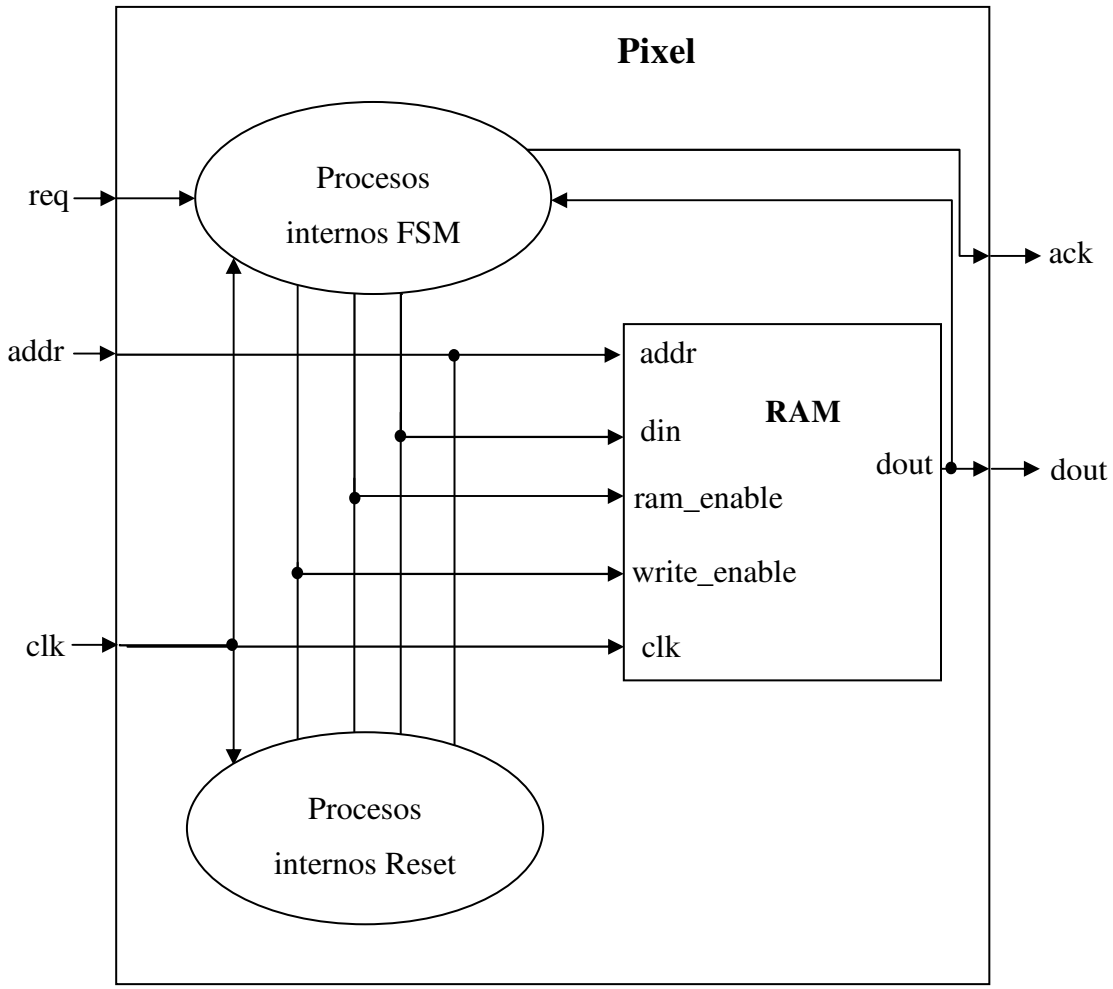


Figura 3.7. Diagrama del módulo final: Pixel

CAPÍTULO 4

SIMULACIONES

En este capítulo se presentan y detallan los resultados obtenidos tras la simulación del módulo **Pixel**, empezando por la **memoria**, seguida de los resultados tras la simulación función **Reset** y **FSM**, dejando para el final cómo se vería el módulo completo fijándose sólo en las entradas y salidas definidas.

Para ello, se emplean dos Testbench: el primero para asegurarse del correcto funcionamiento de la memoria; y el segundo para analizar el módulo en su conjunto. De éste último, se han obtenido los ejemplos que se muestran para ver las simulaciones de los bloques **Reset**, **FSM** y del módulo final **Pixel**.

4.1. Simulación Memoria RAM

Para el Testbench de la memoria, se fija el ciclo de reloj a 100 ns, habiendo por tanto un flanco positivo de reloj cada 50 ns. Este dato es muy útil puesto que todas las variables que dependan de la señal de reloj, se van a actualizar en los flancos positivos. La razón de escoger un ciclo de reloj tan pequeño es el interés de que se trabaje a alta velocidad, dado que la FPGA que se emplea trabaja con una señal de reloj de 200 MHz.

Como se puede ver en la Figura 4.1, inicialmente se configuran las entradas con los siguientes valores:

- addr = 0
- din = 1
- ram_enable = 1
- write_enable = 0

Dados esos valores iniciales, la memoria está en modo lectura. La salida, al no haberse reseteado previamente y desconocer su estado inicial, se indica en rojo y como valor X.

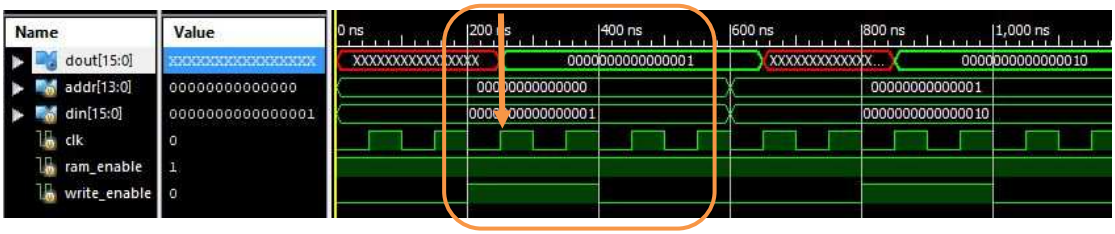


Figura 4.1. Ejemplo 1del funcionamiento de la memoria. Se señala el momento en el que comienza el modo escritura t=200 ns y el instante en el que se almacena t=250 ns.

Pasados 200 ns, se activa **write_enable**, permitiendo así que en el siguiente flanco positivo de reloj se almacene en **dout** el valor que indica **din**, que en ese instante es 1.

En la Figura 4.2 se puede apreciar que aún valiendo **din** =4 (en binario 100), **dout** sólo muestra lo que tiene almacenado en las posiciones de memoria que indica **addr**, con un retardo de 50 ns (cuando se dan los flancos positivos de **clk**).

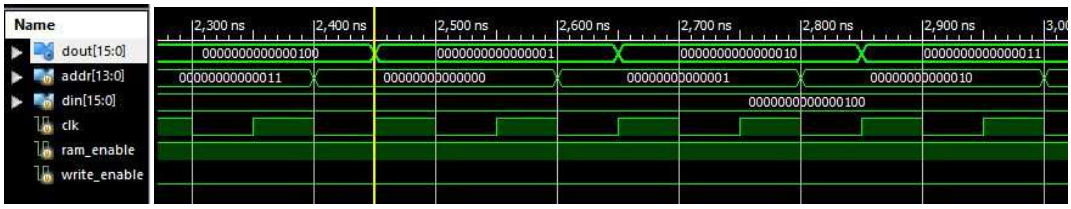


Figura 4.2. Ejemplo 2 del funcionamiento de la memoria. Modo lectura.

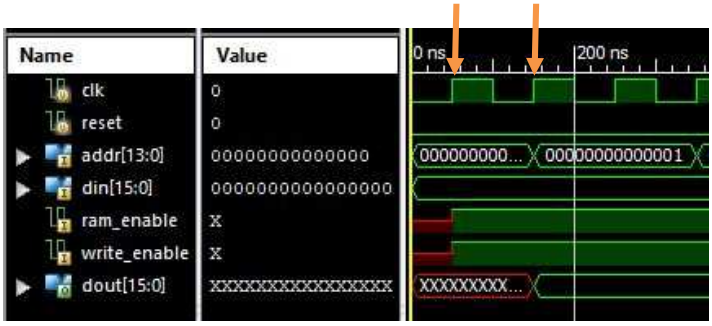
4.2. Simulación Reset

Como se indica en el apartado 3.3, la función Reset parte de la posición de memoria 0 y va incrementando de uno en uno la posición, grabando en cada una el valor 0.

Un momento crítico es nada más iniciarse. Al iniciar el módulo, la memoria no está accesible, debe haber primero un flanco positivo de reloj para que se habilite tanto el acceso como la escritura. Dado ese retardo, si se le pide que almacene y que aumente de posición en el mismo instante, no da tiempo a grabar en la posición 0 el valor 0, quedando por tanto como su valor como desconocido. Para evitar este problema, se incluye un retardo que depende de la señal de reloj **clk**, es decir, para que se cambie la posición de memoria **addr** (en ese momento de la posición 0 a la 1) debe esperar al

siguiente flanco positivo de reloj después de la activación de **write_enable**, como se señala en la Figura 4.3

A)



B)

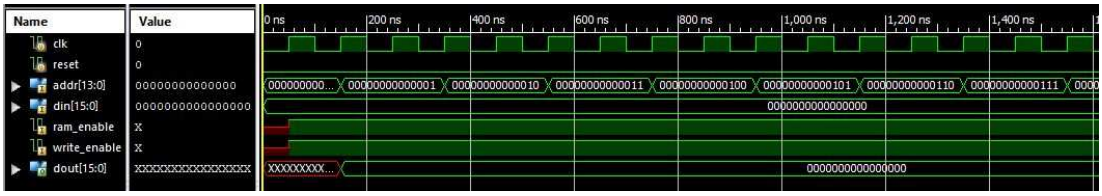


Figura 4.3. A) Inicio de la simulación de la función Reset. En el primer flanco positivo se activan ram_enable y write_enable, mientras que addr no cambia hasta pasado un flanco positivo más. B) Funcionamiento global de la función Reset.

Otro problema que presenta esta función, como ya se ha comentado anteriormente, es que precisa de mucho tiempo para resetear todas las posiciones de la memoria. Para reducir el tiempo habría que establecer una señal de reloj con un periodo mucho menor al que se ha tomado para realizar las simulaciones, el cuál es de 100 ns. Por ese motivo, el testbench se ha configurado para que durante 4 µs **reset** sea igual a 0, lo que equivale a resetear 20 posiciones de memoria con un ciclo de reloj de 100 ns. Pasado ese tiempo **reset** = 1 y con un retarde de 300 ns, comienza la función de la **FSM**.

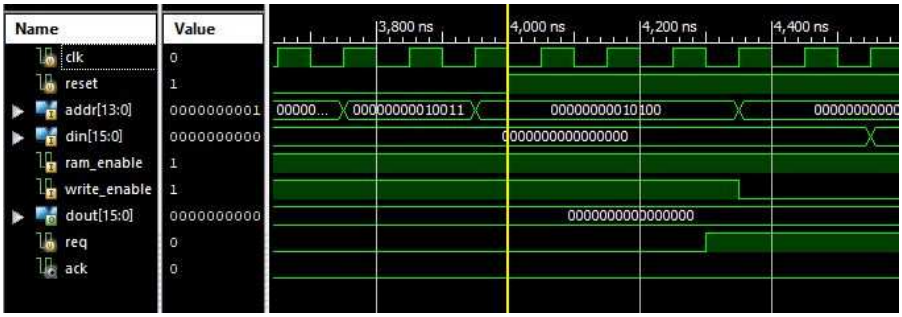


Figura 4.4. Ejemplo final de la función Reset en t=4µs y comienzo de la FSM en t=4,3µs

4.3. Simulación FSM

Como se ha mencionado en el apartado anterior, la máquina de estados comienza una vez que se desactiva la función **Reset** (**reset** = 1).

En esta función, los retardos son fundamentales. Dado que se realizan varias acciones en la memoria, si se le exige cambios con una velocidad inferior a la que necesita para llevar a cabo su cometido, puede conllevar errores y pérdida de información. Esto mismo ocurre en el ejemplo mostrado en la Figura 4.5, en la cual se ve que la dirección **addr** cambia bruscamente cuando la máquina de estados aún no está preparada para recibir nueva información (**write_enable** está aún a 1), de manera que por error se graba en la nueva posición indicada el valor 1, cuando realmente no ha habido ningún requerimiento por parte del sensor para esa dirección (**req** está a 0 en ese momento). En la realidad esto no debe pasar, ya que el cambio en **addr** está ligado a un nuevo requerimiento (**req** = 1). Aún así, el motivo de enseñar este error es el de ver que ocurriría en el casos anormales. Al ser una simulación, podemos poner en el testbench situaciones poco realistas pero que ayudan a comprender el funcionamiento del módulo diseñado.

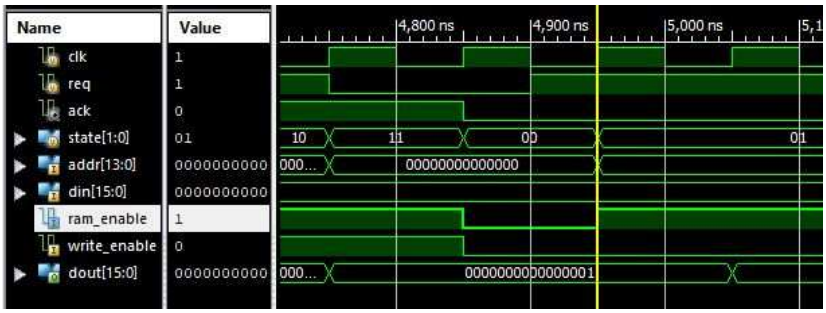


Figura 4.5. Ejemplo fallo en la FSM al cambiar addr cuando no está preparada para un nuevo evento.

Otro posible "fallo" que se puede dar a la hora de simular es que **req** pase a vale 0 antes de que se active **ack**. Puesto que en el testbench se introduce manualmente los valores de las entradas, es posible que en alguna ocasión no se ponga el retardo suficiente antes de cambiar **req** a 0, provocando que en ningún momento coincidan **req** y **ack** a 1. Es un fallo por el hecho de que la condición para que se desactive **req** es que se active **ack**, pero tal y como está diseñada la máquina de estados, no habría ninguna transición indebida ni se quedaría fijo en el estado en el que estaba.

Dejando a un lado esos errores poco frecuentes para el caso que se plantea, lo normal es que ocurra lo que se muestra en la Figura 4.6, donde **addr** cambia cuando hay una nueva solicitud (**req** = 1) y permanece fija durante todo el proceso de la máquina de estados, y donde **req** vale 0 una vez que **ack** vale 1. Se calcula que para un ciclo de reloj de 100 ns, entre la desactivación de **req** para un evento y la activación del siguiente evento, el retardo debe ser de al menos 450 ns y para que **req** valga 0 después de que **ack** valga 1, **req** se debe poner a 0 pasados al menos 200 ns después de que su activación. Este último dato afecta sólo a la hora de hacer un testbench, en la realidad, el sensor lo hará automáticamente.

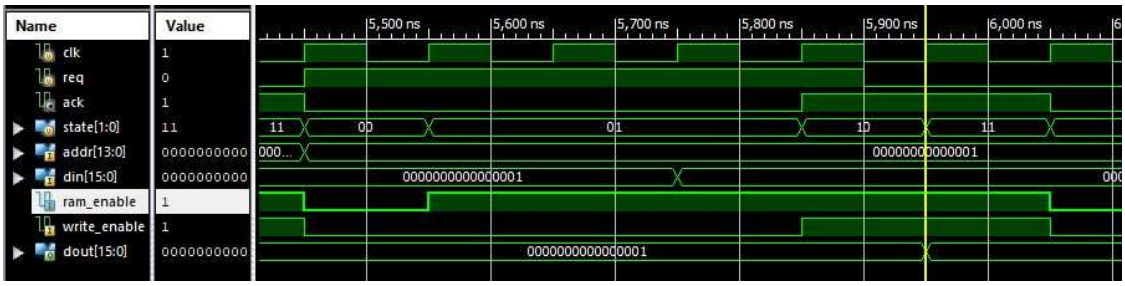


Figura 4.6. Ejemplo correcto funcionamiento de la FSM.

4.4. Pixel

Comprobado el correcto funcionamiento de todas las partes implicadas y analizados los posibles fallos que puedan surgir, se muestra el funcionamiento del módulo Pixel en su totalidad, empezando por el reseteo de la memoria y continuando con la máquina de estados, mostrando sólo las entradas y salidas que se definen para el mismo (**clk**, **reset**, **req** y **addr** como entradas, y **dout** y **ack** como salidas). Véase Figura 4.7 en la siguiente página (vertical).

Se observa que en un plazo de 11 µs, el módulo resetea las 20 posiciones de memoria y atiende posteriormente a las necesidades de comunicación del sensor. También se puede ver que, casi al final de la imagen, **dout** desconoce su valor. Eso se debe a que la dirección de memoria del píxel disparado no se ha reseteado y, por tanto, desconoce su estado previo (**addr** = 6.144, 01100000000000 en binario).

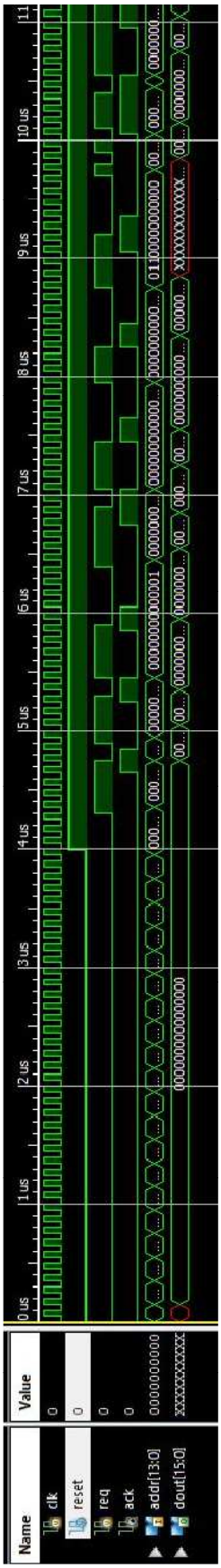


Figura 4.7. Ejemplo simulación módulo completo Pixel.

CAPÍTULO 5

CONCLUSIONES

Este proyecto tiene como fin la creación de una interfaz que mejore la lectura de datos, en este caso, provenientes de un sensor de visión basado en eventos. Dicha interfaz está a su vez basada en el protocolo de comunicación AER, el cual no sólo se puede enfocar para este tipo de proyectos sino también para otros campos como el de la biología, por ejemplo, para las conexiones neuronales con los nervios.

Esta interfaz mejora un montaje real, empleado para fines de investigación en la Escuela Superior de Ingeniería de la Universidad de Cádiz, haciéndolo más eficaz y moderno y permitiendo prescindir de dispositivos intermedios como el datalogger.

Se propone como trabajo a futuro la implementación de dicho módulo en la FPGA del montaje.

CAPÍTULO 6

PRESUPUESTO

Concepto	Precio unitario	Cantidad	Total €
Horas programador	25€/h	40h*	1.000€
Opal Kelly XEM7360	1.699,95$ (1.434,72€ aprox.)	1 ud.	1.434,72€
		Presupuesto final	2.434,72€

*Se incluyen horas para el aprendizaje del lenguaje Verilog.

BIBLIOGRAFÍA

1. M. Silvilotti, *Wiring considerations in analog VLSI systems with application to field-programmable networks*, Ph.D. dissertation, Cal. Inst. of Tech., Pasadena, California, 1991.
2. M. Mahowald, *An Analog VLSI System for Stereoscopic Vision*, Kluwer, 1994.
3. A. Mortara and E. A. Vittoz, *A communication architecture tailored for analog VLSI artificial neural networks: intrinsic performance and limitations*, IEEE Transactions on Neural Networks, vol. 5, no. 3, pp. 459–466, May 1994.
4. Tipos de memorias: *sites.google.com/site/electronicadigitaluvfime/5-1tipos-de-memorias-ram-rom-dram-sram*
5. Tipos de RAM: *es.wikipedia.org/wiki/Memoria_de_acceso_aleatorio*
6. Información técnica FPGA: *www.opalkelly.com/products/xem7360*
7. J. A. Leñero Bardallo, R. Carmona Galán, A. Rodríguez Vázquez, *A Wide Linear Dynamic Range Image Sensor Based on Asynchronous Self-Reset and Tagging of Saturation Events*, IEEE Journal of solid-state circuits (2017).
8. J. A. Leñero Bardallo, L. Farian, J. M. Guerrero Rodríguez, R. Carmona Galán, A. Rodríguez Vázquez, *Sun sensor based on a luminance spiking pixel array*, IEEE Journal.

Bibliografía empleada para el aprendizaje del lenguaje Verilog

1. J. Bhasker, *Verilog HDL Synthesis: A practical primer*, Star Galaxy Publishing, Pensilvania, 1998.
2. M. Morris Mano, *Diseño digital*, Pearson Educación, México, 2003.
3. K. Coffman, *Real World FPGA Design with Verilog*, Prentice Hall PTR, Nueva Jersey, 2000.
4. Thomas L. Floyd, *Fundamentos de sistemas digitales*, Pearson Educación, Madrid, 2006.
5. Thomas L. Floyd, *Digital fundamentals with VHDL*, Pearson Educación, Nueva Jersey, 2003.

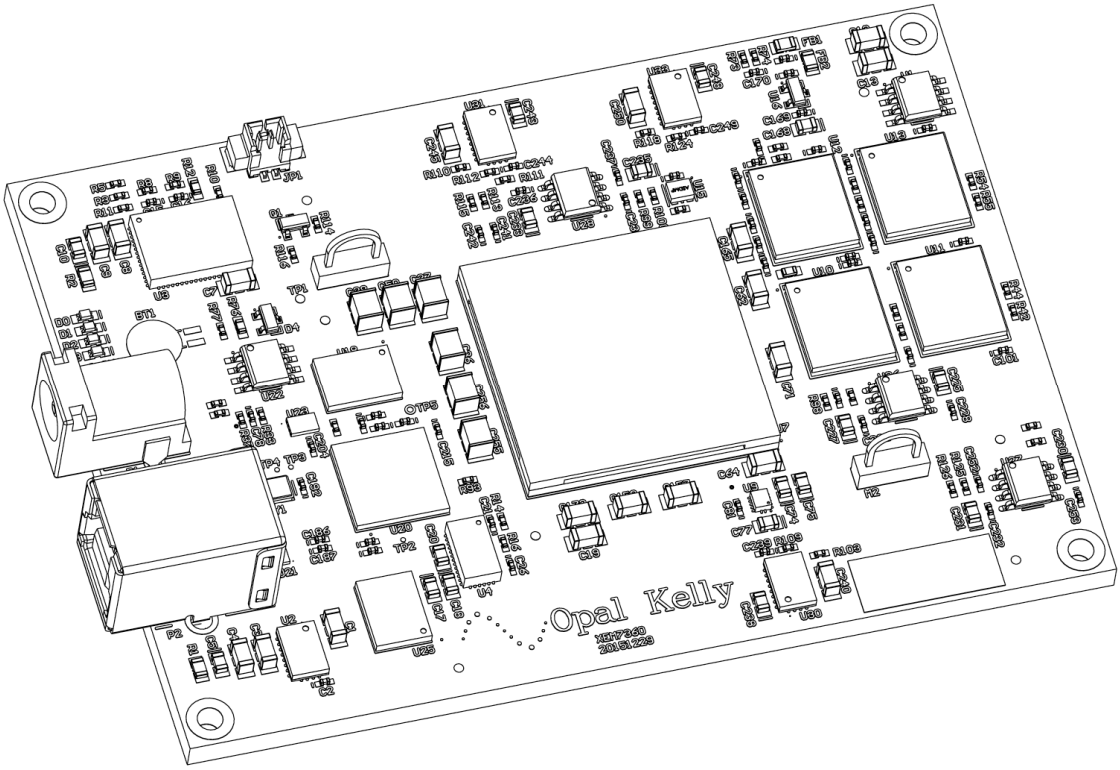
6. Ejemplo FSM en Verilog: *www.altera.com/support/support-resources/design-examples/design-software/verilog/ver_statem.html*

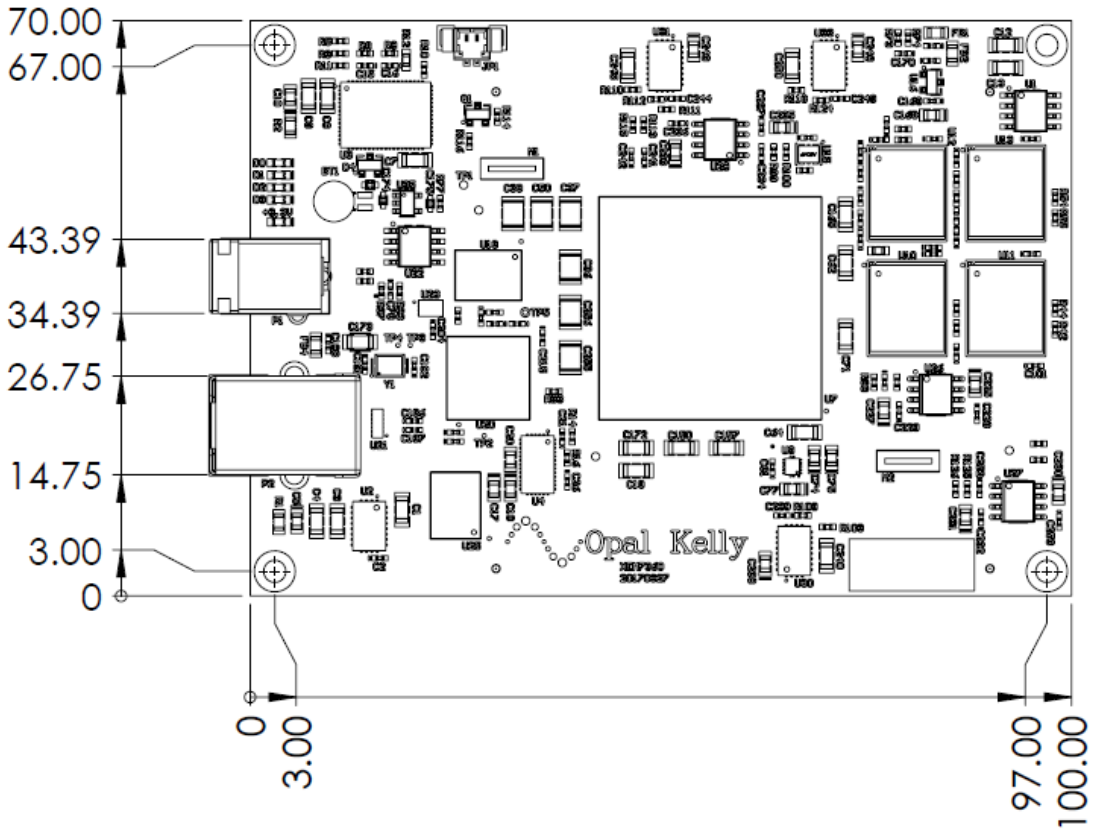
ANEXOS

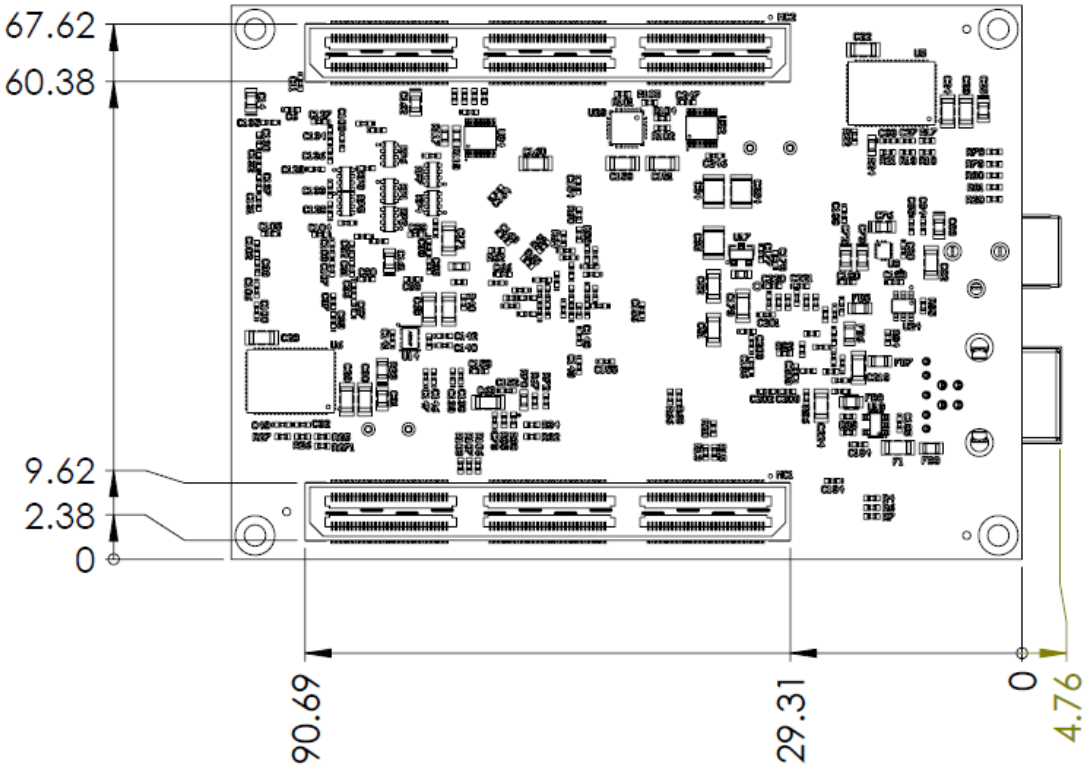
1. Información técnica sensor de visión HDR o basado en eventos

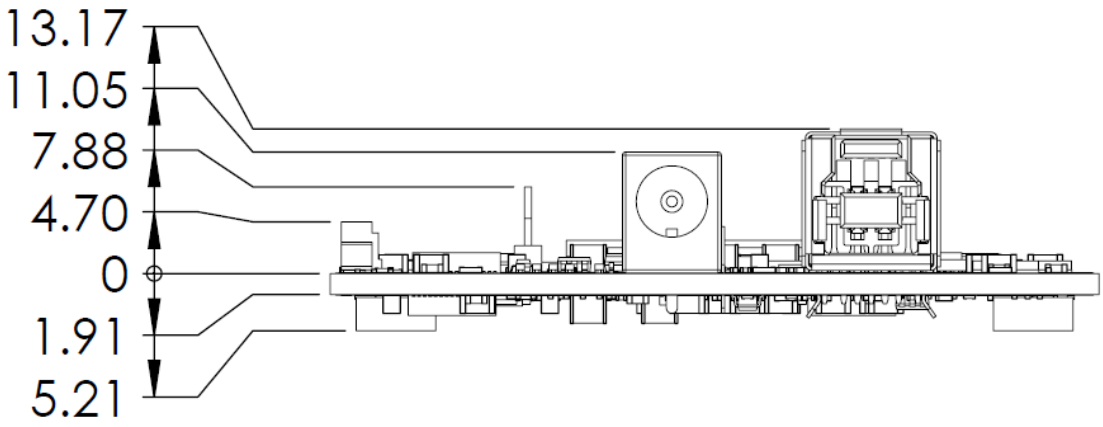
Technology	AMS 0.18µm HV
Power Supply	1.8V/5V
Chip Dimensions	4120µm× 3315µm
Pixel Size	25µm × 25µm
Number of Pixels	96×128
Fill Factor	10%
Dynamic Range	>100dB
Event sensitivity	0.0762events/$\Delta V \cdot lux$
Power Consumption	52mW, 100keps
Sense Node Capacitance	45fF
FPN (event output)	2.6%
Max. event rate	2Meps (same row), 10Meps (different rows)

2. Opal Kelly XEM7360






67.62
60.38
9.62
2.38
0
90.69
29.31
0
4.76


13.17
11.05
7.88
4.70
0
1.91
5.21