Escola Universitària d'Enginyeria
Tècnica Industrial de Barcelona
Consorci Escola Industrial de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

**Memòria Tècnica**

# "PLATAFORMA PER A UNA CÀMERA INTEL·LIGENT"

TFG presentat per optar al títol de GRAU en ENGINYERIA ELECTRÒNICA INDUSTRIAL I AUTOMÀTICA

per **Xavier Pidas i Pagerols**

Barcelona, 8 de Juny de 2016

Director: Jordi Cosp i Vilella
Departament d'EEL
Universitat Politècnica de Catalunya (UPC)

# ÍNDEX MEMÒRIA TÈCNICA

# RESUM

Aquest projecte consisteix en la realització d'una plataforma electrònica pel control i el posterior processament d'una imatge per a una càmera. Aquesta plataforma està basada en un entorn per a FPGA, concretament s'utilitza la *Nexys2 Board* de Digilent que incorpora la *Spartan-3E* de Xilinx. Mitjançant el llenguatge de descripció hardware VHDL, s'aconsegueix dissenyar i connectar els diferents blocs per controlar tots els elements que involucra aquesta tasca.

S'ha dissenyat el dispositiu per tal que segueixi diferents d'algorismes (binarització, erosió, dilatació, obertura i tancament) per dur a terme el processament de la imatge capturada i així poder extreure el contorn de diferents objectes.

# RESUMEN

Este proyecto consiste en la realización de una plataforma electrónica para el control y el posterior procesamiento de una imagen para una cámara. Esta plataforma se basa en un entorno para FPGA, concretamente se utiliza la *Nexys2 Board* de Digilent que incorpora la *Spartan-3E* de Xilinx. Mediante el lenguaje de descripción hardware VHDL, se consigue diseñar y conectar los diferentes bloques para controlar todos los elementos que involucra esta tasca.

Se ha diseñado el dispositivo para que siga diferentes algoritmos (binarización, erosión, dilatación, apertura y cierre) para llevar a cabo el procesamiento de la imagen capturada i así poder extraer el contorno de diferentes objetos.

# ABSTRACT

This project involves the realization of an electronic platform for the control and further processing of an image by a camera. This platform is based on an FPGA environment, specifically is used *Nexys2 Board* from Digilent that incorporates *Spartan-3E* from Xilinx. Using the hardware description language VHDL, different blocks are designed and connected to control the whole elements that involve this task.

The device has been designed to follow some algorithms (binarization, erode, dilate, open and close) to do the processing of the captured image so the contour of different objects can be extracted.

# AGRAÏMENTS

Agraeixo especialment al meu tutor Jordi Cosp l'ajuda i consells que m'ha anat donant al llarg d'aquests mesos.

També vull agrair els ànims rebuts per part dels companys de feina d'AIS Vision Systems.

I a la meva família, per donar-me suport en totes les meves decisions.

# CAPÍTOL 1:

# INTRODUCCIÓ

## 1.1. Introducció i antecedents

La visió artificial és un camp de la ciència que permet adquirir, processar, representar i finalment interpretar una imatge per tal d'extreure'n la informació necessària per prendre decisions i així controlar un procés. És una disciplina que en els últims anys ha evolucionat significativament i ha agafat protagonisme en les línies de producció industrial, ja que un sistema de visió artificial permet eliminar defectes de fabricació mitjançant la inspecció del 100% dels productes, millorant la qualitat tant del procés com del producte i sense cap tipus de contacte físic. S'utilitzen en un gran ventall d'indústries, com poden ser automoció, alimentació o en el sector farmacèutic.

Un element molt important per a un sistema de visió artificial és la càmera que s'utilitza, la qual requereix d'una sèrie de característiques que permetin el control del disparador per capturar la imatge en el moment desitjat i són més complexes que les càmeres convencionals, ja que es necessita velocitat i qualitat per igual. Es poden classificar com a càmeres lineals, que construeixen la imatge línia a línia realitzant un escombrat de l'objecte mentre aquest es desplaça, o com a càmeres matricials, amb les quals es capturen imatges bidimensionals, tot i que en molts casos s'utilitzen les anomenades càmeres intel·ligents. Són dispositius electrònics que capturen una imatge i la processen sense cap ajuda externa d'un computador, és a dir, és una càmera capaç d'analitzar una imatge i, segons com estigui programada, genera diferents senyals de sortida per tal de poder interactuar amb el seu entorn. Tenen moltes possibles aplicacions en la industria degut al gran potencial que tenen per inspeccionar i comprovar les diferents etapes d'una línia automatitzada. Per exemple, en una línia d'etiquetatge i taponat d'ampolles, es pot verificar que l'etiqueta estigui ben impresa i enganxada i a més comprovar que el tap s'ha col·locat correctament. En el cas que algun element

estigui mal col·locat o en mal estat, la mateixa càmera pot avisar el PLC de la línia i així actuar en conseqüència. Altres aplicacions poden ser de verificació de codis de barres o codis matricials, reconeixement de caràcters, fer mesures de precisió, identificació de patrons, classificació i verificació de colors, etc.

En altres sistemes de visió artificial amb diferents característiques també es possible utilitzar processadors de visió, els quals proporcionen la màxima prestació en el processament d'imatges amb una major flexibilitat a l'hora de treballar amb més d'una càmera a la vegada (sistema multicàmera). S'utilitzen en sistemes en els que es necessita analitzar imatges d'alta resolució, en que s'ha d'inspeccionar el producte a alta velocitat o en els que es necessita utilitzar més d'una càmera.

Els softwares que es fan servir en el món de la visió artificial aplicada en la industria han evolucionat considerablement des de la revolució tecnològica que es va iniciar en els anys 80. En el passat no es podien implementar sistemes de visió artificial a temps real degut a que els ordinador d'aquells temps no eren suficientment ràpids per processar tota la informació però, amb l'evolució dels processadors fins als que hi ha actualment, s'ha aconseguit visualitzar les imatges a temps real i processar-les en temps raonables (ms). S'utilitzen softwares especialitzats que treballen directament sobre la imatge i que disposen de diferents eines per tal que els sistemes de visió es puguin aplicar en les diferents aplicacions industrials que s'han anat comentant.

Un exemple d'un sistema comercial similar al que es fa en aquest treball són les càmeres intel·ligents del fabricant Datalogic, el qual també ha desenvolupat un software especialitzat per a aquestes, anomenat IMPACT.

***Figura 1a.*** *Càmeres intel·ligents de les sèries A i T de Datalogic.*


***Figura 1b.*** *Processador de visió de la sèrie MX de Datalogic.*

Tot i que l'aparença exterior és bastant diferent a la del prototip dissenyat, aquestes càmeres també estan basades en un entorn per a FPGA per capturar i processar la imatge i comunicar-se amb l'exterior utilitzant protocols de comunicació RS232, Ethernet o simplement com a I/O. El gran avantatge de treballar amb FPGA és que es poden executar diferents processos en paral·lel fent que es pugui treballar més ràpidament, a diferència dels microcontroladors, els qual segueixen un ordre seqüencial

seguint el codi que se'ls ha programat. En el cas que es vulgui processar les imatges utilitzant un PC o un processador de visió (figura 1b), hi ha l'avantatge que són molt més potents que una FPGA o un simple microcontrolador, tot i que el preu es dispara. És per aquest motiu que en aquest treball s'ha intentat dissenyar un prototip de càmera intel·ligent basat en un entorn FPGA per tal de millorar la velocitat de processament i fent que el cost final del material utilitzat no sigui gaire elevat.

## 1.2. Objecte i abast del treball

L'objectiu d'aquest treball consisteix en realitzar una plataforma electrònica en un entorn per a FPGA per tal de capturar imatges amb una càmera i aplicar sobre elles diferents filtres. Mitjançant el llenguatge de descripció hardware VHDL i utilitzant un seguit d'algorismes i tècniques del camp del processament morfològic, s'aconsegueix detectar i representar en una pantalla els contorns dels objectes capturats. Per dur a terme aquesta tasca s'han de complir un seguit d'objectius:

- Tenir una càmera plenament funcional, és a dir, que es puguin obtenir imatges i presentar-les correctament a la pantalla.
- Cal el correcte disseny i funcionament de quatre blocs principals:
    - El bloc **CAM DRIVER** té la funció de programar la càmera i adquirir les dades de la imatge que s'està capturant.
    - El bloc **RAM DRIVER** gestiona el control de la memòria RAM externa.
    - El bloc **VGA DRIVER** genera els senyals VGA necessaris per representar la imatge capturada a la pantalla.
    - El bloc **PROCESSING IMAGE** és l'encarregat de processar la imatge capturada.
- Tenir control total sobre la programació de la càmera per poder decidir com se sincronitza, el mode de funcionament, el format de les dades de sortida, etc.
- Tenir control total sobre la memòria RAM externa per treballar en mode síncron i asíncron a l'hora d'escriure o llegir dades.
- Tenir control total sobre la pantalla a través del port VGA que incorpora la placa.
- Cal que el bloc encarregat del processament apliqui els algoritmes adients depenent de quin tipus de filtre es vulgui aplicar a la imatge capturada.

# CAPÍTOL 2: DISSENY DEL PROTOTIP

En aquest capítol es presenta el disseny dels blocs programats en VHDL per implementar a la FPGA. S'explica el funcionament dels blocs principals per tal que es garanteixi el funcionament mínim de la càmera, és a dir, únicament la captura d'una imatge i presentar-la correctament en una pantalla.

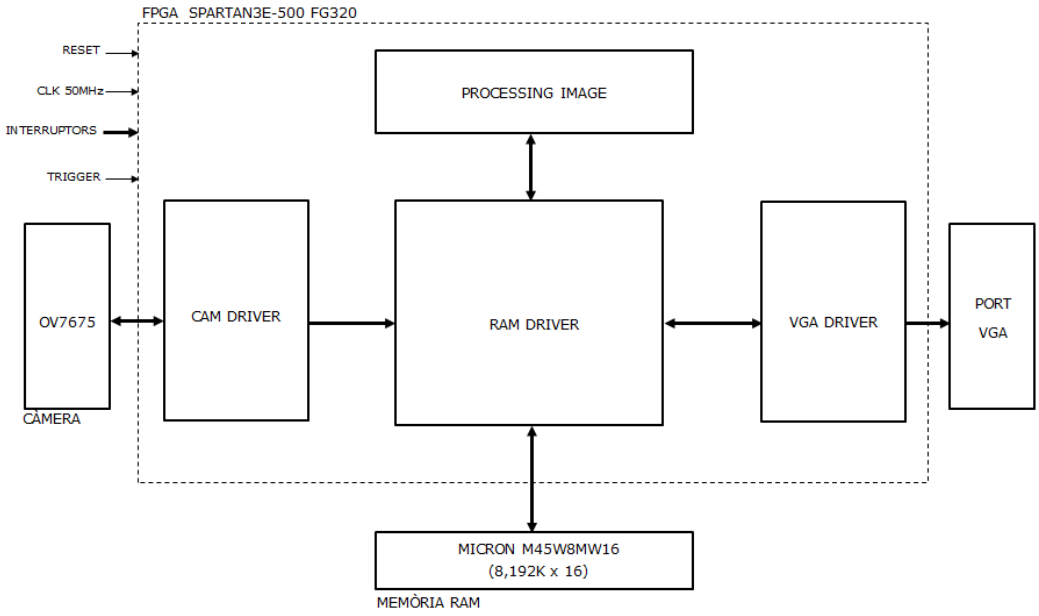


***Figura 2.*** Diagrama de blocs dissenyat simplificat.

## 2.1. Aspectes previs

La plataforma que controla la càmera, processa la imatge i genera els senyals VGA per a la pantalla, està basada en un entorn per a FPGA (*Field-Programmable Gate Array*). Per fer aquest dispositiu s'ha utilitzat una placa per a dissenyar i desenvolupar prototips digitals de l'empresa Digilent, la *Nexys2 Board* que incorpora la FPGA *Spartan3E* de Xilinx. Les característiques d'aquesta placa més destacables i importants pel disseny del dispositiu són les següents:

- La FPGA *Spartan3E* té de l'ordre de 500K portes.
- Es pot alimentar i programar la placa a través d'un cable USB des del PC.
- Incorpora una memòria RAM externa de 16MBytes del fabricant Micron.
- Disposa d'un oscil·lador de 50 MHz per proporcionar senyal de rellotge.
- Té fins a quatre connectors d'expansió de 12 pins per tal de connectar-hi altres dispositius exteriors, en aquest cas la càmera. Aquets connectors poden proporcionar una tensió d'alimentació de 3,3V que s'utilitzaran per alimentar-la.
- Porta integrats diferents elements per interaccionar amb l'exterior com són LEDs, polsadors i interruptors.
- Finalment disposa d'un port VGA amb una resolució de 8 bits (256 colors) per connectar una pantalla.

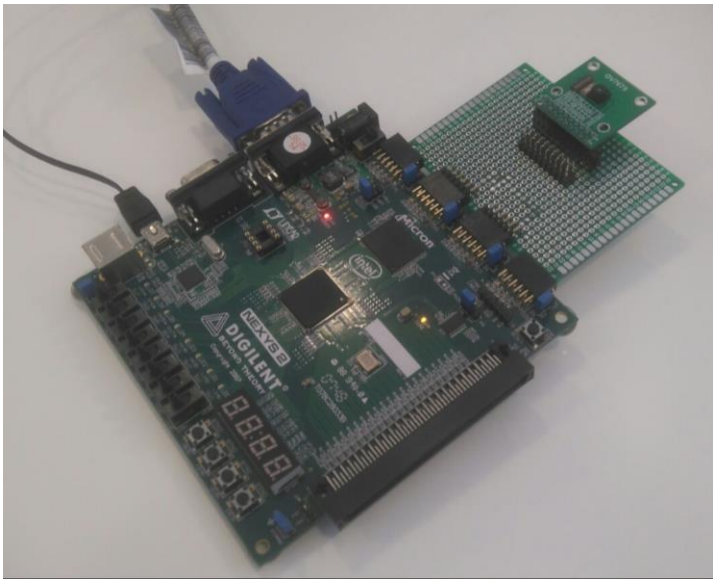


**Figura 3.** *Fotografia de la placa amb la càmera connectada.*

El llenguatge utilitzat per programar la FPGA és el VHDL (*Very high speed integrated circuit – Hardware Despriction Lenguage*), concretament amb l'entorn ISE de Xilinx. Aquest llenguatge va ser desenvolupat i posteriorment adoptat com un estàndard oficial de l'IEEE l'any 1987, anomenat IEEE 1076. Es va fixar l'any 1993 amb el nom de IEEE 1164 (Brown and Vranesic 2009).

És un llenguatge utilitzat per enginyers per descriure circuits digitals. A diferència d'altres llenguatges com C o Pascal, que executen cada una de les seves línies de codi de manera seqüencial, una de les característiques més importants és la seva concurrència, és a dir, totes les línies i tots els mòduls descrits s'executen a l'hora. Juntament amb l'altra característica principal, que és la capacitat de descriure un circuit digital complex com el conjunt de mòduls i sub-mòduls més simples, dóna una gran flexibilitat i és molt adient per l'aplicació ens ocupa.

Hi ha dues maneres de descriure un circuit digital mitjançant aquest llenguatge: amb arquitectura estructural i amb arquitectura de comportament.

- **Arquitectura estructural:** una manera de descriure un circuit digital complex és utilitzant diferents mòduls més simples. Utilitzant aquesta arquitectura es pot descriure com les entrades i sortides dels mòduls es connecten entre elles. És possible que un mòdul estigui descrit com la suma d'altres sub-mòduls connectats, fent que la descripció final d'un circuit digital complex sigui més simple.

- **Arquitectura de comportament:** és la descripció d'un mòdul, és a dir, descriu com funciona i com es comporta un circuit digital concret. Aquest tipus d'arquitectura s'utilitza per descriure els sub-mòduls de més baix nivell per així aconseguir que els mòduls de nivells superiors funcionin.

Seguint aquesta lògica, el circuit digital dissenyat es basa en quatre blocs o mòduls principals. Tres blocs tenen la funció de controlar a tres elements externs respectivament:

- El bloc **CAM DRIVER** és l'encarregat del control de la càmera. Té la funció de programar-la i adquirir les dades de la imatge que s'està capturant.

- El bloc **RAM DRIVER** gestiona el control de la memòria RAM externa per tal de llegir-hi o escriure-hi les dades provinents de la càmera.

- El bloc **VGA DRIVER** genera els senyals VGA necessaris per representar la imatge capturada a la pantalla.

Un quart bloc anomenat **PROCESSING IMAGE** és l'encarregat del processament de la imatge capturada, el qual transforma el dispositiu en una càmera "intel·ligent".

Per últim també hi ha un bloc divisor de freqüència, el qual divideix la freqüència d'entrada de 50 MHz de l'oscil·lador per dos i així s'obté una freqüència de sortida de 25 MHz. Tant la freqüència de 50 MHz com la de 25 MHz s'utilitzen en els diferents blocs depenent de la freqüència de treball que es requereixi.

A continuació es pot observar com estan connectats entre ells els quatre blocs principals. Com s'ha comentat anteriorment, cada bloc és independent dels altres i treballen de forma simultània, de manera que en els següents apartats es detalla com funciona cada bloc per separat així com s'exposen les característiques dels elements externs com són la càmera i la memòria RAM.

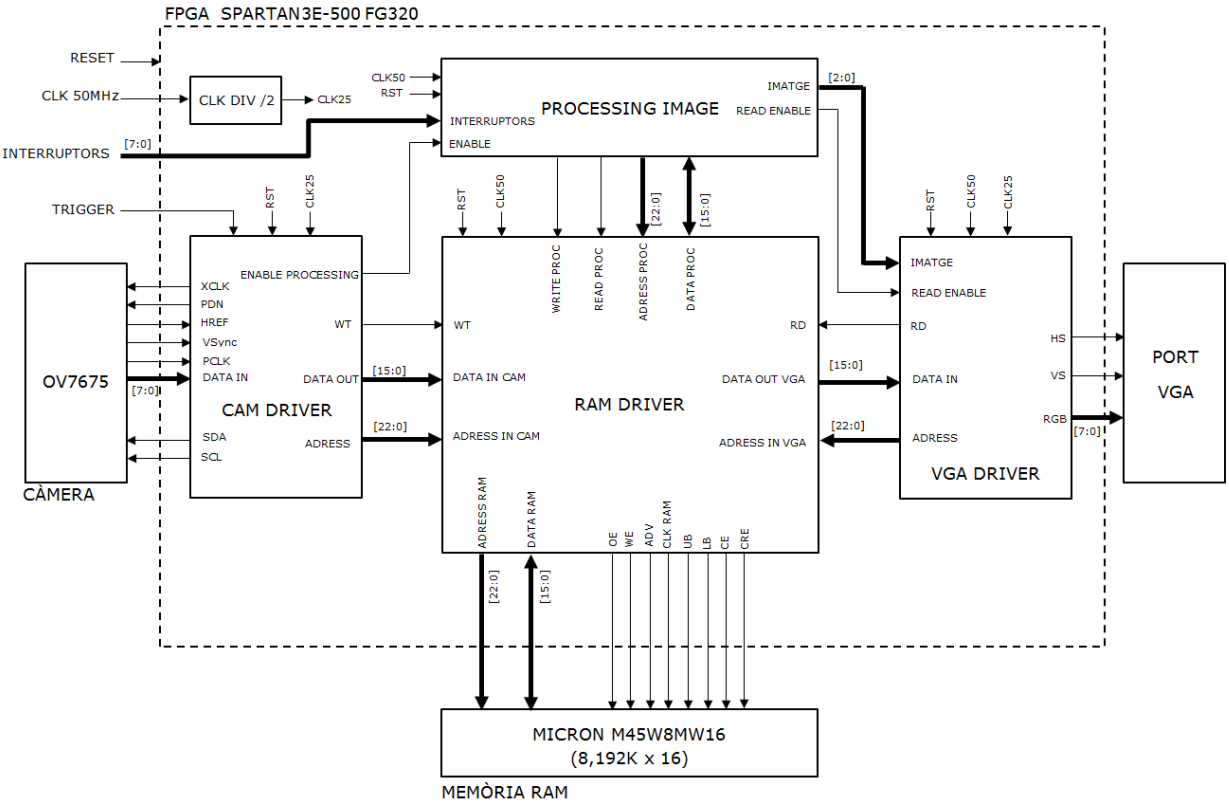


***Figura 4.*** *Diagrama de blocs general dissenyat.*

# 2.2. Bloc controlador de la càmera

En aquest apartat s'explica el funcionament de la càmera OV7675 i el seu control mitjançant la FPGA, gràcies al bloc CAM DRIVER dissenyat per controlar, programar i adquirir les dades de la imatge que s'està capturant.

## *2.2.1. La càmera OV7675*

La càmera escollida és la OV7675 del fabricant OmniVison. És un sensor d'imatge de color del tipus CMOS de dimensions reduïdes (1/9") amb una resolució VGA (640x480 píxels) a una velocitat de 30 fotogrames per segon (fps). La càmera pot ser programada a través d'un bus sèrie anomenat SCCB (*Serial Camera Control Bus*), amb el que l'usuari pot configurar

diferents característiques del seu funcionament com pot ser la sincronització o el format de les dades de sortida.

Les característiques més importants són les següents:

- Resolució: VGA (640x480), QVGA (320x240), QQVGA (160x120).
- Format de les dades de sortida: YUV4:2:2, Raw RGB, ITU656, RGB 565, RGB555 i RGB444.
- Interfície de sortida en paral·lel - DVP (*Digital Video Port*).
- Incorpora un PLL[1] intern (*Phase Lock Loop*).
- Freqüència de rellotge d'entrada entre 1,5 i 27 MHz.
- Comunicació i programació a través del bus SCCB.

Degut a que la càmera en sí és inaccessible per poder-la connectar a la placa *Nexys2*, s'ha optat per comprar una placa d'adaptació amb la càmera incorporada i preparada per ser connectada.


***Figura 5.*** *Placa d'adaptació amb la OV7675.*

Com es pot observar en la figura 5, la placa d'adaptació té disponibles 18 pins per poder interactuar amb l'entorn. A continuació es descriuen tots i cada un d'ells i quina funció tenen:

- **Pin 1 – Pin2:** VCC i GND. Són els pins per on s'alimenta la càmera. S'alimenta a partir de la placa *Nexys2* ja que els seus connectors d'expansió proporcionen una tensió d'alimentació de 3,3 V.
- **Pin 3:** SCL. És el senyal de sincronisme del bus SCCB.
- **Pin 4:** SDA. És el senyal de dades del bus SCCB.
- **Pin 5:** VS. És el senyal de sincronisme vertical. Indica l'inici de captura d'una imatge.

[1] Un PLL és un dispositiu electrònic capaç de copiar freqüències utilitzat en desmodulació de senyals FM, multiplicadors i divisors de freqüència, oscil·ladors de precisió, etc.

- **Pin 6:** HREF. És el senyal de sincronisme horitzontal. Indica quan hi ha dades vàlides a la sortida.

- **Pin 7:** PCLK. És el rellotge de píxel. Indica quan es pot llegir un byte a la sortida.

- **Pin 8:** XCLK. És el senyal de rellotge que es proporciona a la càmera. Des de la *Nexys2* se li proporciona un senyal de rellotge a 25 MHz.

- **Pin 9 – Pin 16:** Dades. Proporcionen la informació dels píxels de la imatge capturada.

- **Pin 19:** PEN (*Power Enable*). Activa els reguladors de tensió que té la placa d'adaptació. S'ha de connectar a 3,3 V.

- **Pin 20:** PDN (*Power Down*). Quan es posa a '1' la càmera es posa en *standby*.

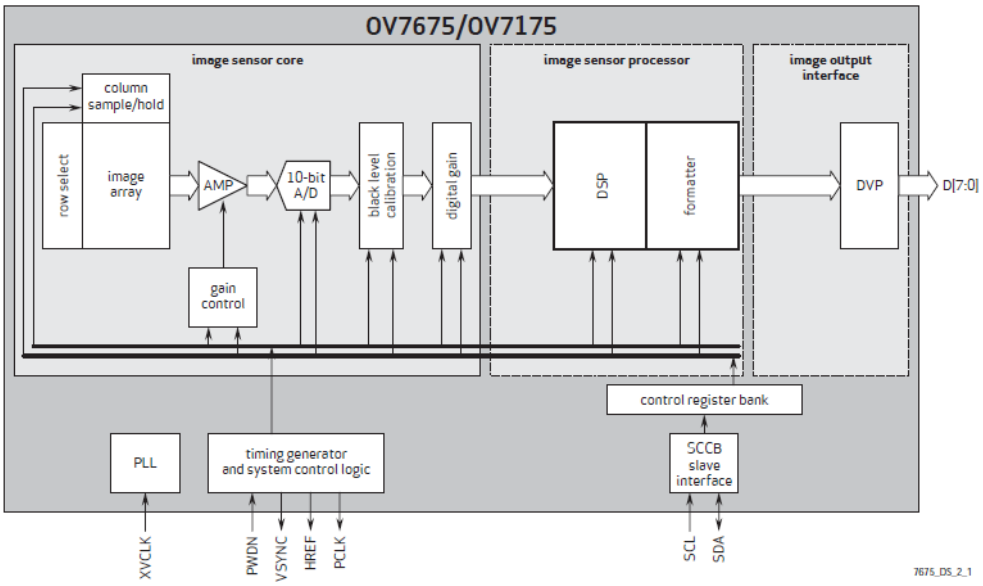


***Figura 6.*** *Diagrama de blocs de la OV7675, amb les entrades i sortides.*

Pel que fa al funcionament a l'hora de transmetre la informació del color de cada píxel, es fa a través dels 8 pins de la interfície de sortida (DVP). La sincronització es fa mitjançant els pins 5 i 6 (VS i HS) seguint el diagrama de la figura 7. Es pot observar que quan es comença a capturar una imatge, el senyal VS es posa a '1' durant un cert període de temps. Tot seguit, el senyal HS es posa a '1' quan es comença a transmetre la informació d'una línia, i així successivament fins que s'ha transmès la informació de les 480 línies. Mentre HS està a '1', la freqüència a la que els bytes es transmeten ve donada pel rellotge PCLK, fent que a cada flanc de pujada estigui disponible un byte a la sortida. Després de transmetre la informació dels 640 píxels de cada línia, el procés torna a començar quan VS es torna a posar a '1'.

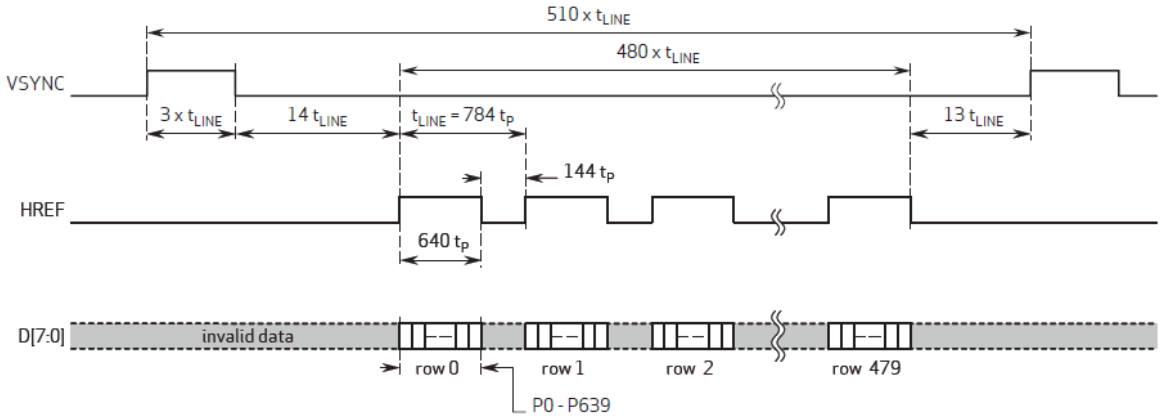


***Figura 7.*** *Diagrama de sincronització de la càmera per a una resolució VGA.*

Com s'ha comentat anteriorment, es pot triar en quin format es vol transmetre la informació. Com que s'utilitza un port VGA, el qual utilitza informació en RGB[2] per a representar les imatges a la pantalla, s'ha optat per programar la càmera en aquest format. Concretament s'ha triat el format RGB444, el qual utilitza quatre bits d'informació per definir la intensitat de cada color primari, fent que cada píxel de la imatge ocupi dos bytes d'informació. Així doncs, per transmetre la informació d'un píxel es necessari fer-ho en dos períodes de PCLK, ja que només es pot transmetre un byte a cada flanc de pujada. En el primer byte que es transmet, s'envia la informació de la intensitat de vermell en els últims quatre bits ($XXXXR_3R_2R_1R_0$). En el segon byte es transmet la intensitat del color verd i blau respectivament ($G_3G_2G_1G_0B_3B_2B_1B_0$), i així successivament per a tots els píxels de la imatge. Cal comentar que la freqüència de PCLK típica que dóna la càmera és de 24 MHz tot i que mitjançant el bus SCCB es pot modificar i dividir-la.

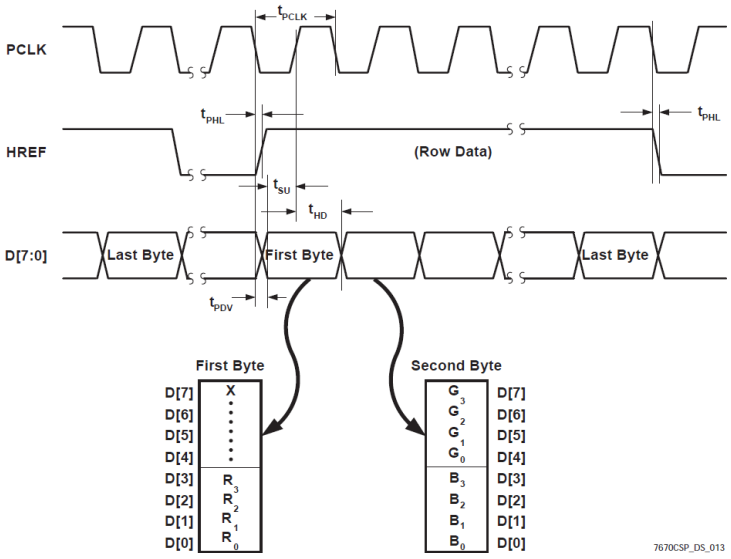


***Figura 8.*** *Diagrama de sincronització pel format RGB444.*

[2] El model de color RGB indica la intensitat de cada un dels tres colors primaris (vermell, verd i blau) per formar qualsevol altre color.

Per configurar la càmera s'utilitza el bus sèrie SCCB, molt semblant a l'estàndard de comunicacions sèrie I2C[3]. El protocol SCCB va ser creat i desenvolupat per l'empresa OmniVision Technologies per controlar les funcions de les seves pròpies càmeres. Està pensat per ser un bus sèrie de tres cables però per a càmeres de dimensions reduïdes, com en aquest cas, el bus SCCB pot treballar només amb dos.

El protocol de transferència de dades dicta que la càmera ha de ser l'esclau i que només el mestre pot iniciar comunicació. En aquest cas el mestre serà la FPGA, concretament el bloc CAM DRIVER, encarregat del control i programació de la càmera, el qual haurà d'actuar sobre les línies SDA i SCL quan es vulgui iniciar comunicació amb la càmera. El senyal SCL es un senyal unidireccional de la FPGA cap a la càmera que serveix de sincronisme per a la transmissió dels bits a través de la línia SDA.

Per què hi hagi una transmissió cal que se segueixin un seguit de seqüències. Primer de tot el mestre ha d'indicar a l'esclau que es vol iniciar una comunicació fent la seqüència *Start* (figura 9a). Tot seguit cal indicar amb quin esclau es vol mantenir la comunicació i transferir les dades. Un cop acabada la transmissió, cal que el mestre faci la seqüència *Stop* (figura 9b) per tal de comunicar a l'esclau que s'ha acabat la transmissió.

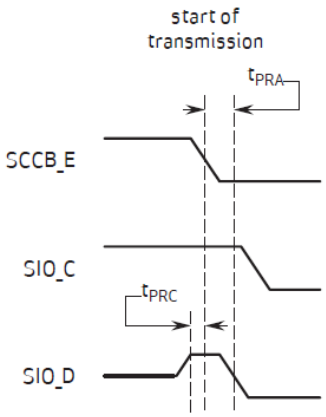


***Figura 9a.*** *Sequència Start.*

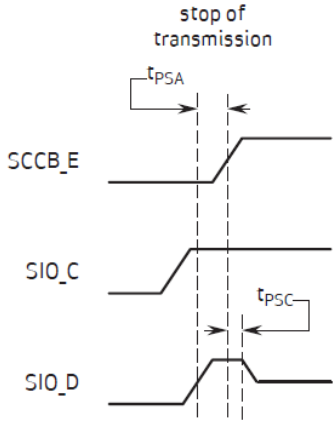


***Figura 9b.*** *Sequència Stop.*

Aquestes dues seqüències es caracteritzen perquè el senyal SDA (equivalent a SIO_D) canvia quan SCL (equivalent a SIO_C) està en estat alt. En la seqüència *Start* el senyal SDA passa a '0' i en la seqüència *Stop* passa a '1' mentre SCL està a '1' en els dos casos. Cal notar que en les figures hi apareix un tercer senyal anomenat SCCB_E que cal ignorar ja que està pensat per quan es treballa en el mode de tres línies.

Un cop feta la seqüència *Start* per iniciar comunicació, comença la seqüència d'escriptura o lectura de dades. En aquesta aplicació només s'escriuran dades ja que només interessa definir el format de les dades de

[3] El bus I2C és un estàndard de comunicació sèrie síncron pensat per interconnectar només amb tres fils (SDA, SCL i GND) circuits integrats que intercanvien grans quantitats d'informació. Va ser desenvolupat l'any 1982 per Philips Semiconductors.

sortida. L'escriptura de dades segueix tres fases, on a cada una s'envien 8 bits d'informació:

- **Fase 1:** el mestre envia l'adreça de l'esclau (*IP adress*) amb el que desitja mantenir comunicació. L'adreça està composta de set bits ordenats del bit 7 al bit 1 i és única per a cada dispositiu connectat al bus. El bit menys significatiu (bit 0) és el selector d'escriptura o lectura i indica si el mestre vol llegir o escriure dades. Si aquest bit és un '1' significa que el mestre vol llegir dades, en canvi si és un '0' significa que vol escriure'n. L'adreça per a la càmera OV7675 és 0x42 per escriure i 0x43 per llegir.

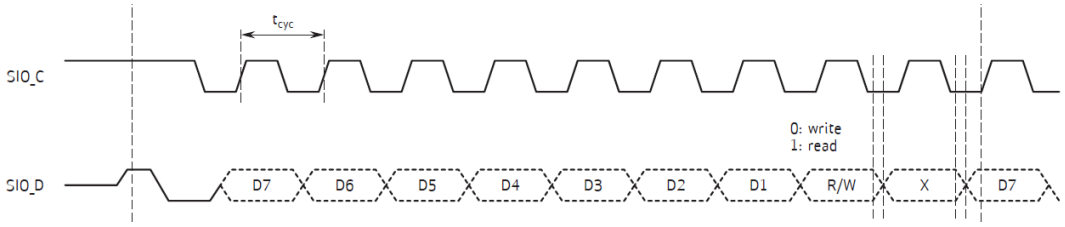


***Figura 10.*** *Seqüència Start seguit de Fase 1: enviar IP adress.*

- **Fase 2:** en aquesta fase el mestre envia l'adreça del registre al que vol accedir (*subadress*).

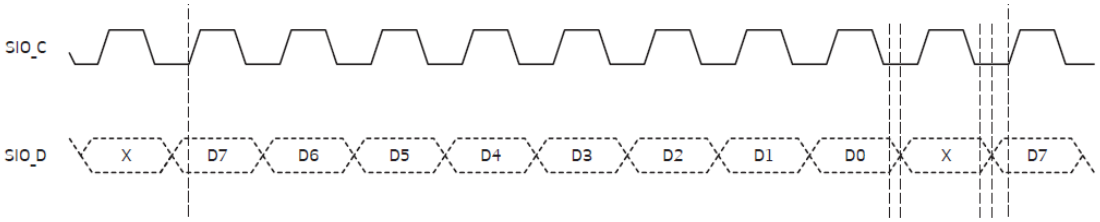


***Figura 11.*** *Fase 2: enviar subadress.*

- **Fase 3:** en aquesta última fase el mestre envia les dades que vol escriure al registre desitjat. Finalment s'executa la seqüència *Stop*.

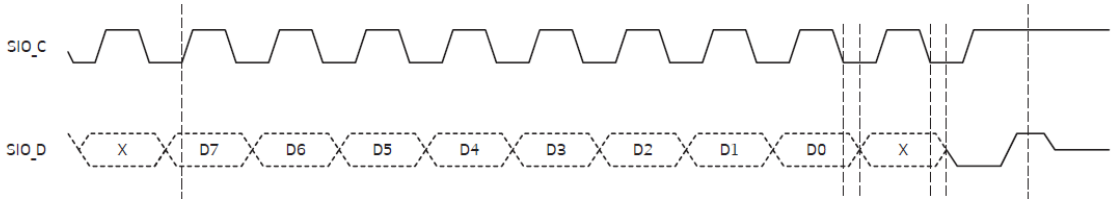


***Figura 12.*** *Fase 3: enviar dades a escriure seguit de seqüència Stop.*

Com es pot observar en les tres fases, el mestre transmet la informació començant pel bit de major pes i s'envien els bits per ordre descendent fins al bit de menor pes. Per enviar un bit el mestre posa l'estat de la línia SDA igual que el bit a enviar ('1' o '0') i la manté mentre genera un pols a la línia SCL. També cal destacar que després d'enviar els vuit bits de cada fase, el mestre provoca un novè pols anomenat ACK a la línia SCL amb l'objectiu de

separar les fases. Finalment cal dir que, seguint les normes del protocol SCCB, la freqüència típica de treball de la línia SCL és de 100 kHz.

A continuació s'exposen tots els registres modificats a través del bus per tal que la càmera treballi en les condicions desitjades.

***Taula 1.*** *Llistat de registres modificats i la seva descripció.*

| Nom del registre | Adreça | Dades | Descripció |
|---|---|---|---|
| MVFP (Només per fer proves) | 0x1E | 0x01 | Serveix per enviar la imatge capturada amb una simetria vertical (bit 5 a '1') o horitzontal (bit 4 a '1'). |
| COM17 (Només per fer proves) | 0x71 | 0x00 | Si s'activa l'opció de *test pattern* (bit 7 a '1'), la càmera envia barres de diferents colors per comprovar el seu funcionament. |
| COM7 | 0x12 | 0x04 | Posant el bit 2 a '1' i el bit 0 a '0' s'aconsegueix que la càmera treballi en mode RGB. |
| COM15 | 0x40 | 0xD0 | Per treballar en mode RGB444 cal que el bit 4 estigui a '1' |
| REG444 | 0x8C | 0x02 | Per habilitar el mode RGB444 cal posar el bit 1 a '1'. Amb el bit 0 es tria l'ordre com s'envien les dades ('0' per xR\|GB o '1' per RG\|Bx) |

### *2.2.2. El bloc CAM DRIVER*

El bloc encarregat de processar tots els senyals de la càmera explicats prèviament i de la seva programació és el CAM DRIVER.

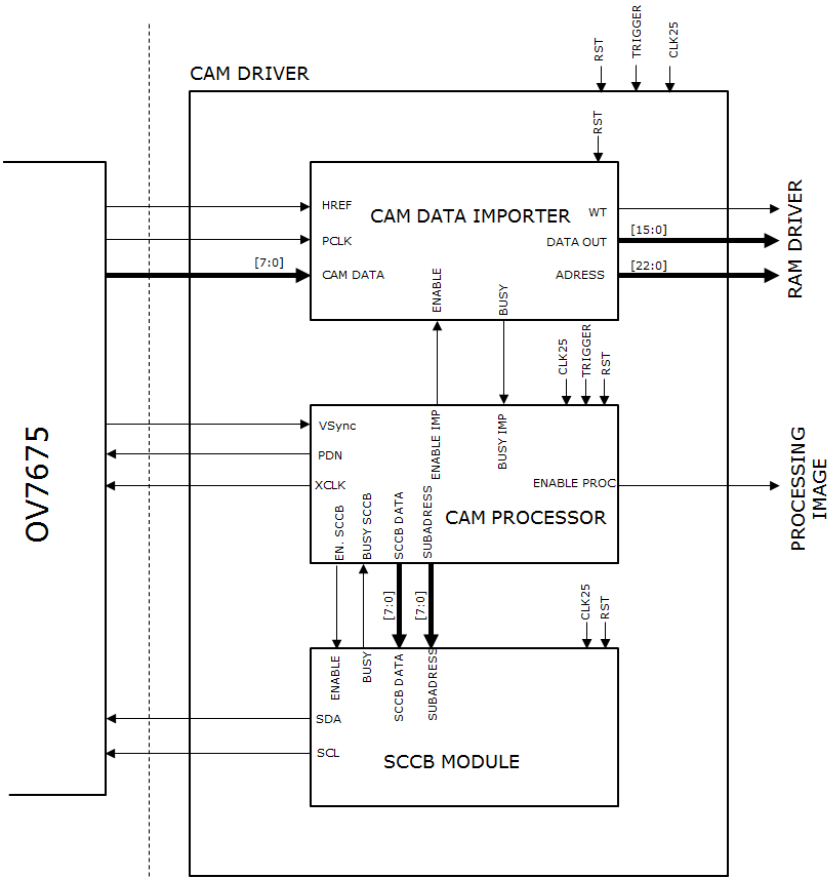


***Figura 13.*** *Diagrama de blocs de CAM DRIVER.*

Com s'ha comentat anteriorment, el llenguatge VHDL permet descriure circuits digitals mitjançant una arquitectura estructural. En aquest bloc s'ha aplicat aquesta arquitectura, ja que està format per tres sub-blocs amb una funció específica una funció específica que s'explicarà en els següents apartats. Es descriu com estan connectats entre ells i com es distribueixen els senyals provinents de la càmera i de la placa *Nexys2,* com són els senyals de RESET, TRIGGER i CLK. El senyal de RESET serveix per reiniciar les màquines d'estats i tots els senyals interns, el senyal de TRIGGER fa la funció de disparador per capturar una imatge i el senyal de CLK és el senyal provinent del divisor de freqüència a 25 MHz.

Els tres sub-blocs que formen CAM DATA estan descrits utilitzant una arquitectura de comportament, és a dir, es descriu el seu funcionament. Tot i que una característica principal del VHDL és la concurrència, utilitzant l'eina *PROCESS()* es pot aconseguir treballar de manera seqüencial. Per indicar al programa quan ha d'avaluar la part seqüencial s'utilitza la llista de sensibilitats, la qual està composta de diferents senyals i quan algun d'aquests canvia d'estat s'avalua el procés. En tots els processos s'utilitza el senyal CLK per la llista de sensibilitats, de tal manera que a cada flanc de pujada o de baixada s'executi el codi seqüencial. Aquest codi seqüencial s'utilitza per generar les màquines d'estat corresponents per a cada sub-bloc, que són:

- **CAM PROCESSOR**: és el "cervell" del bloc que té la funció de controlar els altres dos sub-blocs, indicant-los-hi quan s'han d'activar.

- **CAM DATA IMPORTER:** és l'encarregat d'adquirir les dades que li envia la càmera i enviar-les cap al bloc RAM DRIVER per guardar-les.

- **SCCB MODULE:** manté comunicació amb la càmera a través del bus SCCB per tal de programar-la.

En els següents apartats es detalla el funcionament intern dels tres sub-blocs i es mostren les màquines d'estat dissenyades per a cada un.

### *2.2.3. El sub-bloc CAM PROCESSOR*

El sub-bloc CAM PROCESSOR és l'encarregat de controlar els altres dos sub-blocs donant-los-hi senyals d'*enable* per indica'ls-hi que s'han d'activar. De la mateixa manera, rep un senyal de *busy* de cada un per saber en quin moment han acabat de fer la seva feina. També proporciona el senyal de rellotge XCLK de 25 MHz a la càmera i el senyal PDN (*Power Down*), a més de detectar quan es comença a capturar una imatge comprovant el senyal VSync. Utilitzant un polsador de la placa *Nexys2* se li pot proporcionar el senyal de TRIGGER, indicant-li així que es vol capturar una imatge i que ha d'activar el sub-bloc CAM DATA IMPORTER per rebre les dades. Un cop acabat, dóna un senyal al bloc de processament d'imatge indicant que pot començar a actuar.

A continuació es poden veure les dues màquines d'estats que segueix:

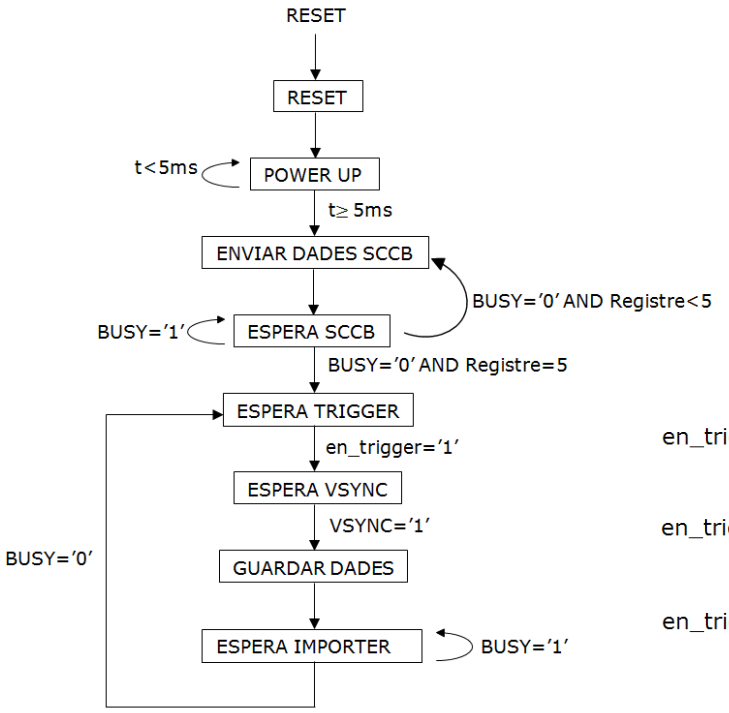


***Figura 14a.*** *Màquina d'estats principal.*

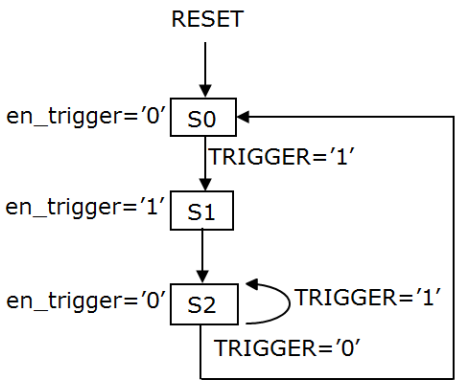


***Figura 14b.*** *Màquines d'estats pel TRIGGER.*

Quan s'activa el botó de RESET s'inicialitzen les màquines d'estats representades en les figures 13a i 13b. La primera és la màquina d'estats principal, la qual serveix per seguir l'ordre lògic per tal de programar i adquirir dades de la càmera. La segona s'utilitza per definir el valor lògic del senyal *en_trigger,* el qual és un senyal intern que s'utilitza com a disparador de la càmera i així avisar que es vol capturar una imatge. Com es pot veure, quan es polsa el botó de TRIGGER, la màquina d'estats passa de l'estat S0 al S1, fent que *en_trigger* passi a valdre '1'. Es mantindrà en aquest estat només un cicle de rellotge (40 ns) ja que en el següent període la màquina passarà a l'estat S2, on *en_trigger* passarà a valdre '0' un altre cop. Es mantindrà en aquest estat mentre el botó de TRIGGER estigui polsat, d'aquesta manera quan s'hagi acabat de capturar la imatge no es tornarà a fer una nova captura ja que el valor de *en_trigger* serà '0'.

En relació a la màquina d'estats principal, a continuació es detalla en pseudocodi l'algorisme que segueix cada estat.

***RESET.*** Es reinicien tots els senyals interns i s'assigna com a pròxim estat POWER UP.

```
en_trig = '0'    -- Enable trigger
Registre = 0     -- nº registre a enviar pel bus SCCB
Preescaler = 0   -- preescalers per fer de comptadors
Preescaler2 = 0
Enable_processing = '0' -- habilitació bloc de processament
Primera_foto = '0' -- No es pot llegir de la RAM fins que
-- no s'hagi capturat la primera imatge
PDN = '1' -- Es posa la càmera en standby
ESTAT = POWER_UP  -- Següent estat
```

***POWER UP.*** Segons les especificacions de la càmera, durant la seqüència de *power up* s'ha de mantenir PDN a '1' durant 5 ms.

```
PDN = '1'
Preescaler = preescaler + 1 -- Cada cicle de rellotge(40ns)
SI preescaler = 125000 ALESHORES -- 125000·40ns = 5ms
      Preescaler = 0
      PDN = '0'
      ESTAT = ENVIAR DADES SCCB -- Següent estat
SINO
      ESTAT = POWER UP
FI SI
```

***ENVIAR DADES SCCB.*** En aquest estat s'activa el bloc SCCB MODULE, el qual se li dóna l'adreça del registre i les dades a enviar.

```
ENABLE_SCCB = '1' -- S'activa el bloc SCCB MODULE
ESTAT = ESPERA_SCCB -- Següent estat
CAS REGISTRE
      QUAN 1 -- MIRROR AND FLIP (per fer proves)
```

```
                SUBADRESS = 0x1E
                SCCB_DATA = 0x01 -- flip 0x10 // mirror 0x20
          QUAN 2 -- TEST PATTERN (per fer proves)
                SUBADRESS = 0x71
                SCCB_DATA = 0x00 -- Activar 0x80
          QUAN 3 -- OUTPUT FORMAT RGB
                SUBADRESS = 0x12
                SCCB_DATA = 0x04
          QUAN 4 -- RGB565
                SUBADRESS = 0x40
                SCCB_DATA = 0xD0
          QUAN 5 -- RGB444 enable
                SUBADRESS = 0x8C
                SCCB_DATA = 0x02
    FI CAS
```

***ESPERA SCCB.*** S'espera que el bloc SCCB MODULE acabi la seva feina enviant les dades. Un cop ha acabat i depenent del nº del registre, torna a l'estat d'enviar dades o passa al següent estat.

```
    ENABLE_SCCB = '0'
    preescaler = preescaler + 1
    SI preescaler => 2 ALESHORES -- Espera de 120ns abans de
-- llegir BUSY per evitar llegir un '0' abans d'hora
          SI BUSY_SCCB = '1' ALESHORES

                ESTAT = ESPERA_SCCB
          SINO
                Preescaler2 = preescaler2 + 1
                SI preescaler2 = 125 ALESHORES -- Espera 5us
    --abans de tornar a enviar dades
                      Preescaler2 = 0
                      SI REGISTRE < 7 ALESHORES
                            REGISTRE = REGISTRE + 1
                            ESTAT = ENVIAR_DADES_SCCB
                      SINO
                            ESTAT = ESPERA_TRIGGER
                            preescaler = 0
                      FI SI
                FI SI
          FI SI
    FI SI
```

***ESPERA TRIGGER.*** Estat d'espera el senyal de *trigger* per començar a capturar una imatge. També es controla el senyal d'*enable* pel bloc de processament, el qual només pot actuar quan ha acabat d'actuar CAM DRIVER i després d'haver capturat una primera imatge.

```
    SI en_trig = '1' ALESHORES
          ESTAT = ESPERA_VSYNC
```

```
            ENABLE_PROCESSING  =  '0'  --Mentre  es  captura  una
--imatge, el bloc de processament no pot actuar
      SINO
            SI primera_foto = '1' ALESHORES -- No és la primera
--imatge capturada
                  ENABLE_PROCESSING = '1'
            SINO -- És la primera imatge capturada
                  ENABLE_PROCESSING = '0'
            FI SI
            ESTAT = ESPERA_TRIGGER --Següent estat
      FI SI
```

***ESPERA VSYNC.*** Un cop s'ha rebut el senyal de *trigger,* s'ha d'esperar que la càmera enviï el senyal de sincronisme vertical que indica que es comença a capturar una imatge.

```
      SI VSYNC = '1' ALESHORES
            ESTAT = GUARDAR_DADES
      SINO
            ESTAT = ESPERA_VSYNC
      FI SI
```

***GUARDAR DADES.*** S'activa el bloc de CAM DATA IMPORTER per rebre la informació que envia la càmera.

```
      ENABLE_DATA_IMPORTER = '1'
      Preescaler = preescaler + 1
      SI preescaler = 2 ALESHORES
            Preescaler = 0
            ESTAT = ESPERA_IMPORTER
      FI SI
```

***ESPERA IMPORTER.*** S'espera que el bloc de CAM DATA IMPORTER acabi la seva feina comprovant el senyal de *busy* que envia. Un cop s'ha acabat de capturar la imatge, es retorna a l'espera del senyal de *trigger.*

```
      ENABLE_DATA_IMPORTER = '0'
      SI BUSY_IMPORTER = '0' ALESHORES
            ESTAT = ESPERA_TRIGGER
            Primera_foto = '1'
      SINO
            ESTAT = ESPERA_IMPORTER
      FI SI
```

### *2.2.4. El sub-bloc CAM DATA IMPORTER*

El sub-bloc CAM DATA IMPORTER és l'encarregat de rebre les dades enviades des de la càmera, transformar-les en el format adequat i enviar-les cap al bloc de RAM DRIVER per tal de que puguin ser desades a la memòria RAM externa.

A continuació es pot veure la màquina d'estats que segueix:

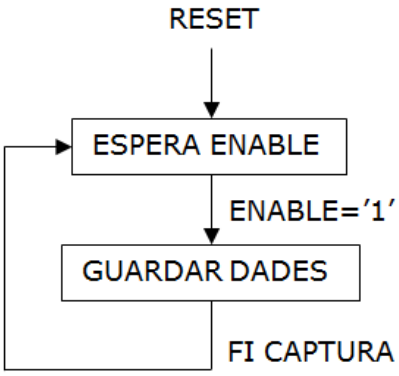


***Figura 15.*** *Màquina d'estats principal.*

***ESPERA ENABLE.*** Es reinicien tots els senyals interns i s'avalua el senyal d'*enable* per començar a capturar dades. Es pot observar en el pseudocodi que el senyal *adress_out_ram,* el qual indica a quina adreça de la memòria RAM es vol guardar una dada, comença amb el valor zero. Això significa que la informació de la imatge capturada comença a partir d'aquesta adreça i el seu motiu s'explicarà més endavant.

```
Adress_out_ram = 0
Count_row = 0 -- comptador de files
Count_pixel = 0 -- comptador de columna
WT = '0' -- Senyal per indicar que es vol guardar una dada
Data1 = 0x00 -- Byte1 d'informació d'un píxel
Data2 = 0x00 -- Byte2 d'informació d'un píxel
Count_data = 1 -- Indica quin byte s'ha de guardar (1 o 2)
SI ENABLE_DATA_IMPORTER = '1' ALESHORES
      ESTAT = GUARDAR_DADES --Següent estat
      BUSY = '1'
SINO
      ESTAT = ESPERA_ENABLE
FI SI
```

***GUARDAR DADES.*** En aquest estat es procedeix a guardar els bytes d'informació de cada píxel que envia la càmera. Com que les dades s'envien en format RGB444 que ocupen dos bytes d'informació per a cada píxel i la interfície de sortida només té 8 bits, s'han de guardar les dades en dos passos. Es fa mitjançant els senyals de Data1 i Data2 amb l'ajuda de Count_data, com es pot veure a continuació. La sincronització es fa utilitzant el senyal HREF que envia la càmera i indica quan hi ha dades vàlides en la interfície de sortida. Cal comentar que en la llista de sensibilitats d'aquesta màquina d'estats hi ha el senyal PCLK en lloc de CLK de 25 MHz, així doncs només s'avaluarà el procés cada vegada que hi hagi un flanc de pujada. Finalment, per saber en quin moment s'ha acabat de capturar una imatge s'utilitzen dos comptadors, un per comptar les files i

l'altre per comptar les columnes. Quan s'arriba al píxel 639 de la fila 479, significa que ja s'han capturat tots els píxels.

```
SI HREF = '1' ALESHORES
      CAS count_data
            QUAN 1
                  Data1 = CAM_DATA
                  Count_data = 2
                  WT = '0'
                  SI no(count_row=0 I count_pixel=0) ALESHORES
                        Adress_out_ram = adress_out_ram + 1
                  FI SI
            QUAN 2
                  WT = '1'
                  Data2 = CAM_DATA
                  Count_data = 1
                  SI count_pixel=639 ALESHORES --Últim píxel
-- de la línia
                  Count_pixel = 0
                        SI count_row = 479 -- Última línia
                              BUSY = '0'
                              ESTAT = ESPERA_ENABLE
                        SINO
                              Count_row = conunt_row + 1
                        FI SI
                  SINO
                        Count_pixel = count_pixel + 1
                  FI SI
      FI CAS
      SINO
            WT = '0'
FI SI
```

Cal comentar dos aspectes importants: el primer és que cada cop que es captura el primer byte d'informació s'augmenta l'adreça de la memòria RAM excepte en el primer píxel (0,0), ja que l'adreça ja és la correcta (0). El segon aspecte es que de manera concurrent i fora del procés seqüencial hi ha la següent sentencia:

```
DATA_OUT = data1 & data2
```

Això significa que en tot moment el senyal de sortida DATA_OUT (de 16 bits) serà en tot moment la concatenació dels senyals data1 i data2.

### *2.2.5. EL sub-bloc SCCB MODULE*

El sub-bloc SCCB MODULE és l'encarregat de transmetre les dades pel bus SCCB per tal de poder programar correctament la càmera. Quan rep el

senyal d'*enable*, comença la seqüència de comunicació seguint els protocols explicats anteriorment.

A continuació es pot veure la màquina d'estats que segueix:

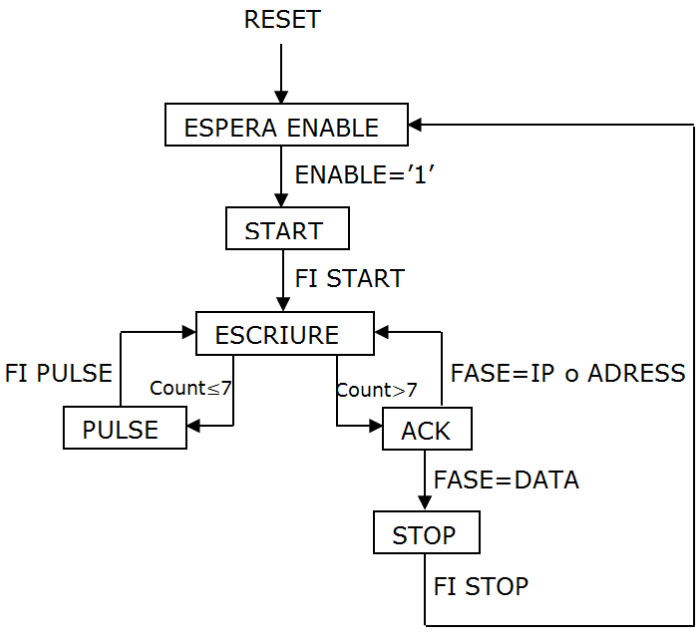


***Figura 16.*** *Màquina d'estats principal.*

***ESPERA ENABLE.*** Es reinicien tots els senyals interns i s'avalua el senyal d'*enable* per començar el protocol de comunicació.

```
SDA = '1'
SCL = '1'
Preescaler = 0
Count = 0 -- Comptador dels bits enviats
BUSY = '0'
FASE = IP -- Fase d'escriptura (IP, ADRESS i DATA)
PULSE_STATE = 0 --Control del pulse
SI ENABLE_SCCB = 1 ALESHORES
      BUSY = '1'
      ESTAT = START
SINO
      ESTAT = ESPERA_ENABLE
FI SI
```

***START.*** En aquest estat es genera la seqüència *Start* per iniciar comunicació amb la càmera. També es prepara el primer byte a enviar, és a dir, l'adreça de l'esclau que en aquest cas és 0x42.

```
Data_to_write = IPadress -- 0x42
SDA = '0'
Preescaler = preescaler + 1
SI preescaler = 125 ALESHORES -- 5us
      Preescaler = 0
```

```
        SCL = '0'
        ESTAT = ESCRIURE
SINO
        ESTAT = START
FI SI
```

***ESCRIURE.*** En aquest estat es prepara la línia SDA, assignant-la al valor del bit 7 del vector *data_to_count,* abans de generar el pols per la línia SCL. En el cas que ja s'hagi enviat tot el byte, es procedeix a generar el pols ACK.

```
SI count <= 7 ALESHORES
        SDA = data_to_write[7]
        ESTAT = PULSE
SINO
        COUNT = 0
        ESTAT = ACK
FI SI
```

***PULSE.*** Es genera un pols a la línia SCL de tal manera que la freqüència sigui de 100 kHz, és a dir, amb un període de 10 µs. Es pot veure que cada pols està compost de tres fases: en la primera es manté SCL a '0' durant 2,5 µs seguit de la segona que posa SCL a '1' durant 5 µs. En la tercera es posa a '0' durant 2,5 µs més i es modifica el vector *data_to_write* de tal manera que els bits corrin una posició a l'esquerra per poder actualitzar la línia SDA.

```
CAS PULSE_STATE
        QUAN 0
                SCL = '0'
                Preescaler = preescaler + 1
                SI preescaler = 62 ALESHORES --2,5us
                        Preescaler = 0
                        PULSE_STATE = 1
                        ESTAT = PULSE
                FI SI
        QUAN 1
                SCL = '1'
                Preescaler = preescaler + 1
                SI preescaler = 125 ALESHORES --5us
                        Preescaler = 0
                        PULSE_STATE = 2
                        ESTAT = PULSE
                FI SI
        QUAN 3
                SCL = '0'
                Preescaler = preescaler + 1
                SI preescaler = 62 ALESHORES --2,5us
                        Preescaler = 0
                        Data_to_write[7:1] = data_to_write[6:0]
```

```
                PULSE_STATE = 0
                ESTAT = ESCRIURE
                Count = count + 1
            FI SI
        FI CASE
```

***ACK.*** És un estat molt semblant a l'anterior ja que la única diferència es troba en l'estat tres del pols. En lloc de fer córrer els bits de *data_to_write* cap a l'esquerra, l'actualitza amb el nou byte que es vol enviar depenent de quina fase d'escriptura es trobi el procés.

```
SDA = '0'
CAS PULSE_STATE
    QUAN 0
        SCL = '0'
        Preescaler = preescaler + 1
        SI preescaler = 62 ALESHORES --2,5us
            Preescaler = 0
            PULSE_STATE = 1
            ESTAT = ACK
        FI SI
    QUAN 1
        SCL = '1'
        Preescaler = preescaler + 1
        SI preescaler = 125 ALESHORES --5us
            Preescaler = 0
            PULSE_STATE = 2
            ESTAT = ACK
        FI SI
    QUAN 3
        SCL = '0'
        Preescaler = preescaler + 1
        SI preescaler = 62 ALESHORES --2,5us
            Preescaler = 0
            PULSE_STATE = 0
            CAS FASE
                QUAN IP
                    FASE = ADRESS
                    Data_to_write = SUBADRESS
                    ESTAT = ESCRIURE
                QUAN ADRESS
                    FASE = DATA
                    Data_to_write = SCCB_DATA
                    ESTAT = ESCRIURE
                QUAN IP
                    FASE = IP
                    ESTAT = ESCRIURE
                    ESTAT = STOP
            FI CAS
        FI SI
```

```
        FI CAS
```

***STOP.*** Finalment, quan s'han enviat tota la informació, es procedeix a generar la seqüència stop per indicar que ja s'ha acabat la comunicació. Es torna a l'estat inicial a l'espera de l'*enable.*

```
SCL = '1'
Preescaler = preescaler + 1
SI preescaler = 125 ALESHORES -- 5us
      Preescaler = 0
      SDA = '1'
      ESTAT = ESPERA_ENABLE
      BUSY = 0
FI SI
```

## 2.3. Bloc controlador de la memòria RAM

En aquest apartat s'explica el funcionament del bloc controlador de la memòria RAM externa que incorpora la placa *Nexys2.* De la mateixa manera, a continuació es detallen les característiques i el funcionament de la pròpia memòria RAM.

### *2.3.1.* La memòria RAM externa

La memòria RAM que incorpora la placa *Nexys2* és la MT45W8MW16 del fabricant Micron. És una memòria del tipus PSDRAM (*Pseudo-Static Dynamic Random Access Memory)* amb tecnologia CMOS organitzada com 8Meg x 16 bits, és a dir, té una capacitat de 128Mb. Aquest tipus de memòria està basada en una memòria DRAM (*Dynamic RAM*), la qual té les cel·les formades per un transistor i un condensador i tenen la característica que han de ser refrescades cada cert període de temps per tal de no perdre la informació. Tot i així, les PSDRAM porten integrat un sistema que automàticament refresca les cel·les DRAM i d'aquesta manera es pot treballar com si fossin SRAM (*Static RAM)*, les quals no necessiten ser refrescades.

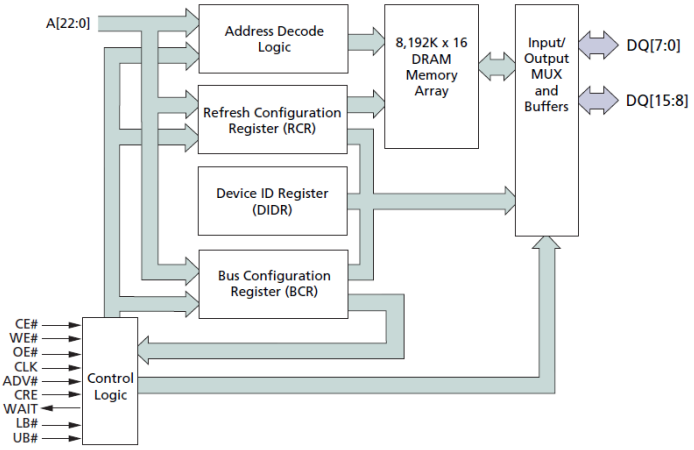


***Figura 17.*** *Diagrama de blocs de la memòria RAM utilitzada.*

Com es pot veure en la figura superior, la memòria RAM es pot comunicar amb l'exterior mitjançant onze senyals que es descriuen a continuació:

- **Adreça [22:0]:** s'utilitza per indicar en quina adreça es vol escriure o llegir informació. També serveix per carregar informació en els registres de configuració BCR i RCR, els quals es comentaran més endavant.

- **Dades [15:0]:** És un senyal d'entrada i de sortida ja que és per on entra i surt la informació de l'adreça requerida.

- **CE:** Chip enable. Quan es posa en estat baix s'activa el dispositiu i quan es posa en estat alt s'inhabilita i es posa en *standby.*

- **WE:** Write enable. Determina si es vol llegir o escriure a la memòria. Quan es posa en estat alt s'hi escriu i quan es posa en estat baix s'hi llegeix.

- **OE:** Output enable. S'activen els *buffers* de la sortida quan es posa en estat baix.

- **CLK:** Clock. S'utilitza quan es treballa en mode *burst* (síncron). Quan es treballa en mode asíncron cal deixar-lo en estat baix.

- **ADV:** Address valid: indica que l'adreça present a l'entrada és vàlida quan es posa en estat baix.

- **CRE:** Control register enable. Quan es posa en estat alt significa que es vol carregar dades en els registres de configuració RCR o BCR.

- **LB, UB:** Lower/Upper byte enable. Quan estan en estat baix, s'habiliten els dos bytes de lectura de l'adreça.

- **WAIT:** És un senyal de sortida que indica quan es pot llegir o escriure degut a les operacions internes de refresc de dades i d'aquesta manera s'eviten col·lisions. Tot i així, en aquesta aplicació no l'utilitzarem per uns motius que s'expliquen més endavant.

Utilitzant els registres BCR i RCR, la memòria RAM pot ser programada per treballar en tres modes de funcionament diferents: asíncron, *page* i *burst* (síncron).

***MODE ASÍNCRON.*** En aquest mode s'utilitza la memòria com si fos una estàndard SRAM, controlada pels senyals CE, OE, WE i LB/UB, mentre que els senyals ADV i CLK han d'estar en estat baix durant l'operació. Quan es treballa en aquest mode, es pot accedir a qualsevol adreça de memòria però només es podrà realitzar una operació de lectura o escriptura. En el

cas que es vulgui accedir a la informació de més d'una adreça, s'han de repetir les operacions que venen a continuació de manera repetitiva, canviant el valor de l'adreça.

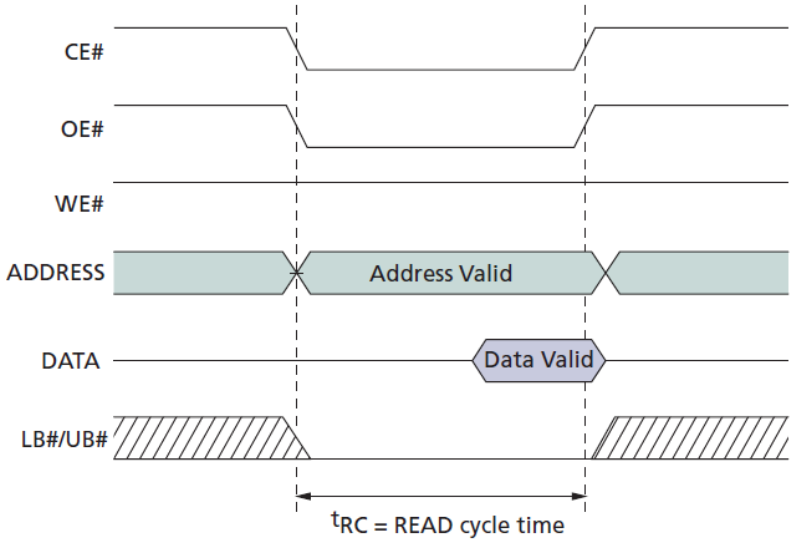


***Figura 18.*** *Diagrama per l'operació de lectura.*

En la figura 18 hi apareix representat el diagrama per l'operació de lectura. Es tracta de posar en estat baix els senyals CE, OE i LB/UB i en estat alt WE, indicant que es vol llegir la informació. Es necessita que el temps $t_{RC}$ sigui superior o igual a 70 ns degut al temps que necessita el dispositiu per processar la informació i entregar les dades. Tot i així, el temps d'accés a les dades pot ser que sigui inferior, ja que com a màxim $t_{AA}$ (*Adress Access time*) serà de 70 ns.

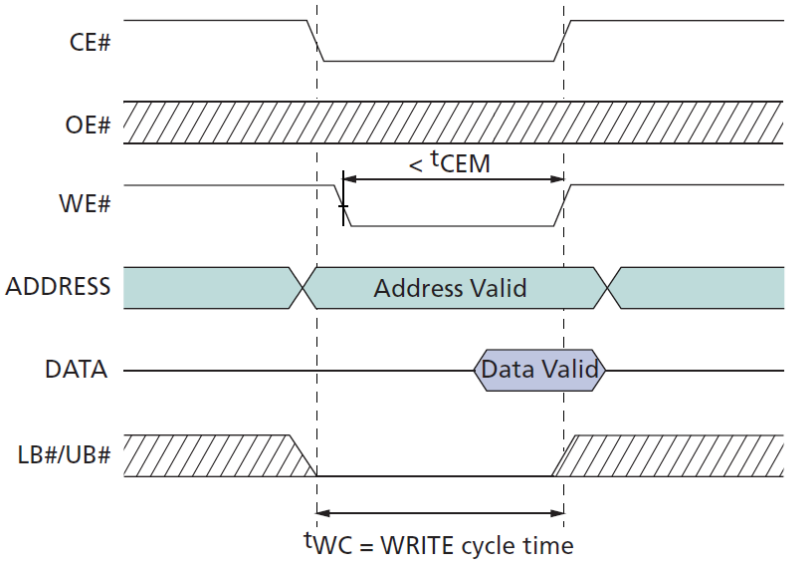


***Figura 19.*** *Diagrama per l'operació d'escriptura.*

L'operació d'escriptura és molt semblant a l'operació de lectura. L'única diferència es troba en el senyal WE, que en aquest cas ha de posar-se en estat baix. De la mateixa manera que en la lectura de dades, es necessita

que el temps $t_{WC}$ sigui superior o igual a 70 ns. També cal vigilar que el temps que WE està en estat baix ($t_{CEM}$) sigui inferior a 4 µs, degut a que el dispositiu necessita accedir a les dades per refrescar-les.

***MODE PAGE.*** És semblant al mode asíncron amb la diferència que es pot llegir la informació d'adreces contigües ràpidament canviant els quatre últims bits del senyal d'adreça, és a dir, es poden llegir fins a setze dades seguides. En aquest mode només no està permès escriure dades, sinó només llegir-ne, seguint el diagrama de la figura 20 que es pot veure a continuació. Cal comentar que per a aquesta aplicació aquest mode no s'utilitzarà degut a que no és el més òptim i indicat per les característiques que presenta el problema.

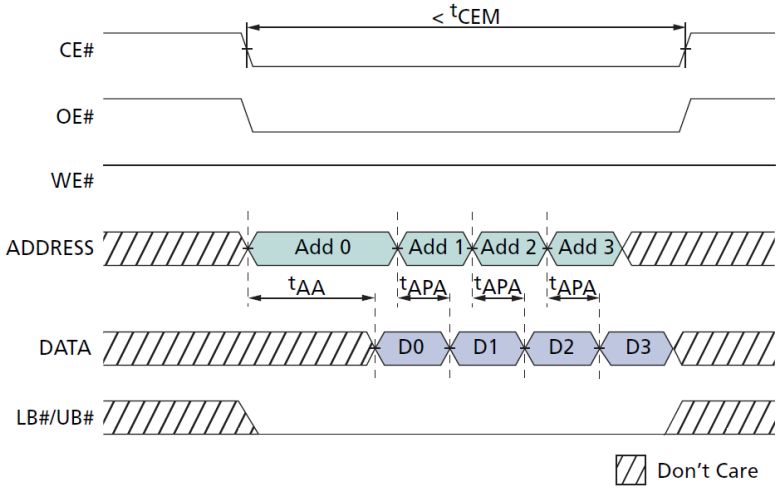


***Figura 20.*** *Diagrama de lectura del mode Page.*

***MODE BURST.*** Aquest mode permet la transferència de dades a una alta velocitat i de manera síncrona utilitzant el senyal CLK. Se li proporciona una adreça inicial a partir de la qual es començarà a llegir o escriure informació i, seguint els diagrames que venen a continuació, es possible transmetre 4, 8, 16 o 32 *words*[4] de manera consecutiva. També hi ha l'opció de fer un *continuous burst*, és a dir, es transmeten les dades de manera seqüencial fins al final de fila, equivalent a 128 *words*.

[4] Una paraula (*word*) en informàtica és un conjunt finit de bits definit prèviament. En aquest cas, la memòria RAM treballa amb *words* de 16 bits.

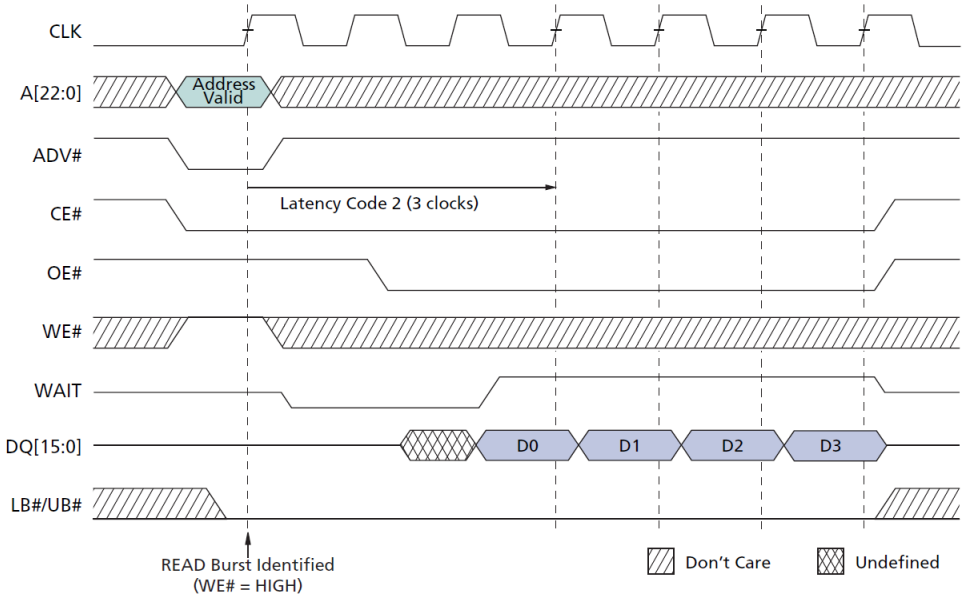


***Figura 21.*** *Diagrama per l'operació de lectura.*

En la figura superior es pot observar una operació de lectura de 4 *words* en mode *burst.* En el primer flanc de pujada del senyal CLK es guarda l'adreça inicial a partir de la qual es començaran a llegir les dades de manera consecutiva. Prèviament però, s'han hagut de posar els senyals ADV i CE en estat baix i WE en estat alt. Després de tres cicles de rellotge i a cada flanc de pujada, estaran disponibles a la sortida les diferents dades per ser llegides. Els cicles de rellotge que cal esperar (*latency*) és configurable a través del registre BCR, que s'explicarà a continuació.

De la mateixa manera que es llegeixen dades, també es poden escriure utilitzant aquest mode, com es pot veure en la figura inferior. L'única diferència destacable és l'estat del senyal WE, que ha d'estar a nivell baix en el primer flanc de pujada del senyal CLK.

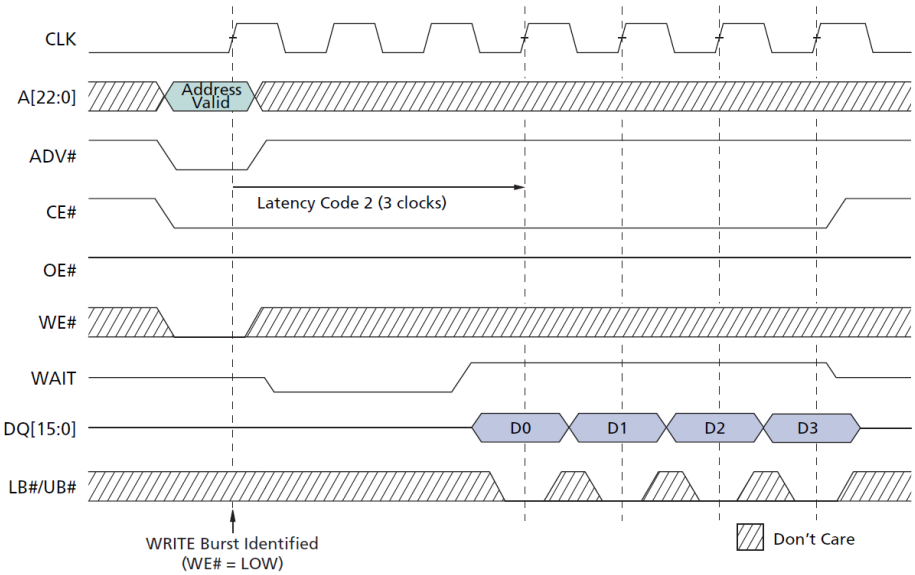


***Figura 22.*** *Diagrama per l'operació d'escriptura.*

Com s'ha comentat anteriorment, el mode de funcionament i altres paràmetres estan definits en els registres de configuració, que son el BCR i el RCR. Amb el registre BCR (*Bus Configuration Register*) es defineixen les característiques principals del funcionament, com pot ser el mode, la llargada del *burst* o altres aspectes que es s'explicaran a continuació. El registre RCR (*Refresh Configuration Register*) s'utilitza per controlar com el dispositiu refresca les cel·les de DRAM internes. Aquests dos registres es carreguen automàticament amb un valor per defecte quan es connecta el dispositiu i poden ser configurats en qualsevol moment.

Per dur a terme la configuració es segueix el diagrama següent:

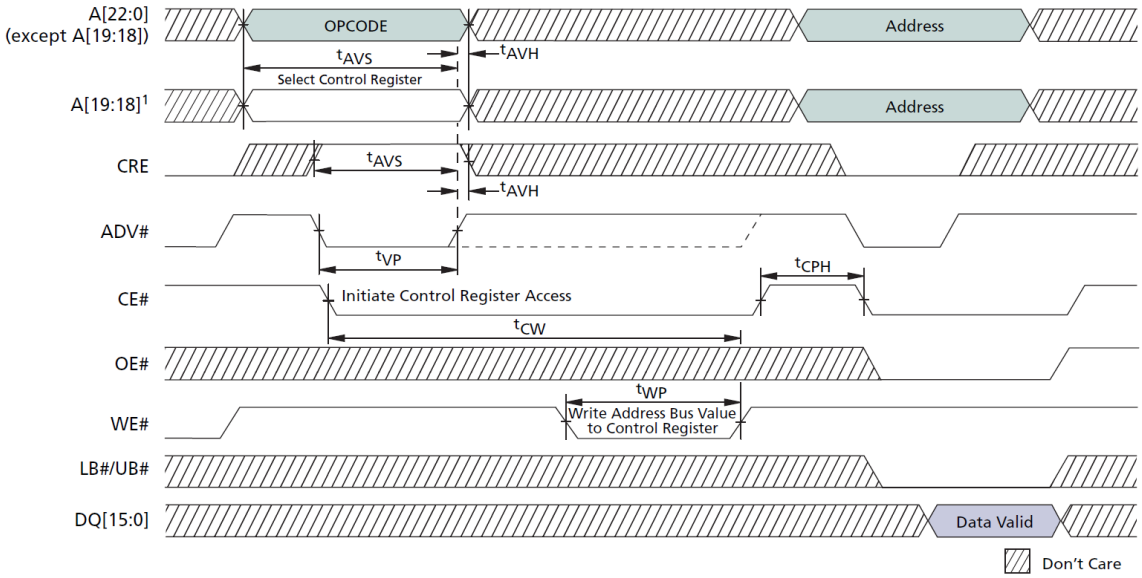


***Figura 23.*** *Diagrama per l'operació de configuració dels registres, seguit d'una operació de lectura.*

Com es pot observar, es tracta de realitzar una operació d'escriptura de manera asíncrona amb la diferència que el senyal CRE ha d'estar a nivell alt, indicant que es vol accedir als registres de configuració. Per indicar quin registre es vol modificar s'utilitzen els bits 19 i 18 de l'adreça: 00b per modificar RCR i 10b per modificar BCR. La resta de bits s'utilitzen per definir el nou valor del registre a modificar. Cal dir que el fabricant recomana que immediatament després de realitzar una operació de configuració, es procedeixi a realitzar una operació de lectura, com es pot observar en la figura 23.

En aquesta aplicació només es modifica el valor del registre BCR que, com s'ha comentat, es fa través del valor de l'adreça. A continuació es descriuen tots els bits de configuració:

- **Bits 22-20, 17-16, 9 i 7-6:** Reservats. Han d'estar a '0'.
- **Bits 19-18:** Selector del registre a modificar. 00b per RCR i 10b per BCR.

- **Bit 15:** Selector del mode. '0' per mode *burst* i '1' per mode asíncron. Si se selecciona el mode *burst* també es poden escriure dades en mode asíncron.

- **Bits 14-11:** quan es treballa en mode *burst* cal esperar uns cicles de rellotge abans d'accedir a les dades i mitjançant els bits 13-11 es pot decidir quants cicles es vol. Amb el bit 14 es pot triar si el temps d'espera es vol fix ('1') o variable ('0'). Si es tria variable, les dades estaran disponibles després del nombre de cicles triat però s'haurà de controlar el senyal WAIT, ja que aquest indica quan és possible accedir a les dades per motius de refresc. En el cas que indiqui que les dades encara no estan disponibles, s'haurà d'esperar tants cicles com calguin. En canvi, si es tria un temps fix, el fabricant t'assegura que les dades estaran disponibles després del nombre de cicles escollit i no caldrà controlar el senyal WAIT.

- **Bit 10:** indica la polaritat del senyal WAIT. '0' per actiu a nivell baix i '1' per actiu a nivell alt.

- **Bit 8 :** indica quan les dades estan disponibles. Amb un '0' les dades estaran disponibles en el flanc de pujada del rellotge immediatament després que el senyal WAIT s'hagi posat en estat alt. Amb un '1', les dades estaran disponibles un cicle més tard després que el senyal WAIT s'hagi posat en estat alt.

- **Bit 5-4:** Configuren el bus de dades per a diferents entorns per tal de minimitzar el soroll generat. En aquesta aplicació es deixarà l'opció per defecte.

- **Bits 3-0:** mitjançant els bits 2-0 es tria la llargada del *burst*: 4, 8, 16 o 32 *words* o un *burst* continu. Amb el bit 3 s'escull el *wrap,* que indica com és la seqüència d'adreces del *burst.* Amb un '1' segueix una seqüència ordenada ascendent mentre que amb un '0' només recorre un nombre d'adreces finit indicat per la llargada del *burst.* Per exemple en un *burst* de llargada 8 començant per l'adreça 5 seguirà el següent ordre: 5-6-7-0-1-2-3-4.

El valor del registre BCR per defecte és 0x9D1F, mentre que se li carrega el 0x5C1A. Aquest nou valor ve definit per les necessitats que requereix el dispositiu dissenyat. S'ha escollit guardar les dades de manera asíncrona, ja que la freqüència a la que el bloc CAM DRIVER envia les dades és aproximadament de 12 MHz i hi ha temps suficient per fer-ho d'aquesta manera. En canvi, la lectura de dades pel bloc VGA DRIVER és en mode *burst,* ja que la freqüència a la que el port ha d'enviar les dades és de 25 MHz (T=40ns) i no hi ha prou temps per llegir-les totes (recordar que el temps mínim per operacions asíncrones és de 70 ns). S'ha escollit treballar amb un *burst* de llargada 8 *words* sense *wrap* amb una espera de 4 cicles de rellotge amb temps fix, és a dir, sense utilitzar el senyal WAIT.

La justificació d'aquesta decisió es detalla en els pròxims apartats, on s'explica el bloc controlador de la memòria RAM i el bloc controlador del port VGA.

### 2.3.2. El bloc RAM DRIVER

El bloc RAM DRIVER té la funció de comunicar-se amb la memòria RAM externa per tal de guardar-hi la imatge capturada i de proporcionar la informació als blocs de processament d'imatge i del port VGA.

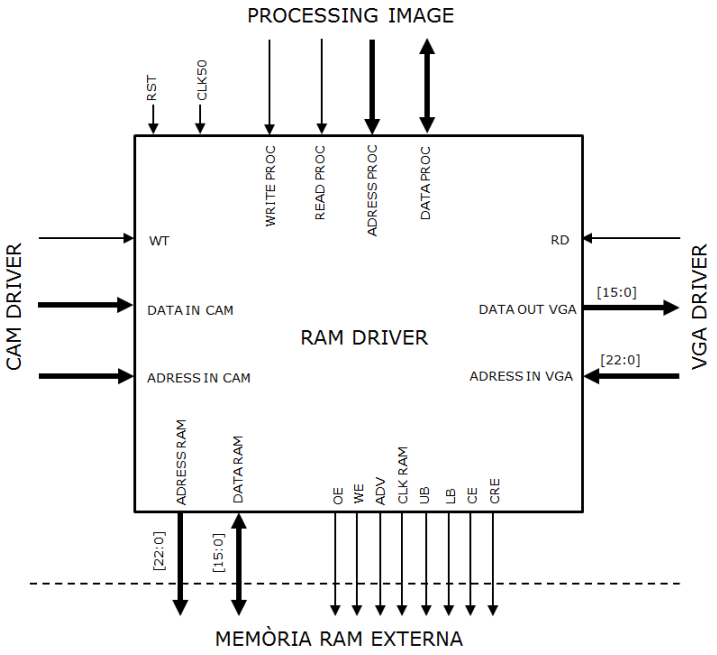


***Figura 24.*** *Bloc RAM DRIVER.*

Com es pot veure en la figura 24, el bloc RAM DRIVER no està compost de sub-blocs, sinó que és un bloc descrit amb una arquitectura de comportament. Està connectat amb els altres tres blocs principals i amb cada un s'hi comunica seguint un protocol diferent. Cal dir que en tot moment només es pot mantenir comunicació amb un sol bloc, ja que la memòria RAM externa només permet que en una operació s'hi llegeixin o escriguin dades.

La comunicació amb el bloc de CAM DRIVER comença quan es llegeix un '1' en el senyal d'entrada WT. Tot seguit envia les dades del senyal DATA IN CAM a l'adreça que li indica el senyal ADRESS IN CAM de manera asíncrona. S'ha decidit fer-ho en aquest mode degut a que el temps necessari per guardar una dada de manera asíncrona és de 70 ns i hi ha temps suficient per realitzar la operació, ja que la freqüència a la que s'han de guardar les dades és aproximadament de 12 MHz, amb període de 83,33 ns.

La comunicació amb el bloc VGA DRIVER comença igual que en el cas anterior, quan es llegeix un '1' en el senyal d'entrada RD. En aquest cas es treballa en mode *burst* a una freqüència de 50 MHz degut a que el port VGA necessita dades a una freqüència de 25 MHz i no hi ha prou temps de llegir

totes les dades. Quan s'ha iniciat la comunicació amb el bloc VGA DRIVER, es comença la seqüència *burst* de llargada 8 *words* a l'adreça que se li indica i s'envien les 8 dades a mesura que es llegeixen.

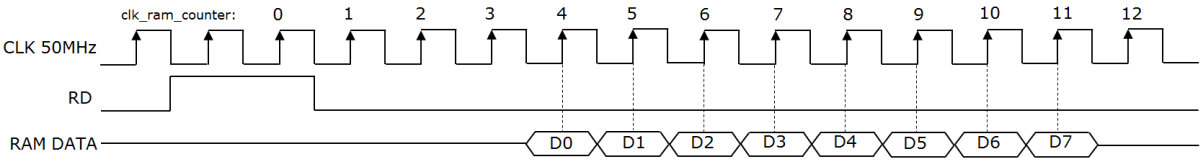


***Figura 25.*** *Diagrama de lectura en mode burst.*

El *burst* és sense *wrap*, és a dir, segueix un ordre ordenat i ascendent d'adreces a partir de l'adreça inicial. També cal dir que s'ha configurat la memòria RAM perquè treballi amb un temps d'espera fix (*fixed latency).* Com es pot veure en la figura 25, des de que es llegeix un '1' a RD fins que es comencen a llegir les dades hi ha un cert nombre de cicles de rellotge que es manté fix per a cada operació de lectura perquè no hi hagi problemes amb el refresc de dades. Amb l'ajuda del senyal intern *clk_ram_counter,* es pot observar que hi ha 5 cicles d'espera, tot i que està configurat per a que siguin 4. Cal dir que el primer cicle serveix d'espera per a què hi hagi una bona sincronització amb el bloc VGA DRIVER. A partir del 5è doncs, es llegeixen les dades a cada flanc de pujada fins que s'hagin llegit els 8 *words.* La manera com el bloc VGA DRIVER tracta les dades rebudes i la seva justificació es detalla en el següent apartat.

La comunicació amb el bloc PROCESSING IMAGE és semblant al protocol que segueix amb el bloc CAM DRIVER. Quan el senyal WRITE_PROC o READ_PROC es posen a '1', es procedeix a llegir o escriure la dada a l'adreça indicada de manera asíncrona.

A continuació es pot veure la màquina d'estats que segueix el bloc RAM DRIVER:

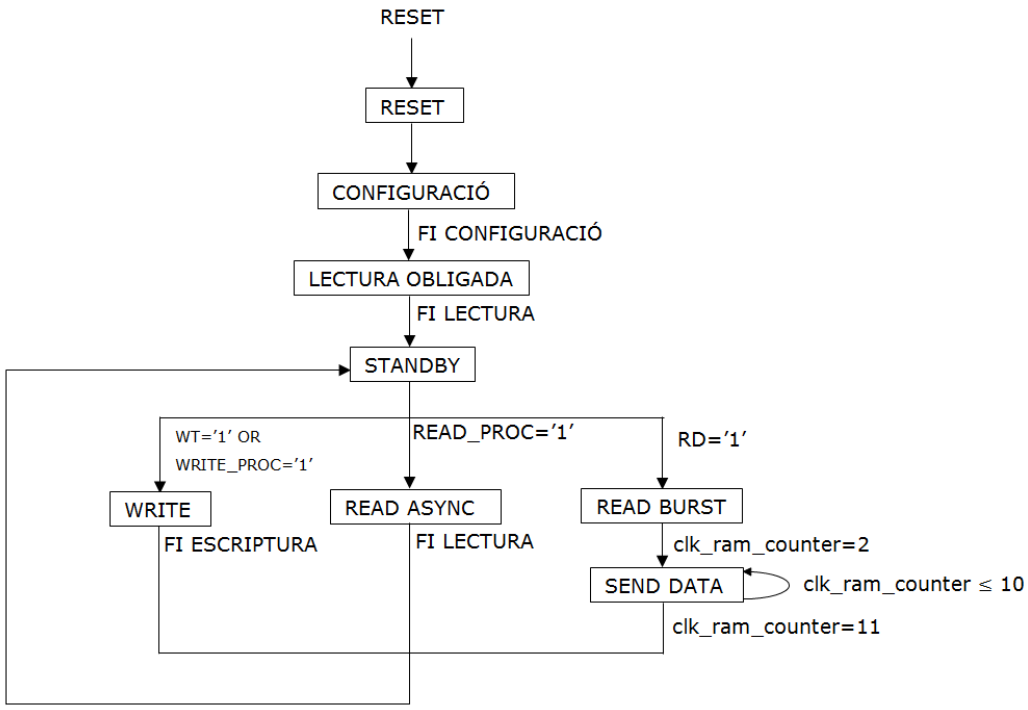


***Figura 26.*** *Màquina d'estats de RAM DRIVER.*

Com es pot observar, després de prémer el botó de reset, es procedeix a configurar la memòria RAM. Tot seguit es fa l'operació de lectura recomanada pel fabricant i es posa en *standby,* a l'espera de rebre el senyal d'algun dels blocs amb els que està connectat. Depenent del que rebi, genera els senyals necessaris per dur a terme les operacions d'escriptura o lectura en mode asíncron o *burst.*

Abans de detallar en pseudocodi cada estat, cal comentar com s'escriu i es llegeix a través dels 15 bits connectats a la memòria externa. De manera concurrent al procés principal, s'ha descrit un circuit digital que permet controlar l'estat lògic del pin de sortida. Concretament, és un *buffer* no inversor tri-estat, com es mostra en la figura 27.

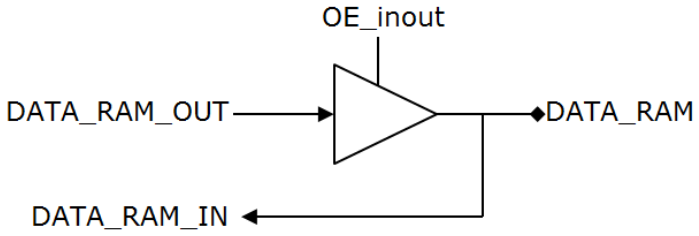


***Figura 27.*** *Buffer no inversor tri-estat.*

El senyal OE_inout (*Output Enable-Inout)* permet posar la sortida del *buffer* en estat d'alta impedància ('Z') de manera que si es vol escriure s'ha de posar en estat alt i si es vol llegir s'ha de posar en estat baix, seguint el següent codi:

```
DATA_RAM_IN = DATA_RAM --Sempre es llegeix DATA_RAM
```

```
PROCÉS(OE_inout, DATA_RAM_OUT)
      SI OE_inout = '1' ALESHORES
            DATA_RAM = DATA_RAM_OUT
      SINO
            DATA_RAM = Z --Alta impedància
      FI SI
FI PROCÉS
```

Com es pot observar, el senyal DATA_RAM_IN sempre és igual que la sortida, és a dir, el senyal DATA_RAM. En canvi, la sortida només serà igual al senyal DATA_RAM_OUT quan OE_inout sigui '1'.

De la mateixa manera, s'ha descrit un altre *buffer* no inversor tri-estat canviant el nom del senyals, per tal de controlar el bus de dades que va des del bloc RAM DRIVER fins al bloc PROCESSING IMAGE, ja que aquest també necessita llegir i escriure dades.

Finalment, hi ha un últim circuit descrit que fa la funció de multiplexor, controlant el senyal de rellotge que va a parar a la memòria RAM externa.

```
AMB clk_ram_enable SELECCIONAR
      CLK_RAM = CLK50 QUAN SIGUI '1'
      CLK_RAM = '0' QUAN SIGUI '0'
```

Com es pot veure, quan el valor del senyal intern clk_ram_enable estigui en estat alt, el valor del senyal CLK_RAM estarà connectat directament al senyal de rellotge de 50MHz. En el cas contrari, es trobarà en estat baix.

En relació a la màquina d'estats principal, a continuació es detalla en pseudocodi l'algorisme que segueix cada estat.

***RESET.*** Es reinicien tots els senyals i es passa a l'estat de configuració.

```
DATA_OUT_VGA = 0
ADRESS_RAM="000100001011100000011010" --Nova configuració
OE = '0'
WE = '1'
ADV = '1'
UB = '0'
LB = '0'
CE = '1'
CRE = '0'
Preescaler
OE_inout = 0
OE_inout_proc = 0
Clk_ram_enable = '0'
Clk_ram_counter = 0
ESTAT = CONFIGURACIÓ
```

***CONFIGURACIÓ.*** En aquest estat es porta a terme la configuració de la memòria RAM externa. Es genera una operació d'escriptura en mode

asíncron i s'esperen 80 ns. A continuació es passa a l'estat de LECTURA OBLIGADA.

```
CRE = '1'
ADV = '0'
CE = '0' --Chip enabled
WE = '0' --Write
SI preescaler < 4 ALESHORES --Espera de 80 ns
      Preescaler = preescaler + 1
SINO
      Preescaler = 0
      ADRESS_RAM = 0
      ADV = '1'
      CE = '1'
      WE = '1'
      CRE = '0'
      ESTAT = LECTURA_OBLIGADA
FI SI
```

***LECTURA OBLIGADA.*** En aquest estat es fa una operació de lectura en mode asíncron, recomanada pel fabricant.

```
ADV = '0'
CE = '0' --Chip enabled
SI preescaler < 4 ALESHORES --Espera de 80 ns
      Preescaler = preescaler + 1
SINO
      Preescaler = 0
      ADV = '1'
      CE = '1'
      WE = '1'
      ESTAT = STANDBY
FI SI
```

***STANDBY.*** Aquest és l'estat de repòs mentre el bloc RAM DRIVER no s'utilitza. Està a l'espera de posar-se en funcionament per a realitzar les operacions d'escriptura o lectura necessàries.

```
ADRESS_RAM=0
ADV = '1'
CE = '1' --Chip disabled
WE = '1'
OE_inout = '0'
OE_inout_proc = '0'
Clk_ram_enable = '0'
Preescaler = 0
Clk_ram_counter = 0
SI WT = '1' ALESHORES
      ESTAT = WRITE
      ADRESS_RAM = ADRESS_IN_CAM
      DATA_RAM_OUT = DATA_IN_CAM
```

```
        OE_inout = '1'
        CE = '0'
        ADV = '0'
        WE = '0' --Write
    SINO SI RD = '1' ALESHORES
        ESTAT = READ_BURST
        ADRESS_RAM = ADRESS_IN_VGA
    SINO SI READ_PROC  = '1' ALESHORES
        ESTAT = READ_ASYNC
        ADRESS_RAM = ADRESS_PROC
        OE_inout = '0'
        OE_inout_proc = '1'
        CE = '0'
        ADV = '0'
        WE = '1' --Read
    SINO SI WRITE_PROC = '1' ALESHORES
        ESTAT = WRITE
        ADRESS_RAM = ADRESS_PROC
        DATA_RAM_OUT = DATA_PROC_IN
        OE_inout = '1'
        CE = '0'
        ADV = '0'
        WE = '0' --Write
    SINO
        ESTAT = STANDBY
    FI SI
```

Com es pot veure en el pseudocodi, s'avalua l'estat dels senyals que indiquen que es vol actuar sobre la memòria RAM. Quan es vol escriure de manera asíncrona, és a dir, quan hi ha un '1' a WT o a WRITE_PROC, es copia el valor de l'adreça al senyal de sortida ADRESS_RAM i les dades al senyal intern DATA_RAM_OUT. Per tal que el *buffer* deixi d'estar en estat d'alta impedància, es posa el senyal OE_inout en estat alt. Tot seguit es procedeix a fer l'operació d'escriptura posant en estat baix els senyals CE, ADV i WE. De la mateixa manera, quan es vol escriure de manera asíncrona (READ_PROC='1') es copia l'adreça a ADRESS_RAM però en aquest cas es deixa el *buffer* connectat a la memòria RAM en estat d'alta impedància per poder llegir correctament les dades que envia. D'altra banda, el senyal OE_inout_proc es deixa en estat alt per tal que es puguin enviar correctament les dades llegides cap al bloc de processament. Per últim, quan es vol llegir en mode *burst* es copia l'adreça a partir de la qual es realitzaran les lectures i es passa a l'estat de READ_BURST.

***WRITE.*** En aquest estat es guarden les dades de manera asíncrona utilitzant un comptador per controlar l'espera. El fabricant recomana que l'espera sigui de 70 ns tot i que, mirant la figura 19 (pàgina 33), es pot veure que les dades es guarden abans d'aquest temps. Degut a que la freqüència a la que s'han de guardar les dades que envia la càmera és de

12 MHz aproximadament, amb un període de 83,33 ns, s'ha optat per un temps d'espera de 60 ns. Cal comentar que el procés principal s'avalua cada 20 ns[5] i, per tant, els temps d'espera seran múltiples d'aquest. Si s'escollís un temps de 80 ns, no hi hauria temps suficient entre operacions d'escriptura.

```
SI preescaler < 2 ALESHORES --Espera de 60 ns
      Preescaler = preescaler + 1
SINO
      Preescaler = 0
      ADV = '1'
      CE = '1'
      WE = '1'
      ESTAT = STANDBY
FI SI
```

***READ ASYNC.*** En aquest estat es llegeixen les dades de manera asíncrona seguint el mateix algorisme d'espera de l'estat anterior. En aquest cas si que es fa una espera de 80 ns.

```
SI preescaler < 3 ALESHORES --Espera de 80 ns
      Preescaler = preescaler + 1
SINO
      Preescaler = 0
      DATA_PROC_OUT = DATA_RAM_IN
      ESTAT = STANDBY
FI SI
```

***READ BURST.*** En aquest estat es tracta de preparar la memòria RAM per tal de fer una lectura de 8 dades en mode *burst*.

```
OE_inout = '0'
CE = '0'
WE = '1'
SI clk_ram_counter < 1 ALESHORES
      ADV = '0'
SINO
      ADV = '1'
FI SI
SI preescaler < 1 ALESHORES
      Preescaler = preescaler + 1
SINO
      Clk_ram_enable = '1'
      Clk_ram_counter = clk_ram_counter + 1
      SI clk_ram_counter = 2 ALESHORES
            ESTAT = SEND_DATA
      FI SI
FI SI
```

[5] El procés principal s'avalua a cada flanc de pujada del rellotge de 50 MHz, que té un període de 20 ns.

Es posa el *buffer* en estat d'alta impedància per fer la lectura i la memòria en mode lectura. Cal espera un cicle per activar el senyal CKL_RAM per tal que es comencin a comptar els cicles per motius de sincronització amb el bloc VGA DRIVER. Un cop realitzat el tercer cicle, es passa a l'estat SEND_DATA, ja que a partir del quart cicle es començaran a llegir les dades.

***SEND DATA.*** En aquest estat s'envien les dades llegides cap al bloc VGA DRIVER. Un cop s'han llegit i enviat les vuit, es torna a l'estat STANDBY.

```
DATA_OUT_VGA = DATA_RAM_IN
SI clk_ram_counter <= 10 ALESHORES
      Clk_ram_counter = clk_ram_counter + 1
SINO
      CE = '1'
      Clk_ram_counter = 0
      Clk_ram_enable = '0'
      ESTAT = STANDBY
FI SI
```

## 2.4. Bloc controlador del port VGA

En aquest aparat es detalla com es controla el port VGA encarregat de representar en una pantalla les imatges capturades per la càmera. En els següents subapartats s'expliquen les característiques de l'estàndard VGA i el funcionament del bloc controlador anomenat VGA DRIVER.

### *2.4.1.* Video Graphics Array

Les sigles VGA corresponen a *Video Graphics Array* i es refereix a tres elements: a una pantalla analògica estàndard introduïda per IBM l'any 1987, al connector DE-15 i a la pròpia resolució de 640x480 píxels. Tot i que la sincronització VGA es va fer especialment per a pantalles analògiques de rajos catòdics, les pantalles actuals amb tecnologia LED s'han dissenyat de tal manera perquè puguin utilitzar els mateixos senyals de sincronisme. Aquestes pantalles més modernes poden adoptar diferents resolucions canviant els senyals de sincronisme, els quals són polsos a 3,3V o 5V. La mida real d'un píxel és funció de la mida real de la pantalla i del nombre de files i columnes, els quals venen determinats pel sincronisme.

El sincronisme es basa en generar els polsos HS i VS coordinar-se amb el rellotge de píxel, que defineix quant de temps es disposa per representar la informació d'un píxel. El senyal VS defineix la freqüència de refresc, és a dir, quantes imatges per segon es representen a la pantalla i el nombre de línies de cada imatge ve donat pel senyal HS. Es tracta de fer un escombrat de tota la pantalla, d'esquerra a dreta i de dalt a baix, línia per línia on a cada píxel se li proporciona la informació del color que ha de tenir.

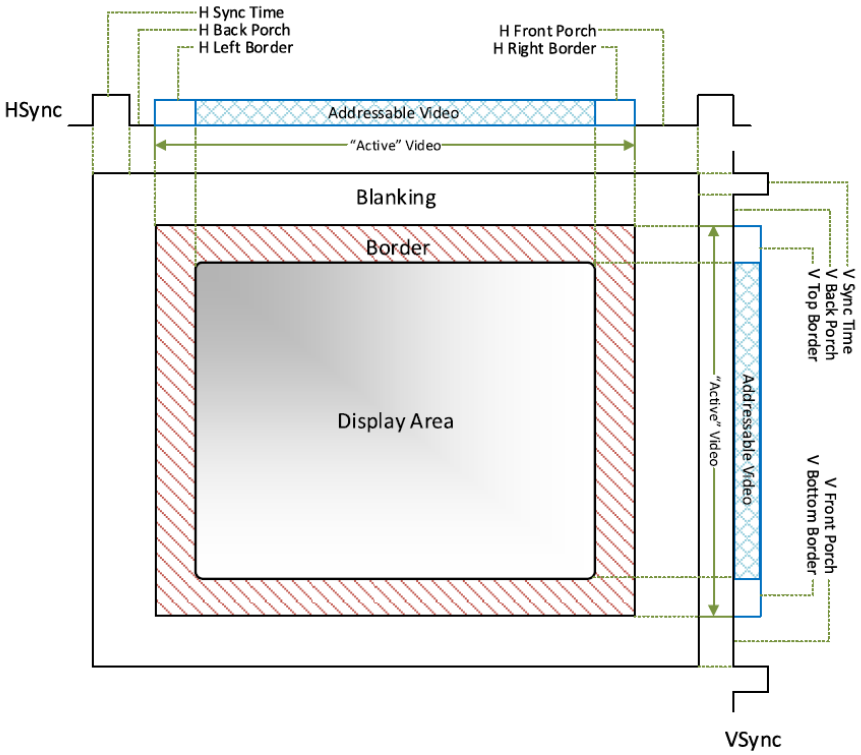


***Figura 28.*** *Sincronisme VGA.*

En aquesta aplicació es treballa amb una resolució VGA, és a dir, de 640x480 píxels a una freqüència de refresc de 60 Hz. Les especificacions per tal resolució a continuació:

***Taula 2.*** *Especificacions per a una resolució 640x480 a 60 Hz.*

| Descripció | Temps | Amplada/Freqüència |
|---|---|---|
| Rellotge de píxel | 39,7 ns (± 5%) | 25,175 MHz |
| H Sync | 3,813 µs | 96 píxels |
| H Back | 1,907 µs | 48 píxels |
| H Front | 0,636 µs | 16 píxels |
| H Addr. Video | 25,422 µs | 640 píxels |
| H Left/Right | 0 µs | 0 píxels |
| V Sync | 0,064 ms | 2 línies |
| V Back | 1,048 ms | 33 línies |
| V Front | 0,318 ms | 10 línies |
| V Addr. Video | 15,253 ms | 480 línies |
| V Top/Bottom | 0 ms | 0 línies |

Com es pot observar en la taula 2, el rellotge de píxel ha de ser de 25,175 MHz amb una certa tolerància, fet que permet que es pugui treballar amb una freqüència de 25 MHz, generada a partir de la freqüència de l'oscil·lador de 50 MHz. Això significa que cada 40 ns s'ha d'actualitzar la informació RGB que s'envia a través del port VGA, ja que és el temps que es disposa

per representar cada píxel. També es pot veure que cada imatge conté 525 línies de 800 píxels cada una, tot i que la zona vàlida (*Display Area)* és de 640x480 píxels.

La informació s'envia a través del el connector DE-15, conegut popularment com a connector VGA. És un connector de 15 pins tot i que en aquesta aplicació només se n'utilitzen 5, els quals són els dos senyals de sincronisme HS (pin 13) i VS (pin 14) i els tres senyals per a la informació del color en RGB (pins 1-3). Aquets tres són senyals analògics que van des dels 0V fins als 0,7V, indicant la intensitat de cada color primari (0V per mínima intensitat i 0,7V per màxima intensitat).

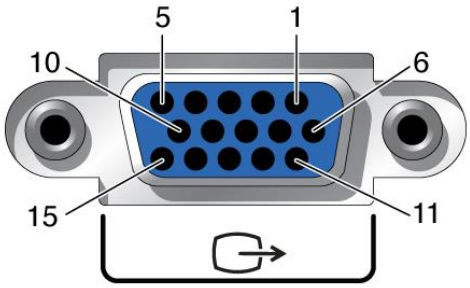


***Figura 29.*** *Connector DE-15 famella.*

Per generar els senyals analògics RGB, la placa *Nexys2* utilitza divisors de tensió amb resistències com es pot veure en la figura 30.

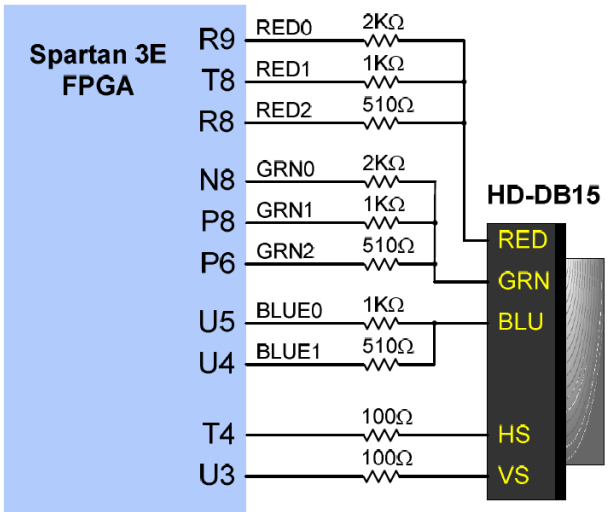


***Figura 30.*** *Connexió entre el port VGA i la FPGA.*

Els colors vermell i verd es controlen a través de 3 bits i el color blau a través de 2, ja que aquest és menys sensible als ulls humans. Utilitzant aquets 8 bits connectats als divisors de tensió, s'aconsegueix crear 8 nivells de tensió pels colors vermell i verd i 4 nivells pel color blau, entre 0V i 0,7V. Amb aquest circuit es poden crear fins a 256 colors diferents, un per a cada combinació dels 8 bits.

### 2.4.2. El bloc VGA DRIVER

El bloc VGA DRIVER és l'encarregat de proporcionar la sincronització juntament amb la informació de cada píxel a través del port VGA per tal de representar la imatge capturada per la càmera en una pantalla. És un bloc que sempre està en funcionament ja que en tot moment s'ha d'enviar informació a la pantalla.

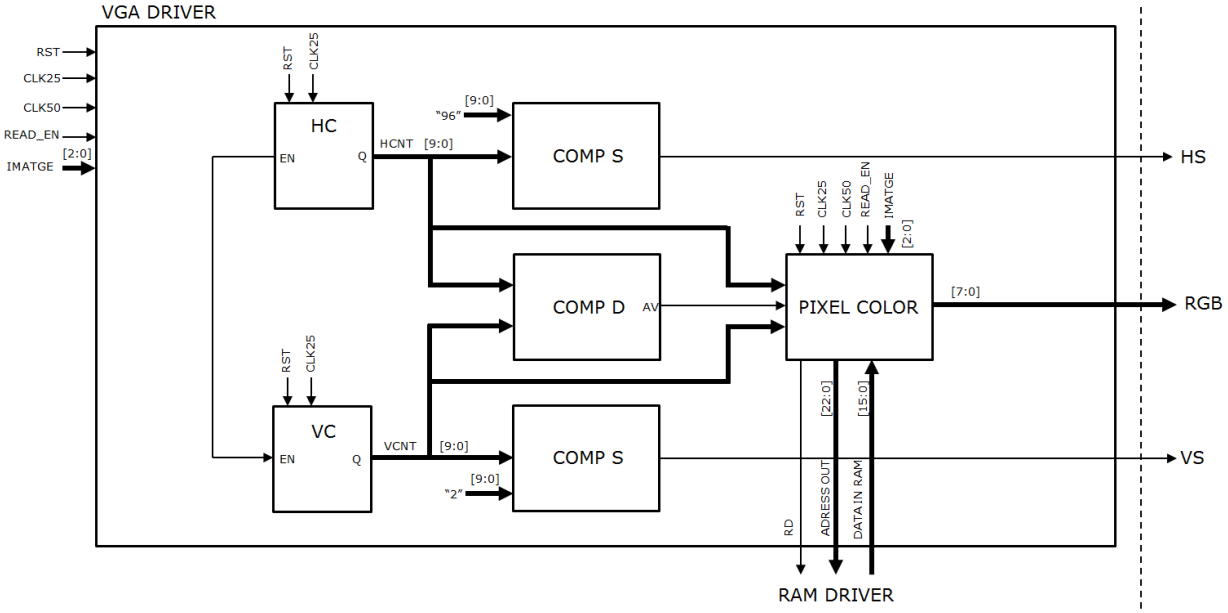


***Figura 31.*** *Diagrama de blocs de VGA DRIVER.*

Aquest bloc està format per sub-blocs seguint una arquitectura estructural i el seu funcionament és bastant simple. Té dos comptadors per controlar l'escombrat de la pantalla: el comptador HC compta les columnes i s'incrementa a una freqüència de 25 MHz (igual al rellotge de píxel). Quan arriba a 799 indica que s'ha acabat una línia i, per tant, es reinicia i dóna pas al comptador VC perquè s'incrementi. El valor dels comptes (HCNT i VCNT) van a parar a uns comparadors (COMP S) per tal de generar els polsos HC i VC en el moment correcte. També es connecten amb un altre comparador (COMP D) per tal de saber quan s'està en la zona vàlida (*Display Area*) per representar la imatge. Finalment hi ha un bloc anomenat PIXEL COLOR de vital importància ja que és l'encarregat d'enviar la informació del color de cada píxel en el moment precís. Per aquest motiu, està connectat als comptadors per saber en quin píxel es troba l'escombrat i al bloc RAM DRIVER, per obtenir la informació de cada píxel.

En els subapartats següents es detalla el funcionament de cada sub-bloc.

### 2.4.3. Els comptadors

Tal i com s'ha comentat, aquest dos sub-blocs anomenats HC i VC tenen la funció de realitzar l'escombrat dels píxels de la pantalla. El comptador horitzontal HC és l'encarregat de comptar els píxels de cada línia i el comptador vertical VC té la funció de comptar les línies. El comptador

horitzontal s'incrementa a una freqüència igual a la del rellotge de píxel, de tal maner que cada 40 ns hi haurà un canvi de píxel. Quan aquest arriba al final de línia (píxel 799), dóna pas al comptador vertical per incrementar-se. Aquest procés es va repetint fins que s'arriba a la última línia (línia 524), moment quan HC i VC es reinicien i el procés torna a començar.

A continuació el pseudocodi que segueix el comptador horitzontal HC:

```
PROCÉS (CLK25)
      SI FLANC DE PUJADA ALESHORES
            SI RST = '1' ALESHORES
                  Count = 799
                  EN = '1'
            SINO SI count = 799 ALESHORES
                  Count = 0
                  EN = '0'
            SINO SI count = 798 ALESHORES
                  Count = count + 1
                  EN = '1'
            SINO
                  Count = count + 1
                  EN = '0'
            FI SI
      FI SI
FI PROCÉS
```

Tal i com es pot observar, quan es prem el botó de reset es posa el compte a 799 per tal de que en el primer flanc després de que no ho estigui, el compte valgui zero. Quan el compte arriba a 798 el senyal d'*enable* es posa a '1' per tal que en el següent flanc el comptador vertical es pugui incrementar ja que HC valdrà 799.

A continuació el pseudocodi que segueix el comptador vertical VC:

```
PROCÉS (CLK25)
      SI FLANC DE PUJADA ALESHORES
            SI RST = '1' ALESHORES
                  Count = 524
            SINO SI EN = '1' ALESHORES
                  SI count = 524 ALESHORES
                        Count = 0
                  SINO
                        Count = count + 1
            FI SI
      FI SI
FI PROCÉS
```

De la mateixa manera que en el comptador horitzontal, quan es prem el botó de reset el compte es posa 524. Quan el senyal d'*enable* es troba en estat alt, el compte s'incrementa o es reinicia depenent del valor anterior.

### 2.4.4. Els comparadors

Hi ha dos tipus de comparadors: els simples (COMP S) i el doble (COMP D). Els comparadors simples tenen la funció de generar els senyals de sincronisme HS i VS i el comparador doble té la funció d'indicar quan s'està en zona vàlida. Tots tres treballen seguint la següent figura:


***Figura 32.*** *Valors per generar el sincronisme.*

El comparador simple encarregat de generar el senyal HS compara el valor del comptador horitzontal amb el valor 96. Mentre el compte sigui inferior a aquest nombre, cal que el senyal de sortida sigui '1' i en cas contrari que valgui '0'. El mateix principi val pel comparador simple encarregat de generar el senyal VS, ja que compara el valor del comptador vertical amb el nombre 2. A continuació el pseudocodi que segueixen aquests dos comparadors:

```
PROCÉS (count, max)
      SI count >= 0 I count < max ALESHORES
            OUT = '1'
      SINO
            OUT = '0'
      FI SI
FI PROCÉS
```

El comparador doble compara els valors dels comptadors per tal d'indicar al bloc PIXEL COLOR quan s'està en zona vàlida per representar una imatge. És necessari ja que quan l'escombrat es troba en zona de *blanking*, cal enviar la informació del color negre a través dels pins RGB. A continuació el pseudocodi que segueix:

```
PROCÉS (countH, countV)
      SI countH >= 144 I countH < 784 I countV >= 35 I
countV < 515 ALESHORES
            AV = '1'
      SINO
            AV = '0'
      FI SI
FI PROCÉS
```

### 2.4.5. Els sub-bloc PIXEL COLOR

Aquest sub-bloc és l'encarregat de d'enviar la informació del color del píxel que s'està representant a la pantalla en tot moment. Per dur a terme

aquesta tasca manté una comunicació amb el bloc RAM DRIVER per tal d'obtenir la informació adequada en el moment desitjat guiant-se amb el valor dels comptadors. També fa la funció de descodificador, ja que cal transformar la informació rebuda per adequar-la al format de 8 bits.

Està format per dos sub-blocs: el sub-bloc RAM DATA IMPORTER i el sub-bloc DATA DECODER. Com el seu nom indica, el primer és l'encarregat de comunicar-se amb el bloc controlador de la memòria RAM per importar les dades i el segon és l'encarregat de la descodificació.

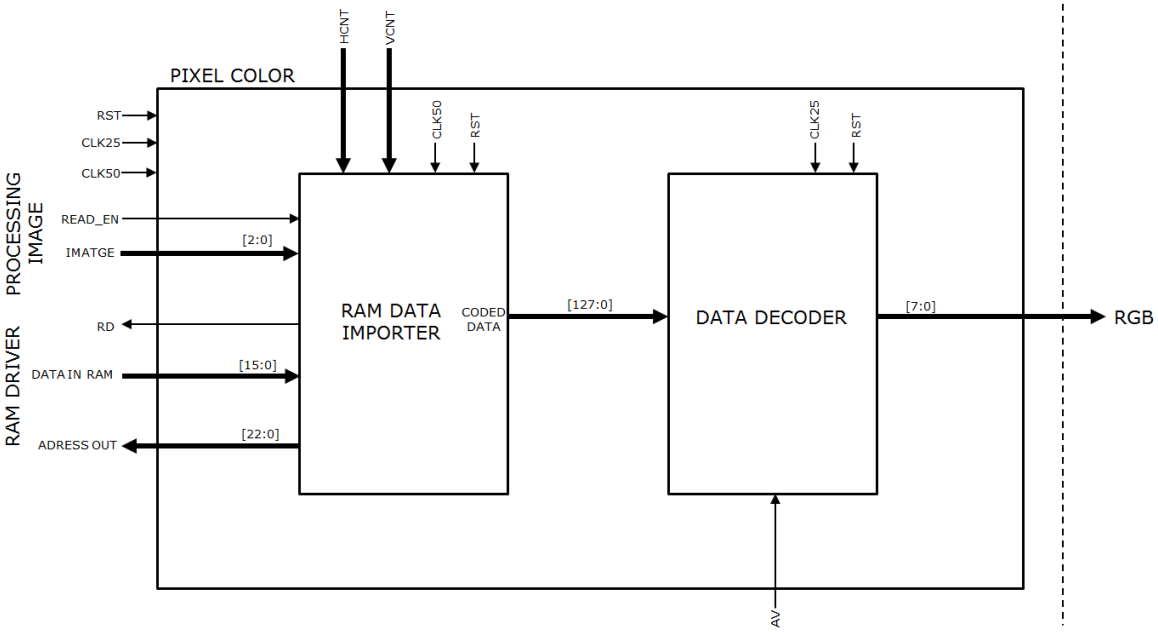


***Figura 33.*** *El sub-bloc PIXEL COLOR.*

El funcionament és el següent: el bloc RAM DATA IMPORTER, utilitzant els valors dels comptadors, envia el senyal de lectura al bloc RAM DRIVER en el moment que l'escombrat arriba a la zona vàlida. Com s'ha comentat anteriorment, l'operació de lectura és en mode *burst* a una freqüència de 50 MHz ja que no hi ha prou temps per llegir i entregar la informació si es fes en mode asíncron. La llargada del *burst* és de 8 *words*, per tant la informació que rep aquest sub-bloc és de paquets de 8 píxels. La idea general és que mentre s'està entregant la informació de 8 píxels pels pins RGB de manera consecutiva, a l'hora s'estigui llegint la informació dels 8 píxels següents. Quan s'ha enviat la informació de l'últim píxel, s'actualitza el vector d'informació que s'envia al sub-bloc descodificador i es procedeix a llegir la informació dels 8 píxels següents i així successivament fins al final de l'escombrat. Cal destacar que només es llegeix de la memòria RAM quan ho permet el senyal READ_EN, ja que procedeix del bloc encarregat del processament de la imatge i indica quan es pot començar a representar la imatge a la pantalla, és a dir, quan es pot llegir de la memòria externa. També li arriba un altre senyal provinent de PROCESSING IMAGE anomenat IMATGE que indica quina imatge guardada a la memòria externa s'ha de representar a la pantalla.

El vector que s'envia al bloc descodificador inclou la informació de 8 píxels, per tant ocupa 128 bits ja que cada píxel ocupa 16 bits d'informació. La funció que fa el bloc descodificador és separar la informació de cada píxel i entregar-la en el moment precís. A més a més, ha de transformar el format de les dades ja que el port VGA que incorpora la placa *Nexys2* té 8 bits de resolució i, com s'ha comentat anteriorment, la informació RGB que envia la càmera està en format RGB444, és a dir, 4 bits d'informació per a cada color primari.

A continuació es pot observar el diagrama de sincronisme que segueix el bloc RAM DATA IMPORTER:

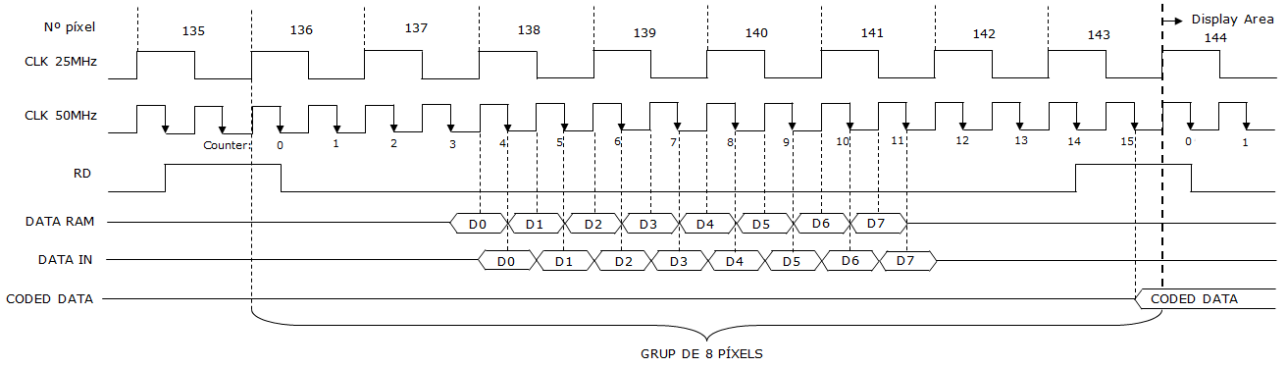


***Figura 34.*** *Diagrama de sincronització del bloc RAM DATA IMPORTER.*

Els blocs RAM DRIVER i RAM DATA IMPORTER utilitzen el mateix senyal de rellotge de 50 MHz tot i que el primer executa la seva màquina d'estats en el flanc de pujada i el segon en el flanc de baixada. Es pot observar que quan l'escombrat arriba al píxel 135 (a partir de la línia 35) envia el senyal de lectura al bloc controlador de la RAM per tal de realitzar la primera operació de lectura en mode *burst*. El bloc RAM DRIVER llegeix les dades en el flanc de pujada i les envia al bloc RAM DATA IMPORTER per tal que les llegeixi en el flanc de baixada, tal i com es pot veure amb els senyals DATA RAM i DATA IN[6]. Un cop s'ha acabat la lectura, s'envia la informació del paquet de 8 píxels al bloc descodificador just en el moment abans d'entrar en la zona vàlida, a partir del píxel 144. El mateix procés es va repetint contínuament fins a arribar al final de l'escombrat.

La màquina que segueix el bloc RAM DATA IMPORTER és la següent:

[6] El senyal DATA RAM és el bus d'informació entre el bloc RAM DRIVER i la memòria externa i el senyal DATA IN es el bus d'informació entre el bloc RAM DRIVER i el bloc PIXEL COLOR.

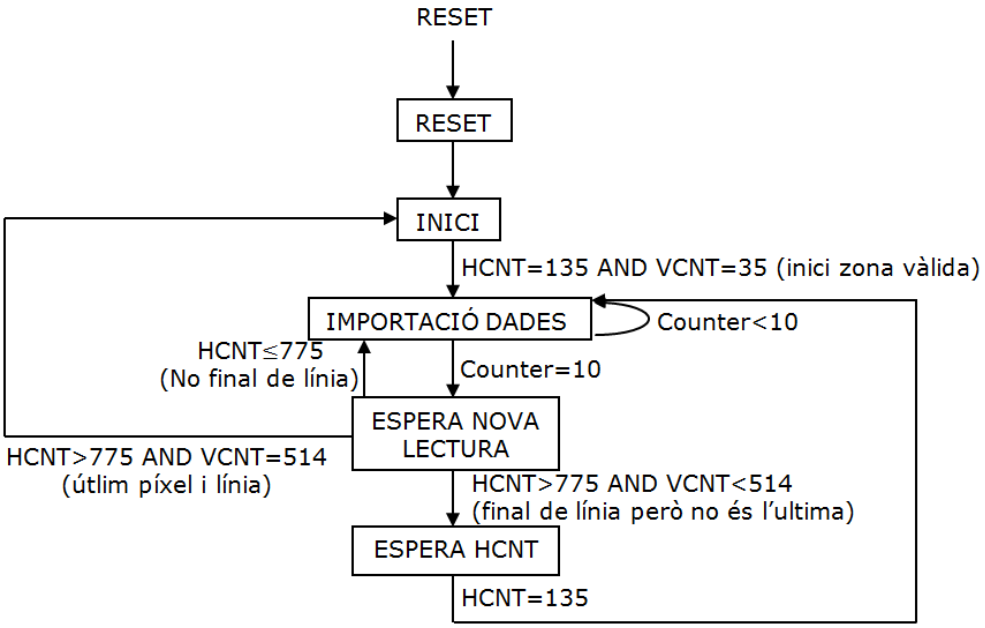


***Figura 35.*** *Màquina d'estats del bloc RAM DATA IMPORTER.*

***RESET.*** En es reinicien tots els senyals interns i es passa a l'estat d'inici.

```
Data1 = "1111...1111"
Data2 = "1111...1111"
Counter = 15
Data_enabled = '0' --'0' per data1//'1' per data2
Preescaler = '0'
Adress_out = 0
ESTAT = INICI
```

Cal comentar que els senyals *data1* i *data2* són vectors de 128 bits on es guarda la informació dels paquets de 8 píxels. Se n'utilitzen dos degut a que mentre un es enviat al bloc descodificador, a l'altre s'hi guarda la informació dels 8 següents i viceversa. Utilitzant el senyal *data_enabled* es tria a quin dels dos vectors toca guardar-hi la informació.

***INICI.*** En aquest estat s'espera que l'escombrat arribi al primer píxel per tal de començar a llegir informació de la memòria RAM.

```
CAS IMATGE
      QUAN "000"
            Adress_out = 0
      QUAN "001"
            Adress_out = 307200
      QUAN "010"
            Adress_out = 614400
      QUAN "011"
            Adress_out = 921600
      QUAN "100"
            Adress_out = 1228800
      QUAN "101"
```

```
            Adress_out = 1536000
      QUAN "110"
            Adress_out = 1843200
      QUAN ALTRES
            Adress_out = 0
FI CAS
Counter = 15
Data_enabled = '0'
Preescaler = '0'
SI HCNT = 135 I VCNT = 35 ALESHORES
      RD = '1'
      SI preescaler = '0' ALESHORES
            Preescaler = '1'
      SINO
            Preescaler = 0
            ESTAT = IMPORTACIÓ_DADES
      FI SI
SINO
      ESTAT = INICI
FI SI
```

Quan s'arriba al píxel 135 de la línia 35 significa que s'ha de llegir el primer paquet d'informació ja que a partir del píxel 144 s'entra en zona vàlida. Es posa el senyal RD en estat alt i s'espera un cicle de rellotge per tal que el comptador (*counter*) es posi a zero en el píxel 136[7]. També es pot veure com es defineix l'adreça inicial a partir de la qual es començarà a llegir de la memòria RAM en funció del senyal IMATGE. Cada valor correspon a l'adreça inicial on està guardada cada imatge.

***IMPORTACIÓ DADES.*** En aquest estat es guarda la informació rebuda des del bloc RAM DRIVER en un dels dos vectors (*data 1* o *data2*) utilitzant el valor del senyal *counter*.

```
RD = '0'
CAS counter
      QUAN 3
            SI data_enabled = '0' ALESHORES
                  Data1[127:112] = DATA_IN_RAM
            SINO
                  Data2[127:112] = DATA_IN_RAM
            FI SI
      QUAN 4
            SI data_enabled = '0' ALESHORES
                  Data1[111:96] = DATA_IN_RAM
            SINO
                  Data2[111:96] = DATA_IN_RAM
            FI SI
      QUAN 5
```

[7] S'han dividit les línies en grups de 8 píxels per tal d'organitzar les operacions de lectura. El píxel 136 és el primer píxel de l'últim grup abans que l'escombrat entri en zona vàlida.

```
            SI data_enabled = '0' ALESHORES
                  Data1[95:80] = DATA_IN_RAM
            SINO
                  Data2[95:80] = DATA_IN_RAM
            FI SI
      QUAN 6
            SI data_enabled = '0' ALESHORES
                  Data1[79:64] = DATA_IN_RAM
            SINO
                  Data2[79:64] = DATA_IN_RAM
            FI SI
      QUAN 7
            SI data_enabled = '0' ALESHORES
                  Data1[63:48] = DATA_IN_RAM
            SINO
                  Data2[63:48] = DATA_IN_RAM
            FI SI
      QUAN 8
            SI data_enabled = '0' ALESHORES
                  Data1[47:32] = DATA_IN_RAM
            SINO
                  Data2[47:32] = DATA_IN_RAM
            FI SI
      QUAN 9
            SI data_enabled = '0' ALESHORES
                  Data1[31:16] = DATA_IN_RAM
            SINO
                  Data2[31:16] = DATA_IN_RAM
            FI SI
      QUAN 10
            ESTAT = ESPERA_NOVA_LECTURA
            SI data_enabled = '0' ALESHORES
                  Data1[15:0] = DATA_IN_RAM
            SINO
                  Data2[15:0] = DATA_IN_RAM
            FI SI
      QUAN 15
            Counter = 0
      QUAN ALTRES

FI CAS
SI counter < 15 ALESHORES
      Counter = counter + 1
FI SI
```

Com es pot observar, a partir de *counter*=3 es va guardant la informació en els 16 bits corresponents en un dels dos vectors. Quan s'ha guardat la informació dels 8 píxels (*counter*=10), es passa a l'estat d'espera per a una nova lectura.

***ESPERA NOVA LECTURA.*** En aquest estat s'hi entra un cop s'han guardat les dades i el senyal intern *counter* val 11. Si no s'ha arribat al final de la línia que s'està llegint, es torna a l'estat d'IMPORTACIÓ DADES. En canvi, si l'escombrat passa per l'últim píxel de la línia però no és l'última, es procedeix a esperar que el comptador horitzontal torni a arribar al píxel 135 per iniciar les noves lectures. Finalment, si l'escombrat es troba en l'últim píxel de l'última línia, es passa a l'estat inicial a l'espera d'un nou escombrat.

```
CAS counter
      QUAN 12
            SI HCNT != 782 ALESHORES
                  Adress_out = adress_out + 8
            FI SI
      QUAN 13
            SI HCNT <= 767 ALESHORES
                  RD = '1'
            FI SI
      QUAN 14
            SI HCNT <= 775 ALESHORES --No és final de línia
                  ESTAT = IMPORTACIO DADES
                  SI data_enabled = '0' ALESHORES
                        CODED_DATA = data1
                        Data_enabled = '1'
                  SINO
                        CODED_DATA = data2
                        Data_enabled = '0'
                  FI SI
            SINO -- Final de línia
                  CODED_DATA = "0000...0000"
                  SI VCNT = 514 ALESHORES --Última línia
                        ESTAT = INICI
                  SINO --No és l'última línia
                        ESTAT = ESPERA_HCNT
                  FI SI
            FI SI
      QUAN ALTRES
            ESTAT = ESPERA_NOVA_LECTURA
FI CAS
Counter = counter + 1
```

Quan el comptador val 12, el valor de l'adreça s'augmenta de 8 en 8 per tal de realitzar la lectura dels 8 píxels següents. Només s'augmenta si el valor de HCNT és diferent a 782, ja que no cal que se segueixi llegint després d'aquest píxel perquè s'entra en zona de *blanking*. Quan el comptador val 13, es determina l'estat del senyal RD: mentre no s'arribi al final de línia s'ha de llegir contínuament fins a arribar al píxel 767[8], moment de realitzar

[8] A partir del píxel 767 no s'envia cap més senyal de lectura, ja que falten dos grups de 8 píxels per arribar al final de la zona vàlida i no cal llegir més dades.

l'última operació de lectura. Quan el comptador val 14 s'envia la informació al bloc descodificador mentre no s'arribi al final de línia. Si és el cas, s'avalua si és l'últim píxel de l'última línia i en cas que ho sigui, es torna a l'estat inicial. Si no és l'última línia, es passa a l'estat ESPERA HCNT.

***ESPERA HCNT.*** En aquest estat s'hi entra quan s'ha acabat de llegir la informació d'una línia però no és l'última i s'espera a que l'escombrat torni a arribar en el píxel 135 per tal de procedir a llegir de la RAM.

```
Counter = 15
Data_enabled = '0'
Preescaler = '0'
SI HCNT = 135 ALESHORES
      RD = '1'
      SI preescaler = '0'
            Preescaler = '1'
      SINO
            Preescaler = '0'
            ESTAT = IMPORTACIÓ_DADES
      FI SI
SINO
      ESTAT = ESPERA_HCNT
FI SI
```

A continuació es detalla el bloc encarregat de la descodificació de les dades per tal de poder enviar la informació RGB correctament pel port VGA.

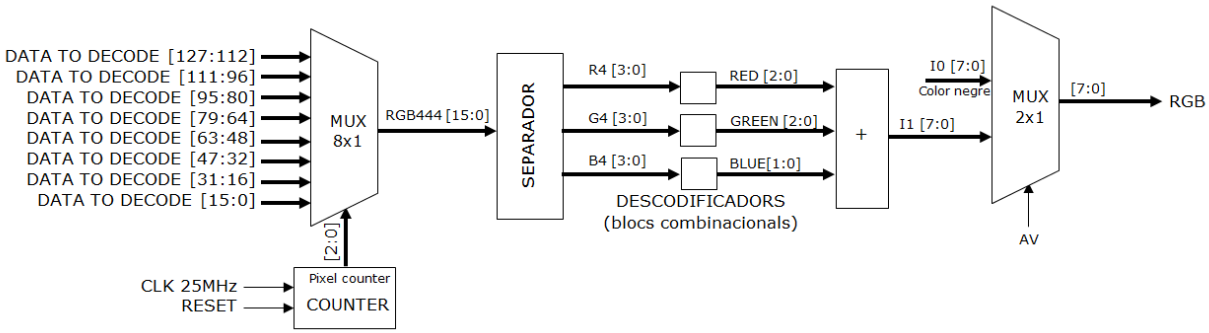


***Figura 36.*** *Esquema del bloc DATA DECODER.*

Com es pot observar, utilitzant un multiplexor 8x1 es tria de quin píxel es vol descodificar la informació. Gràcies a un comptador a una freqüència igual a la del rellotge de píxel (25 MHz) es genera el senyal de control de 3 bits per seleccionar cada un dels 8 píxels. Com s'ha comentat anteriorment, les línies s'han dividit en grups de 8 píxels per tal d'organitzar les operacions de lectura i, a més a més, per tal que estigui sincronitzat amb el comptador de tal manera que quan l'escombrat arriba al píxel 144 (el primer píxel de la zona vàlida) el valor del senyal de control és "000", és a dir, se selecciona la informació del primer píxel del paquet.

La informació seleccionada està formada per 16 bits, 12 dels quals amb informació útil, la qual utilitzant el bloc separador se separa la informació de 4 bits de cada color primari. Fent servir un seguit de blocs combinacionals es transforma la informació de 4 bits a 3, en el cas dels colors vermell i verd i a 2 en el cas del color blau. Aquesta nova informació s'encadena seguint l'ordre RGB per obtenir el vector de 8 bits que s'enviarà a través del port VGA. Hi ha un multiplexor 2x1 controlat pel senyal provinent del comparador doble que indica quan s'està en zona vàlida. En el cas que s'estigui en zona de *blanking* cal que s'enviï la informació del color negre (tot zeros) a través del port.

La lògica combinacional que segueixen els blocs descodificadors és la següent:

***Taula 3.*** *Descodificació RGB444 a RGB de 8 bits.*

| RGB444 | Red / Green | Blue |
|---|---|---|
| 0000 | 000 | 00 |
| 0001 | 000 | 00 |
| 0010 | 001 | 00 |
| 0011 | 001 | 00 |
| 0100 | 010 | 01 |
| 0101 | 010 | 01 |
| 0110 | 011 | 01 |
| 0111 | 011 | 01 |
| 1000 | 100 | 10 |
| 1001 | 100 | 10 |
| 1010 | 101 | 10 |
| 1011 | 101 | 10 |
| 1100 | 110 | 11 |
| 1101 | 110 | 11 |
| 1110 | 111 | 11 |
| 1111 | 111 | 11 |

Finalment, el procés seqüencial que segueix el comptador pel control del multiplexor 8x1 és el següent:

```
PROCÉS (CLK25)
     SI FLANC DE PUJADA ALESHORES
          SI RST = '1' ALESHORES
               Píxel_counter = "111"
          SINO
               SI píxel_counter < "111" ALESHORES
                    Píxel_counter = píxel_counter + 1
               SINO
                    Píxel_counter = "000"
                    DATA_TO_DECODE = CODED_DATA --latch
               FI SI
```

```
            FI SI
        FI SI
    FI PROCÉS
```

La descripció en VHDL dels blocs multiplexors i descodificadors es pot trobar en els annexos.

# CAPÍTOL 3: PROCESSAMENT DE LA IMATGE

En aquest capítol s'explica el funcionament del bloc encarregat del processament de la imatge capturada. Es detallen tots els algorismes i les tècniques utilitzades des de transformar la imatge a blanc i negre fins a l'obtenció del contorn dels objectes.

## 3.1. Processament morfològic

Les tècniques i algorismes utilitzats en aquest treball pel processament de la imatge són del camp del processament morfològic, basat en la morfologia matemàtica. La morfologia matemàtica és una eina utilitzada per extreure components d'interès de la imatge com pot ser el contorn d'objectes, l'esquelet, etc. També s'utilitza per a fer un pre o post processat de la imatge, com poden ser tècniques morfològiques d'aprimament o poda.

Les tècniques de processament morfològic s'apliquen sobretot en imatges binàries, és a dir, en imatges on cada un dels seus píxels són o blanc ('1') o negre ('0'), tot i que també es poden aplicar en imatges amb nivells de grisos. La informació d'una imatge binària es troba en un espai 2D, on cada píxel té unes coordenades (x,y) en un pla bidimensional.

Per obtenir la informació desitjada d'una imatge binària es treballa amb el que s'anomena element estructurant: és un petit subconjunt d'una forma predefinida utilitzat per examinar tot el conjunt de la imatge per extreure'n les propietats d'interès. L'element estructurant bàsic i més comú és un quadrat de 3x3 píxels tot i que es poden utilitzar altres elements sempre que siguin rectangulars, ja que la imatge a inspeccionar és rectangular i cal

que l'element hi càpiga completament quan el seu origen es trobi en una cantonada. La idea principal és que l'element estructurant escombri la imatge binària i, aplicant un seguit d'algorismes explicats a continuació, s'obtingui una nova imatge transformada amb unes característiques diferents a la original.

En aquest treball s'apliquen les tècniques d'erosió i dilatació per tal d'aplicar els algorismes d'obertura i tancament amb l'objectiu d'extreure el contorn dels objectes que apareguin a la imatge. Les tècniques d'erosió i dilatació són fonamentals en el processament morfològic ja que són bàsiques per aplicar altres algorismes morfològics més complexos.

A continuació es detallen les tècniques d'erosió i dilatació, així com els algorismes d'obertura i tancament. Finalment s'explica l'algorisme d'extracció de contorn.

***EROSIÓ.*** És una de les dues tècniques bàsiques en el processament morfològic. La seva definició matemàtica és la següent: sent A i B conjunts en $Z^2$, l'erosió de A per B és:

$$A \ominus B = \{z|(B)_z \subseteq A\} \quad (1)$$

Aquesta equació indica que l'erosió del conjunt A feta per l'element estructurant B és el conjunt de tots els punts z tal que B, traslladat per z, estigui contingut en A.

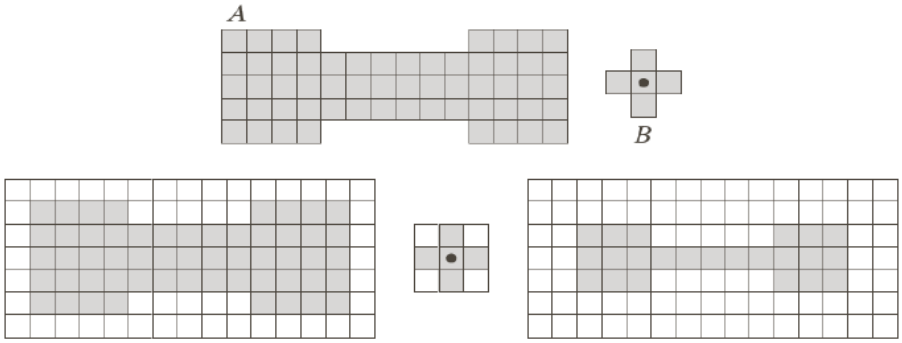


***Figura 37.*** *Tècnica d'erosió.*

Per entendre-ho millor cal observar la figura superior, on es pot veure l'element que es vol erosionar A utilitzant l'element estructurant B. Com s'ha comentat, cal que els dos elements siguin rectangulars i per tal motiu s'omplen amb píxels blancs de fons. Es tracta que l'origen de B (punt negre) passi per tots els píxels de A i s'avaluï si B està dins de A, és a dir, si tots els píxels negres de B estan a sobre de píxels negres de A. Si és el cas, el píxel on hi ha situat l'origen de B de la nova imatge serà negre i si no és el cas, serà blanc. La imatge final erosionada de la figura 37 es pot observar a baix a la dreta.

Aquesta tècnica permet eliminar detalls irrellevants com pot ser soroll en forma de píxels saltejats, ja que l'element estructurant serà més gran i els

eliminarà. Tot i així, també cal observar que els elements s'aprimen i es fan més petits.

***DILATACIÓ.*** És la segona tècnica bàsica del processament morfològic. La seva definició matemàtica és la següent: sent A i B conjunts en $Z^2$, la dilatació de A per B es defineix com:

$$A \oplus B = \{z|(B)_z \cap A \neq \emptyset\} \quad (2)$$

L'equació indica que la dilatació del conjunt A feta per l'element estructurant B és el conjunt de tots els punts z tal que B i A se solapin almenys en un píxel.

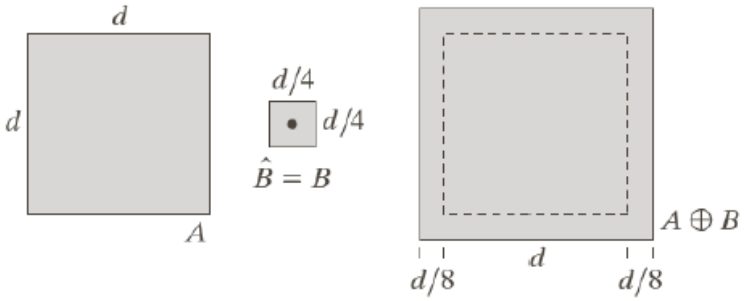


***Figura 38.*** *Tècnica de dilatació.*

Explicat d'una manera més simple, segueix el mateix principi de funcionament que la tècnica d'erosió amb la diferència que a l'hora de definir l'estat del píxel ('1' blanc o '0' negre) de la nova imatge s'avalua si algun píxel negre de B està sobre algun píxel negre de A. Si és el cas, l'estat del nou píxel serà negre i si no és el cas serà blanc.

Aquesta tècnica permet omplir petits buits que han quedat blancs en els elements ja que l'element estructurant és més gran i els ompliria de color negre. També cal observar que els elements s'engreixen i es fan més grans.

***OBERTURA.*** És una tècnica molt important per a operacions morfològiques ja que tendeix a suavitzar els contorns dels objectes eliminant el soroll en forma de protuberàncies que hi pugui haver i també trenca istmes entre dos elements més grans fent que se separin.

L'obertura d'un conjunt A per un element estructurant B es defineix com:

$$A \circ B = (A \ominus B) \oplus B \quad (3)$$

Com es pot observar en l'equació 3, la tècnica d'obertura tracta de realitzar una operació d'erosió a A per B, seguida d'una dilatació del resultat per B.

La interpretació geogràfica és simple, ja que com es pot observar a la figura que ve a continuació, l'obertura de A per B és el conjunt de tots els B que caben a A.

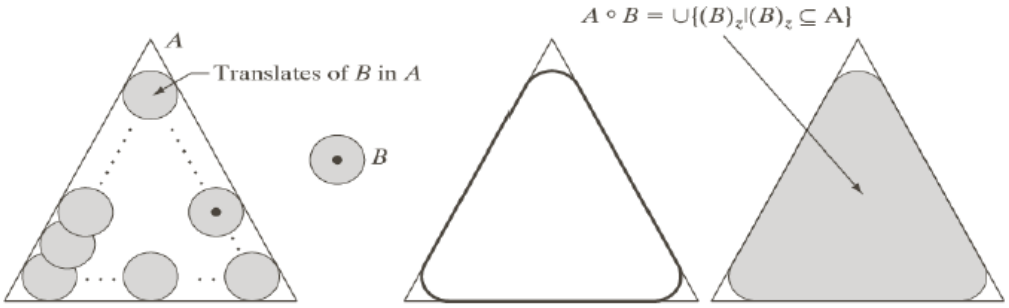


***Figura 39.*** *Tècnica d'obertura.*

***TANCAMENT.*** És una altra tècnica molt important per a operacions morfològiques ja que també tendeix a suavitzar els contorns dels objectes omplint possibles "golfs" o espais que hi pugui haver i també elimina petits forats en els elements.

El tancament d'un conjunt A per un element estructurant B es defineix com:

$$A \bullet B = (A \oplus B) \ominus B \tag{4}$$

Com es pot observar en l'equació 4, la tècnica d'obertura tracta de realitzar una operació de dilatació a A per B, seguida d'una erosió del resultat per B.

La interpretació geogràfica també és simple però a diferència de la tècnica d'obertura, l'element estructurant no "roda per dins" sinó per fora, com es pot veure a continuació.

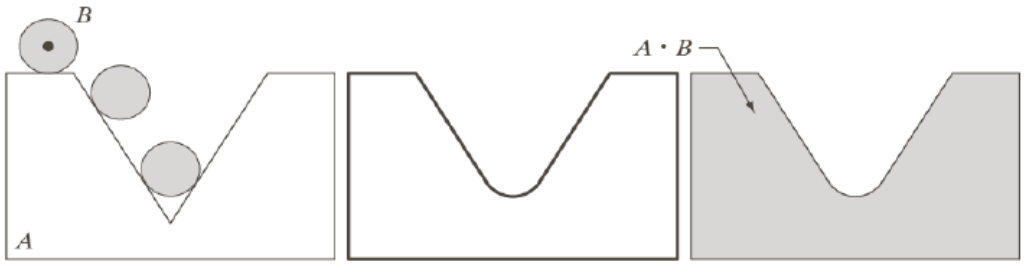


***Figura 40.*** *Tècnica de tancament.*

***EXTRACCIÓ DE CONTORN.*** És un algoritme que utilitza les tècniques vistes fins ara que té com a aplicació extreure el contorn dels objectes que apareixen en la imatge.

L'algorisme d'extracció de contorn d'un objecte A es defineix de la següent manera:

$$\beta(A) = A - (A \ominus B) \tag{5}$$

El contorn, anomenat β(A), es pot obtenir erosionant A per B seguit de la diferència entre A i l'erosió.

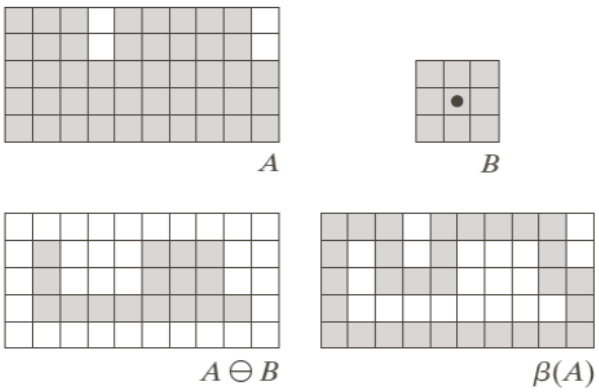


***Figura 41.*** *Algorisme per l'extracció de contorn.*

Com es pot observar, per extreure el contorn del conjunt A s'utilitza un element estructurant de 3x3 per aplicar-li la tècnica d'erosió (a baix a l'esquerra). Comparant el resultat de l'erosió amb el conjunt original es pot obtenir d'una manera senzilla el contorn de A. En definitiva, el contorn que es representa són els píxels que s'han erosionat en el procés d'erosió.

## 3.2. El bloc PROCESSING IMAGE

El bloc encarregat del processament de la imatge, aplicant les tècniques i algorismes que s'han explicat prèviament és el PROCESSING IMAGE. És un bloc descrit amb una arquitectura de comportament, és a dir, no està format per altres sub-blocs.

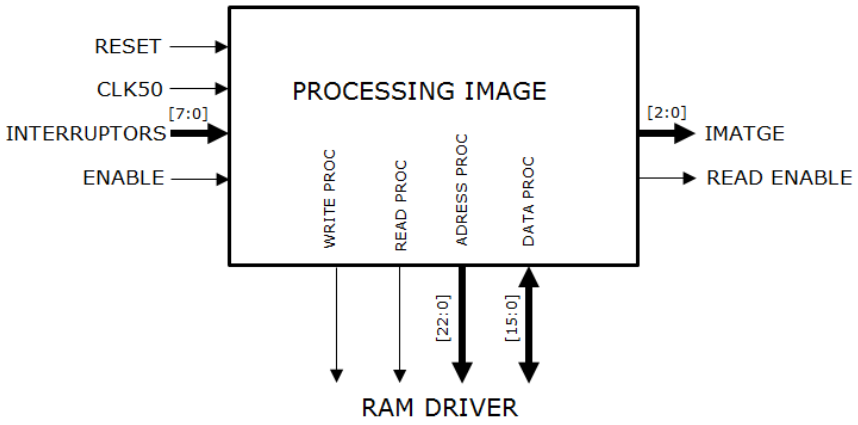


***Figura 42.*** *El bloc PROCESSING IMAGE.*

Com es pot observar, a aquest bloc li arriben fins a tres senyals exteriors els quals són el senyal de reset i el senyal de rellotge de 50 MHz, que serveixen per controlar la màquina d'estats, i el senyal dels interruptors, que serveix per determinar quin filtre es vol aplicar a la imatge. També li arriba un senyal provinent del bloc CAM DRIVER, el qual serveix d'*enable* i indica quan pot començar aplicar el procés seleccionat a la imatge capturada. Un cop ha acabat envia dos senyals al bloc VGA DRIVER: el senyal READ ENABLE li indica que ja pot començar a llegir i el senyal

IMATGE li indica quina de les imatges guardades a la memòria RAM s'ha de representar al a pantalla.

Es comunica amb el bloc controlador de la memòria RAM a través de quatres senyals, dos dels quals per indicar si es vol llegir o escriure i els altres dos serveixen de bus de dades i per indicar l'adreça respectivament. Les operacions de lectura i escriptura es fan de manera asíncrona, ja que es necessita accedir a adreces no consecutives per a realitzar els algorismes dissenyats.

Es poden aplicar fins a deu filtres diferents, els quals són:

- **Color:** es presenta la imatge tal qual s'ha capturat, és a dir, en color.
- **Blanc i negre:** es presenta la imatge binaritzada a partir de l'original. Els píxels només poden ser de dos colors: blanc o negre.
- **Escala de grisos:** es passa la imatge original de color a escala de grisos. Els píxels poden ser de quatre colors: blanc, gris clar, gris fosc o negre.
- **Erosió:** s'aplica l'algorisme d'erosió a la imatge binaritzada.
- **Dilatació:** s'aplica l'algorisme de dilatació a la imatge binaritzada.
- **Obertura:** s'aplica l'algorisme d'obertura a la imatge binaritzada.
- **Tancament:** s'aplica l'algorisme de tancament a la imatge binaritzada.
- **Obertura-tancament:** s'aplica l'algorisme d'obertura seguit del de tancament. Aquest algorisme és molt útil per prepara la imatge per a l'extracció de contorn ja que suavitza els contorns i elimina el soroll.
- **Contorn:** s'extreu el contorn dels objectes directament de la imatge binaritzada.
- **Contorn obertura-tancament:** s'extreu el contorn dels objectes després d'haver aplicat l'algorisme d'obertura-tancament.

Cada filtre segueix un algorisme diferent però el funcionament general és semblant per tots, com es pot veure en la taula a continuació:

***Taula 4.*** *Posicions de memòria ocupades per cada filtre.*

| | **Nº imatge i adreces** | | | | | | |
|---|---|---|---|---|---|---|---|
| **Filtre** | 0 (0-307199) | 1 (307200-614399) | 2 (614400-921599) | 3 (921600-1228799) | 4 (1228800-1535999) | 5 (1536000-1843199) | 6 (1843200-2150399) |
| Color | Original | - | - | - | - | - | - |
| Blanc i Negre | Original | BN | - | - | - | - | - |
| Grisos | Original | Grisos | - | - | - | - | - |
| Erosió | Original | BN | Erosió | - | - | - | - |
| Dilatació | Original | BN | Dilat. | - | - | - | - |
| Obertura | Original | BN | Erosió | Dilat. | - | - | - |
| Tancament | Original | BN | Dilat. | Erosió | - | - | - |
| Ober-Tanc | Original | BN | Erosió | Dilat. | Dilat. | Erosió | - |
| Contorn | Original | BN | Contorn | - | - | - | - |
| Contorn O-T | Original | BN | Erosió | Dilat. | Dilat. | Erosió | Contorn |

La imatge original en color es guarda a partir de l'adreça zero de memòria. Com es pot veure, s'ha dividit la memòria en grups de 307.200 adreces, ja que cada imatge té 307.200[9] píxels, i cada cop que s'aplica un algorisme es guarda la nova imatge just a continuació de la imatge a la que se li ha aplicat, de tal manera que es va ocupant la memòria d'imatges consecutives. Un cop s'ha acabat d'aplicar el filtre, s'indica al bloc VGA DRIVER quina imatge s'ha de representar a la pantalla.

A continuació es poden veure les dues màquines d'estats que regeixen el comportament del bloc:

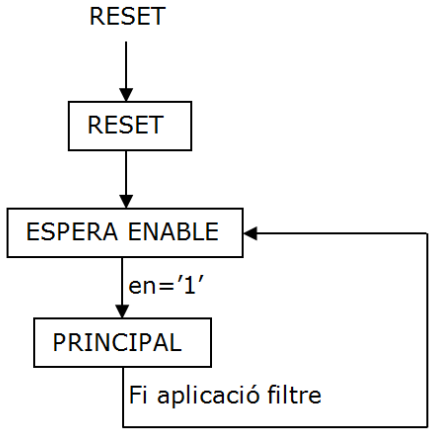


***Figura 43a.*** *Màquina d'estats principal.*

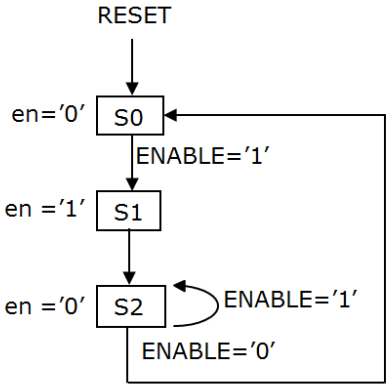


***Figura 43b.*** *Màquines d'estats per l'enable.*

[9] Una imatge de resolució VGA 640x480 té 307.200 píxels (0,3 megapíxels).

En la figura 43b es pot veure com es controla l'estat del senyal *enable* (*en*), per tal que s'activi el bloc de processament quan el bloc CAM DRIVER ho indiqui. Com que el senyal que s'envia es manté a '1' un cop s'ha acaba de capturar la imatge, es necessari treballar a partir del senyal *en* de tal manera que el bloc de processament no es quedi actiu un cop hagi acabat d'aplicar el filtre. La màquina d'estats funciona igual que en el bloc CAM PROCESSOR per controlar el *trigger*, ja que mentre el senyal *ENABLE* estigui actiu, el senyal *en* estarà a nivell baix.

La màquina d'estats principal es compon de més estats dels que es poden veure en la figura 43a degut a la seva complexitat, ja que l'estat principal és el "centre de control" on hi ha definits tots els passos a seguir per cada filtre. A partir d'aquest es passa a altres estats per aplicar els algorismes necessaris i es torna a l'estat principal per obtenir la informació del següent algorisme a aplicar.

A continuació es descriuen el senyals interns utilitzats per tal de facilitar la comprensió.

- **DATA_IN:** conté la informació enviada des del bloc RAM DRIVER.

- **DATA_OUT:** conté la informació que es vol enviar cap al bloc RAM DRIVER.

- **DATA_OUT_AUX, DATA_OUT_BN, DATA_OUT_BNAUX:** degut a que no es pot assignar un valor des de dins i des de fora d'un *process* en un mateix senyal a l'hora, s'han hagut de crear diferents senyals connectats entre ells per un multiplexor per evitar aquests conflictes. S'explicarà més endavant com estan connectats entre ells.

- **BN:** Senyal de selecció del multiplexor encarregat de seleccionar quin senyal s'ha de connectar amb DATA_OUT.

- **OE_inout:** és el senyal de control per a un *buffer* tri-estat no inversor per controlar quan s'han de llegir o escriure dades del bus de dades connectat amb el bloc RAM DRIVER.

- **Mask1, mask2, ..., mask9 i centre:** senyals per guardar l'adreça on s'ha de llegir la informació de la màscara per l'escombrat dels algorismes d'erosió i dilatació. Cal comentar que la màscara està composta per 9 píxels, ja que s'ha escollit treballar amb una màscara quadrada de 3x3. L'adreça del píxel central és guarda en el senyal anomenat *centre*.

- **DATA_mask1, DATA_mask2, ..., DATA_centre:** on es guarda la informació de cada píxel de la màscara.

- **Mask:** comptador per controlar quin píxel de la màscara s'ha de llegir.

- **Preescaler:** utilitzat per controlar les operacions d'escriptura i lectura a la memòria RAM. Es fan de manera asíncrona.

- **Row_conunter, column_counter:** utilitzats per controlar els diferents escombrats que s'han de fer sobre la imatge.

- **En:** senyal comentat anteriorment per controlar quan s'ha d'activar el bloc de processament.

- **Fase_Contorn:** comptador utilitzat per controlar l'algorisme d'extracció de contorn.

- **Counter_contorn:** comptador utilitzat per controlar l'escombrat de la imatge en l'algorisme d'extracció de contorn.

- **Píxel_original, píxel_erosió:** utilitzat per guardar els dos píxels que s'han de comparar per l'algorisme d'extracció de contorn.

- **Copy_data:** utilitzat en els algorismes de binarització i escala de grisos, per tal de transformar la informació píxel per píxel.

- **Fase_copy:** indica en quin estat es troba l'algorisme de transformar la informació a blanc i negre o a escala de grisos.

- **Pas:** comptador per saber en quin pas es troba l'execució de cada filtre.

- **R4, G4, B4, red, green, blue:** utilitzats per la transformació de la informació de cada píxel a blanc i negre o a escala de grisos.

- **FILTRE:** indica quin filtre s'ha d'aplicar.

- **ALGORISME:** indica quin algorisme s'ha d'aplicar, si erosió o dilatació, en cada pas de cada filtre.

A continuació es detalla el pseudocodi dels estats de RESET i ESPERA ENABLE:

***RESET.*** En aquest estat es reinicien tots els senyals interns.

```
Mask1 = 0
Mask2 = 0
...
Mask9 = 0
Centre = 0
IMATGE = "000"
OE_inout = '0'
Preescaler = 0
Row_counter = 0
Column_counter = 0
READ_EN = '0'
FASE_CONTORN = 1
FASE_COPY = RD
Pas = 1
BN = '0'
Counter_contorn = 0
ESTAT = ESPERA_ENABLE
```

***ESPERA ENABLE.*** En aquest estat s'espera que arribi el senyal d'*enable* del bloc CAM DRIVER per començar a aplicar el filtre a la imatge capturada. Quan es rep el aquest senyal es defineix quin filtre s'ha d'aplicar depenent de l'estat dels interruptors i a continuació es passa a l'estat principal per tal d'executar el filtre escollit.

```
SI en = '1' ALESHORES
      READ_EN = '0'
      CAS INTERRUPTORS
            QUAN "00000000"
                  FILTRE = COLOR
                  ESTAT = PRINCIIPAL
            QUAN "00000001"
                  FILTRE = COLOR
                  ESTAT = PRINCIIPAL
            QUAN "00000011"
                  FILTRE = COLOR
                  ESTAT = PRINCIIPAL
            QUAN "00000010"
                  FILTRE = COLOR
                  ESTAT = PRINCIIPAL
            QUAN "00000100"
                  FILTRE = COLOR
                  ESTAT = PRINCIIPAL
            QUAN "00001000"
                  FILTRE = COLOR
                  ESTAT = PRINCIIPAL
            QUAN "00010000"
                  FILTRE = COLOR
                  ESTAT = PRINCIIPAL
```

```
                QUAN “00100000”
                      FILTRE = COLOR
                      ESTAT = PRINCIIPAL
                QUAN “01000000”
                      FILTRE = COLOR
                      ESTAT = PRINCIIPAL
                QUAN “10000000”
                      FILTRE = COLOR
                      ESTAT = PRINCIIPAL
                QUAN ALTRES
                      ESTAT = ESPERA_ENABLE
          FI CAS
          SINO
                ESTAT = ESPERA_ENABLE
    FI SI
```

Com s’ha pogut veure en la taula 4, tots els filtres es basen aplicant els mateixos algorismes, els quals són els algorismes morfològics d’erosió, dilatació i extracció de contorn i els algorismes per passar la imatge a blanc i negre (binarització) o a escala de grisos. En el següents apartats es detallen aquests algorismes juntament amb la màquina d’estats que segueixen.

Cal comentar que no es detalla en pseudocodi les màquines d’estat que segueixen els filtres d’obertura, tancament, obertura-tancament i contorn amb obertura-tancament ja que és la concatenació d’aplicar els algorismes comentats (binarització, erosió, dilatació i extracció de contorn) en l’ordre corresponent. En els annexos es troba disponible el codi en VHDL del bloc de processament on es poden seguir clarament els passos que segueix cada filtre.

### *3.2.1.* Blanc i negre / Escala de grisos

Aquests algorismes s’utilitzen per passar la imatge original en color a blanc i negre (binarització) o a escala de grisos. En el procés de binarització s’aconsegueix una nova imatge on els píxels només poden tenir dos colors (blanc o negre) i es a partir d’aquesta que es poden aplicar els algorismes d’erosió i dilatació. En el procés de transformació a escala de grisos s’aconsegueix una nova imatge on els píxels poden tenir fins a quatre colors (blanc, gris clar, gris fosc i negre). A continuació la màquina d’estats que segueix:

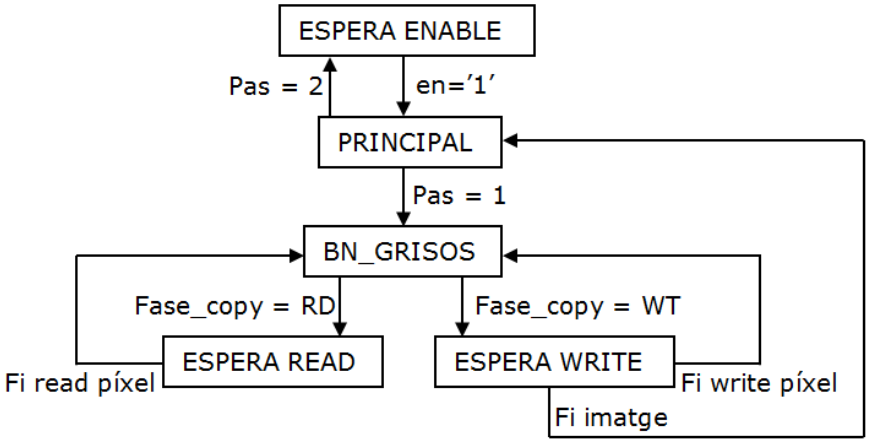


***Figura 44.*** *Màquina d'estats per al filtre BN/Grisos.*

***PRINCIPAL.*** En aquest estat hi ha definits els passos a seguir per a cada filtre. En els filtres de binarització i escala de grisos són els següents:

```
CAS FILTRE
      QUAN COLOR
            IMATGE = "000"
            ESTAT = ESPERA_ENABLE
            READ_EN = '1'
      QUAN BN
            CAS pas
                  QUAN 1
                        Centre = 0
                        ESTAT = BN_GRISOS
                        BN = '1'
                  QUAN ALTRES
                        Pas = 1
                        BN = '0'
                        ESTAT = ESPERA_ENABLE
                        READ_EN = '1'
                        IMATGE = "001"
            FI CAS
      QUAN GRISOS
            CAS pas
                  QUAN 1
                        Centre = 0
                        ESTAT = BN_GRISOS
                        BN = '1'
                  QUAN ALTRES
                        Pas = 1
                        BN = '0'
                        ESTAT = ESPERA_ENABLE
                        READ_EN = '1'
                        IMATGE = "001"
            FI CAS
```

```
...
FI CAS
```

Es pot observar que els passos que segueixen els dos filtres són iguals, ja que l'algorisme per binaritzar o passar a escala de grisos és el mateix amb l'única diferència de com es defineix el nou color dels píxels. També es pot veure que si s'ha escollit no aplicar cap filtre a la imatge (COLOR) directament s'envia la senyal de lectura al bloc VGA DRIVER amb la informació que s'ha de representar la imatge 0.

***BN_GRISOS.*** En aquest estat es genera un escombrat per tota la imatge píxel per píxel de tal manera que primer es llegeix el píxel corresponent de la imatge original en color, es transforma i finalment s'escriu la informació en les mateixes coordenades però de la imatge següent, és a dir, a la imatge 1.

```
SI FASE_COPY = RD ALESHORES
      ADRESS_RAM = centre
      READ_PROC = '1'
      ESTAT = ESPERA_READ_BNGRISOS
SINO
      WRITE_PROC = '1'
      DATA_OUT_BNAUX = COPY_DATA
      OE_inout = '1'
      ESTAT = ESPERA_WRITE_BNGRISOS
      ADRESS_RAM = centre + 307200
FI SI
```

Es pot veure que en el cas que s'hagi de llegir el píxel es passa a l'estat d'espera de lectura amb l'adreça igual al valor del senyal *centre*, el qual indica en quin píxel es troba l'escombrat. Quan s'escriuen les dades, es observar que s'escriu en l'adreça *centre+307200*, ja que s'ha de guardar en l'espai de la memòria destinat a la imatge següent.

***ESPERA_READ.*** En aquest estat s'espera a que la memòria RAM entregui les dades indicades. Quan ha acabat, es torna a l'estat BN_GRISOS per tal d'escriure-hi la informació que toqui.

```
READ_PROC = '0'
SI preescaler < 5 ALESHORES
      Preesaler = preescaler + 1
SINO
      Preescaler = 0
      COPY_DATA = DATA_IN
      ESTAT = BN_GRISOS
      FASE_COPY = WT
FI SI
```

***ESPERA_WRITE.*** En aquest estat s'espera a que la memòria RAM escrigui les dades indicades. Es pot observar que s'avalua si s'ha acabat l'escombrat i si és el cas es torna a l'estat principal passant al següent pas. En cas contrari és torna a l'estat BN_GRISOS amb el valor de *centre+1*.

```
READ_PROC = '0'
SI preescaler < 3 ALESHORES
      Preesaler = preescaler + 1
SINO
      Preescaler = 0
      OE_inout = '0'
      FASE_COPY = RD
      SI centre < 614399 ALESHORES
            Centre = centre + 1
            ESTAT = BN_GRISOS
      SINO
            ESTAT = PRINCIPAL
            Pas = pas + 1
      FI SI
FI SI
```

Es pot observar que per controlar el bus de dades connectat amb el bloc controlador de la RAM es treballa amb un *buffer* no inversor tri-estat controlat amb el senyal OE_inout, com en el bloc RAM DIRVER.

Com s'ha comentat anteriorment, per controlar el valor del senyal DATA_OUT s'utilitza un multiplexor ja que no es poden assignar valors a un mateix senyal des de diferents processos degut a que la conversió a blanc i negre o a escala de grisos es fa des de fora del procés principal. S'ha creat el següent circuit digital per tal d'evitar aquest conflicte:

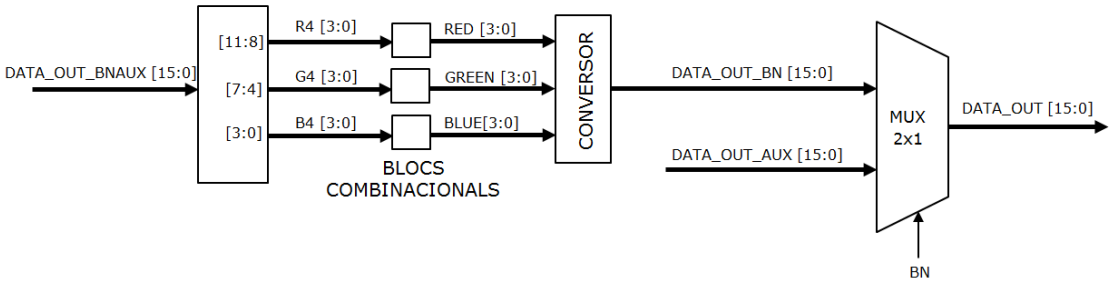


***Figura 45.*** *Circuit digital per controlar DATA_OUT.*

Des del procés principal es controla el valor del senyal DATA_OUT_AUX i l'estat lògic de BN per tal de controlar el multiplexor. En el moment que es vol fer la conversió a blanc i negre o a escala de grisos es treballa amb el senyal DATA_OUT_BNAUX per tal que la informació passi per un seguit de blocs combinacionals per tal que es produeixi la conversió. Se separa la informació de cada color i depenent de la seva intensitat es passa a blanc o negre seguint la taula següent:

***Taula 5.*** *Blocs combinacionals per la descodificació RGB444 a BN.*

| RGB444 | Red / Green/Blue |
|---|---|
| 0000 | 0000 |
| 0001 | 0000 |
| 0010 | 0000 |
| 0011 | 0000 |
| 0100 | 0000 |
| 0101 | 0000 |
| 0110 | 1111 |
| 0111 | 1111 |
| 1000 | 1111 |
| 1001 | 1111 |
| 1010 | 1111 |
| 1011 | 1111 |
| 1100 | 1111 |
| 1101 | 1111 |
| 1110 | 1111 |
| 1111 | 1111 |

Es pot observar que el canvi de blanc a negre no es fa a la meitat (“1000”) sinó a “0110” degut a que les imatges quedaven molt fosques.

Per decidir el color final del píxel la informació es processa en el bloc anomenat CONVERSOR, el qual segueix el següent procés:

```
PROCÉS (RED, GREEN, BLUE)
     SI FILTRE = GRISOS ALESHORES
          SI RED=“0000” I GREEN“0000” I BLUE=“0000”
          ALESHORES
               DATA_OUT_BN=“0000000000000000” --Negre
          SINO SI (RED=”0000” I GREEN=”0000”) O
          (RED=”0000” I BLUE=”0000”) O (GREEN=”0000” I
          BLUE=”0000”) ALESHORES
               DATA_OUT_BN=”0000010001000100”--Gris Fosc
               SINO SI RED=“111” I GREEN“111” I
               BLUE=“111”  ALESHORES
                    DATA_OUT_BN=”1111111111111111”--Blanc
               SINO
                    DATA_OUT_BN=”0000100010001000”--Gris
                                                  --Clar
          FI SI
     SINO --Filtre BN
          SI (RED=”0000” I GREEN=”0000”) O
          (RED=”0000” I BLUE=”0000”) O (GREEN=”0000” I
          BLUE=”0000”) ALESHORES
               DATA_OUT_BN=“0000000000000000” --Negre
          SINO
```

```
                    DATA_OUT_BN="1111111111111111"--Blanc
        FI SI
    FI PROCÉS
```

Es pot observar que es tracta d'un procés que té els senyals RED, GREEN i BLUE a la llista de sensibilitats. Quan un d'aquets tres senyals canvia, s'executa el codi, el qual avalua de quin filtre es tracta (Blanc i negre o escala de grisos) i defineix el color que tindrà el píxel. En el cas que sigui el filtre d'escala de grisos, si tots tres són "0000" el color del píxel serà negre i blanc en cas que siguin "1111". Si només dos són "0000" el color final serà gris fosc i serà gris clar en el cas que només dos siguin "1111". De la mateixa manera, pel filtre de blanc i negre el color final serà negre si mínim dos són "0000" i blanc en cas contrari.

### *3.2.2.* Erosió / Dilatació

Aquests algorismes són bàsics per dur a terme la resta de filtres, ja que són la combinació d'aquests dos. L'algorisme es regeix per la següent màquina d'estats:

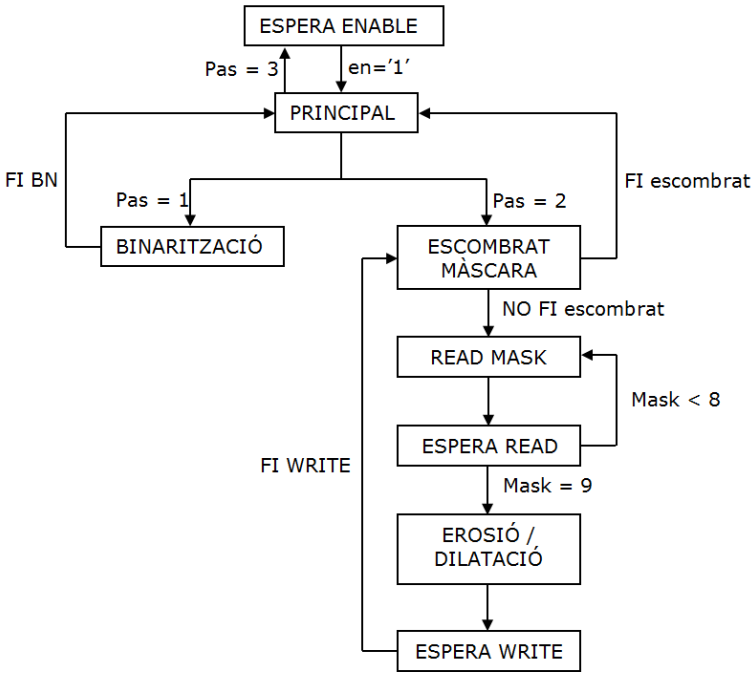


***Figura 46.*** *Màquina d'estats per erosió i dilatació.*

Com es pot veure, el primer pas que cal fer es binaritzar la imatge seguint l'algorisme de l'apartat anterior. Un cop la imatge està binaritzada es procedeix a aplicar l'algorisme d'erosió o dilatació.

***PRINCIPAL.*** En aquest estat hi ha definits els passos a seguir per a cada filtre. En els filtres d'erosió i dilatació són els següents:

```
CAS FILTRE
...
      QUAN EROSIÓ
            CAS pas
                  QUAN 1
                        Centre = 0
                        ESTAT = BN_GRISOS
                        BN = '1'
                  QUAN 2
                        BN = '0'
                        Centre = 307200
                        ESTAT = ESCOMBRAT_MÀSCARA
                        ALGORISME = EROSIO
                  QUAN ALTRES
                        Pas = 1
                        ESTAT = ESPERA_ENABLE
                        READ_EN = '1'
                        IMATGE = "010"
            FI CAS
      QUAN DILATACIÓ
            CAS pas
                  QUAN 1
                        Centre = 0
                        ESTAT = BN_GRISOS
                        BN = '1'
                  QUAN 2
                        BN = '0'
                        Centre = 307200
                        ESTAT = ESCOMBRAT_MASCARA
                        ALGORISME = DILATACIÓ
                  QUAN ALTRES
                        Pas = 1
                        ESTAT = ESPERA_ENABLE
                        READ_EN = '1'
                        IMATGE = "010"
            FI CAS
...
FI CAS
```

Es pot observar que els passos que segueixen els dos filtres són iguals amb la diferència que el senyal ALGORISME es defineix de manera diferent.

***ESCOMBRAT MÀSCARA.*** En aquest estat es genera i controla l'escombrat de la màscara de 3x3 píxels per la imatge.

```
SI column_counter=0 O columnt_counter=639 O row_counter=0 O
row_counter=479 ALESHORES
      DATA_OUT_AUX = "1111111111111111" --Blanc
      WRITE_PROC = '1'
```

```
        ADRESS_RAM = centre + 307200
        OE_inout = '1'
        ESTAT = ESPERA_WRITE_PROC
SINO
        ESTAT = READ_MASK
        Mask1 = centre - 641 --superior esquerre
        Mask2 = centre - 640 --superior central
        Mask3 = centre - 639 --superior dreta
        Mask4 = centre - 1 --central esquerre
        Mask6 = centre + 1 --central dreta
        Mask7 = centre + 639 --inferior esquerre
        Mask8 = centre + 640 --inferior central
        Mask9 = centre + 641 --inferior dreta
FI SI
SI column_counter < 639 ALESHORES
        Column_counter = columnt_counter + 1
SINO
        Column_counter = 0
        SI row_counter < 479 ALESHORES
                Row_counter = row_counter + 1
        SINO
                ESTAT = PRINCIPAL
                Pas = pas + 1
                WRITE_PROC = '0'
                OE_inout = '0'
                Preescaler = 0
                Row_counter = 0
                Column_counter = 0
        FI SI
FI SI
```

Com es pot veure, es comprova que l'escombrat no estigui en els marges de la imatge. Si és el cas, s'escriu el píxel en qüestió de color blanc. En cas contrari, es procedeix a definir les adreces que corresponen als 9 píxels de la màscara per tal de llegir-ne les dades. Finalment s'avalua per on passa l'escombrat per tal de saber quan s'ha acabat i tornar a l'estat principal.

***READ MASK.*** En aquest estat es llegeix la informació dels píxels que formen la màscara i es passa a l'estat ESPERA READ mentre el controlador RAM DRIVER envia les dades.

```
CAS MASK
        QUAN 1
                ADRESS_RAM = mask1
        QUAN 2
                ADRESS_RAM = mask2
        QUAN 3
                ADRESS_RAM = mask3
        QUAN 4
```

```
            ADRESS_RAM = mask4
      QUAN 5
            ADRESS_RAM = centre
      QUAN 6
            ADRESS_RAM = mask6
      QUAN 7
            ADRESS_RAM = mask7
      QUAN 8
            ADRESS_RAM = mask8
      QUAN 9
            ADRESS_RAM = mask9
FI CAS
READ_PROC = '1'
ESTAT = ESPERA_READ
```

***ESPERA READ.*** En aquest estat es guarda la informació de la màscara en els senyals DATA_mask$_n$. Quan es té tota la informació es passa a l'estat EROSIÓ/DILATACIÓ.

```
READ_PROC = '0'
SI preescaler < 5 ALESHORES
      Preescaler = preescaler + 1
SINO
      Preescaler = 0
      CAS MASK
            QUAN 1
                  DATA_mask1 = DATA_IN
                  ESTAT = READ_MASK
                  MASK = MASK + 1
            QUAN 2
                  DATA_mask2 = DATA_IN
                  ESTAT = READ_MASK
                  MASK = MASK + 1
            QUAN 3
                  DATA_mask3 = DATA_IN
                  ESTAT = READ_MASK
                  MASK = MASK + 1
            QUAN 4
                  DATA_mask4 = DATA_IN
                  ESTAT = READ_MASK
                  MASK = MASK + 1
            QUAN 5
                  DATA_centre = DATA_IN
                  ESTAT = READ_MASK
                  MASK = MASK + 1
            QUAN 6
                  DATA_mask6 = DATA_IN
                  ESTAT = READ_MASK
                  MASK = MASK + 1
            QUAN 7
```

```
                    DATA_mask7 = DATA_IN
                    ESTAT = READ_MASK
                    MASK = MASK + 1
              QUAN 8
                    DATA_mask8 = DATA_IN
                    ESTAT = READ_MASK
                    MASK = MASK + 1
              QUAN 9
                    DATA_mask9 = DATA_IN
                    ESTAT = EROSIÓ_DILATACIÓ
                    MASK = 1
        FI CAS
FI SI
```

***EROSIÓ_DILATACIÓ.*** En aquest estat s'avalua el color del píxel de la nova imatge que s'està creant a partir de la informació que aporta la màscara. Depenent del senyal ALGORISME s'aplicarà la tècnica d'erosió o la de dilatació.

```
SI ALGORISME = EROSIÓ ALESHORES
      SI TOTS ELS PÍXELS DE LA MÀSCARA SÓN NEGRES ALESHORES
            DATA_OUT_AUX = "0000000000000000" --Negre
      SINO
            DATA_OUT_AUX = "1111111111111111"--Blanc
SINO
      SI ALGUN PÍXEL DE LA MÀSCARA ÉS NEGRE ALESHORES
            DATA_OUT_AUX = "0000000000000000" --Negre
      SINO
            DATA_OUT_AUX = "1111111111111111"--Blanc
FI SI
WRITE_PROC = '1'
ADRESS_RAM = centre + 307200
OE_inout = '1'
ESTAT = ESPERA_WRITE
```

***ESPERA_WRITE.*** En aquest estat s'espera a que la informació s'escrigui a la memòria ram i es torna a l'esta de ESCOMBRAT_MÀSCARA per tal de seguir amb l'escombrat.

```
WRITE_PROC = '0'
SI preescaler < 3 ALESHORES
      Preescaler = preescaler + 1
SINO
      Preescaler = 0
      OE_inout = '0'
      ESTAT = ESCOMBRAT_MÀSCARA
      Centre = centre + 1
FI SI
```

### *3.2.3.* Extracció del contorn

Aquest algorisme s'utilitza per extreure el contorn dels objectes presents en la imatge. Prèviament, però, cal que s'hagi binaritzat la imatge original en color i s'hagi aplicat un algorisme d'erosió, tal i com s'ha vist en l'apartat 3.1.

La màquina d'estats que segueix aquest algorisme és la següent:

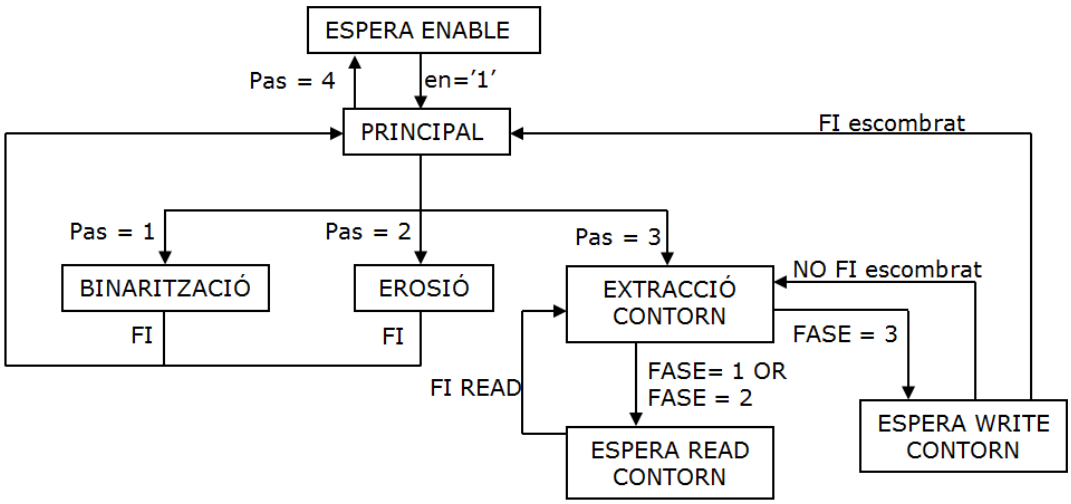


***Figura 47.*** *Màquina d'estats per l'extracció de contorn.*

Com es pot veure, el primer pas que cal fer es binaritzar la imatge i tot setguit realitzar la tècnica d'erosió seguint l'algorisme de l'apartat anterior. Un cop la imatge està binaritzada i erosionada es procedeix a aplicar l'algorisme d'extracció de contorn.

***PRINCIPAL.*** En aquest estat hi ha definits els passos a seguir per a cada filtre. En el filtre d'extracció de contorn són els següents:

```
CAS FILTRE
...
      QUAN EXTRACCIÓ_CONTORN
            CAS pas
                  QUAN 1
                        Centre = 0
                        ESTAT = BN_GRISOS
                        BN = '1'
                  QUAN 2
                        BN = '0'
                        Centre = 307200
                        ESTAT = ESCOMBRAT_MÀSCARA
                        ALGORISME = EROSIO
                  QUAN 3
                        Counter_contorn  = 0
                        Centre = 307200
                        ESTAT = EXTRACCIÓ_CONTORN
```

```
                    QUAN ALTRES
                          Pas = 1
                          ESTAT = ESPERA_ENABLE
                          READ_EN = '1'
                          IMATGE = "010"
              FI CAS
...
FI CAS
```

***EXTRACCIÓ CONTORN.*** En aquest estat es controla l'algorisme per l'extracció de contorn mitjançant tres fases. En la fase 1 es llegeix la informació dels píxels de la imatge binaritzada, en la fase 2 es llegeix la informació de la imatge erosionada a partir de la binaritzada i en la fase 3 es comparen els píxels per tal de determinar el color del nou píxel. En el cas que siguin iguals (els dos blancs o els dos negres) el color del nou píxel serà blanc i en cas contrari serà negre.

```
CAS FASE_CONTORN
      QUAN 1
            ADRESS_RAM = centre
            READ_PROC = '1'
            ESTAT = ESPERA_READ_CONTORN
      QUAN 2
            ADRESS_RAM = centre + 307200
            READ_PROC = '1'
            ESTAT = ESPERA_READ_CONTORN
      QUAN 3
            SI (píxel_original=0xFFFF I píxel_erosió=0xFFFF)
            O (píxel_original=0x0000 I píxel_erosió=0x0000)
            ALESHORES
                  DATA_OUT_AUX = "1111111111111111" --Blanc
            SINO
                  DATA_OUT_AUX = "0000000000000000" --Negre
            FI SI
            ADRESS_RAM = centre + 307200
            OE_inout = '1'
            WRITE_PROC = '1'
            ESTAT = ESPERA_WRITE_CONTORN
FI CAS
```

***ESPERA READ CONTORN.*** En aquest estat es llegeixen els píxels de la imatge original binaritzada i de la imatge erosionada. Quan s'ha llegit i guardat la informació, es torna a l'estat EXTRACCIÓ CONTORN.

```
READ_PROC = '0'
SI preescaler < 5 ALESHORES
      Preescaler = preescaler + 1
SINO
      Preescaler = 0
      CAS FASE_CONTORN
            QUAN 1
```

```
                    Píxel_original = DATA_IN
                    ESTAT = EXTRACCIÓ_CONTORN
                    FASE_CONTORN = FASE_CONTORN + 1
               QUAN ALTRES
                    Píxel_erosió = DATA_IN
                    ESTAT = EXTRACCIÓ_CONTORN
                    FASE_CONTORN = FASE_CONTORN + 1
          FI CAS
FI SI
```

***ESPERA WRITE CONTORN.*** En aquest estat s'escriu el resultat de l'algorisme per l'extracció de contorn. També s'avalua si s'ha acabat l'escombrat per tota la imatge utilitzant el senyal *counter_contorn.* Cal destacar que el resultat s'escriu sobre la imatge erosionada.

```
WRITE_PROC = '0'
FASE_CONTORN = 1
SI preescaler < 3 ALESHORES
     Preescaler = preescaler + 1
SINO
     Preescaler = 0
     OE_inout = '0'
     SI counter_contorn<307199 ALESHORES
          Centre = centre + 1
          Counter_contorn = counter_contorn + 1
          ESTAT = EXTRACCIÓ_CONTORN
     SINO
          ESTAT = PRINCIPAL
          Pas = pas + 1
     FI SI
```

### *3.2.4.* Temps de processament

El temps de processament és diferent per a cada filtre que s'aplica, ja que com s'ha vist, cada un necessita ocupar diferents posicions de memòria i aplicar diferents algorismes

El temps de processament per presentar la imatge original en color es nul, tot i que s'ha de tenir en compte el temps que es tarda a capturar la imatge. Seguint el diagrama que proporciona el fabricant i que s'ha pogut veure en la figura 7 (pàgina 17), es calcula el temps de captura:

$$t_c = (510\text{-}13)t_L = 497 \cdot 784 t_P = 497 \cdot 784 \cdot 83{,}33 \text{ ns} = 32{,}74 \text{ ms} \quad (6)$$

El temps de processament pels algorismes de binarització i escala de grisos es pot aproximar de la següent manera, tenint en compte que es necessita llegir i escriure cada píxel de la imatge, és a dir, un total de 307200 píxels:

$$t_{BN} = 70 \text{ ns} \cdot 2 \cdot 307200 = 43 \text{ ms} \quad (7)$$

El temps de processament pels algorismes d'erosió i dilatació es pot aproximar de la següent manera, tenint en compte que es necessiten llegir els 9 píxels de la màscara per a cada píxel de l'escombrat i es necessiten escriure 307200 píxels per formar la nova imatge:

$$T_{A\ E/D} = 9 \cdot 70\ ns \cdot 307200 + 70\ ns \cdot 307200 = 215{,}04\ ms \quad (8)$$

El temps de processament pels filtres d'erosió i dilació contempla el temps en aplicar l'algorisme de binarització ($t_{BN}$) i el temps d'aplicar l'algorisme d'erosió o dilatació ($t_{A\ E/D}$):

$$T_{T\ E/D} = 43\ ms + 215{,}04\ ms = 258{,}04\ ms \quad (9)$$

El temps de processament pels filtres d'obertura i tancament tenen en compte el temps de binarització ($t_{BN}$) i el temps d'aplicar dos algorismes d'erosió i dilatació ($t_{A\ E/D}$):

$$T_{O/T} = 43\ ms + 2 \cdot 215{,}04\ ms = 473{,}08\ ms \quad (10)$$

En canvi, per calcular el temps del filtre obertura-tancament s'ha de tenir en compte que s'aplica quatre vegades l'algorisme d'erosió/dilatació:

$$T_{OT} = 43\ ms + 4 \cdot 215{,}04 = 903{,}16\ ms \quad (11)$$

El filtre d'extracció de contorn simple només utilitza l'algorisme de binarització, l'algorisme d'erosió i l'algorisme d'extracció de contorn, el qual ha de llegir dos píxels per a cada píxel que s'escriu. El temps de processament aproximat és el següent:

$$t_{CONT} = 43\ ms + 215{,}04\ ms + 3 \cdot 70\ ns \cdot 307200 = 322{,}56\ ms \quad (12)$$

Finalment, el filtre d'extracció de contorn amb OT es pot aproximar de la següent manera:

$$T_{CONT\ OT} = 43\ ms + 5 \cdot 215{,}04\ ms + 3 \cdot 70\ ns \cdot 307200 = 1{,}18\ s \quad (13)$$

En la taula següent es pot observar de forma resumida el temps de processament i el temps total (processament + captura) de cada filtre:

***Taula 6.*** *Resum de temps de processament i temps totals.*

| Filtre | Temps de processament | Temps total |
|---|---|---|
| Color | 0 ms | 32,47 ms |
| Blanc i negre | 43 ms | 75,47 ms |
| Grisos | 43 ms | 75,47 ms |
| Erosió | 258,04 ms | 290,51 ms |
| Dilatació | 258,04 ms | 290,51 ms |
| Obertura | 473,08 ms | 505,55 ms |
| Tancament | 473,08 ms | 505,55 ms |
| Ober-tanc | 903,16 ms | 936,63 ms |
| Contorn | 322,56 ms | 355,03 ms |
| Contorn amb O-T | 1,18 s | 1,21 s |

# CAPÍTOL 4: SIMULACIONS I RESULTATS PRÀCTICS

En aquest capítol s'exposen les simulacions realitzades i s'expliquen els resultats obtinguts a partir dels blocs descrits en VHDL, detallats en els capítols 2 i 3.

## 4.1. Simulacions

Per realitzar les simulacions s'ha utilitzat l'eina que proporciona l'entorn ISE de Xilinx anomenat ISim. A continuació es mostren les simulacions realitzades en els diferents blocs dissenyats per tal de comprovar el seu correcte funcionament.

### *4.1.1.* CAM DRIVER

Primerament es mostren les simulacions realitzades sobre el bloc CAM DRIVER. Com s'ha comentat anteriorment, aquest bloc està format per tres sub-blocs, a partir dels quals s'han realitzat les simulacions. La primera simulació s'ha realitzat sobre el sub-bloc CAM PROCESSOR, com es pot veure a continuació:

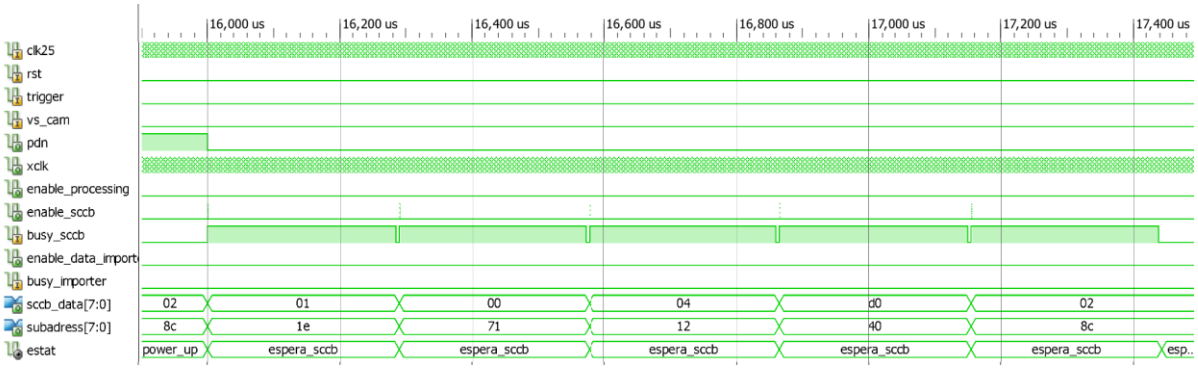


***Figura 48.*** *Simulació de CAM PROCESSOR.*

En la figura superior es pot veure que just després del període de *power up* de la càmera amb el senyal *Power Down* en estat alt, es procedeix a activar el bloc SCCB MODULE per tal de programar correctament la càmera posant el senyal *enable_sccb* en estat alt. Es pot observar que el bloc respon enviant el senyal de *busy* a '1' fins que ha acabat d'enviar les dades. El procés es repeteix tantes vegades com registres es vulgui modificar.

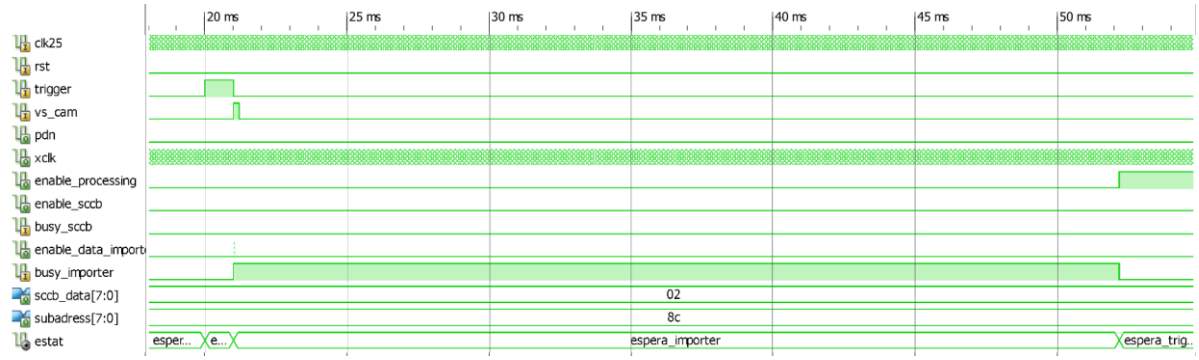


***Figura 49.*** *Simulació de CAM PROCESSOR.*

En la figura 49 es pot observar com es rep el senyal de *trigger* indicant que es vol capturar una imatge i el bloc entra a l'estat on espera la sincronització vertical enviada des de la càmera. Quan aquest senyal es posa en estat alt, activa el bloc CAM DATA IMPORTER per tal de guardar la informació a la memòria RAM. Es pot veure que el senyal de *busy* es manté a '1' fins que s'ha acabat la captura i just després s'activa el senyal per avisar el bloc PROCESSING IMAGE que ja es pot processar la imatge.

A continuació es mostra la simulació del bloc SCCB MODULE:

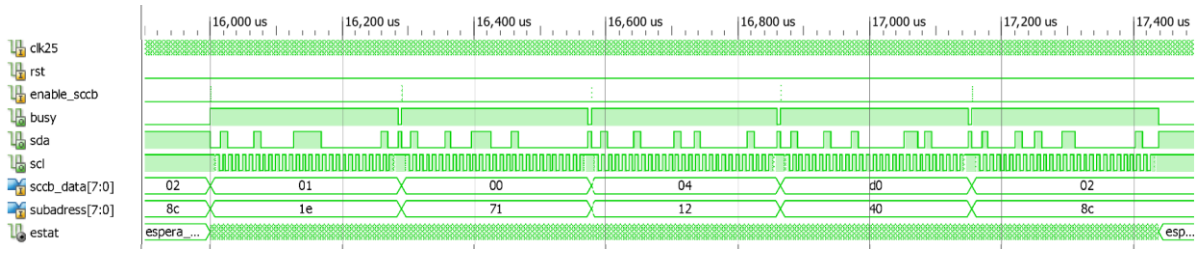


***Figura 50.*** *Simulació de SCCB MODULE.*

En la figura superior es pot apreciar que quan el bloc SCCB MODULE rep el senyal d'*enable,* envia la informació seguint el protocol SCCB explicat en el capítol 2 i manté el senyal de *busy* en estat alt mentre està ocupat.

El tercer sub-bloc a simular és el CAM DATA IMPORTER:

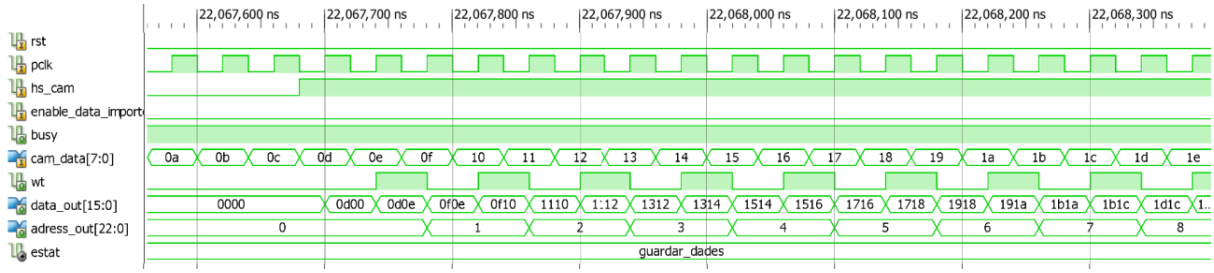


***Figura 51.*** *Simulació de CAM DATA IMPORTER.*

Com es pot veure, quan el senyal de sincronisme horitzontal es posa en estat alt, es procedeix a guardar la informació que arriba per CAM DATA, també simulada. Quan s'han llegit i guardat dos bytes seguits en el senyal DATA OUT s'activa el senyal WT per tal d'indicar al bloc RAM DRIVER que es volen guardar les dades. També es pot veure com s'incrementa el valor de l'adreça a mesura que es va guardant la informació dels píxels.

### *4.1.2.* RAM DRIVER

En la figura que es mostra a continuació es pot observar com es generen els senyals necessaris per tal de guardar les dades que s'envien des de CAM DRIVER. Com s'ha comentat anteriorment, aquestes dades es guarden de manera asíncrona i, per tant, només cal jugar amb el senyals WE, ADV i CE.

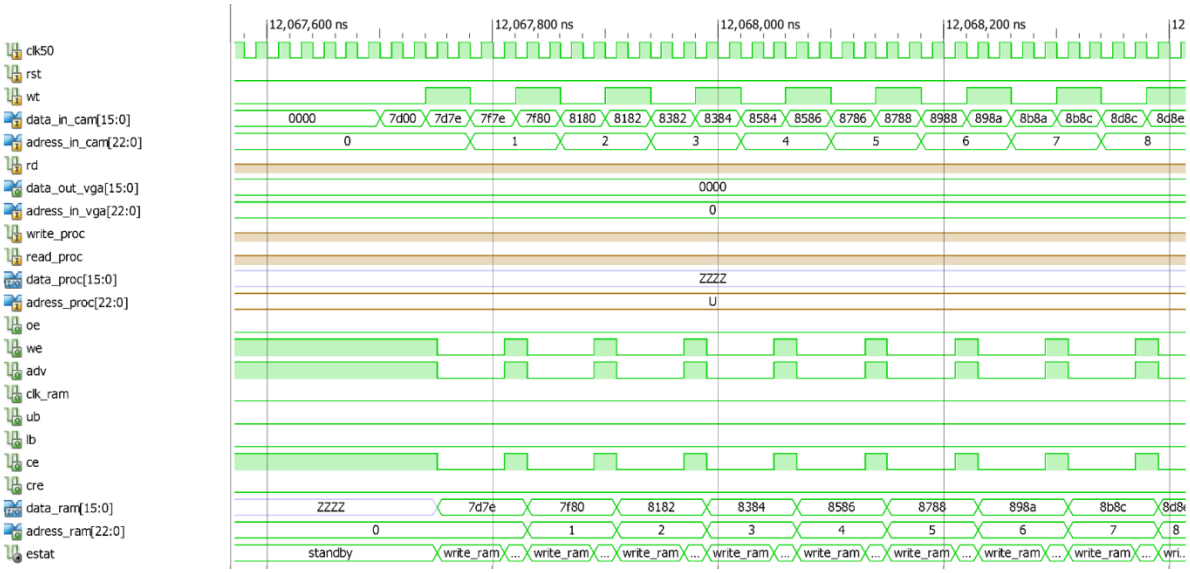


***Figura 52.*** *Simulació de les operacions d'escriptura asíncrona.*

En canvi, les operacions de lectura pel bloc VGA DRIVER són de manera síncrona tal i com s'ha explicat anteriorment. Es pot observar en la figura

53 com es manipulen els senyals ADV, CLK_RAM i CE per tal de dur a terme la lectura en mode *burst* de 8 *words.*

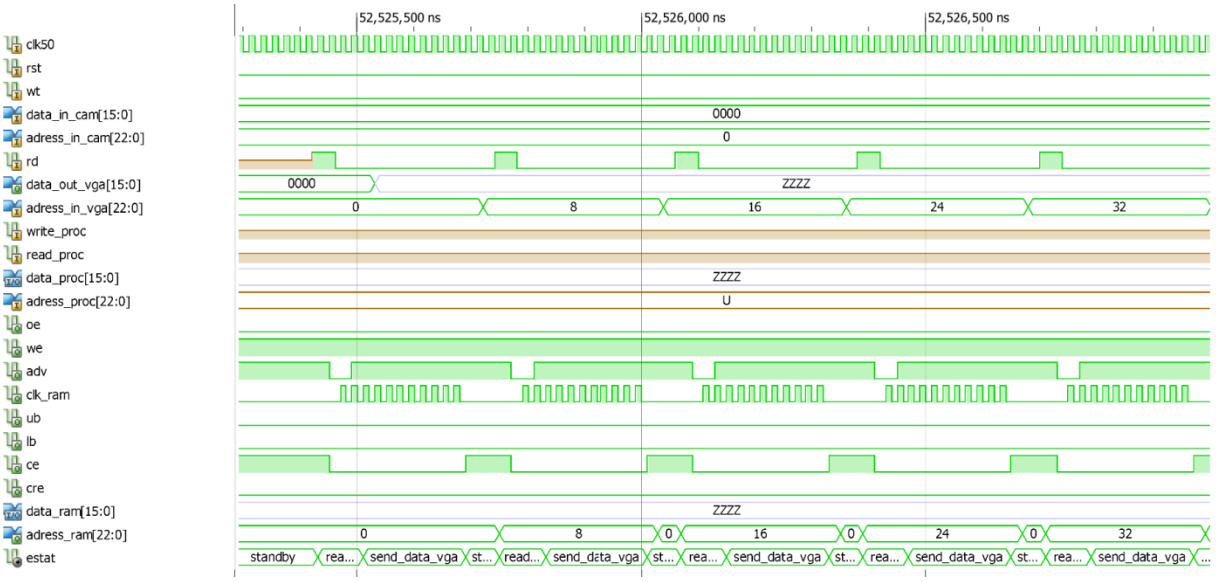


***Figura 53.*** *Simulació de les operacions de lectura en mode burst.*

Cal comentar que les operacions de lectura i escriptura pel bloc de processament d'imatge són en mode asíncron i per tant són iguals que les operacions mostrades en la figura 52.

### *4.1.3.* VGA DRIVER

En la primera captura de la simulació es mostra com treballen els senyals de sincronització vertical i horitzontal. Es pot observar que tant els comparadors com els comptadors funcionen correctament.

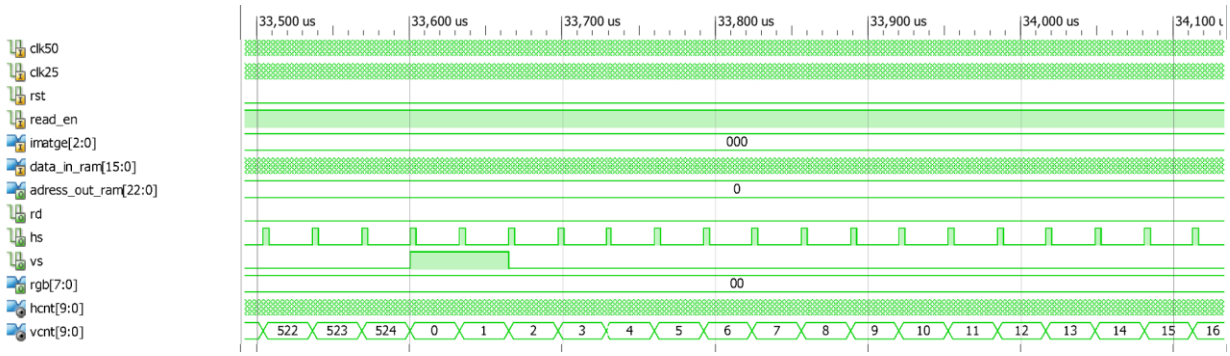


***Figura 54.*** *Simulació de VGA DRIVER.*

En la següent simulació es mostra el funcionament intern del bloc RAM DATA IMPORTER:

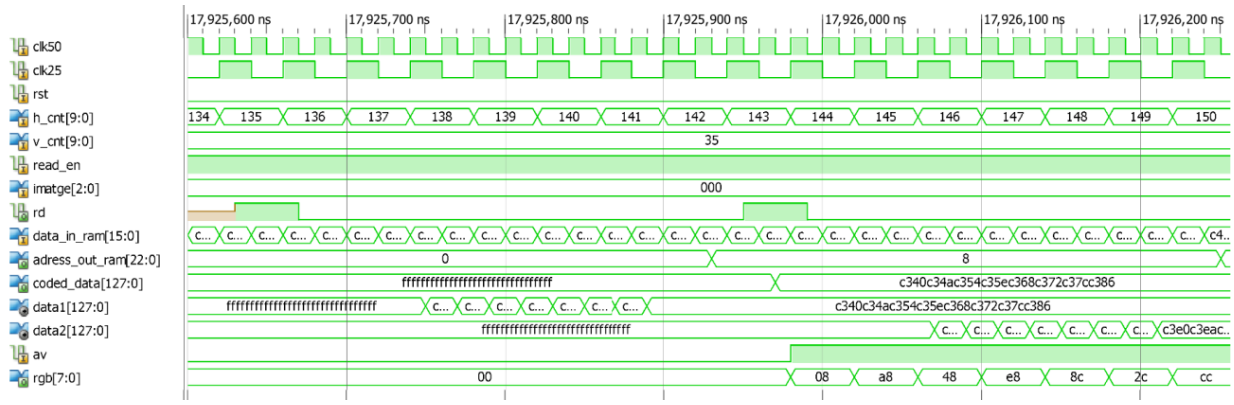


***Figura 55.*** *Simulació de RAM DATA IMPORTER.*

Es pot observar que quan s'arriba al píxel 135, s'activa el senyal de lectura RD per tal d'obtenir la informació del primer paquet de 8 píxels, ja que a partir del píxel 144 s'entra en zona vàlida tal i com es pot veure amb el senyal AV. Es pot veure que el primer paquet es guarda en el senyal *data1* i el segon en el senyal *data2,* i així successivament. També cal observar que s'envia la informació cap al bloc descodificador mitjançant el senyal *CODED_DATA,* que s'actualitza just en el moment anterior d'entrar en un nou grup de 8 píxels. Finalment, es pot veure que el senyal RGB canvia a una freqüència igual a la del rellotge de píxel, és a dir, a 25 MHz.

Per últim, en la següent figura hi ha la simulació del bloc descodificador:

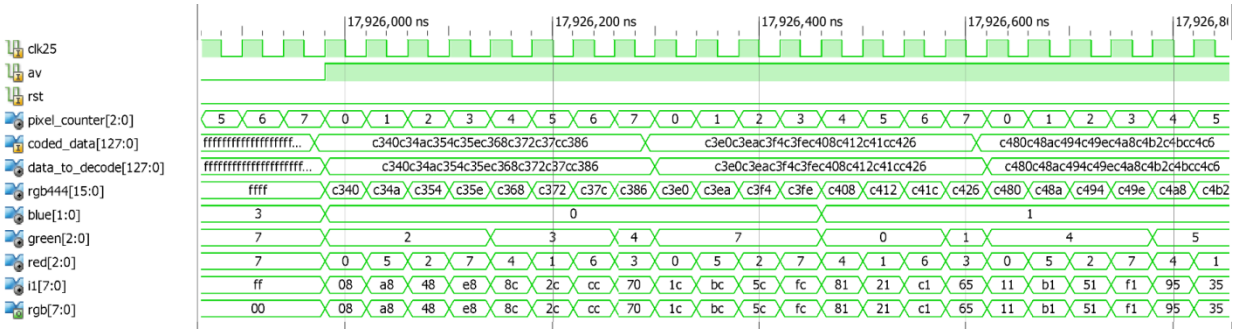


***Figura 56.*** *Simulació de DATA DECODER.*

Es pot observar que quan s'entra en zona vàlida, indicat pel senyal AV, el comptador es reinicia a zero i s'agafa la informació del primer píxel, la qual es descodificada i finalment enviada a través del port VGA. Es pot veure que la informació descodificada canvia cada 40 ns, és a dir, a una freqüència de 25 MHz. Quan s'ha descodificat la informació d'un paquet de 8 píxels, el comptador es reinicia i es rep un nou paquet d'informació.

## 4.2. Resultats pràctics

El primer objectiu del treball era obtenir una imatge en color i representar-la a la pantalla sense ser processada. Gràcies a l'opció de *test pattern* que incorpora la càmera OV7675, es poden obtenir unes barres de diferents colors per tal de saber si s'està tractant be la informació rebuda. Així doncs,

es va programar la càmera per treballar en aquest mode de test i es va obtenir la següent imatge i com es pot veure apareixen vuit barres amb els colors correctes:

***Figura 57.*** *Test pattern obtingut.*

Un cop realitzades les proves de test per tal de comprovar si s'està tractant bé la informació, es va procedir a capturar una imatge real.

***Figura 58a.*** *Imatge original.*

***Figura 58b.*** *Imatge capturada.*

Com es pot observa en la figura 58b, a la pantalla hi apareix la captura de la imatge original de la figura 58a. Es pot apreciar que la imatge capturada no té molta resolució, degut a que es treballa amb una resolució VGA de 640x480 píxels. A més a més, el port VGA que incorpora la placa *Nexys2* té una resolució de 8 bits per definir el color, com s'ha comentat anteriorment, i això fa que el canvi d'un color a un altre sigui més brusc que si es fes amb una resolució de més bits. Cal recordar que amb una resolució de 8 bits només es poden obtenir fins a 256 colors.

A partir d'aquí s'han aplicat els filtres detallats en el capítol 3 sobre les imatges capturades. A continuació es comenta el resultat de la imatge final després d'aplicar-hi els diferents filtres.

La captura de la imatge original és la que es mostra en la figura 59, que com s'ha comentat no té molt bona resolució i hi ha canvis de color bruscs. Es pot veure que es tracta d'unes tisores i d'unes alicates sobre un fons blanc.

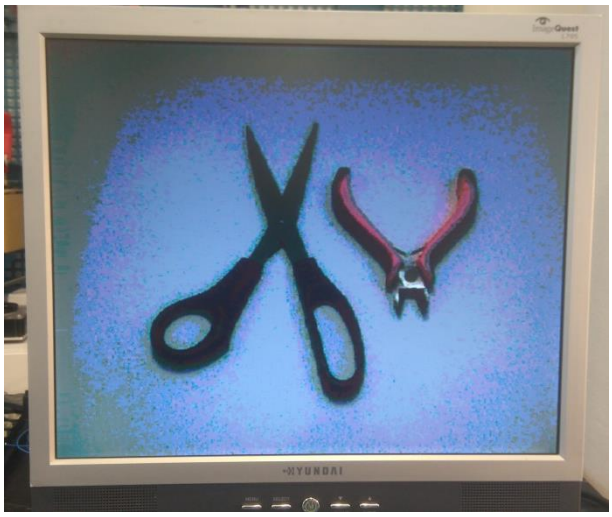

***Figura 59.*** *Imatge original sense aplicar cap filtre.*

El primer filtre que se li aplica és el de passar la imatge en escala de grisos. Com es pot veure en la següent figura, les tisores es queden de color negre degut a que son molt fosques però es pot observar que el color vermell de les alicates es torna de color gris clar i fosc. També es pot observar que hi ha bastant de soroll al voltant dels dos objectes.

***Figura 60.*** *Resultat després d'aplicar el filtre d'escala de grisos.*

Un filtre molt important de cara a aplicar els següents és el de binarització, és a dir, passar la imatge capturada a blanc i negre. Es pot observar que els dos objectes queden majoritàriament de color negre mentre que el fons queda de color blanc malgrat el soroll que hi pugui haver. Es pot veure que els contorns dels objectes no queden molt ben definits degut a possibles ombres que hi podia haver en la imatge original.

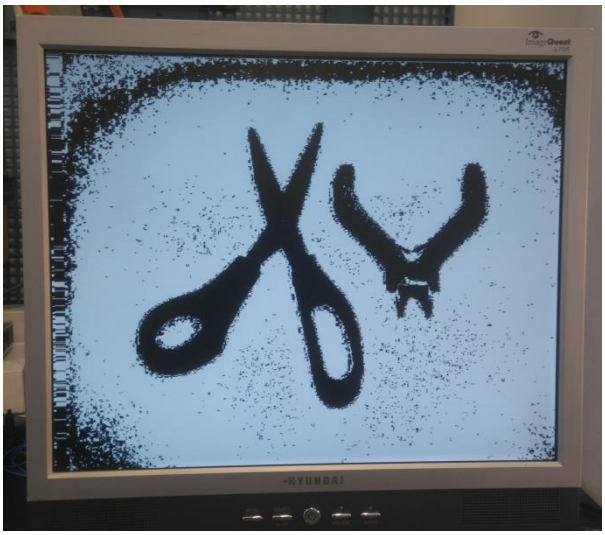

***Figura 61.*** *Resultat després d'aplicar la binarització.*

Un cop la imatge ha estat binaritzada, es poden aplicar la resta de filtres. El primer que s'aplica és el filtre d'erosió, com es pot veure en la figura 62. Es pot observar que el soroll que hi havia al voltant dels objectes desapareix a causa que el que fa la erosió és "menjar-se" els píxels negres seguint l'algorisme explicat en el capítol 3. S'observa que els objectes es fan més prims.

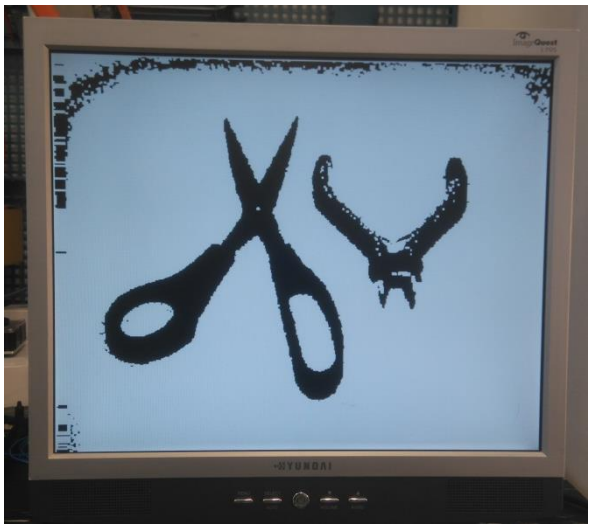

***Figura 62.*** *Resultat després d'aplicar el filtre d'erosió.*

A diferència del filtre d'erosió, el filtre de dilatació el que fa es "fer créixer" el negre, com es pot veure en la següent figura. S'observa que el soroll al

voltant dels objectes es fa molt més gros i abundant. També es pot veure que els propis objectes es fan més gruixuts.

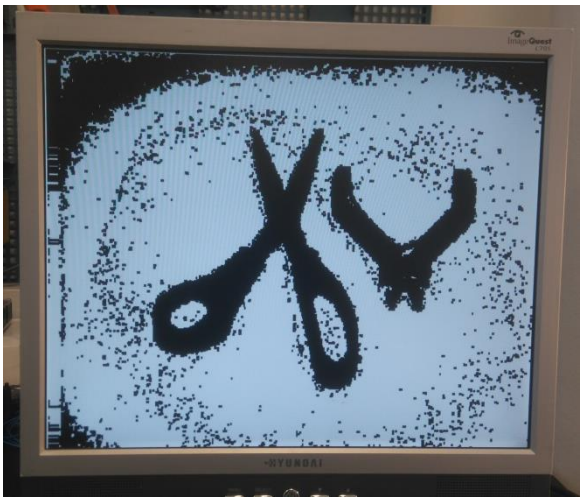

***Figura 63.*** *Resultat després d'aplicar el filtre de dilatació.*

Utilitzant aquests dos filtres es poden aplicar els filtres d'obertura i tancament. En la següent figura es pot observar com s'ha aplicat el filtre d'obertura. Les característiques d'aquest filtre són que suavitza els contorns eliminant protuberàncies i trenca petits istmes entre cossos més grossos fent que el color negre se separi i, com es pot observar, ha desaparegut el soroll al voltant dels objectes i s'ha suavitzat el contorn dels mateixos.

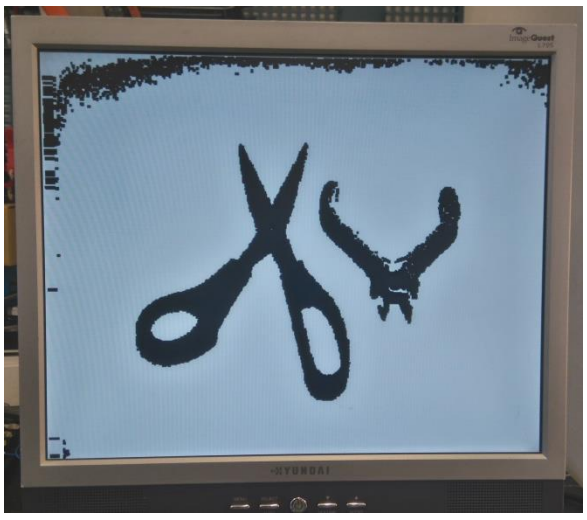

***Figura 64.*** *Resultat després d'aplicar el filtre d'obertura.*

En canvi, el filtre de tancament el que fa és suavitzar els contorns eliminant petits forats o "golfs" que pugui haver-hi i també elimina petits forats dintre dels objectes fent que el color negre s'ajunti. En la figura inferior es pot observar el resultat d'aplicar aquest filtre, on el soroll al voltant dels objectes no hagi desaparegut, sinó que s'hagi intentat ajuntar.

***Figura 65.*** *Resultat després d'aplicar el filtre de tancament.*

Per obtenir una imatge perfectament preparada per l'extracció del contorn cal aplicar un filtre d'obertura seguit del filtre de tancament. D'aquesta manera es deixa el contorn el més suavitzat possible i s'aconsegueix eliminar el soroll que hi pugui haver en el fons. Com es pot veure en la següent figura, els dos objectes estant perfectament definits i no hi ha soroll al voltant dels objectes.

***Figura 66.*** *Resultat després d'aplicar el filtre d'obertura i tancament.*

Un cop s'han aplicat aquests dos filtres es procedeix a extreure el contorn dels objectes presents. En la figura 67a es pot observar el resultat de l'extracció sobre la imatge prèviament preparada, on es pot veure que el contorn queda perfectament definit i sense soroll al voltant. En la figura 67b es pot veure com s'ha aplicat el filtre d'extracció de contorn sense haver preparat la imatge prèviament. S'observa que queda molt soroll al voltant i que els contorns no queden gaire ben definits.

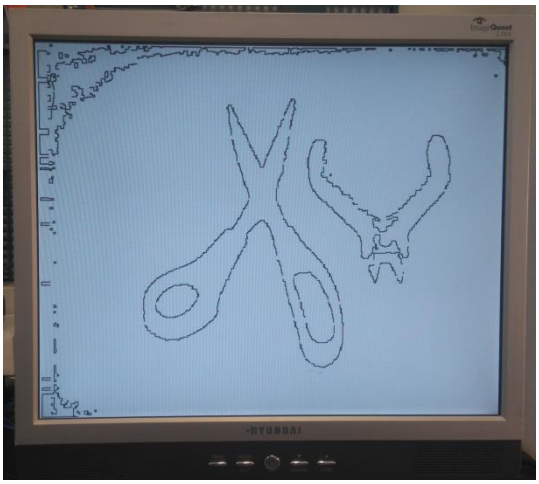

***Figura 67a.*** *Extracció amb O-T.*

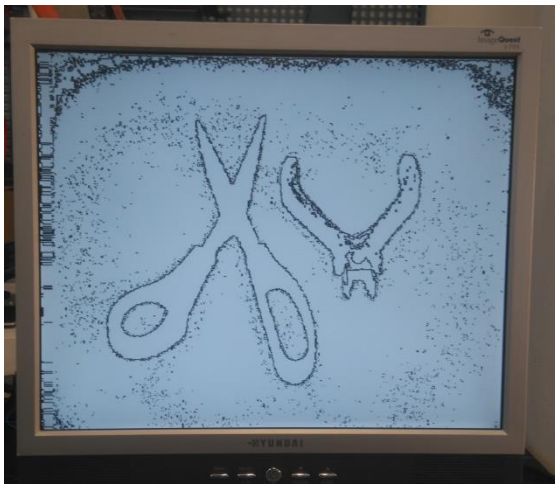

***Figura 67b.*** *Extracció sense O-T.*

# CAPÍTOL 5: CONCLUSIONS I TREBALL FUTUR

## 5.1. Conclusions

En aquest projecte s'ha dissenyat i implementat en una FPGA un sistema capaç de capturar una imatge, processar-la i finalment representar-la en una pantalla. Els objectius inicials plantejats s'han complert, ja que:

- S'ha aconseguit obtenir un sistema plenament funcional que permet capturar, processar i presentar imatges.

- Els blocs dissenyats funcionen correctament ja que es pot programar la càmera i adquirir les dades que envia, es gestiona correctament la informació guardada en la memòria RAM externa, es generen els senyals VGA per tal de representar la imatge i finalment, s'ha aconseguit realitzar el processat de les imatges sense l'ajuda externa d'un ordinador, fent que es puguin aplicar fins a nou filtres diferents.

El prototip però, no es comporta com una càmera intel·ligent en el sentit estricte de la paraula. Una càmera intel·ligent és un dispositiu capaç d'analitzar una imatge i, segons com estigui programada, genera diferents senyals de sortida per tal d'interactuar amb el seu entorn. Es programen utilitzant softwares especialitzats que treballen directament sobre la imatge i que disposen d'una sèrie d'eines per tal que el sistema de visió es pugui implementar en diferents aplicacions industrials. En canvi, utilitzant el VHDL i treballant a tant baix nivell és molt complicat poder realitzar un programa de visió artificial que funcioni correctament. A més a més, és necessari que la càmera tingui una bona resolució i, sobretot, es necessita una bona

il·luminació que moltes vegades també està controlada per la càmera. Així doncs, s'ha aconseguit un prototip capaç de capturar, aplicar diferents filtres i representar una imatge però no és exactament una càmera intel·ligent.

## 5.2. Treball futur

El prototip dissenyat presenta alguns inconvenients en quan a la comoditat per treballar-hi. Això es degut a que es treballa sobre una placa de proves, la qual té més elements dels que realment es necessiten. Durant les proves realitzades sobre el prototip actual es va intentar connectar la càmera amb el mòdul d'expansió utilitzant una cinta plana per tal de millorar la comoditat, però va resultar ser una mala idea ja que el soroll introduït per aquesta no permetia que les dades es transmetessin correctament, tal i com es pot veure en la figura 68. Per tant, es va decidir prescindir de la cinta plana i connectar directament la càmera al mòdul d'expansió, fent que capturar una imatge sigui més incòmode.

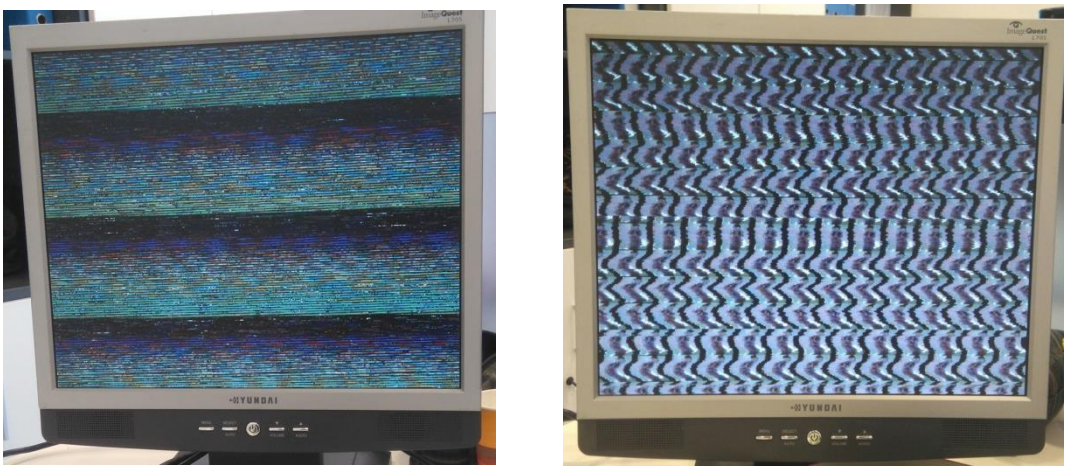

***Figura 68.*** *Imatges mal capturades.*

Un altre problema a destacar és el temps de processament en aplicar els filtres, com l'obertura-tancament i l'extracció de contorn amb obertura-tancament, que tarden 903,16 ms i 1,18 s respectivament. Aquests dos filtres tenen la característica que necessiten ocupar 6 i 7 posicions de memòria respectivament per aplicar tots els algorismes i això fa que la imatge final que s'ha de representar tardi més temps en aparèixer comparat amb resta de filtres, que semblen més "immediats".

Crec que el següent pas per millorar aquest projecte seria dissenyar una placa PCB especial per a aquesta aplicació, amb els components exclusivament necessaris pel seu bon funcionament i s'hauria de pensar una manera més còmoda i més ràpida per capturar i processar les dades. El problema del processament es podria solucionar incrementant la freqüència de treball, tot i que el que realment limita el prototip és el temps mínim d'accés en una operació asíncrona de la memòria RAM (70 ns). Així doncs, s'hauria de pensar en una memòria RAM d'accés mínim més petit o canviar

l'algorisme que segueix el bloc de processament per un altre molt més òptim, tenint en compte que es treballa sobre una FPGA i que es poden fer diferents processos en paral·lel. També s'hauria de trobar una manera per tal que la càmera es pugi connectar a la placa mitjançant una cinta plana o algun altre tipus de cable, de tal manera que no introdueixi soroll i que permeti tenir separat la càmera i la placa de processament per tal de treballar més còmodament.

# CAPÍTOL 6:

# BIBLIOGRAFIA

## 6.1. Referències bibliogràfiques

Brown, S. and Vranesic, Z. 2000. Fundamentals of Digital Logic with VHDL design, McGraw Hill, third edition. Department of Electrical and Computer Engineering. University of Toronto.

## 6.2. Bibliografia de Consulta

Arboleda, J.P. Apunts de l'assignatura Automatització i Robòtica Industrial.

Brown, S. and Vranesic, Z. 2000. Fundamentals of Digital Logic with VHDL design, McGraw Hill, third edition. Department of Electrical and Computer Engineering. University of Toronto.

Cosp, J. Apunts de l'assignatura Electrònica Digital i Microprocessadors.

Digilent Inc. 2011. Digilent Nexys2 Board - Reference manual. http://store.digilentinc.com/ (visitada el 20 de desembre de 2015).

Digilent Inc. 2015. VGA Display Controller. https://learn.digilentinc.com (visitada el 21 de gener de 2016).

Gàmiz, J. Apunts de l'assignatura Disseny de Sistemes pel Control de Processos.

Micron Technology, Inc. 2004. Async/Page/Burst CellularRAM 1.5. Product Specification. https://www.micron.com (visitada el 19 de desembre de 2015).

OmniVision Technologies Inc. 2007. OmniVision Serial Camera Control Bus (SCCB) - Functional Specification. http://www.ovt.com/ (visitada el 19 de desembre de 2015).

OmniVision Technologies Inc. 2009. OV7675 Product Specification. http://www.ovt.com/ (visitada el 19 de desembre de 2015).

Rafael C. Gonzalez and Richard E. Woods. 2002. Digital Image Processing, Prentice Hall, second Edition.

Shieber, A. and Vainter, Y. 2013. Connecting Kinetis MCU with CMOS Sensor Interface through GPIO. QQVGA imatge trasnfer to Kinetis internal SRAM. Freescale Semiconductor, Inc. http://www.nxp.com/ (visitada el 14 de febrer de 2016).