EYEBORG 2.0

Matias Lizana García

Director: Enric Xavier Martin i Rull

Gener 2011

Agraïments

L'agraïment és per a totes aquelles persones que, a la seva manera, han contribuït a fer aquest projecte, i m'han ajudat a tirar-lo endavant:

Al meu tutor Enric, que m'ha donat les guies necessàries per avançar, i aquests entretinguts dijous a la tarda comentant el projecte.

Al Manel, per donar-me l'oportunitat que em va donar i interessar-se per tot el que faig.

Al Frede, per ensenyar-me tantes coses, i haver passat tant bones estones programant i aprenent.

Als meus pares, que m'han donat suport a casa, i que em desitgen sempre el millor.

I a tu Marta, per tenir-te sempre al meu costat.

Índex

1 Plantejament del projecte

Per començar, es situa el projecte en el seu context. Neil Harbisson i la malaltia d'acromatòpsia, són els termes principals que desencadenen el desenvolupament d'aquest projecte.

1.1 Acromatòpsia

Categoritzada com a "malaltia estranya", l'acromatòpsia o monocromatisme és la incapacitat de percebre els colors. És un problema de la visió, més concretament una retinopatia perifèrica bilateral. És una malaltia estacionària, i pot ser congènita (de naixement) o adquirida amb un dany al diencèfal o al còrtex cerebral i afecta a una de cada 30.000 persones, per igual a ambdós sexes.

Els ulls normals tenen 6 milions de cons fotoreceptors, situats majoritàriament al centre de la retina, que són les cèl·lules encarregades de percebre el color, i detectar canvis en la imatge més ràpidament, ja que la resposta als estímuls de la llum és molt més ràpida, a diferència dels bastons, que s'encarreguen de captar el nivell d'il·luminació, no tenen una resposta tant ràpida i existeixen en una quantitat de 100 milions, situats a la perifèria de la retina. Per això hi ha molts tipus d'acromatòpsia, variant en la quantitat de cons i bastons dels que disposa l'ull humà.

Els primers símptomes detectats poden ser:

- Evidentment, la no percepció del color, que pot ser descoberta aviat en edats escolars, al no reconèixer bé els colors, o no relacionar-los i un detall de la visió pobre.
- Fotofòbia (sensibilitat a la llum, i aversió)
- Moviment involuntari dels ulls (altrament conegut com nistagme).

1.2 Neil Harbisson

Figura 1: Neil Harbisson

Neil Harbisson és un artista i compositor que, des de que se li va diagnosticar l'acromatòpsia, ha dedicat la major part del seu temps a intentar percebre el color, des d'un punt de vista diferent.

El primer pas realitzat va ser l'invent anomenat "eyeborg", un sistema que li permet "escoltar els colors". Format per una càmera connectada a un portàtil i uns auriculars, aquest sistema funciona de la següent manera:

La càmera, col·locada al seu front, capta imatges de l'entorn des d'un angle similar al de la seva visió. Aquestes imatges són processades pel portàtil que duu a una motxilla que, mitjançant un software específic, capta de cada imatge un color en particular. Aquest color és associat a un so mitjançant l'anomenada Escala sonocromàtica pura de Harbisson, que s'explicarà més endavant.

Tot aquest sistema permet a Neil solucionar el seu problema d'acromatòpsia, podent captar cada color com un so determinat. Segons ell, la seva condició es defineix amb la paraula *sonocromatòpsia* (sono: so en llatí, chromat: color en grec, -opsia: condició visual en grec). Creu que l'acromatòpsia no defineix la seva condició, perquè els que pateixen aquesta malaltia no poden distingir els colors ni percebre'ls, i ell si. És considerat el primer "cyborg", combinació de persona i ordinador, reconegut per un govern.

2 Solució: Eyeborg 2.0

L'Eyeborg antic és un sistema pesant, de dimensions massa grans, ja que requereix d'un ordinador portàtil per realitzar el processament.

Aquest projecte que es presenta, anomenat Eyeborg 2.0, pretén incorporar la mateixa funcionalitat però amb un sistema miniaturitzat i compacte, incorporant-ho tot en una plataforma hardware de visió, capaç de fer tot el processat alhora que agafa imatges de l'entorn i genera els tons necessaris. D'aquesta manera, la plataforma es podrà incorporar de forma ergonòmica al cap de l'usuari, i estalviar una càrrega de pes i volum innecessaris.

Així doncs, l'objectiu principal del projecte és obtenir un color representatiu d'una imatge, per transformar-lo posteriorment a un to concret. Per fer això, és necessari obtenir una plataforma hardware de visió de dimensions petites que permeti:

- Captar imatges a color
- Generar sons
- Tenir una mínima autonomia d'energia
- Integrar-se al cap de l'usuari

A continuació es comenta quin és el sistema escollit per realitzar aquest projecte, i com s'aconseguiran assolir tots els objectius.

2.1 CMUCam3

L'objectiu de CMUCam3 és proporcionar capacitats de visió a un petit sistema incrustat, basat en un entorn de desenvolupament de codi lliure. Per tant es té una plataforma de hardware programable, combinada amb una càmera que dona les imatges necessàries per obtenir visió de l'entorn.

Està formada per un microcontrolador NXP LPC2106, connectat a un mòdul amb sensor de càmera Omnivision, i es programa amb un port del *GNU toolchain* per ARM.

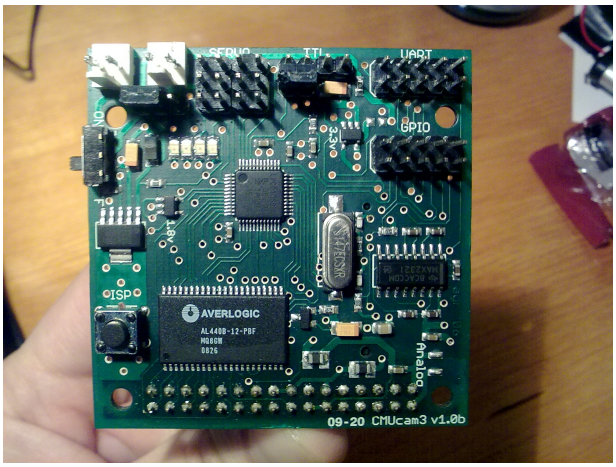


Figura 2: CMUCam3

NXP LPC2106

El component principal de CMUCam3 és el xip NXP LPC2106, que conté aquestes característiques:

- Processador ARM7TDMI-S amb CPU operant fins a 60 Mhz.
- Memòria flash de 128 kilobytes i RAM de 64 kilobytes.
- Controlador de vector d'interrupcions.
- 2 Canals UART, per la comunicació sèrie amb el PC.
- Unitat PWM fins a 6 sortides i pins GPIO.

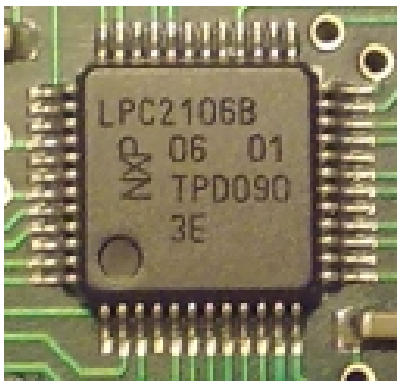


Figura 3: Processador LPC2106

El xip utilitza 3 busos diferents per accedir a les funcionalitats del sistema:

- Un bus local (ARM7 local bus) per accedir als 2 mòduls de memòria (flash i RAM).
- Un bus AMB (Advanced High-performance Bus) per accedir al vector d'interrupcions.
- Un bus VPB, bus per controlar els serveis perifèrics com el PWM, UART i les GPIO.

Aquí es pot veure el diagrama de blocs del LPC2106 amb el processador ARM:

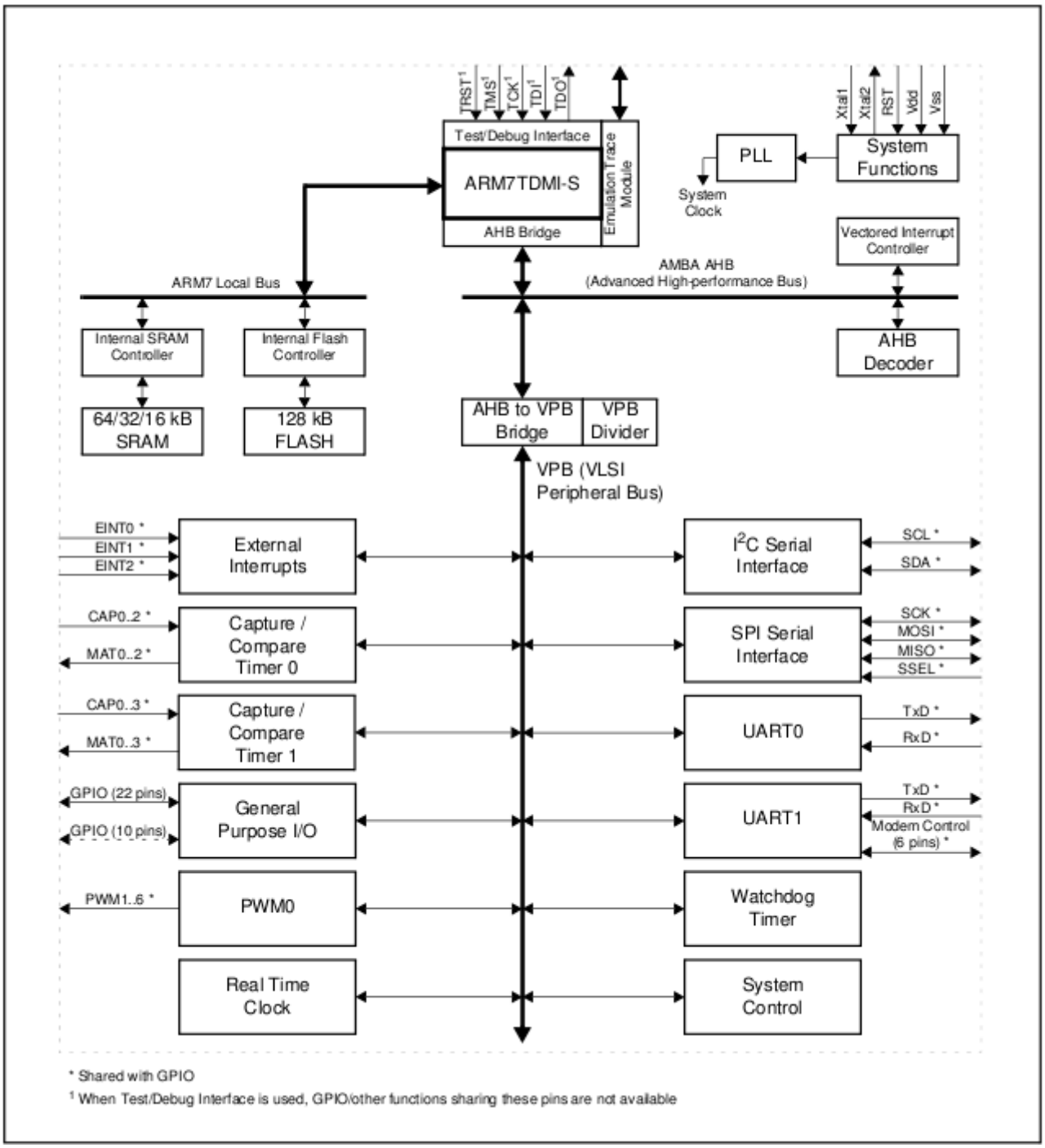


Figura 4: Diagrama de blocs LPC2106

ARM7TDMI-S

La família del processador ARM7TDMI-S, introduït al 1994, ha arribat a vendre unes 10 bilions d'unitats degut a la seva versatilitat tant en cost com en potència. Aquest model és un microprocessador de 32 bits de propòsit general, consumeix molt poca energia donant unes bones prestacions. Utilitza l'arquitectura RISC i això proporciona una resposta molt bona de temps real en les interrupcions, enfront la complexitat de les arquitectures CISC.

També utilitza una estratègia d'arquitectura anomenada THUMB. Aquesta estratègia es basa en un conjunt d'instruccions molt reduïdes, bàsicament contenint el conjunt estàndard d'instruccions ARM de 32 bits, i després un altre conjunt THUMB d'instruccions de 16 bits. L'avantatge principal és que utilitza els mateixos registres de 32 bits de les instruccions ARM però agafant la rapidesa que dona l'accés a registres de 16 bits.

Mòdul Omnivision OV6620

Omnivision OV6620 és un mòdul que integra el sensor d'una càmera a color amb un xip per controlar el dispositiu, tot en un paquet de mida reduïda. Proporciona imatges de fins a 352x288 píxels a uns 60 fotogrames per segon.

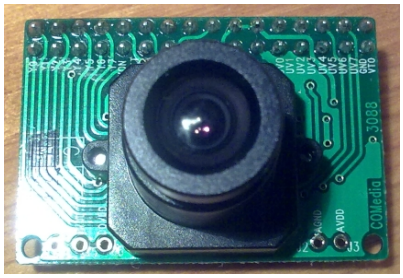
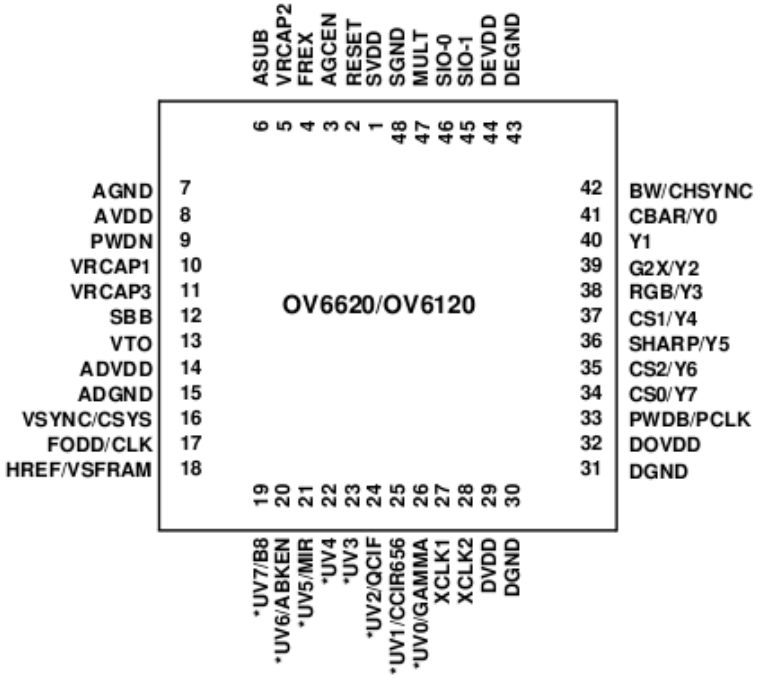


Figura 5: Mòdul Omnivision OV6620

S'acobla directament a la plataforma CMUCam3 transmetent la informació per el bus de dades de la càmera. Aquesta transmissió està controlada per el rellotge del mòdul de la càmera. Les imatges d'aquesta es passen a una FIFO que emmagatzema les dades i són servides al LPC2106 quan es requereix.



* *Note: Outputs UV0-UV7 are not available on the OV6120. The inputs associated with these respective pins are still functional.*

OV6620/OV6120 PIN ASSIGNMENTS

Array Elements (CIF) (QCIF)	356 x 292 (176 x 144)
Pixel Size	9.0 x 8.2 µm
Image Area	3.1 x 2.5 mm
Max Frames/Sec	Up to 60 FPS
Electronic Exposure	Up to 500 : 1 (for selected FPS)
Scan Mode	progressive
Gamma Correction	0.45/.55/1.0
Min. Illumination (3000K)	OV6620 - < 3 lux @ f1.2 OV6120 - < 0.5 lux @ f1.2
S/N Ratio (Digital Camera Out)	> 48 dB (AGC = Off, Gamma = 1)
FPN	< 0.03% V_{P-P}
Dark Current	< 0.2 nA/cm^2
Dynamic Range	> 72 dB
Power Supply	5VDC, ±5% (Anal.) 5VDC or 3.3VDC (DIO)
Power Requirements	< 80mW Active < 30µW Standby
Package	48 pin LCC

Figura 6: Esquema xip Omnivision

I en aquesta gràfica s'observa la resposta que té el sensor a les diferents amplituds d'ona de l'espectre de la llum. Es pot comprovar com té una pitjor resposta de recepció en les amplituds d'ona baixes, en la corba de blau.

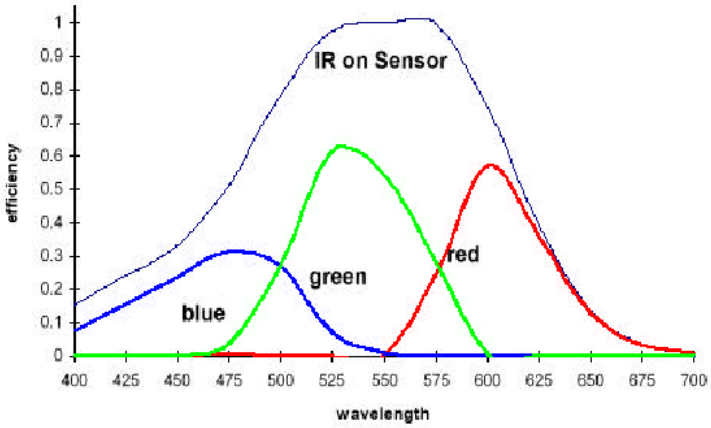


Figura 7: Resposta de la lent

Els colors més vius en canvi, com el verd o el vermell, els capta molt millor. Això es comprovarà en l'apartat de test de color, on aquesta resposta es veurà reflectida.

Les característiques de la òptica són:

- Distància focal davantera: 4.9mm
- Distància focal posterior: 5.1mm
- Apertura F: 2.8
- Camp de visió: Diagonal 70º, horitzontal 56º, vertical 42º

Sistema de color RGB i HSV

Primerament, s'ha de pensar quin serà el millor sistema de color per interpretar les imatges que donarà la càmera, bàsicament, s'ha d'escollir entre dos dels sistemes més coneguts, RGB i HSV.

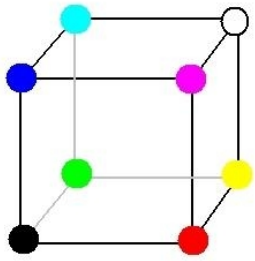


Figura 8: Cub RGB

RGB és un sistema de color format per 3 components (R,G,B) que corresponen a la quantitat de vermell, verd i blau. Aquest sistema, basant-se en l'adició dels colors primaris, crea una relació directe per saber quina quantitat d'aquests components té el color. Es representa, en la majoria de casos, amb valors per cada component en l'interval [0,255], per representar-ho en el format estàndard de byte per component.

La manera més comú de mostrar l'espai RGB és amb un cub, com es veu a la figura. La diagonal del cub on les tres components són idèntiques, representa clarament l'escala de grisos. El problema però és que no és fàcil determinar la saturació del color, ni tampoc la brillantor, ja que s'han de combinar coeficients entre les tres components, i no s'interpreta directament.

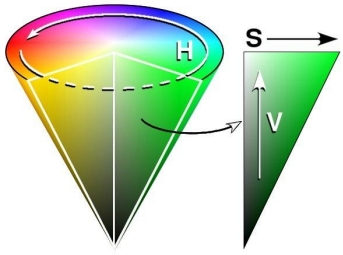


Figura 9: Con HSV

HSV és un altre sistema de color molt conegut, i proporciona una altra visió del color, la separació de components per to (*Hue*), saturació (*Saturation*) i lluminositat (*Value*). És una transformació directe del model RGB, mitjançant unes condicions sobre els valors mínims i màxims del color en RGB. Es representa normalment en forma de con, ja que és més fàcil representar la component de to en un cercle, per poder-la escollir fàcilment a través d'un angle, i després desplaçar-se sa través del con per enfosquir el color (lluminositat) o donar-li més pigment (saturació).

Aquest sistema permet fàcilment interpretar per separat el color, de la seva lluminositat i saturació, cosa que és perfecte per el projecte ja que interessa primerament la component de to per composar l'escala.

Port de GNU toolchain per ARM

Per desenvolupar en el llenguatge de la càmera, és necessari executar instruccions de 32-bits sobre el format d'instruccions del processador ARM. Per això, s'ha de buscar algun llenguatge d'alt nivell amb el que es pugui programar còmodament, però que tingui un port traduïble a instruccions ARM. Aquest és el GNU toolchain per ARM.

Sourcery G++ és una eina que permet compilar per vàries plataformes, des del llenguatge C o C++. Juntament amb un conveni amb ARM, mantenen el port GNU toolchain per ARM, per programar per aquesta arquitectura.

2.2 Obtenció del color

Abans s'ha descrit l'objectiu principal del projecte, obtenir un color d'una imatge per després generar un to concret. Cal tenir clar com s'obtindrà aquest color, ja que no és una decisió fàcil de prendre.

Partir d'una imatge en concret i preguntar-se: Quin és el color d'aquesta imatge? Sovint s'acosta més a un criteri artístic que no pas científic. Es pot obtenir un color d'una imatge de moltes maneres, però s'ha d'aconseguir captar un color que representi tota la imatge.

Evidentment, si es fixa el punt de vista en un objecte que ocupa tot el camp de visió i només té un color, la solució seria fàcil. Però es vol aconseguir un procés que obtingui un sol color resultant d'un espai obert, per exemple.

Si es preguntés a una persona, de quin color és una habitació, que respondria? En una habitació hi ha quadres, mobles i molts més objectes. Com pot determinar un color concret? Tot i poder semblar difícil, la resposta és senzilla, el més segur és que escollís el color de les parets. Això pot passar perquè les parets ocupen més espai en el seu camp de visió, i per tant l'àrea del color en la imatge influeix alhora de determinar aquest color resultant. Per tant, per aquest projecte, es farà servir el que s'anomenarà color representatiu, que no és res més que el color que més es repeteix en tota la imatge.

Per fer això, s'han de rebutjar sistemes com la mitjana aritmètica de tots els colors de la imatge, ja que això produiria una barreja que no determinaria en absolut el color representatiu. El sistema que s'utilitzarà és el dels histogrames.

Un histograma no és res més que un vector, en aquest cas de 360 tons (referenciant a l'angle del to del sistema HSV) i per cada posició del vector, es fa un recompte del nombre d'aparicions d'aquell to. Per tant només cal comptar per cada to, quants cops apareix a la imatge, cosa que implicarà tant el component de color com la de àrea, que és justament el que s'estava buscant.

3 Funcionament hardware

En aquest apartat es descriurà tot el que fa referència a la part hardware del projecte. Abans de començar a programar res, és completament necessari conèixer a fons la plataforma a nivell físic.

Es faran constants referències a les guies d'especificació d'aquesta, ja que allà està tot descrit en detall. També cal dir que hi ha parts de codi que s'expliquen en aquest apartat, en comptes de fer-ho a l'apartat de software, ja que s'accedeix a registres del processador i requereix una explicació més propera al hardware. Per accedir a aquests registres, s'utilitzarà la macro REG:

```
#define REG(addr) (*(volatile unsigned long *)(addr))
```

3.1 Accés als pins del LPC2106

En aquest esquema es poden visualitzar tots els pins del microcontrolador:

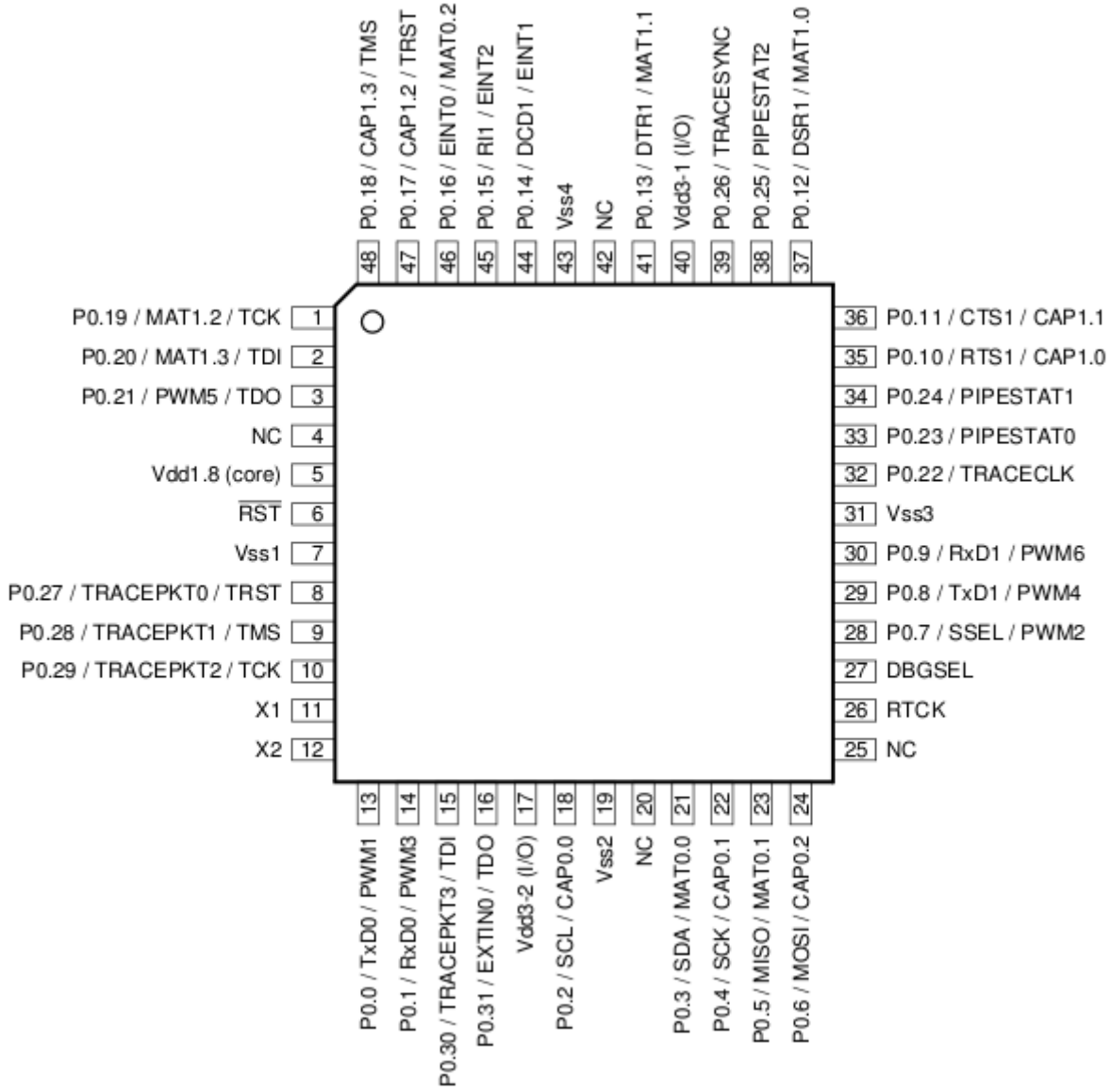


Figura 10: Esquema pins LPC2106

L'accés als pins del LPC2106 serà necessari per comunicar-se amb l'exterior del microcontrolador i per fer-ho, es disposa de les anomenades GPIO (*General Purpose Input Output*). Els pins s'han etiquetat del 0 al 31, de manera que es poden reconèixer en l'esquema anterior.

Pin Name	Type	Description
P0.0 - P0.31	Input/ Output	General purpose input/output. The number of GPIOs actually available depends on the use of alternate functions.

Figura 11: Pins GPIO

Cal tenir en compte que els pins tenen funcions compartides, com es pot veure en l'esquema, i s'ha de configurar un registre per poder utilitzar una funció o una altra.

Pinsel0 and Pinsel1 Values		Function	Value after Reset
0	0	Primary (default) function, typically GPIO Port	00
0	1	First alternate function	
1	0	Second alternate function	
1	1	Reserved	

Figura 12: Registre Pinsel0 i Pinsel1

El registre PINSEL0 controla els pins de 0 a 15 i el PINSEL1 de 16 a 31. Per utilitzar les GPIO cal que cada entrada del pin en aquests registres tingui els dos bits a 0, com mostra la taula anterior.

Per accedir als pins i activar-los o desactivar-los, el microcontrolador disposa d'uns registres pel control d'aquestes GPIO, que són els següents:

Address	Name	Description	Access
0xE0028000	IOPIN	GPIO Pin value register. The current state of the port pins can always be read from this register, regardless of pin direction and mode.	Read Only
0xE0028004	IOSET	GPIO 0 Output set register. This register controls the state of output pins in conjunction with the IOCLR register. Writing ones produces highs at the corresponding port pins. Writing zeroes has no effect.	Read/Set
0xE0028008	IODIR	GPIO 0 Direction control register. This register individually controls the direction of each port pin.	Read/Write
0xE002800C	IOCLR	GPIO 0 Output clear register. This register controls the state of output pins. Writing ones produces lows at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing zeroes has no effect.	Clear Only

Figura 13: Registres GPIO

Aquests registres, de 32 bits cadascun (corresponents als 32 pins que es poden controlar com a GPIO), s'utilitzen de la següent manera:

- IOPIN : Permet llegir el valor dels pins.
- IOSET: Posar a 1 els pins seleccionats.
- IOCLR: Posar a 0 els pins seleccionats.
- IODIR: Es pot indicar la direcció de cada pin, si és d'entrada o de sortida.

3.2 Interrupcions

Un punt clau del projecte és conèixer les interrupcions, que permetran executar funcions contínuament per controlar, per exemple, la captura d'imatges o el PWM, que s'expliquen més endavant.

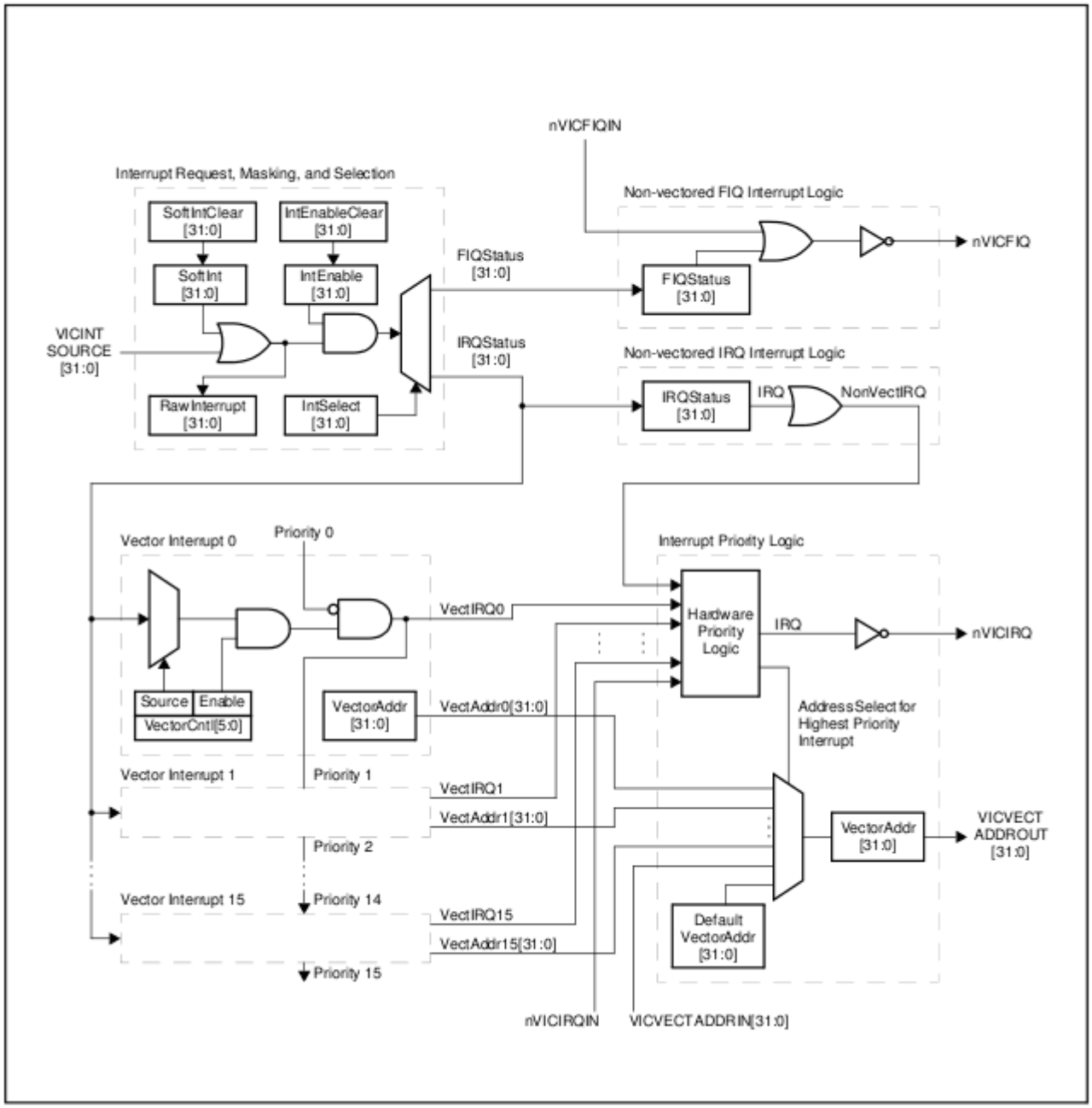


Figura 14: Diagrama de blocs del VIC

El controlador de vector d'interrupcions (VIC) controla 32 inputs d'interrupció, i aquests es poden assignar a 3 categories: FIQ, IRQ vectoritzades i IRQ no vectoritzades, amb prioritats en aquest ordre de major a menor. Això permet dinàmicament a través de programació, canviar la prioritat de les interrupcions. Aquest és el diagrama de blocs de les interrupcions:

Bàsicament, en el diagrama es pot veure l'entrada de 32 interrupcions (VICINT SOURCE). El registre *SoftInt* força interrupcions a nivell de software (per si interessa generar-ne una), i el *SoftIntClear* les esborra. D'aquí surt el registre *RawInterrupt* que són les interrupcions reals a tractar.

Seguidament, amb el registre *IntEnable*, s'indica quines interrupcions de les que vénen estan habilitades (o deshabilitades, amb el registre *IntEnableClear*), per això es fa una AND que anul·larà les que es vulguin desactivar. Amb el registre *IntSelect*, s'escull la categoria de les interrupcions a FIQ o a IRQ. *IRQStatus* simplement és un registre per saber l'estat de les 32 interrupcions.

El vector d'interrupcions té 16 entrades, per cada una (vector interrupt 0 – 15) hi ha un registre de control *VectorCntl* que conté un bit per indicar si està activa aquesta posició del vector, i 4 bits per indicar l'adreça a la que s'assigna la interrupció. En la següent taula es poden veure les interrupcions assignades a cada entrada del vector IRQ:

Block	Flag(s)	VIC Channel #
WDT	Watchdog Interrupt (WDINT)	0
-	Reserved for software interrupts only	1
ARM Core	Embedded ICE, DbgCommRx	2
ARM Core	Embedded ICE, DbgCommTx	3
Timer 0	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	4
Timer 1	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	5
UART 0	Rx Line Status (RLS) Transmit Holding Register empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI)	6
UART 1	Rx Line Status (RLS) Transmit Holding Register empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Status Interrupt (MSI)	7
PWM0	Match 0 - 6 (MR0, MR1, MR2, MR3, MR4, MR5, MR6) Capture 0 - 3 (CR0, CR1, CR2, CR3)	8
I2C	SI (state change)	9
SPI	SPIF, MODF	10
-	reserved	11
PLL	PLL Lock (PLOCK)	12
RTC	RTCCIF (Counter Increment), RTCALF (Alarm)	13
System Control	External Interrupt 0 (EINT0)	14
System Control	External Interrupt 1 (EINT1)	15
System Control	External Interrupt 2 (EINT2)	16

Figura 15: Vector d'interrupcions

Finalment, dir que els registres *VectorAddr* serveixen per emmagatzemar l'adreça a on es situa la rutina per tractar cada interrupció. En la part de software ja es veurà com es gestionen les interrupcions a nivell de codi.

3.3 Càmera

El que interessa saber de la càmera són dues coses: Com es configura i com s'obtenen les imatges. Per això, cal analitzar bé el mòdul de la càmera Omnivision i veure com es comunica amb el LPC2106. A continuació es mostra un esquema de blocs entre aquests:

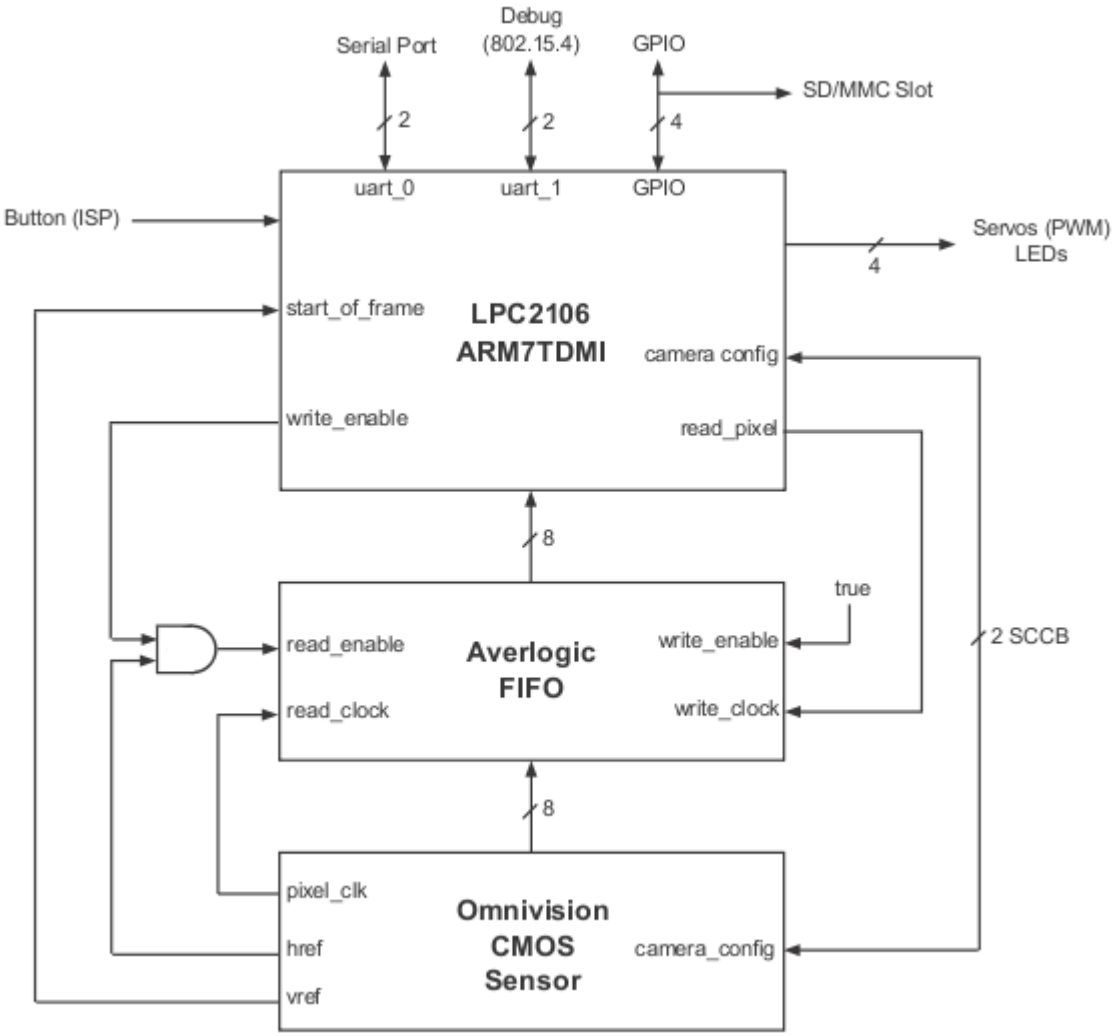


Figura 16: Connexions entre sensor i microcontrolador

Com es pot veure, el microcontrolador i el mòdul de la càmera no estan connectats directament. S'utilitza un buffer col·locat entremig dels dos, així aquest buffer recull les dades del sensor i són servides a l'usuari quan aquest ho demana.

Averlogic AL440B

Figura 17: Memòria Averlogic AL440B

Aquesta memòria intermèdia és una memòria FIFO de 4Mbits que consta de ports independents d'entrada i sortida de 8 bits, funcionant a una freqüència màxima de 80MHz. La memòria recollirà les dades del mòdul de la càmera, i des del microcontrolador caldrà accedir a aquesta memòria per obtenir les

dades. En el següent esquema se'n detallen els pins:

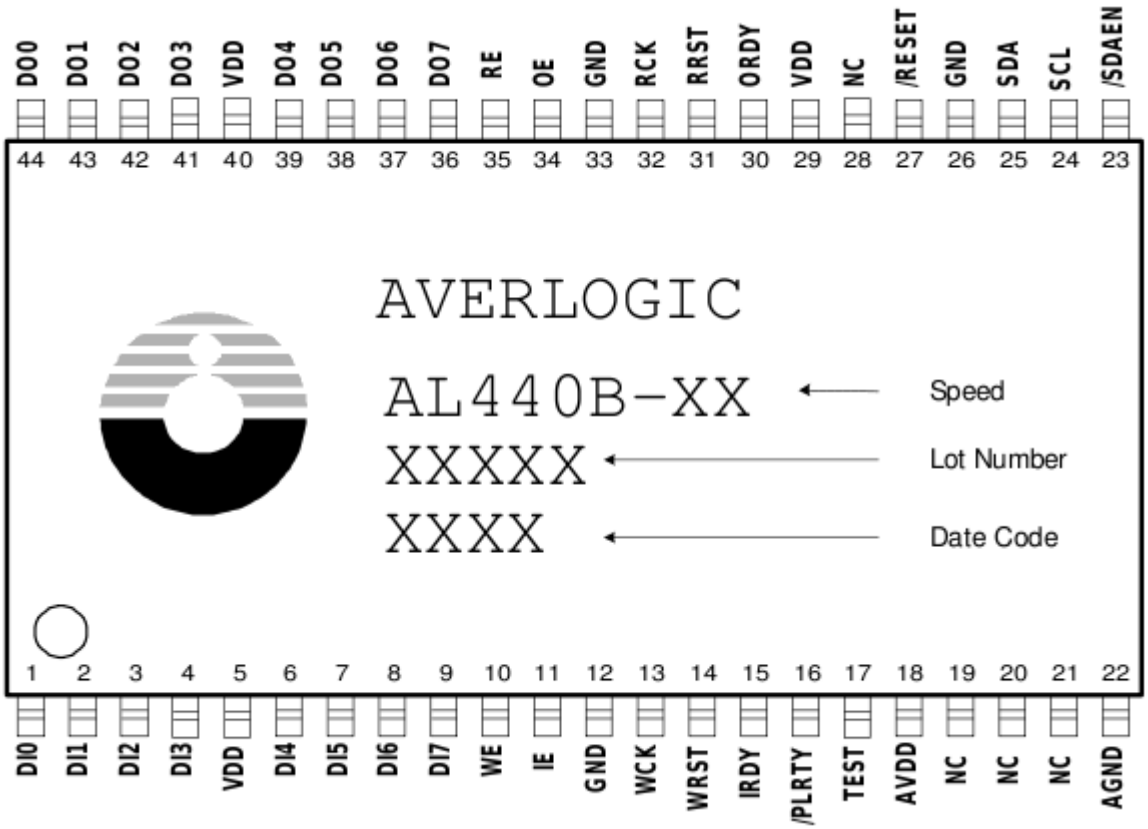


Figura 18: Pins Averlogic AL440B

I seguidament es pot veure un diagrama de blocs representant el funcionament de la memòria:

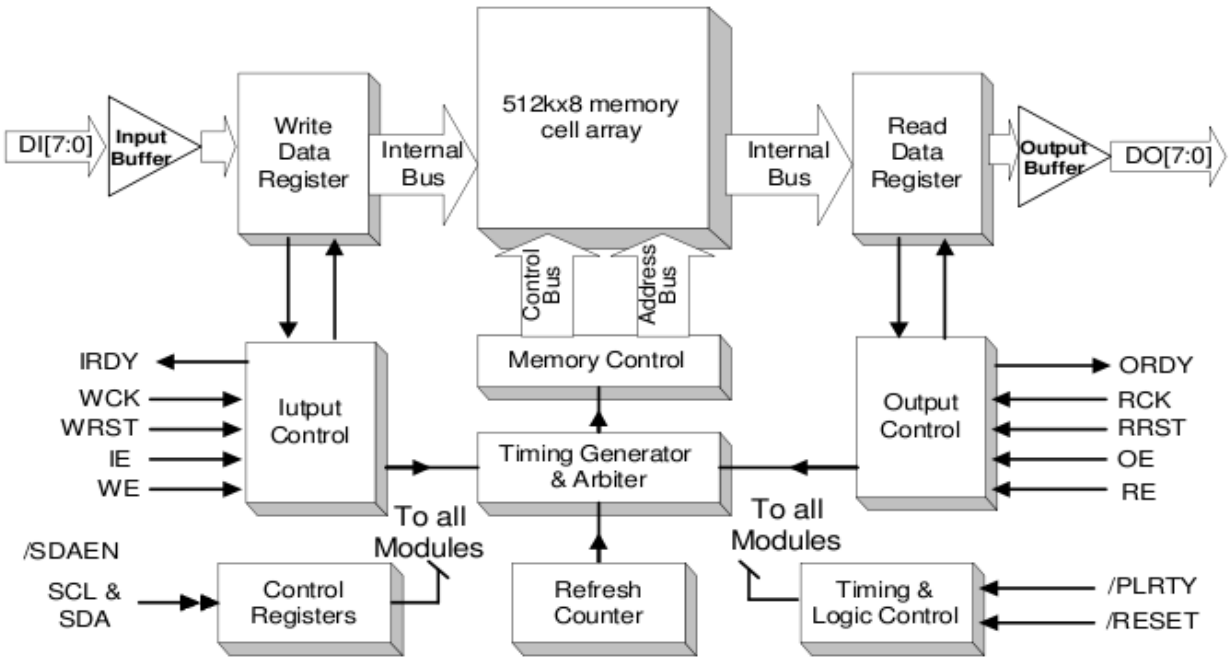


Figura 19: Diagrama de blocs de AL440B

La figura anterior mostra simplement un buffer d'entrada i un de sortida (ports de 8 bits) que permeten llegir i escriure al bloc de memòria intern de 4MB, mitjançant els blocs de control (Input Control i Output Control). Aquests mòduls de control van gestionats per l'àrbitre que controla l'execució d'aquestes ordres.

Ara s'observa un esquema de les connexions externes al AL440B:

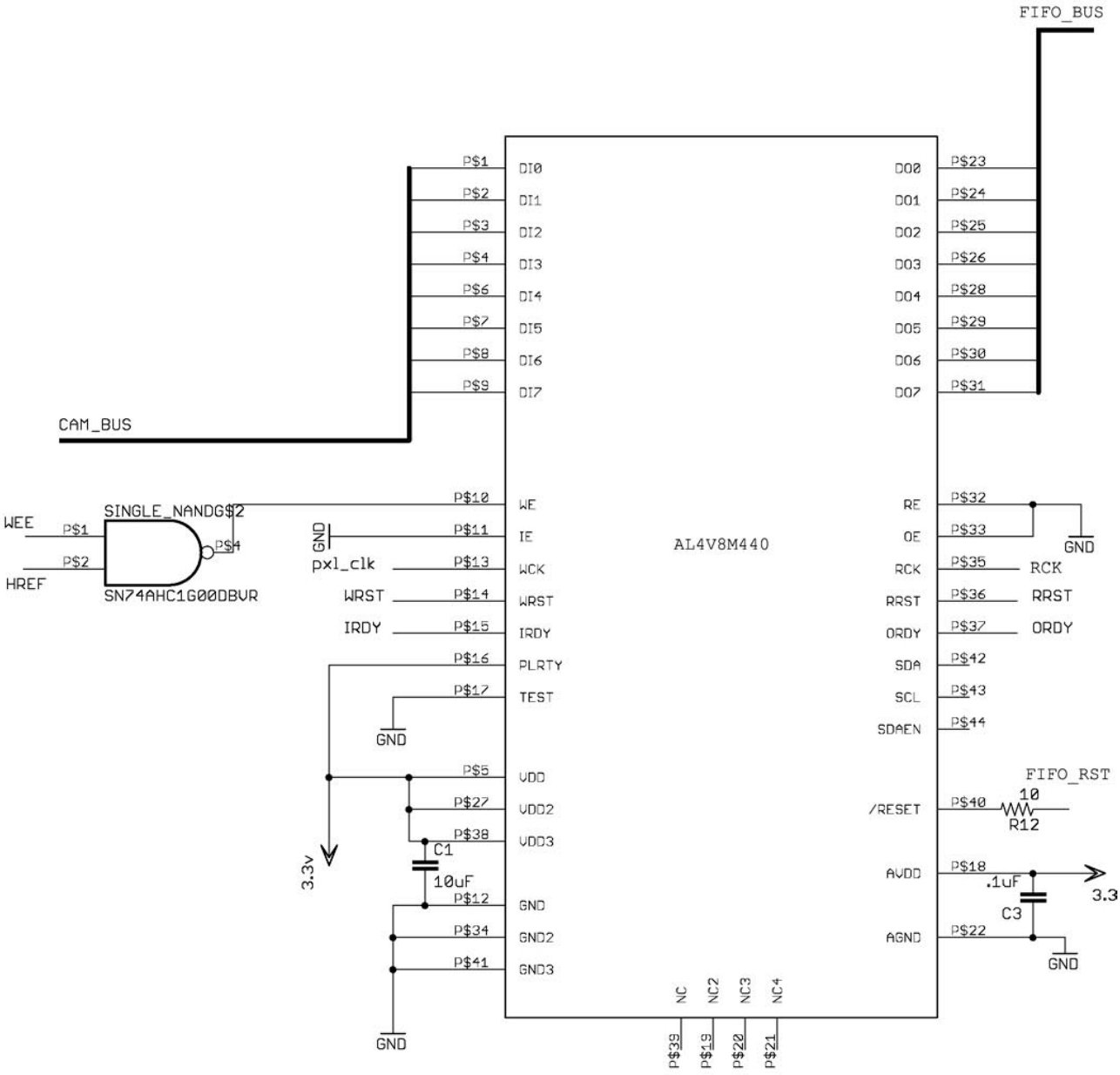


Figura 20: Connexions de AL440B

El CAM_BUS ve del mòdul de la càmera, que envia les dades. I el FIFO_BUS és el que utilitzarà el microcontrolador per llegir-les. La utilització de la resta de pins s'anirà comentant a mida que sigui necessari.

Seguidament es pot contemplar l'esquema de connexions externes per el LPC2106:

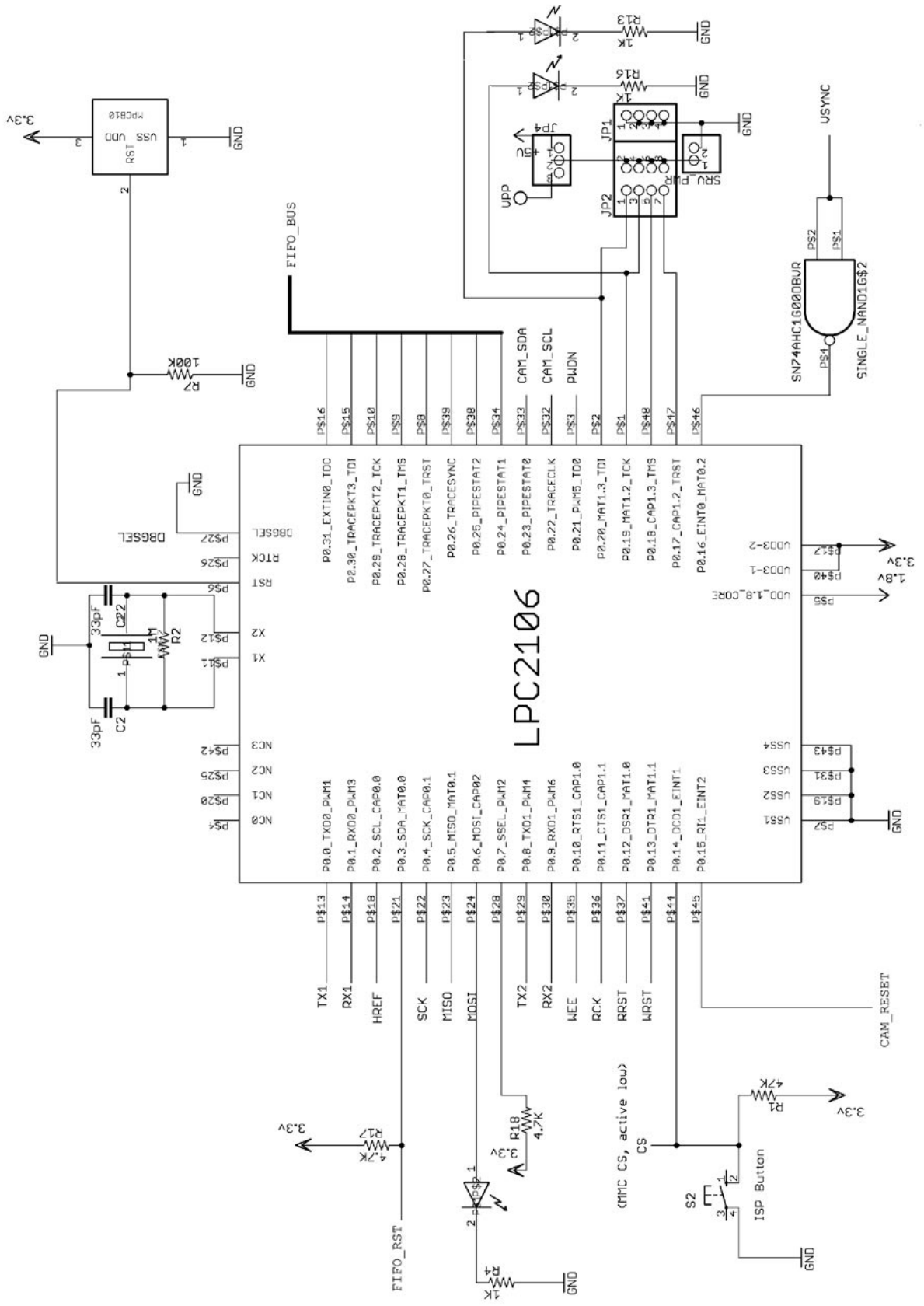


Figura 21: Connexions LPC2106

I finalment, les connexions amb el mòdul de la càmera:

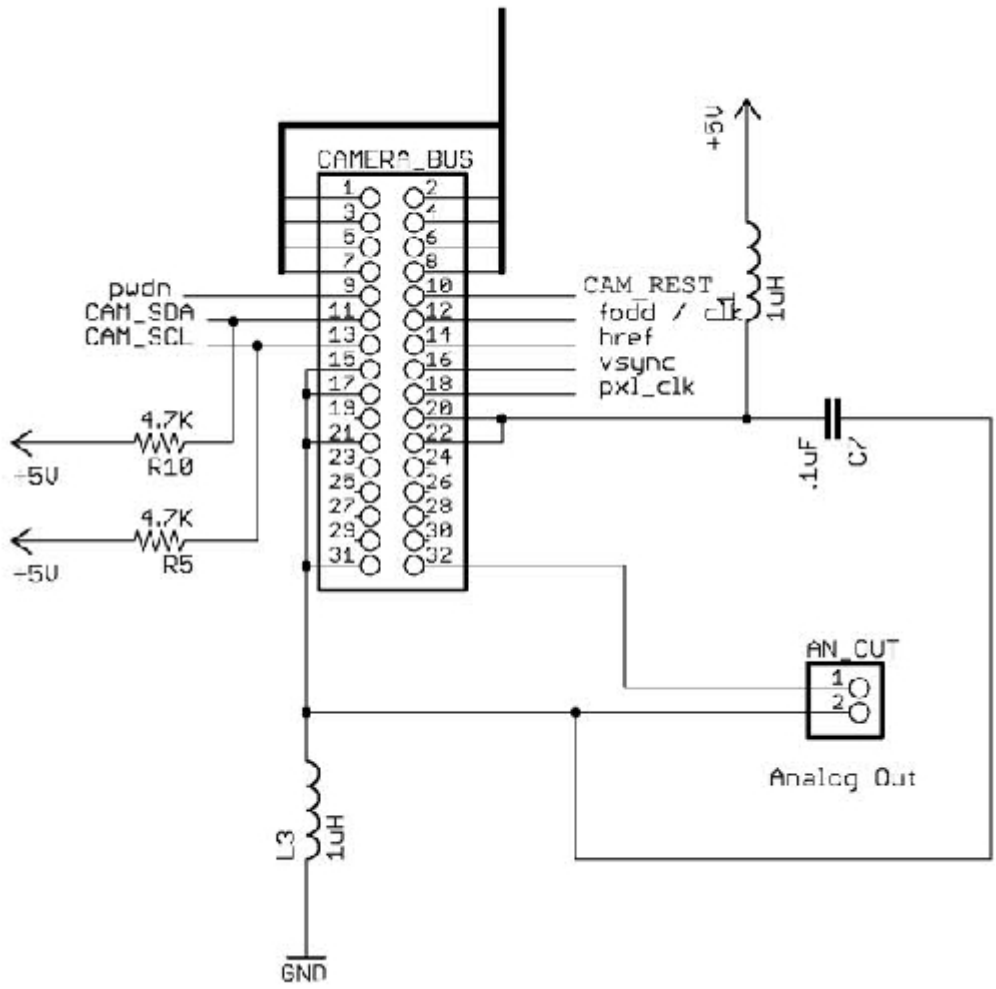


Figura 22: Connexions mòdul càmera

Un cop presentats tots els esquemes, ja es poden comentar els dos punts presentats a l'inici.

Configuració de la càmera

Es procedeix a configurar la càmera, fent referència a la funció de codi *cc3_camera_init* que s'utilitzarà en l'apartat de software. Aquí s'analitzarà la inicialització dels registres pertinents.

```
REG (GPIO_IODIR) = _CC3_DEFAULT_PORT_DIR;
```

Amb aquesta instrucció, s'assigna la direcció dels ports. La constant _CC3_DEFAULT_PORT_DIR indica, dels 32 ports, quins seran d'entrada i quins de sortida, que es veuran més endavant.

```
#define _CC3_DEFAULT_PORT_DIR     0x003EBDC9
```

A continuació, s'ha de reiniciar la càmera, per assegurar-se de que es troba en l'estat inicial, mitjançant la funció *_cc3_camera_reset*. Per fer-ho, s'utilitza el pin 15 GPIO de sortida (que està connectat directament a CAM_RESET, el pin del mòdul de la càmera per resetejar-la). Per fer bé l'activació del reset, primer es comprova que no estigui a 1 el pin posant-lo a 0, seguidament

s'activa per fer el reset, i finalment es torna a deixar a 0 perquè torni a estar estable. Entremig fa una petita pausa per estabilitzar el valor en el pin.

```
#define _CC3_CAM_RESET      0x8000      //1000 0000 0000 0000 Pin 15
```

```
void _cc3_camera_reset ()
{
  REG (GPIO_IOCLR) = _CC3_CAM_RESET;
  _cc3_delay_us_4 (1);
  REG (GPIO_IOSET) = _CC3_CAM_RESET;
  _cc3_delay_us_4 (1);
  REG (GPIO_IOCLR) = _CC3_CAM_RESET;
  _cc3_delay_us_4 (1);
}
```

Seguidament, s'ha de fer un reset complet al buffer FIFO. Per fer-ho, es crida la funció _cc3_fifo_reset.

```
void _cc3_fifo_reset () {
  REG (GPIO_IOCLR) = _CC3_BUF_RESET;
  _cc3_delay_us_4 (1);
  REG (GPIO_IOSET) = _CC3_BUF_RESET;
  REG (GPIO_IOCLR) = _CC3_BUF_WEE;
  REG (GPIO_IOCLR) = _CC3_BUF_WRST;
  REG (GPIO_IOCLR) = _CC3_BUF_RRST;
  REG (GPIO_IOCLR) = _CC3_BUF_RCK;
  REG (GPIO_IOSET) = _CC3_BUF_RCK;
  REG (GPIO_IOCLR) = _CC3_BUF_RCK;
  _cc3_delay_us_4 (1);
  REG (GPIO_IOSET) = _CC3_BUF_WRST;
  REG (GPIO_IOSET) = _CC3_BUF_RRST;

}
```

Per veure'n el funcionament, s'han d'entendre els següents pins del mòdul de memòria:

```
#define _CC3_BUF_WRST       0x2000 //0010 0000 0000 0000 Pin 13
#define _CC3_BUF_RRST       0x1000 //0001 0000 0000 0000 Pin 12
#define _CC3_BUF_RCK        0x800  //     1000 0000 0000 Pin 11
#define _CC3_BUF_WEE        0x400  //     0100 0000 0000 Pin 10
#define _CC3_BUF_RESET      0x8    //               1000 Pin 3
```

- WRST / RRST, connectats als pins 13 i 12 GPIO respectivament, de sortida, són les senyals que fan el reset al punter de memòria de lectura i d'escriptura. S'ha de fer un pols de baixada a cadascun d'ells, però per materialitzar el pols s'ha d'esperar al flanc del RCK.
- RCK, connectat al pin 11 GPIO, de sortida, segueix la connexió RCK i és el rellotge de lectura de dades, les quals s'obtenen sincronitzades amb aquest rellotge. És necessari el flanc de pujada del rellotge per fer que les entrades canviades es processin. Per això s'executa quan les entrades WRST, RRST s'han posat a 0, per materialitzar aquest valor. Després es posen a 1, però s'estabilitzaran en el pròxim flanc de rellotge.
- WEE, connectat al pin 10 GPIO, de sortida, segueix la connexió WEE. És el pin que permet l'escriptura al mòdul. Es posa a 0 per deixar a l'inici que no es pugui escriure. No necessita l'espera del rellotge ja que és una senyal de control.
- RESET, connectat al pin 3 GPIO, de sortida, segueix la connexió FIFO_RST. Fa un reset a la lògica de control, deixant-lo amb les configuracions inicials per defecte. Un flanc de baixada és suficient per resetejar la memòria. El pin RESET és independent del rellotge, per tant no li cal esperar al flanc de RCK.

Cal notar que per fer un reset estable al punter d'escriptura WRST, s'hauria de donar el flanc de rellotge al WCK (rellotge d'escriptura de dades), però no hi ha connexió directa des del microcontrolador en aquest pin, ja que el rellotge d'escriptura va regit per el rellotge que dóna el mòdul de la càmera (pxl_clk). Per tant en certa manera, s'aprofita la senyal automàtica de l'altre rellotge, es posa a 0 la WRST i quan ve el flanc de rellotge de pxl_clk, ja es materialitza.

Emmagatzemar imatge al buffer

La càmera va enviant les dades al buffer FIFO, i la idea es tracta de permetre l'escriptura en aquest buffer per emmagatzemar-les.

Per saber com s'obtenen imatges de la càmera, es farà referència a la funció *cc3_pixbuf_load*:

```
#define _CC3_CAM_POWER_DOWN 0x200000 //0010 0000 0000 0000 0000 0000 Pin 21
```

El pin 21, configurat de sortida, està connectat directament al pin de *power_down* del mòdul de la càmera. L'estat *powerdown* de la càmera permet rebaixar el consum d'energia del sistema, mentre no s'esta fent servir. En aquest projecte no s'utilitzarà ja que la càmera està permanentment prenent imatges de l'entorn.

```
if (REG (GPIO_IOPIN) & _CC3_CAM_POWER_DOWN) return;
```

El que es fa primer és comprovar si aquest pin esta a 1 (mode baix consum activat). Si ho està, no ha de ser capaç de capturar imatges. També es comprova que hi hagi alguna dada als GPIO, perquè si no n'hi ha, no cal capturar res encara que estigui en mode de baix consum.

Quan es vol capturar una imatge, s'ha de posar el punter de dades del buffer FIFO a 0, per començar a llegir-les des del principi. Això és el que fa la funció *_cc3_pixbuf_write_rewind*, que com s'ha vist abans a la configuració de la càmera, posa el pin de control d'escriptura WEE a 0 (per no permetre l'escriptura) i seguidament fa el flanc de pujada i baixada al WRST per inicialitzar el punter a 0:

```
void _cc3_pixbuf_write_rewind ()
{
  REG (GPIO_IOCLR) = _CC3_BUF_WEE;
  REG (GPIO_IOCLR) = _CC3_BUF_WRST;
  _cc3_delay_us_4 (1);
  REG (GPIO_IOSET) = _CC3_BUF_WRST;
}
```

Abans de tornar a habilitar l'escriptura per llegir les noves dades de la càmera, s'ha d'esperar a que comenci la imatge des d'el primer píxel. Això com es sap? Amb el VSYNC.

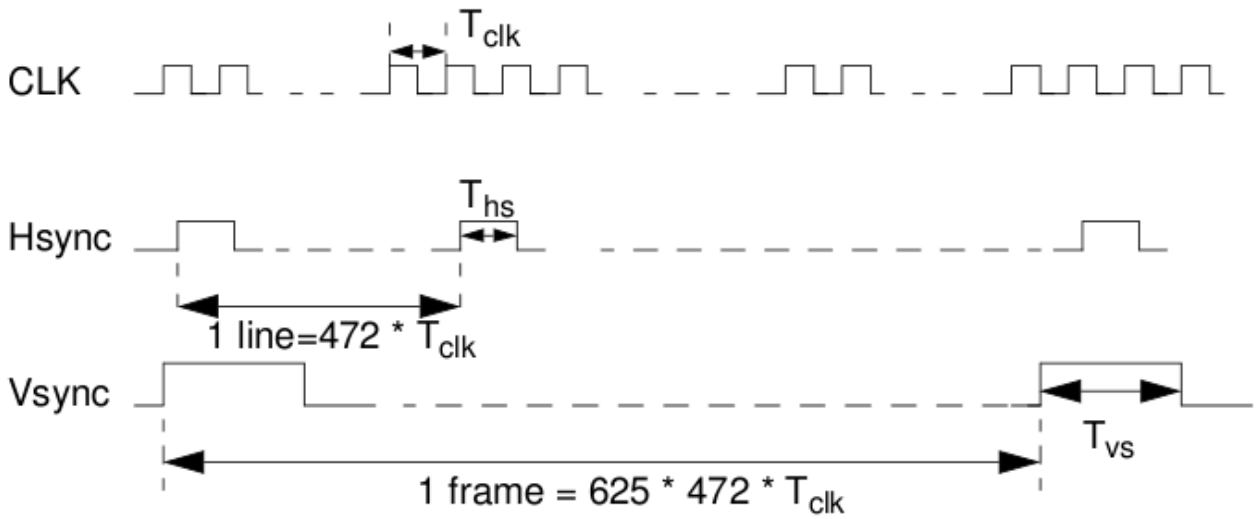


Figura 23: VSync i obtenció de la imatge

VSYNC no és res més que una senyal que indica quan s'han recorregut totes les files (així com HSYNC indica que s'ha recorregut tots els píxels d'una sola fila). Per tant, per començar a escriure s'ha d'esperar aquesta senyal VSYNC.

```
#define _CC3_CAM_VSYNC      0x10000 //0001 0000 0000 0000 0000 Pin 16
```

El pin 16 esta connectat d'entrada, per llegir el valor de VSYNC, que ve directe del mòdul de la càmera.

```
while (!(REG (GPIO_IOPIN) & _CC3_CAM_VSYNC));
while (REG (GPIO_IOPIN) & _CC3_CAM_VSYNC);
```

Amb aquests bucles es comprova que VSYNC estigui a 0, i després s'espera a que VSYNC passi a 1. Un cop s'ha activat VSYNC, s'activa també el permís d'escriptura a la memòria:

```
REG (GPIO_IOSET) = _CC3_BUF_WEE;
```

Ara, s'ha d'esperar a que VSYNC baixi (com es mostra a l'esquema anterior):

```
while (!(REG (GPIO_IOPIN) & _CC3_CAM_VSYNC));
```

Un cop finalitzada la senyal de VSYNC, s'ha de deixar preparada la memòria per començar a llegir-la. Primer s'espera una estona a que el buffer hagi escrit almenys un parell o tres d'entrades a la memòria, perquè el buffer de lectura no avanci el buffer d'escriptura (tot i que no hauria de passar, però es per assegurar-se).

Després, es fa un reset al punter de lectura per preparar la FIFO per començar a llegir la imatge des del principi, de la manera que ja s'ha vist abans (fent el reset al RRST):

```
REG (GPIO_IOCLR) = _CC3_BUF_RRST;
REG (GPIO_IOCLR) = _CC3_BUF_RCK;
REG (GPIO_IOSET) = _CC3_BUF_RCK;
REG (GPIO_IOCLR) = _CC3_BUF_RCK;
REG (GPIO_IOSET) = _CC3_BUF_RRST;
```

Ara ja esta tot a punt per llegir la imatge. No s'ha d'oblidar però que, un cop la imatge s'hagi llegit, s'ha de desactivar l'escriptura a la FIFO, ja que la càmera continua enviant dades i encara està el pin de WEE indicant el permís d'escriptura com a actiu. Per desactivar l'escriptura però, s'ha d'esperar a que la imatge actual acabi d'escriure's a memòria, i aquest temps no es pot malgastar esperant.

Això es soluciona mitjançant interrupcions. Es programa de manera que el permís d'escriptura es desactivi automàticament al pròxim VSYNC. Primer de tot, abans de sortir de la rutina, s'activen les interrupcions:

```
enable_ext_interrupt ();
```

I ara es programa una interrupció que arribi controlada per VSYNC. Això es pot fer perquè el pin que llegeix VSYNC (GPIO 16) està compartit amb la interrupció EINT0 del vector d'interrupcions (Canal 14).

```
if (REG (VICRawIntr) & 0x4000) {
    REG (GPIO_IOCLR) = _CC3_BUF_WEE;
    _cc3_pixbuf_write_rewind ();
    disable_ext_interrupt ();
}
```

Quan ha rebut la senyal de VSYNC, es genera la interrupció i es desactiva el pin de control d'escriptura. Seguidament, es posa el punter d'escriptura a l'inici per el pròxim cop que s'escrigui, fer-ho des d'el principi, i finalment es deshabilita aquesta interrupció.

Amb això ja s'ha vist com obtenir la imatge de la càmera i emmagatzemar-la a la FIFO. Ara s'ha de veure com llegir la imatge.

Llegir imatge del buffer

En aquest punt es farà referència a la funció *_read_pixel*, que és la que s'encarrega de llegir un píxel del buffer FIFO. En l'apartat de software també s'hi farà referència per saber com funciona.

Fins ara s'ha explicat com s'emmagatzemen les dades al buffer però no s'ha explicat en quin format ni en quin ordre. Sense saber això, no es podran interpretar les dades ni es sabrà com llegir-les.

La càmera captura el bolcat de la imatge 356x288 i ofereix a la sortida diferents tipus de formats. El que s'utilitza aquí per defecte és el del canal RGB a 8-bits, aquí es pot veure el format de la matriu de la càmera:

R\C	1	2	3	4	.	353	354	355	356
1	B_{11}	G_{12}	B_{13}	G_{14}		B	G	B	G
2	G_{21}	R_{22}	G_{23}	R_{24}		G	R	G	R
3	B_{31}	G_{32}	B_{33}	G_{34}		B	G	B	G
4	G_{41}	R_{42}	G_{43}	R_{44}		G	R	G	R
5	B_{51}	G_{52}	B_{53}	G_{54}		B	G	B	G
.									
289	B	G	B	G		B	G	B	G
290	G	R	G	R		G	R	G	R
291	B	G	B	G		B	G	B	G
292	G	R	G	R		G	R	G	R

Figura 24: Dades RGB

L'estructura d'aquesta matriu ve donada per l'anomenat filtre de Bayer. El filtre o patró de Bayer no és res més que un format d'adquisició d'imatges en color, és a dir, una estructuració dels sensors per tal d'intentar millorar l'eficiència del color en les imatges captades.

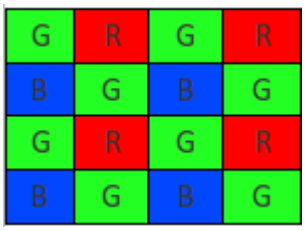


Figura 25: Patró de Bayer

Com es pot veure, l'estructura és la mateixa que la figura anterior. El patró de Bayer pretén utilitzar més sensors de crominància verds que blaus i vermells, per imitar l'ull humà, ja que els sensors verds tenen més sensibilitat a la llum, i això simula el percentatge de bastons més elevat que el de cons.

Les dades s'envien des de la càmera seguint l'ordre G R G B en forma de quadrat. Per tant, fent referència a la figura de la matriu de la càmera anterior, s'obté la sortida G12 R22 G21 B11 G14 R24 G23 B13 i així successivament. Per tant es generen 2 píxels amb els mateixos components R i B, però utilitzant un dels dos components G.

Per veure com s'obtenen els píxels, s'explica la funció *_read_pixel* de la que es fa referència a l'apartat de software:

```
void _read_pixel (uint8_t * pixel, uint8_t * saved)
{

    if (_cc3_second_green_valid) {
      _cc3_second_green_valid = false;
      *(pixel + 0) = *(saved + 0);
      *(pixel + 1) = _cc3_second_green;
      *(pixel + 2) = *(saved + 2);
      return;
    }

    *(pixel + 1) = _cc3_pixbuf_read_subpixel ();          // G
    *(pixel + 0) = _cc3_pixbuf_read_subpixel ();          // R
    _cc3_second_green = _cc3_pixbuf_read_subpixel ();     // G
    *(pixel + 2) = _cc3_pixbuf_read_subpixel ();          // B
    _cc3_second_green_valid = true;
}
```

Com es pot veure, es manté una variable anomenada _cc3_second_green_valid, que és per saber si s'ha d'agafar el primer verd o el segon de la quadrícula. La variable *pixel* es una direcció de memòria que es veurà a l'apartat de software, a la que se li guardarà la informació del píxel.

Al primer pas, no s'entra a la condició i és llegeixen les quatre components del bloc en l'ordre que s'ha comentat abans. El segon verd, es guarda a una variable global. Per tant s'ha aconseguit guardar el primer píxel.

Després, al tornar a cridar la funció el segon cop, com que es té guardat el segon verd, s'entra a la condició i es guarda el següent píxel utilitzant la component R i B anteriors i el segon verd guardat.

La funció per llegir cada component del píxel és *_cc3_pixbuf_read_subpixel*, i fa el següent:

```
uint8_t _cc3_pixbuf_read_subpixel (void) {
  uint8_t result = REG (GPIO_IOPIN) >> 24;
  REG (GPIO_IOSET) = _CC3_BUF_RCK;
  REG (GPIO_IOCLR) = _CC3_BUF_RCK;
  return result;
}
```

Simplement agafa els 8 últims bits del registre GPIO, que és el FIFO_BUS que s'ha comentat anteriorment, com a valor del component del píxel. Seguidament, s'aplica un flanc de pujada a la senyal RCK, i així s'augmenta el punter RRST per llegir el pròxim valor.

Ja s'ha explicat tot el funcionament de la càmera. Ara des de l'apartat de software es podran obtenir fàcilment els píxels i tractar-los convenientment.

3.4 PWM

En els objectius del projecte s'anuncia que la plataforma és capaç de generar sons, però no ho fa des del punt de vista de reproducció d'arxius mitjançant algun programa, com ho podria fer un ordinador, sinó que es genera directament amb el hardware, electrònicament. Abans d'explicar la part hardware que ho fa funcionar però, s'ha d'entendre la part teòrica.

El PWM és una tècnica que permet controlar sistemes analògics a través de fonts digitals. Aquest procés és molt interessant perquè, amb qualsevol microcontrolador, es poden generar senyals digitals que acabin simulant una senyal analògica. PWM significa *Pulse Width Modulation*, és a dir, modulació per ample de pols.

Quan s'emet una senyal digital, es pot jugar bàsicament amb un paràmetre, el temps, ja que el valor d'aquesta sortida només pot ser o 0 o 1. Per tant sent T un període constant, la senyal digital es podria posar a 1 durant un cert temps D.T i a 0 durant un temps T – D.T, i es veuria una senyal tal com la següent:

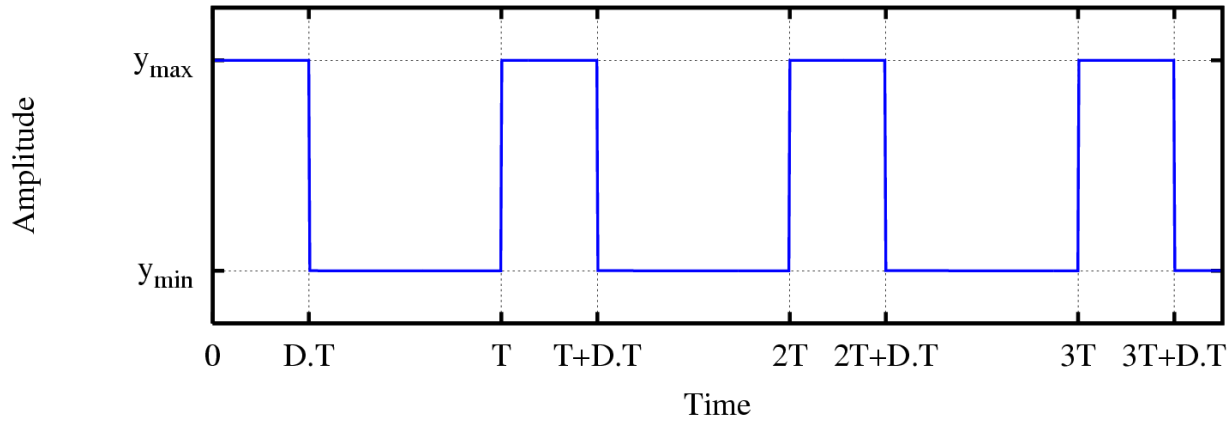


Figura 26: Duty cicle

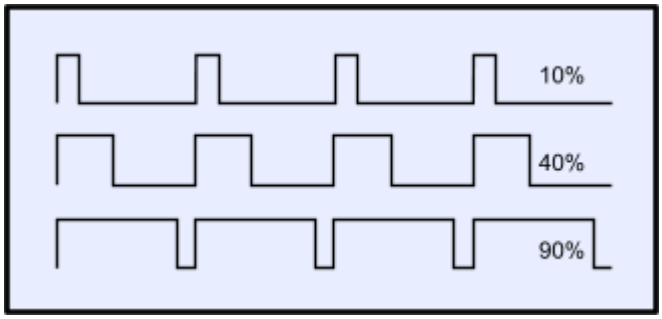


Figura 27: Exemples duty cicle

El temps D.T s'anomena *duty cicle*, i sovint s'expressa com un percentatge. Per tant, es podrien generar senyals com les que es veuen al costat, amb *duty cicles* del 10%, 40% i 90%.

Ara bé, què s'aconsegueix realment amb això? La part interessant es comprova amb el següent exemple:

Si es té un led, el voltatge del qual va controlat per una sortida digital, només es pot o encendre o apagar, posant el voltatge a 5V (valor 1 digital) o 0V (valor 0 digital). Però al fer canvis d'estat en el voltatge, es genera una "senyal mitja" (com es veu a la següent figura) que correspon a una espècie de suma sobre el període del voltatge donat. Si el component s'activa o es desactiva 1 cop per segon, es veuen clarament els dos estats d'encès i apagat, perquè el període és molt gran. Però si s'augmenta la freqüència de la senyal (i es fa més petit el D.T) els canvis són tant ràpids

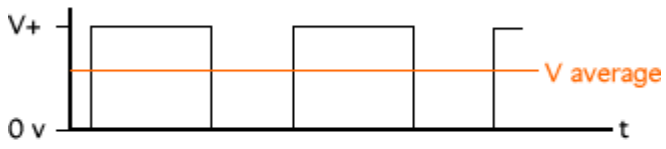


Figura 28: Senyal mitja PWM

que el component no els percep com a tals, sinó que interpreta el valor del voltatge com a valor de la senyal mitja esmentada.

Agafant els valors de la figura d'exemples del duty cicle, es podria veure com en una sortida digital de 0V a 5V, s'obtindria per els 3 casos uns voltatges resultants a la sortida de 0,5V (10%), 0,2V (40%) i 4V (90%). Amb aquestes sortides, es podria alimentar el led amb diferents voltatges i aconseguir modificar la seva intensitat, per tant, es comportaria com un sistema analògic, no digital.

Ara es presenta el cas d'aquest projecte, la generació d'àudio. El que es vol aconseguir és generar un to, que sigui agradable a l'oïda si pot ser, per representar la freqüència de so necessària. Uns altaveus funcionen de la mateixa manera que pot funcionar un led, si apliquem un voltatge mitjà concret generat per un PWM, a una freqüència determinada, generarà un to. Quan més alta sigui aquesta freqüència, el to serà més agut ja que la membrana de l'altaveu vibrarà més ràpid, i més greu si la freqüència és més baixa.

Amb aquest sistema ja s'obtindria l'objectiu, ja que es podria generar un so i variar el to respecte la freqüència del PWM, però es pot millorar, ja que la senyal resultant és una senyal quadrada (conegudes com a *square wave*). El problema d'aquest tipus de senyal és que té un so força punxant i estrident, degut a l'espai que hi ha entre pics de pujada i baixada, genera buits i sona entretallat, no és un so gaire agradable.

El que es pretén és, fent un control del PWM més complex, generar una ona sinusoïdal (*sine wave*) que representa un to "pur", és a dir, la ona de so en el seu estat més natural. Aquest tipus d'ona és molt més suau a l'oïda que la quadrada, per tant es genera un millor to.

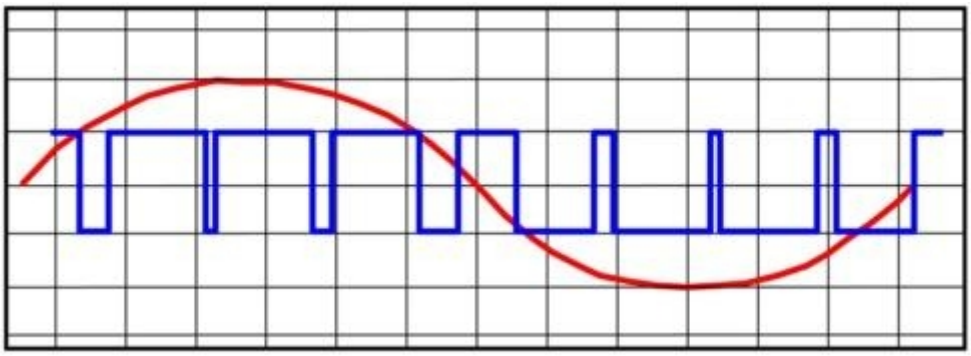


Figura 29: Ona sinusoïdal generada amb el PWM

La manera de generar aquest tipus d'ona, com es veu a la figura superior, és variant el duty cicle a cada període, d'aquesta manera el que s'aconsegueix és variar la tensió mitjana al llarg del temps. Cal adonar-se de que per generar una ona sinusoïdal de període T, cal fer N canvis de duty cicle dins d'aquesta ona, per tant el període d'un pols del PWM serà com a mínim N vegades més petit que T. Com es veurà més endavant, s'intueix que quan més gran sigui aquest N, més resolució hi haurà, i per tant la ona quedarà més ben definida.

Ara bé, un cop es programi el PWM a través de software, s'aconseguirà la senyal quadrada de color blau de la figura anterior, i el que es vol és generar la ona en forma sinusoïdal. Aquest tractament es veurà al capítol de millora de la senyal d'àudio.

Aquesta és la teoria per generar els sons en la plataforma, però abans de poder-ho programar s'ha d'estudiar quin hardware té disponible la plataforma, tant els registres per controlar el PWM com els pins necessaris per connectar un auricular i poder sentir el so.

Ports PWM i connexió amb l'auricular

Abans de comentar la part de registres de control del PWM, cal tenir clar les sortides físiques que donaran la senyal resultant per el so. A continuació es pot veure un esquema:

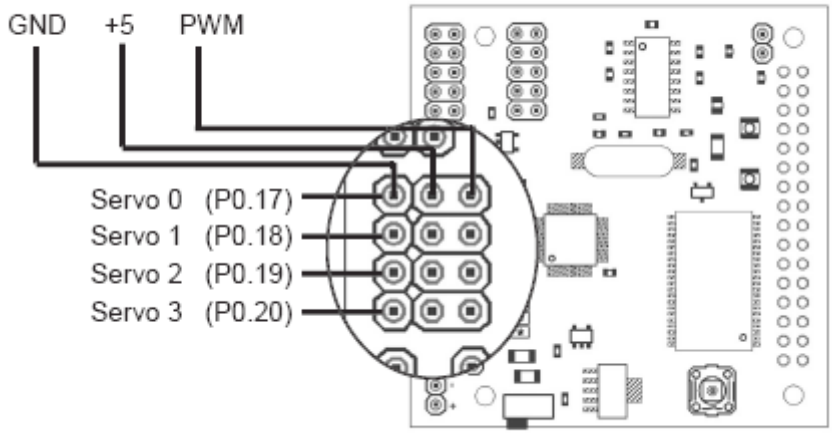


Figura 30: Pins del PWM

Es tenen 4 ports per controlar 4 senyals de PWM diferents i independents. En aquest cas, només se'n farà servir una, ja que només es vol una senyal d'àudio. La senyal pròpia sortirà per el pin de la columna PWM, fila P0.17 anomenat com Servo 0 (aquests ports s'anomenen Servo ja que estan dissenyats amb el propòsit inicial de controlar servos o motors, tot i que en aquest cas es farà servir per generar senyal d'àudio).

Els auriculars que s'utilitzaran (com es veurà a la part de muntatge de la carcassa), com tots els auriculars que permeten estèreo, tenen 3 cables: Un per cada auricular i el de terra. Per tant s'utilitzarà el pin de la senyal (P0.17 senyal PWM) per connectar-lo als dos cables dels auriculars, i el pin de terra (P0.17 senyal GND) per el terra dels auriculars.

Registres per el control del PWM

S'han vist les connexions per passar l'àudio que es generarà a uns cascs qualsevol. Ara toca veure de quins registres es disposa per poder controlar el PWM. Seguint un esquema d'aquests registres, serà més fàcil entendre'n el funcionament:

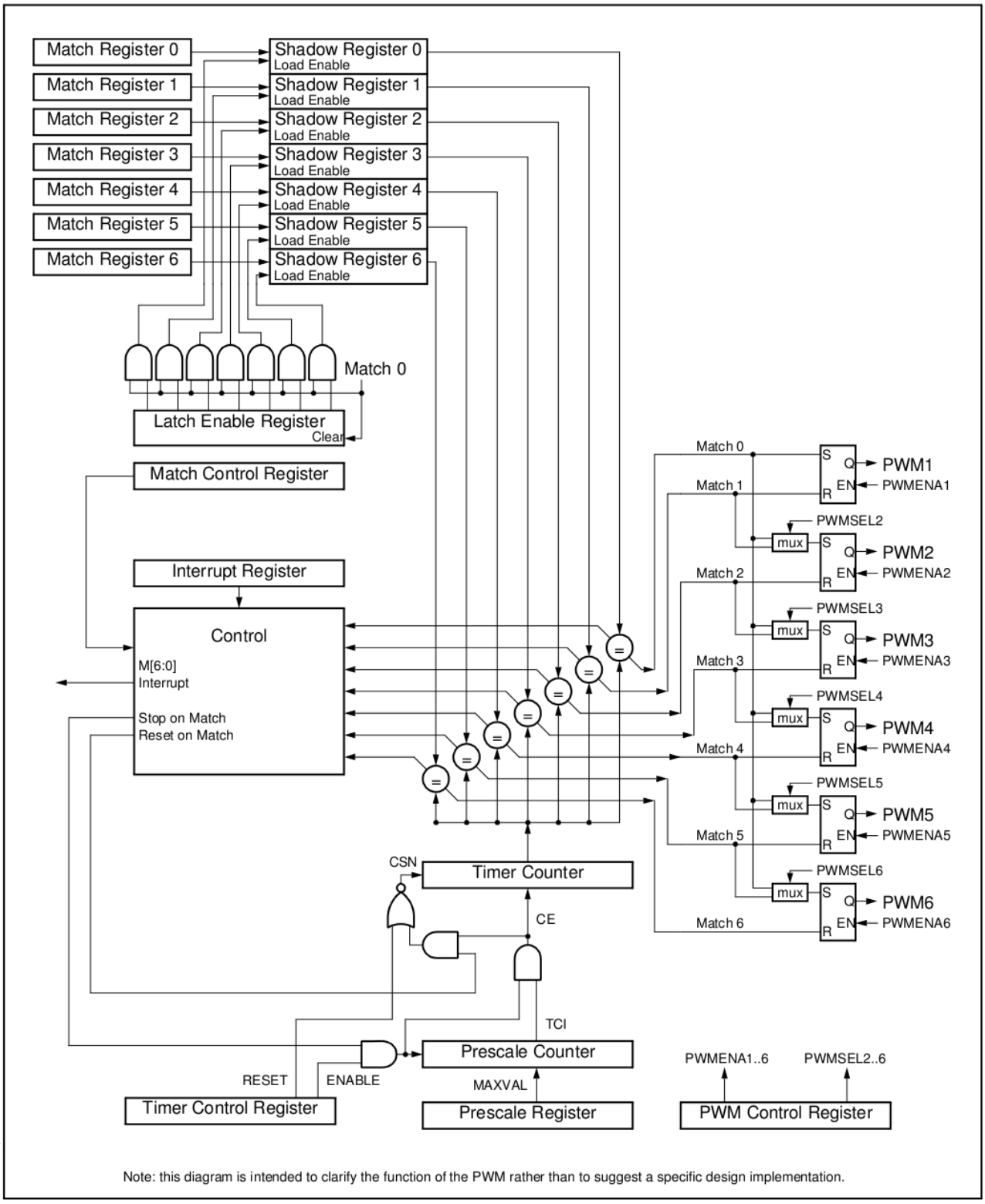


Figura 31: Esquema registres PWM

A la dreta del tot de l'esquema, es veuen les sortides del PWM. Existeixen registres per controlar-ne fins a 6, en realitat però només existeixen físicament els pins per a 4 sortides PWM. En el cas d'aquest projecte només es farà servir la sortida PWM1, ja que només es vol una senyal d'àudio.

El que interessa és controlar la pujada i baixada del flanc de la senyal, així es podrà calcular el temps del duty cicle on la senyal val 1 i baixar-ho el temps restant a 0. El disseny està fet de manera que aquesta pujada i baixada del flanc es pot controlar de 2 maneres:

- PWM d'eix simple: Una senyal Match 0 per indicar el període en tics del PWM i una senyal Match X per indicar quan s'ha de produir el flanc de baixada. El flanc de pujada es fa automàticament al arribar a final de període. Inicialment sempre està a 1.
- PWM d'eix doble: Dos senyals Match X i Match Y per indicar quan s'ha de produir el flanc de baixada i quan el de pujada respectivament.

Hi ha 7 senyals Match X (amb X de 0 a 6) per controlar cadascuna de les sortides PWM. Cadascuna de les sortides s'ha de configurar amb el mode d'eix simple o doble, per saber com s'indiquen els flancs de pujada i baixada. Per configurar-ho, es fan servir les senyals PWMSELX (amb X de 1 a 6) que controlen el multiplexor connectat a cada sortida. El biestable que controla la sortida té el bit de control de reset i el de set. A les entrades de set esta connectada la senyal Match 0, combinada en el multiplexor amb una senyal Match X, per tant el flanc de pujada (activar el SET del biestable) s'activarà per Match 0 si és PWM d'eix simple, i s'activarà per un dels altres Match X si és d'eix doble.

En el cas d'aquest projecte, com que només es necessita controlar el flanc de baixada (ja que el de pujada s'ha d'activar sempre a final de període), es pot deixar controlat el flanc de pujada per la senyal Match 0 i amb la Match 1 controlar el flanc de baixada. Cal notar que la senyal PWM1 no pot ser controlada per eix doble, només per eix simple, per tant ja va bé fer servir aquesta senyal.

Ara bé, d'on vénen aquestes senyals de control de Match? En algun lloc s'ha d'indicar un valor temporal perquè el microcontrolador sàpiga en quin moment ha d'activar el flanc de pujada o el de baixada de la senyal PWM. Això es fa mitjançant uns registres que actuen com a comptadors (MRX) i utilitzant un registre de timer (TC) que actua com a rellotge.

El TC és un registre rellotge, que va comptant el nombre de tics del rellotge de processador que es generen. Aquest registre pot ampliar el temps del seu comptatge mitjançant el típic registre *preescaler* (PR). Per tant cada augment en el rellotge de TC es calcula així:

$$TC = pclk / (preescaler + 1)$$

Hi ha 7 registres MRX, un per cada senyal més el base (senyal Match 0). Aquests registres contenen un valor concret, que indica el valor que ha de tenir TC per fer “match”, és a dir, quan el TC arriba al valor que conté el registre MRX, es dirà que s'ha produït un match al registre MRX i s'activarà la senyal Match X. Això es pot veure representat en l'esquema, on hi ha uns comparadors entre el TC i els *shadow registers* que provenen dels match registers. Els shadow registers no són res més que la materialització del valor del match register, és a dir, quan s'indica un valor a un MRX, s'actualitza només si activem un bit de *enable* en el registre *latch enable*. Aleshores el valor de MRX s'envia al shadow register X, on d'allà s'agafarà el valor actualitzat.

A continuació es pot veure un exemple de com es produeix un match en un dels registres MRX:

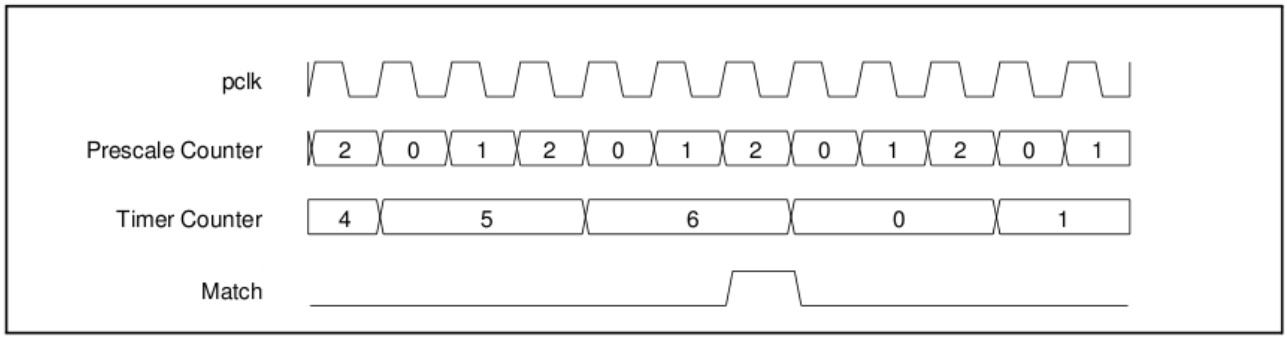


Figura 32: Match amb TC

El PR està posat a 2, i el MRX a 6. Per tant, fent servir la formula anterior comptant els tics de rellotge transcorreguts:

$$TC = pclk / preescaler = 18/(2+1) = 6$$

Un cop passats 18 tics de rellotge, amb el preescaler posat a 2, el TC val 6 i per tant fa match amb el registre MRX.

A la figura següent es pot veure el funcionament de l'eix simple i eix doble, i la senyal PWM5 representa el cas que es tractarà en aquest projecte, configurat amb eix simple:

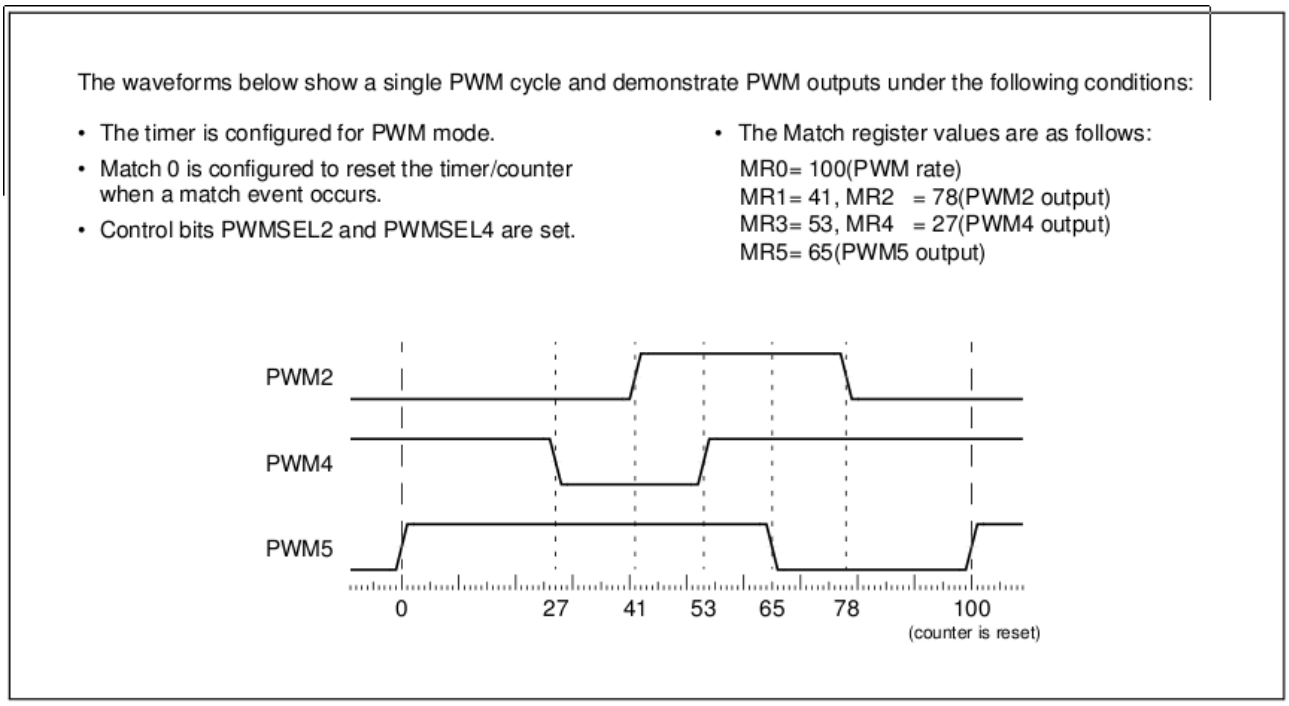


Figura 33: Exemple de PWM amb els registres

Com diu la figura, el registre MR0 està a 100 (que controla la pujada de flanc a cada període del PWM) i el MR5 en aquest cas controla la baixada del flanc (posat a 65).

Com es veurà en l'apartat de software, el control de PWM es farà mitjançant interrupcions (amb un registre anomenat PWMIR). D'això s'encarrega el bloc situat a l'esquerra de la figura anomenat "Control" que pot generar les interrupcions necessàries quan es produeix un match en un dels registres, a més de controlar el valor del TC, com per exemple aturar el comptatge o resetejar el valor.

Càlcul per generar les freqüències

En la introducció d'aquest capítol s'ha comentat com es generarà el PWM. L'objectiu és aconseguir generar una freqüència de so determinada i per fer-ho, és necessari conèixer tots els elements que participen en el càlcul d'aquesta freqüència, que inclou des del rellotge del processador fins als registres de control del PWM.

Com s'ha vist fa un moment, els registres TC i PR compten sobre el temps marcat per la senyal pclk, per tant s'ha d'averiguar d'on surt aquesta senyal i quin valor té.

Tot parteix del cristall oscil·lador de la plataforma. Com es pot intentar apreciar a la figura, posa S147ECSXR. El número 147 es refereix a que el cristall funciona a una freqüència d'oscil·lació de 14,7 Mhz, i se li dirà *Fosc*.

Figura 34: Cristall oscil·lador S147ECSXR

Seguidament de la senyal de rellotge del cristall, hi ha col·locat un bloc anomenat PLL (Phase Locked Loop). Aquest bloc es basa en la senyal controlada d'un oscil·lador, que es compara amb la senyal d'entrada i genera una nova senyal de sortida. Permet multiplicar o dividir aquesta freqüència, podent treballar així amb una freqüència de sortida des de 10MHz fins a 60MHz. Com que interessa tenir la resolució més alta possible, s'aprofitarà per multiplicar la freqüència per arribar al màxim permès de 60Mhz, per tant s'obtindrà *cclk* que és la freqüència de sortida al PLL:

$$cclk = F_{OSC} * M = 14745000 * 4 = 58980000\ Hz$$

Un cop passat el PLL, hi ha un altre bloc anomenat VPB divider. Representa que la senyal *cclk* ja és la que farà servir el processador, però per poder controlar elements externs a aquest, pot ser necessari reduïr aquesta freqüència si funcionen a altres velocitats. Per tant aquest bloc permet dividir la freqüència *cclk* i se n'obtindrà la senyal *pclk*, que ja és la utilitzen els registres de control del PWM. En aquest cas, el PWM no necessita rebaixar la freqüència, per tant aprofitarem el màxim valor per tenir una resolució més bona:

$$pclk = cclk = 58980000\ Hz$$

A continuació es veu un diagrama de la senyal *Fosc* fins a transformar-se en la senyal *pclk*:

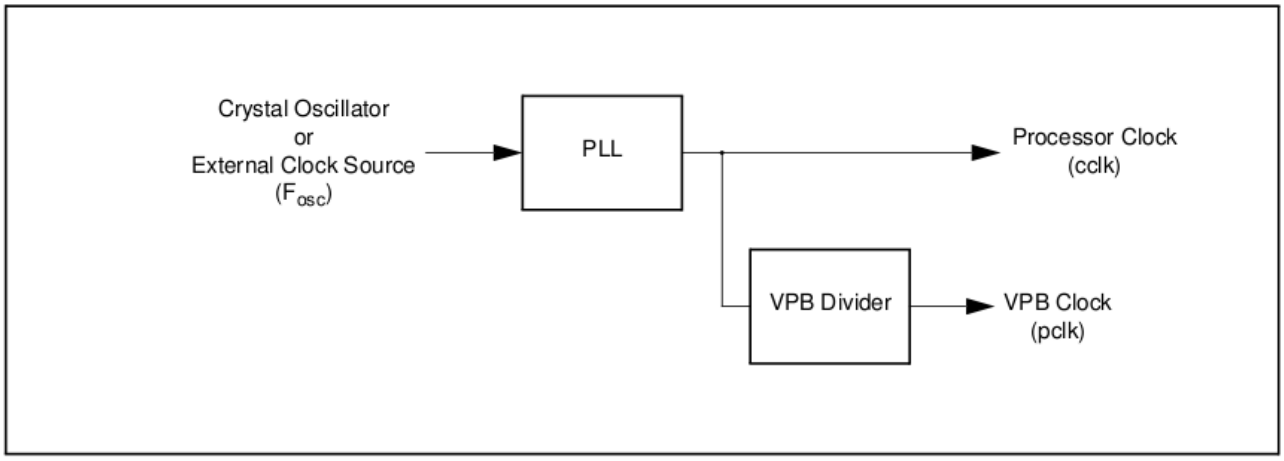


Figura 35: Origen de la senyal pclk

Ara es té el pclk, que és la freqüència de rellotge a la que comptaran els registres del PWM vistos anteriorment. Els registres que participen en el PWM són:

- PR (preescaler): Determina la divisió del pclk.
- MR0 (Match 0): Determina el període de cada pols de PWM.

I a més a més, per crear la ona sinusoïdal s'ha vist que es necessiten N polsos PWM per aconseguir la freqüència de la ona desitjada, per tant es tindrà una variable que es pot dir "resolució", que participa també en el càlcul de la freqüència, que és el següent:

$$F_{sinus} = pclk / PR / MR0 / resolució$$

A l'apartat de software es discutirà com ajustar aquests paràmetres, i amb quins criteris, per poder generar la millor ona possible.

3.5 Port sèrie

El port sèrie serà l'eina principal per poder-se comunicar amb la plataforma.

La comunicació sèrie funciona en base a dos paràmetres importants:

- Baud rate: Velocitat a la que transmet el port sèrie en bauds (símbols per segon). En aquest cas, cal ser configurat a una velocitat de 115200 bauds.

- mode: S'ha d'especificar la llargada del caràcter a transmetre (8 bits),si està habilitat el bit de paritat (en aquest cas no s'activa) que s'utilitza per la comprovació d'errors en les dades, i el bit de parada (1 bit). El mode s'anomena 8N1 (8 bits de paraula, no bit de paritat i 1 bit de parada).

Aquí es pot veure la funció *_cc3_uart0_setup* que configura els registres per el port sèrie:

```
void _cc3_uart0_setup(uint16_t baud, uint8_t mode, uint8_t fmode) {
  REG(PCB_PINSEL0) = (REG(PCB_PINSEL0) & ~UART0_PINMASK) | UART0_PINSEL;

  REG(UART0_IER) = 0x00;
  REG(UART0_IIR) = 0x00;
  REG(UART0_LSR )= 0x00;

  REG(UART0_LCR )= (1<<UART0_LCR_DLAB);
  REG(UART0_DLL )= (uint8_t)baud;
  REG(UART0_DLM )= (uint8_t)(baud >> 8);
  REG(UART0_LCR )= (mode & ~(1<<UART0_LCR_DLAB));
  REG(UART0_FCR) = fmode;
}
```

Les tres primeres línies són per netejar els dos registres d'interrupcions creades per el port sèrie (IER, IIR) i el registre LSR que conté la informació llegida del port sèrie.

El registre DLL i DLM es configuren per posar el baud rate (que s'especifica amb la part alta i baixa del valor) i el registre LCR és el que s'utilitza per configurar el mode 8N1.

Finalment, el registre FCR serveix per configurar la FIFO del port sèrie, que és una memòria interna de 16 bytes que permet emmagatzemar i controlar temporalment les dades que es reben o s'envien.

3.6 Alimentació

Un dels punts més importants del projecte és l'alimentació. Per dissenyar el sistema que alimentarà la plataforma, cal prendre la decisió tenint en compte que:

- L'autonomia ha de ser màxima
- La complexitat ha de ser mínima

Jugant amb aquest parell de paràmetres, s'ha d'aconseguit trobar la bateria ideal per fer funcionar el sistema. Per veure el consum de la plataforma, es referencia l'apartat del test de consum.

Figura 36: Bateria lipo

Hi ha molts tipus de bateries que es poden escollir, des de piles fins a bateries més grans i complexes. Les piles són molt petites, però no aporten suficient autonomia. Les que s'ha cregut que millor resultat poden donar són les bateries LIPO (liti - polímer). La particularitat d'aquestes és que tenen una mida reduïda, són de poc pes, i la densitat d'energia és força alta.

La plataforma necessita 5V per alimentar-se, i un inconvenient que s'ha trobat és que rarament es troben bateries de liti d'aquest voltatge, ja que les fabriquen per estàndards de 3.7V i 7.4V. Per tant s'ha escollit l'opció de comprar la de 3.7V i intentar augmentar aquest voltatge a part.

El DCDC és un circuit que permet, donat un voltatge indicat, proporcionar-ne un altre de més elevat, fent la funció d'amplificador de voltatge. S'ha comprat un DCDC amb aquestes característiques:

	Output / Current				
Input	5v	12v	18v	25v	34v
3.7v	1000mA	440mA	270mA	190mA	130mA
5v		780mA	480mA	320mA	210mA
6v		1400mA	640mA	400mA	370mA
12v			1700mA	900mA	620mA
18v				1900mA	1200mA
25v					1800mA

Figura 38: Taula voltatges DCDC

Figura 37: DCDC

En aquest cas, s'agafarà el voltatge de 3.7V que proporciona la bateria i s'amplificarà fins a 5V, que és el que necessita la plataforma. Per fer-ho, es connecta el DCDC a una font d'alimentació i, ajustant el cargol inferior esquerre (que varia el voltatge) i mesurant amb un tester, es va comprovant fins que la sortida de voltatge és la correcta.

Finalment s'ha comprat un carregador per la bateria LIPO. Com que el *ratio* de càrrega màxim és de 2.4A, i la capacitat és de 2400mAh, es carrega en una hora aproximadament.

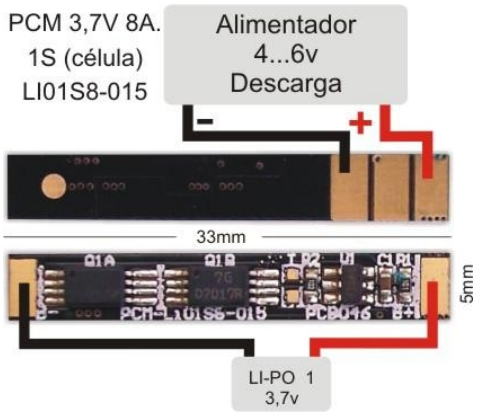


Figura 39: PCM

Només falta un punt per acabar de fer el muntatge. Les bateries són perilloses en el sentit de que, alhora de carregar-les, s'ha de controlar el flux de corrent i si supera la càrrega màxima, parar de carregar. Si s'endollés el carregador a la bateria, i no s'aturés passada la càrrega màxima, aquesta es començaria a inflar i esclataria en flames. També cal dir que si la bateria es descarrega del tot, es podria fer malbé.

Per evitar això, s'ha comprat un circuit anomenat PCM (Protection Circuit Module) que es col·loca entre la bateria i el carregador, i entre la bateria i el circuit, i talla la corrent en el moment de càrrega i descàrrega màxima. Així s'eviten els problemes comentats anteriorment.

Finalment, es mostra un esquema de les connexions per muntar el sistema d'alimentació:

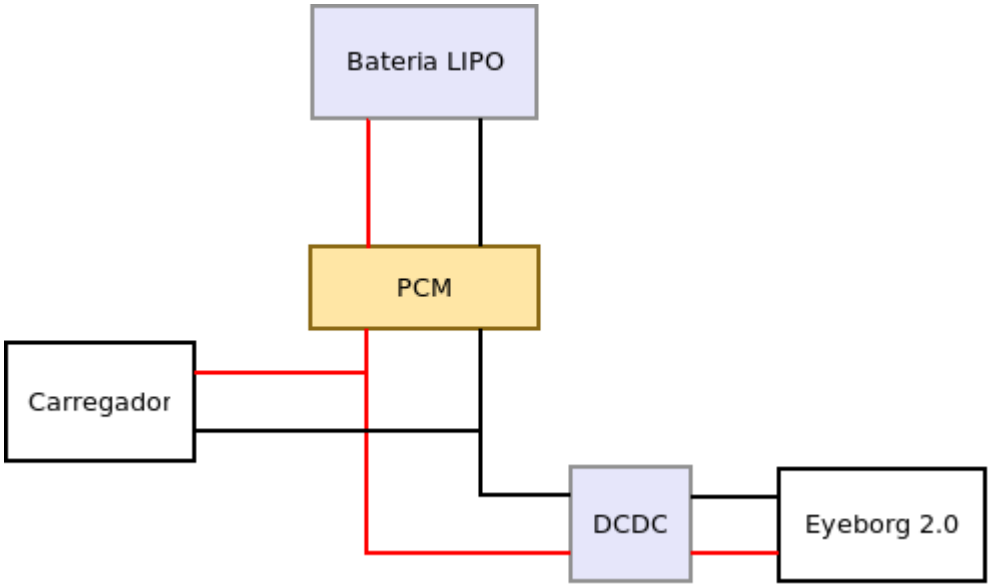


Figura 40: Muntatge del sistema d'alimentació

4 Funcionament software

En aquest apartat es començarà descrivint com es programa la plataforma. Després es veurà una visió general sobre el programa que s'anirà desglossant pas a pas per comentar de manera detallada tots els processos que conté. Finalment es veurà un apartat d'optimització, on s'ha aconseguit reduir el temps d'execució del programa.

4.1 Programació de CMUCam3

En l'apartat anterior, s'ha vist tot el funcionament del hardware, utilitzant com a referència la documentació del processador LPC2106. Per això, abans de començar a dissenyar i a programar per la plataforma, cal saber de quines eines es disposa per fer-ho.

Per començar, a l'annex I es descriu tot el procediment per preparar l'entorn de treball per aquesta plataforma. S'utilitzarà la carpeta "cc3" que conté una API especifica amb funcions per programar la CMUCam3, junt amb una guia de referència, tot proporcionat per els desenvolupadors de la plataforma.

A mesura que es descrigui el procés del programa, s'aniran mencionant certes funcions que fan referència a aquesta API (sovint amb el prefix cc3), i s'analitzaran per veure el seu funcionament.

Per consultar les funcions completes, cal dirigir-se a l'annex de codi font.

4.2 Visió global del Procés

Aquí es pot observar el següent esquema que il·lustra el procés sencer:

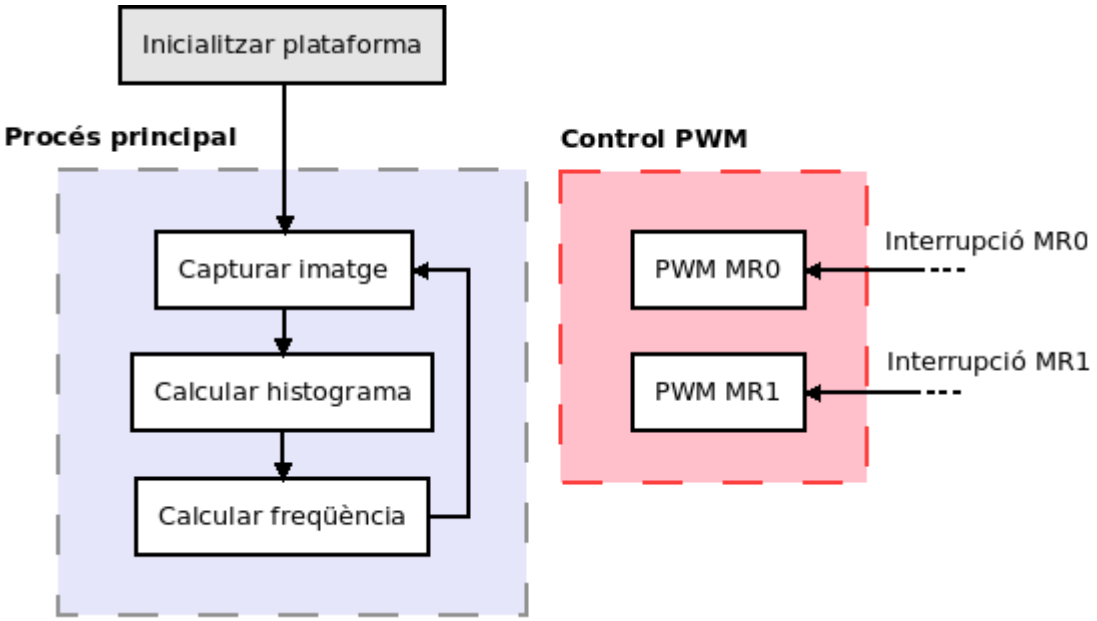


Figura 41: Procés global del programa

En general, el programa consta d'una inicialització del dispositiu i funciona executant un bucle infinit com a codi principal i externament, va rebent peticions d'interrupció que es controlaran en un bloc apart. El procés principal s'encarrega de capturar la imatge i processar el color. El bloc de control PWM, s'encarrega de modificar el PWM per generar l'àudio associat al to obtingut.

El primer bloc és la inicialització de la plataforma. Conté des de la inicialització de variables globals per el codi, fins a preparació física de registres per configurar la càmera.

Com que tant les variables globals com les inicialitzacions, són pròpies de cada procés que es descriurà a continuació, el bloc d'inicialització s'explicarà per fragments, mentre s'explica detalladament cadascun d'aquests processos.

4.3 Capturar imatge

Aquest és el procés principal del programa:

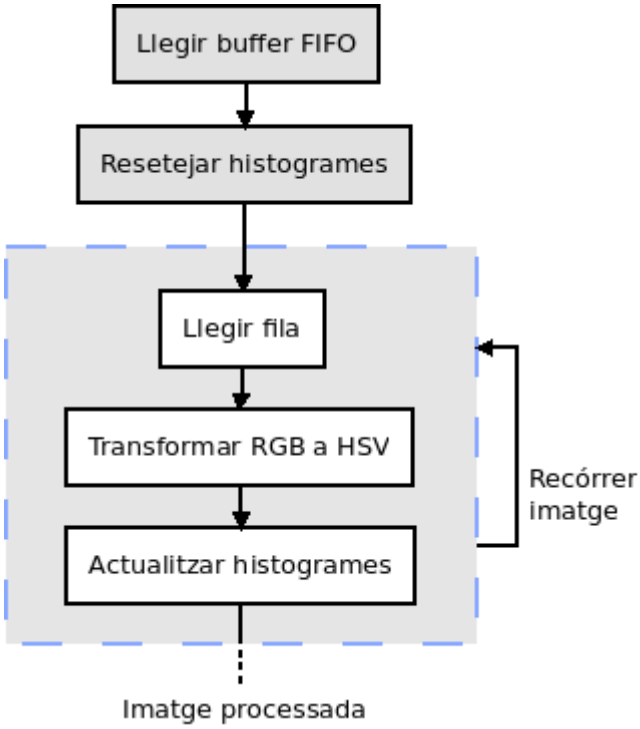


Figura 42: Procés global de la funció captura_imatge

L'objectiu d'aquest bloc és, donada una imatge presa per la càmera, llegir la imatge, transformar-la de RGB a HSV i obtenir la informació sobre els colors dels píxels.

Configurar càmera

Primer, al bloc d'inicialització de la plataforma, s'ha de configurar la càmera:

```
printf("\t> Camera... ");
if(cc3_camera_init()) printf("OK\n");
else {
    printf("Error!\n");
    return 0;
}
```

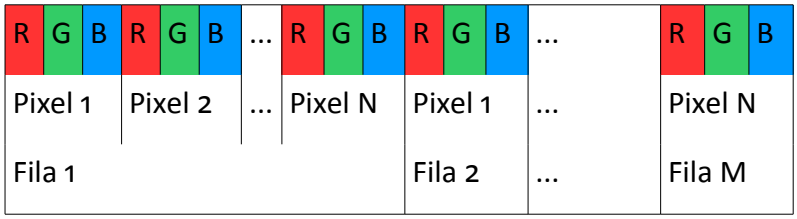
La càmera s'inicialitza amb la funció *cc3_camera_init* configurant els registres que s'han vist a l'apartat de hardware per inicialitzar la càmera i es fa un tractament d'error per comprovar que s'ha inicialitzat correctament.

Inicialització de variables

Com s'ha comentat a l'apartat d'obtenció del color, s'ha decidit obtenir un color significatiu mitjançant el valor màxim d'un histograma, fent servir el sistema de color HSV. Per tant necessitarem tres vectors per emmagatzemar els histogrames, un per cada component. També guardarem una variable extra per cada histograma, per accedir directament al valor màxim d'aquest:

```
int hue[361];
int maxhue = 0;
int sat[101];
int maxsat = 0;
int val[101];
int maxval = 0;
```

Abans de veure com s'inicialitzen les variables per llegir la imatge, s'ha de veure com s'estructurarà la imatge en si mateixa a memòria (aquesta és l'estructura que s'utilitza en el programa, no s'ha de confondre amb l'estructura que envia la càmera al buffer FIFO).



La imatge a memòria s'emmagatzema amb les components R,G,B de cada píxel una rere l'altre sempre en el mateix ordre, i fila per fila. Aquestes components es guarden en format 8 bits.

Per recórrer la imatge caldrà emmagatzemar-la a memòria, ja que no es pot accedir a tractar les dades directament al buffer FIFO. Per tant s'haurà de declarar un punter a una posició de memòria de tamany 8 bits (fent servir el tipus de C estàndard uint8_t):

```
uint8_t *row;
```

Llegir buffer FIFO

Per capturar una imatge nova, només és necessari actualitzar el buffer FIFO que s'ha explicat al capítol de hardware, on s'emmagatzema la imatge que es vol capturar en un moment concret:

```
cc3_pixbuf_load ();
```

Aquesta funció s'ha descrit també a l'apartat de hardware, explicant quin és el procés per carregar una imatge al buffer FIFO.

Resetejar histogrames

Cada cop que es captura una imatge, cal posar a zero tots els histogrames, per començar de nou el comptatge.

```
memset(hue,'\0',sizeof(hue));
memset(sat,'\0',sizeof(sat));
memset(val,'\0',sizeof(val));
```

Amb la funció memset estàndard de C, se li passa un punter a una direcció base i omple, amb el caràcter que li diem (en aquest cas omplir de zeros), tota una secció de memòria especificada amb la mida escollida (la mida dels vectors corresponents). Es podria fer tranquil·lament amb bucles, però amb aquesta funció queda més directe i s'estalvia la declaració variables extres.

Recórrer imatge

S'ha carregat la imatge al buffer FIFO i s'han inicialitzat els histogrames, ara esta tot llest per començar a processar la imatge. Per fer-ho, es recorrerà la imatge de la manera tradicional, per files i columnes. El codi és el següent:

```
for (uint16_t y = 0; y < cc3_g_pixbuf_frame.height; y++) {
    ...
    for (uint16_t x = 0; x < cc3_g_pixbuf_frame.width * 3U; x+=3) {
        ...
    }
}
```

La variable que compta files (variable y) la comparem amb l'altura de la imatge i la que compta columnes (variable x) amb l'amplada, utilitzant els camps *height* i *width* respectivament de l'estructura cc3_g_pixbuf_frame, que és una estructura global de les llibreries de la plataforma, que conté les dades de la imatge.

Llegir fila

La imatge es llegirà per files. En l'apartat d'optimització d'aquest capítol, hi ha una discussió sobre la quantitat de files òptim a llegir, que s'ha fixat finalment en un. Per tant, al bloc d'inicialitzar plataforma, caldrà reservar espai per una fila de la imatge, així:

```
row = cc3_malloc_rows(1);
```

Aquesta funció agafa el numero de canals (3 si és en color, 1 si és blanc i negre) i l'amplada de la imatge (cc3_g_pixbuf_frame.width) i retorna un punter a memòria utilitzant la funció estàndard de C malloc, multiplicant el numero de files demanades, per la seva amplada i el numero de canals de la imatge, per reservar una estructura tal com s'ha vist abans.

```
uint8_t *cc3_malloc_rows (uint32_t rows) {
  int channels = cc3_g_pixbuf_frame.channels;
  int width = cc3_g_pixbuf_frame.width;
  return (uint8_t *) malloc (width * channels * rows);
}
```

Un cop reservada la memòria al principi del programa, seguim amb la iteració del bucle. A cada

iteració del bucle de files, es llegirà una fila del buffer FIFO i s'emmagatzemarà a la memòria que s'ha reservat per llegir la imatge (variable row), de la següent manera:

```
cc3_pixbuf_read_row(row);
```

Aquesta funció que es veu a continuació, s'ha reduït a l'apartat d'optimització:

```
void cc3_pixbuf_read_row (uint8_t * mem)
{
    _cc3_second_green_valid = false;
    for (uint16_t j = 0; j < cc3_g_pixbuf_frame.width; j++) {
          uint8_t *p = mem + j * 3;
                _read_pixel (p, p - 3);
    }
}
```

Es recorre tota la memòria reservada per la fila de la imatge (que es passa per paràmetre), i a cada 3 posicions crida a la funció _read_pixel per llegir un píxel del buffer FIFO, que també ha estat treballada i optimitzada, i s'ha vist a l'apartat de hardware.

Transformació RGB a HSV

Un cop es té la fila llegida, es procedeix a transformar els valors de RGB a HSV, mitjançant la crida:

```
cc3_rgb2hsv_row((cc3_pixel_t*)row,cc3_g_pixbuf_frame.width);
```

Es passa el punter de la fila actual que es vol transformar, i la mida d'aquesta.

En l'apartat d'optimització s'analitza detingudament per dins aquesta funció.

Actualitzar histogrames

Un cop transformats els valors, s'han d'actualitzar els histogrames per poder obtenir finalment el valor màxim de cada component. La funció que transforma els colors donarà els valors de les components H, S i V en els intervals [0,255],[0,100] i [0,100]. En aquest cas, el valor i la saturació ja venen en el format que interessa. El to però, com es veurà a l'apartat d'optimització, no es pot donar en l'interval [0,360] per tant l'interval del to s'haurà de modificar aquí:

```
hue[(row[x]*360)/255]++;
sat[row[x+1]]++;
val[row[x+2]]++;
```

Un cop fet això, es procedeix a les següents files fins a tenir la imatge processada.

4.4 Calcular histograma

Ara s'ha d'agafar el valor màxim de cada histograma per obtenir el color significatiu de la imatge, això es fa amb un simple bloc de codi que recorre els histogrames i obté el valor màxim buscat:

```
void calculate_histogram() {
    int maxh = 0, maxs = 0, maxv = 0;
    for(int k=0; k<361; k++) {
        if(hue[k] > maxh) {
            maxh = hue[k];
            maxhue = k;
        }
    }
    for(int k=0; k<101; k++) {
        if(sat[k] > maxs) {
            maxs = sat[k];
            maxsat = k;
        }
        if(val[k] > maxv) {
            maxv = val[k];
            maxval = k;
        }
    }
}
```

4.5 Calcular freqüència

Finalment, un cop es té el color resultant, cal associar-lo a una freqüència de so determinada.

El primer que cal determinar és com s'obtindrà la freqüència desitjada. Si es recorda de l'apartat de hardware, s'ha explicat una formula per calcular aquesta freqüència en funció dels valors dels registres. El que s'ha de fer en realitat és, mirar la freqüència a la que es vol arribar, i modificar adequadament els registres per obtenir aquesta freqüència, ara bé, quins registres s'han de modificar?

Discussió sobre la funció dels registres

Es recorda la formula per calcular la freqüència, i es comenten les possibilitats que ofereix cada valor:

$$F_{sinus} = pclk / PR / MR0 / resolució$$

- Primer es fixarà el valor *Fsinus* per saber a quina freqüència es vol generar el to.
- *pclk* és un valor fix, degut a que és el rellotge del processador, i ja s'ha posat al màxim per obtenir la màxima resolució possible.
- *PR* és un registre a modificar, per dividir el temps de rellotge, no ha de tenir cap valor especial.
- *MR0* és el registre que marca el període del PWM i és molt important. Quan més gran sigui, es podran tenir més valors entremig per definir el duty cicle. Però quan més gran es fa, més es divideix la freqüència i per tant no es podran generar tons tant aguts.
- Resolució és una variable que permet definir els valors del duty cicle que s'utilitzaran per

generar la ona sinusoïdal. Quants més valors es tinguin, més resolució tindrà la senyal, però alhora, com el MR0, es faran més divisions a la freqüència i per tant no es podran generar tons tan aguts.

S'ha de pensar que es té un rang de freqüències a cobrir, ja que s'han de generar 7 tons, per tant és important que els valors es puguin moure de manera que es puguin arribar a generar tots. La limitació és, evidentment, el processament de la plataforma. S'ha de tenir en compte que tot el tractament de PWM es genera a través d'interrupcions, però per sota s'executa contínuament el codi de lectura d'imatges, que intenta ocupar el 100% de CPU per anar el més ràpid possible a capturar imatges.

Com s'ha vist doncs, el registre MR0 i la variable resolució s'han d'escollir en uns valors fixes per tal de tenir una bona resolució i poder generar tots els tons. S'ha provat amb diferents valors, tant observant la senyal en un oscil·loscopi com escoltant-la amb els auriculars, i s'han decidit els valors següents:

- MR0 = 10 : Permet dividir poc la freqüència ja que té un valor baix, i alhora permet resolució ja que el registre MR1, que marca el període del duty cicle, es defineix amb valors del 0 al 10 i es podrà apreciar el comportament del sinus.
- Resolució = 90 : Els valors del sinus es podrien prendre sobre cada grau de la circumferència (360 valors), però s'ha vist poca millora i auditivament és indistingible. Igualment s'ha prioritzat el nombre de punts de resolució sobre augmentar el període de MR0, ja que és preferible tenir quantes més mostres de la senyal millor.

Per tant, l'únic registre que es podrà modificar és el preescaler (PR) i serà la manera de generar diferents freqüències. Per veure la taula de freqüències del projecte, cal dirigir-se a l'apartat del test de color, tot i que les freqüències de la taula s'han modificat. El motiu és que les freqüències superiors als 500 Hz aproximadament amb els paràmetres definits, són impossibles d'assolir perquè el processador no és capaç de tractar les interrupcions a tant alta freqüència, i es penja. Per tant s'ha escollit dividir les freqüències per la meitat, de manera que generen els mateixos tons desitjats, sense crear cap problema a l'usuari, però a una octava més greus.

Posem el cas que es vol calcular el valor del PR per la freqüència del color groc (462 Hz dividit a la meitat), s'aïlla el valor de PR i es calcula:

$$PR = pclk / MR0 / resolució / F_{sinus} = 58980000 / 10 / 90 / (462/2) = \mathbf{284}$$

Això es repeteix per totes les freqüències, i els valors es posen en un vector:

```
int freq[7] = {360, //182 Hz - VERMELL
               298, //220 Hz - TARONJA
               284, //231 Hz - GROC
               274, //239 Hz - VERD
               238, //275 Hz - CIAN
               229, //286 Hz - BLAU
               216, //303 Hz - VIOLETA
              };
```

Un cop triats els valors, i utilitzant els llindars que es poden consultar al capítol de test de color, es genera el següent codi:

```
void calculate_frequency() {
    if(maxhue > 332) {
          REG(PWM0_PR) = freq[0];                    //VERMELL
    }else if(maxhue <= 11) {
          if(maxsat >= 70)
                REG(PWM0_PR) = freq[0];              //VERMELL
          else
                REG(PWM0_PR) = freq[1];              //TARONJA
    }else if(maxhue > 11 && maxhue <= 35) {
          REG(PWM0_PR) = freq[1];                    //TARONJA
    } else if(maxhue > 35 && maxhue <= 105) {
          REG(PWM0_PR) = freq[2];                    //GROC
    }else if(maxhue > 105 && maxhue <= 170) {
          REG(PWM0_PR) = freq[3];                    //VERD
    }
    else if(maxhue > 170 && maxhue <= 178) {
          if(maxval >= 59)
                REG(PWM0_PR) = freq[3];              //VERD
          else
                REG(PWM0_PR) = freq[4];              //CIAN
    } else if(maxhue > 178 && maxhue <= 213) {
          REG(PWM0_PR) = freq[4];                    //CIAN
    }else if(maxhue > 213 && maxhue <= 234) {
          REG(PWM0_PR) = freq[5];                    //BLAU
    }else if(maxhue > 234 && maxhue <= 244) {
          if(maxval >= 51)
                REG(PWM0_PR) = freq[6];              //VIOLETA
          else
                REG(PWM0_PR) = freq[5];              //BLAU
    } else if(maxhue > 244 && maxhue <= 332) {
          REG(PWM0_PR) = freq[6];                    //VIOLETA
    }
}
```

4.6 Control PWM

En aquest apartat es descriurà, com ja s'ha dit en l'apartat de hardware, el control per generar el PWM a través de software aprofitant les interrupcions hardware dels registres de match.

Com es pot veure a la figura de descripció del procés general, el bloc de control PWM és un bloc independent. Això és perquè no s'executa consecutivament amb el procés de càlcul de la imatge sinó que s'executa a través d'interrupcions puntuals.

El procediment per controlar el PWM serà capturar les interrupcions generades per un match en els registres MR0 i MR1, indicant el moment on es crearà el flanc de pujada i el de baixada respectivament. Un cop fet això, s'anirà modificant el duty cicle del PWM per crear la forma sinusoïdal i es modificarà la freqüència de la ona per canviar de to.

Punt d'entrada a les interrupcions

Per controlar el PWM, primer s'ha de veure des d'on es captaran les interrupcions a nivell software. Habitualment, el procediment per capturar una interrupció és declarar una funció de codi que s'executi cada cop que el processador llenci una interrupció. Després, aquella rutina es desglossa en una sèrie de condicions comprovant cada bit del registre d'interrupcions, per saber quina d'elles ha estat activada.

S'ha d'inserir un bloc de codi abans de la càrrega a memòria del programa principal "main" que faci saltar a la instrucció que conté l'inici de la rutina d'interrupcions. Aquí es pot veure el codi:

```
stmfd   sp!, { r0-r3, r12, lr }
bl      interrupt
ldmfd   sp!, { r0-r3, r12, lr }
```

Aquest codi apareix en el fitxer "startup.s" que és un fitxer de codi assemblador específic per l'ARM7TDMI-S. Bàsicament, la primera instrucció guarda l'estat dels registres de treball, la segona indica que s'ha de fer un "branch with link", un salt a la funció etiquetada com a "interrupt" que és la rutina on es tractaran les interrupcions, i la tercera finalment restaura els registres de treball. Per tant la rutina que es programarà per tractar les interrupcions és la funció interrupt.

Tractament de les interrupcions

Com s'ha anunciat a l'apartat de hardware, les interrupcions es materialitzaran en un registre anomenat PWMIR. Aquest registre informa per cada bit de les interrupcions que s'han produït. Per tant només caldrà comprovar el bit corresponent a la interrupció esperada, seguint la següent taula (només interessen les interrupcions generades per el match register 0 i 1):

PWMIR	Function	Description	Reset Value
0	PWMMR0 Interrupt	Interrupt flag for PWM match channel 0.	0
1	PWMMR1 Interrupt	Interrupt flag for PWM match channel 1.	0

Figura 43: Bits d'interrupció del PWM

La funció interrupt que s'està programant és part de les llibreries estàndard de la plataforma, per tant, perquè quedi tot més ordenat, el tractament de cada interrupció es farà en una funció pròpia del codi del projecte:

- Primer es cridarà una funció pròpia del codi del projecte que es dirà pwm_MR0 o pwm_MR1 segons el cas, i allà es farà el control del PWM.
- Després s'haurà de desactivar la interrupció, per fer-ho, segons l'especificació del registre, escrivint un 1 al bit de la interrupció, se li fa un reset.

```
void interrupt (void) {
    ...
    if (REG (PWM0_IR) == 0x2) {
          pwm_MR1();
```

```
            REG (PWM0_IR) = 0x2;
      }
      if (REG (PWM0_IR) == 0x1) {
            pwm_MR0();
            REG (PWM0_IR) = 0x1;
      }
}
```

Configuració inicial del PWM

Per controlar el PWM, s'han de configurar primerament tota una sèrie de registres. Això es farà en una funció anomenada config_pwm.

Primer cal activar les interrupcions del PWM al vector d'interrupcions (com s'ha vist a l'apartat d'interrupcions del hardware), per fer-ho s'activa el bit corresponent a les interrupcions del PWM, que és el bit 8 (es pot comprovar a la taula del vector d'interrupcions de l'apartat d'interrupcions a hardware):

```
REG (VICIntEnable) = 0x00000100;
```

Seguidament es presenta el registre PWMTCR:

PWMTCR	Function	Description	Reset Value
0	Counter Enable	When one, the PWM Timer Counter and PWM Prescale Counter are enabled for counting. When zero, the counters are disabled.	0
1	Counter Reset	When one, the PWM Timer Counter and the PWM Prescale Counter are synchronously reset on the next positive edge of pclk. The counters remain reset until TCR[1] is returned to zero.	0
2	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
3	PWM Enable	When one, PWM mode is enabled. PWM mode causes shadow registers to operate in connection with the Match registers. A program write to a Match register will not have an effect on the Match result until the corresponding bit in PWMLER has been set, followed by the occurrence of a PWM Match 0 event. Note that the PWM Match register that determines the PWM rate (PWM Match 0) must be set up prior to the PWM being enabled. Otherwise a Match event will not occur to cause shadow register contents to become effective.	0

Figura 44: Registre PWMTCR

Aquest registre configura el comptador TC del PWM. Primer de tot, es posa a 0 per deshabilitar el comptatge fins que no s'hagin configurat tots els registres:

```
REG(PWM0_TCR) = 0;
```

Els registres PWMTC i PWMPC són els comptadors corresponents al TC i al PR, per tant es posen a 0 per començar el comptatge:

```
REG(PWM0_TC) = 0;
REG(PWM0_PC) = 0;
```

Al preescaler se li posa el valor inicial del primer to per poder començar a comptar (que després s'anirà substituint per variar la freqüència del PWM):

```
REG(PWM0_PR) = freq[0];
```

I es configuren els match registers per controlar el flanc de pujada i baixada, inicialment en duty cicle 50%:

```
REG(PWM0_MR0) = 10;
REG(PWM0_MR1) = 5;
```

Com s'ha comentat a l'apartat de hardware, els valors dels match registers no es fan efectius fins que no s'activen els bits corresponents al latch enable register, que està descrit així:

PWMLER	Function	Description	Reset Value
0	Enable PWM Match 0 Latch	Writing a one to this bit allows the last value written to the PWM Match 0 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0
1	Enable PWM Match 1 Latch	Writing a one to this bit allows the last value written to the PWM Match 1 register to be become effective when the timer is next reset by a PWM Match event. See the description of the PWM Match Control Register (PWMMCR).	0

Figura 45: Registre PWMLER

Per tant s'han d'activar els dos bits de menys pes del registre per actualitzar MR0 i MR1:

```
REG(PWM0_LER) = 3;
```

A continuació, es configura el registre PWM_PCR, que conté la selecció d'eix simple o eix doble per la sortida desitjada i l'habilitació de la pròpia senyal.

PWMPCR	Function	Description	Reset Value
1:0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	PWMSEL2	When zero, selects single edge controlled mode for PWM2. When one, selects double edge controlled mode for the PWM2 output.	0
3	PWMSEL3	When zero, selects single edge controlled mode for PWM3. When one, selects double edge controlled mode for the PWM3 output.	0
4	PWMSEL4	When zero, selects single edge controlled mode for PWM4. When one, selects double edge controlled mode for the PWM4 output.	0
5	PWMSEL5	When zero, selects single edge controlled mode for PWM5. When one, selects double edge controlled mode for the PWM5 output.	0
6	PWMSEL6	When zero, selects single edge controlled mode for PWM6. When one, selects double edge controlled mode for the PWM6 output.	0
8:7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
9	PWMENA1	When one, enables the PWM1 output. When zero, disables the PWM1 output.	0
10	PWMENA2	When one, enables the PWM2 output. When zero, disables the PWM2 output.	0
11	PWMENA3	When one, enables the PWM3 output. When zero, disables the PWM3 output.	0
12	PWMENA4	When one, enables the PWM4 output. When zero, disables the PWM4 output.	0
13	PWMENA5	When one, enables the PWM5 output. When zero, disables the PWM5 output.	0
14	PWMENA6	When one, enables the PWM6 output. When zero, disables the PWM6 output.	0
15	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Figura 46: Registre PWMPCR

Com que es vol la sortida PWM1 controlada per eix simple i habilitada, s'activarà només el bit 9 que habilita la senyal, ja que la sortida PWM1 només es pot configurar d'eix simple.

```
REG(PWM0_PCR) = 0x0200;
```

A continuació, s'ha de configurar el registre PWMMCR, que controla les interrupcions i el comportament del TC. Aquí es pot veure l'esquema:

PWMMCR	Function	Description	Reset Value
0	Interrupt on PWMMR0	When one, an interrupt is generated when PWMMR0 matches the value in the PWMTC. When zero this interrupt is disabled.	0
1	Reset on PWMMR0	When one, the PWMTC will be reset if PWMMR0 matches it. When zero this feature is disabled.	0
2	Stop on PWMMR0	When one, the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR0 matches the PWMTC. When zero this feature is disabled.	0
3	Interrupt on PWMMR1	When one, an interrupt is generated when PWMMR1 matches the value in the PWMTC. When zero this interrupt is disabled.	0

Figura 47: Registre PWMMCR

Com que el registre MR0 controla el període del PWM, es farà un reset al TC cada cop que hi hagi un match 0. Per altra banda, s'activaran les interrupcions del MR0 i MR1 cada cop que hi hagi un match, per executar el codi corresponent. Per tant el registre tindrà aquest valor:

```
REG(PWM0_MCR) = 0x0B;
```

Finalment, els valors del registre PWMTCR es carregaran adequadament per activar el comptatge i iniciar el control del PWM. Per fer-ho, s'activarà el bit 0 que activa els comptadors TC i preescaler i també el bit 3, que activa el mode de PWM (i habilita els shadow registers vistos a l'apartat de hardware per passar els valors a la lògica de control).

```
REG(PWM0_TCR) = 9;
```

Un cop definits tots els registres, cal controlar una variable que indica que s'ha completat la configuració del PWM i que el sistema està llest per executar-se. Les interrupcions, al estar declarades abans que el programa principal, es poden executar abans de que ho faci aquest. Per assegurar que les interrupcions només es comencen a executar un cop s'estigui dins el programa principal, es declara aquesta variable que actua com a bandera:

```
eyeborg_init = true;
```

Generar la ona sinusoïdal

Aquest és l'últim pas que permetrà controlar el PWM. Com s'ha vist abans, a cada interrupció de MR0 i MR1, es fa una crida a les funcions pwm_MR0 i pwm_MR1 respectivament. Es tracta de veure què s'ha de fer a cadascuna d'elles per generar la ona sinusoïdal.

Primerament, caldrà tenir els valors adequats per el duty cicle, per generar aquesta ona. Així doncs, es guardaran els valors corresponents a la funció sinus però adaptats per entrar dins l'interval màxim que controla el MR0, és a dir 10. Com que es tenen 90 valors, es tindrà un valor del sinus per cada 4 graus de la circumferència. A continuació es mostra el càlcul per cada grau:

$$valor[i] = (\sin(grau_i) \cdot 5) + 5$$

Amb això s'aconsegueix tenir un vector de 90 posicions, on de cada grau es calcula el sinus i es porta el valor sobre l'escala de 5 per moure's en l'interval de 0 a 10.

Un problema és que els valors dels registres MR0 i MR1 no poden estar junts, és a dir, no poden valdre 10 i 10, i per seguretat, 9 i 10 tampoc, perquè estan molt a prop l'un de l'altre i les interrupcions es generen massa a prop i la placa es penja (això s'ha comprovat amb diversos valors tant de MR0 com de MR1). Per això s'agafen tots els valors del vector que siguin 0, 1, 9 i 10 (que estan a dos nombres del mínim i màxim del període de MR0) i es canvien per 2, 2, 8 i 8 respectivament. El vector queda així:

```
int duty[90] =
{5,5,6,6,6,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,7,7,7
,7,6,6,6,5,5,5,4,4,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,2,2,2,2,2,2,2,3,3,3,4,4,5};
```

Per controlar el valor de l'angle actual, que produirà el duty cicle que s'esta generant pel PWM, es declararà una variable anomenada angle:

```
int angle = 0;
```

A continuació es pot veure el codi generat en la interrupció del flanc de pujada (Match 0):

```
void pwm_MR0() {
    if(eyeborg_init) {
          REG(GPIO_IOSET) = _CC3_SERVO_0;
          REG(PWM0_MR1) = duty[angle];
          REG(PWM0_LER) = 2;
          angle++;
          if(angle >= 90) {
                angle = 0;
          }
    }
}
```

Cada cop que es completi el període controlat per MR0, s'ha de posar la senyal a 1, per marcar el flanc de pujada. Això es fa posant un bit del registre GPIO_IOSET a 1, corresponent a la sortida de PWM que es vol activar, en aquest cas la 1, mitjançant la constant _CC3_SERVO_0.

Seguidament, es posa el valor actual del sinus al registre MR1, que marcarà el duty cicle, mitjançant el vector que s'ha definit abans, i després s'activa el penúltim bit del registre PWM0_LER, que com s'ha vist anteriorment, fa que el valor de MR1 es faci efectiu.

Per acabar, s'incrementa el comptador de l'angle i es controla el final de vector per posar-lo a 0 quan arriba al final.

La condició amb la variable bandera *eyeborg_init* s'utilitza per evitar que es tractin interrupcions abans d'acabar d'inicialitzar la càmera.

Pel que fa la interrupció del flanc de baixada només cal fer el següent:

```
void pwm_MR1() {
    if(eyeborg_init) REG(GPIO_IOCLR) = _CC3_SERVO_0;
}
```

Bàsicament es genera el flanc de baixada, no cal fer res més.

4.7 Comunicació sèrie

És necessari establir una comunicació via port sèrie amb la plataforma, d'aquesta manera es podran enviar missatges d'estat a un dispositiu extern que vulgui escoltar què passa a l'eyeborg. El port sèrie serà la via de comunicació estàndard amb l'exterior.

Per fer-ho s'utilitza la funció *cc3_uart_init* que requereix els paràmetres que ja s'han comentat per configurar el port sèrie.

```
cc3_uart_init(0,CC3_UART_RATE_115200,
                CC3_UART_MODE_8N1,
                CC3_UART_BINMODE_TEXT);
```

Aquesta rutina internament crida a la funció *_cc3_uart0_setup*, comentada a l'apartat de hardware que configura els registres pertinents.

4.8 Optimització

Un cop s'ha tingut el codi en estat funcional, sorgeix un problema considerable. La funció de capturar imatge que s'ha vist del procés principal, triga 730 mili-segons a executar-se. Això és un inconvenient real, perquè alhora de captar el color, no pot ser que es trigui gairebé un segon a detectar el pròxim to.

Per això, s'ha d'analitzar a fons el codi des d'una perspectiva d'optimització i veure quins són els punts conflictius de codi a on es pot reduir el temps d'execució. Es centrarà la discussió només a la funció de capturar imatge del procés principal, ja que les altres no tenen rellevància a nivell de codi.

Les funcions en el codi final ja estan optimitzades. En les explicacions que es faran, s'aniran mostrant fragments de codi de les funcions no optimitzades, s'explicarà què feien i es comentarà l'optimització aplicada. Per veure les funcions sense optimitzar, cal veure l'annex de funcions no optimitzades.

Optimització de reducció de la imatge

La primera pregunta que es pot plantejar és: Per obtenir el color majoritari, que és l'objectiu principal, cal tenir la resolució màxima de la càmera? Dit d'una altra manera, es podria reduir la mida de la imatge?

Aquesta és la primera optimització que s'ha dut a terme, i la resposta és si, es pot reduir la mida de la imatge i podem obtenir els mateixos resultats de color i reduir el temps d'execució.

La càmera té dos configuracions de resolució que venen per defecte:

- Màxima: 352x288
- Mínima: 176x144

La documentació de referència del sensor Omnivision diu que la diferència entre les dos resolucions és que la màxima agafa totes les dades que capta la càmera, i la mínima simplement fa el mateix però saltant-se un pas a cada iteració (tant de files com de columnes), obtenint així una imatge amb la mateixa qualitat de color i lluminositat, però amb una mida reduïda a la meitat. Aquesta és una optimització doncs molt potent, perquè tot el càlcul necessari per 352 columnes i 288 files és redueix a la meitat per cada dimensió.

Per fer que la càmera tingui la resolució mínima, simplement s'ha de cridar a la funció:

```
cc3_camera_set_resolution (CC3_CAMERA_RESOLUTION_LOW);
```

S'ha aconseguit una optimització que passa dels 730 ms inicials a 195 ms.

Optimització en llegir les files i els píxels

El problema que pot donar utilitzar les llibreries ja fetes de la plataforma, és que són massa genèriques. Aquest és el cas de les funcions *_cc3_pixbuf_read_rows() i _cc3_read_pixel()*.

cc3_pixbuf_read_rows() és, la que llegeix un seguit de files de píxels. La versió original de les llibreries no optimitzada està dissenyada per fer les següents funcions:

- Contemplar diversos sistemes de color (RGB, YcbCr)
- Fer *cropping* de la imatge
- Llegir per canals de color (R,G,B)
- Llegir múltiples files

Per accedir als píxels, depenent del sistema de color, s'utilitzen uns determinats *offsets* que ordenen les components del píxel (ja que la càmera emmagatzema les components en ordre diferent depenent del sistema de color escollit).

```
if (_cc3_g_current_camera_state.colorspace == CC3_COLORSPACE_RGB) {
    off0 = 0; off1 = 1; off2 = 2;
}else if (_cc3_g_current_camera_state.colorspace == CC3_COLORSPACE_YCRCB) {
    off0 = 1; off1 = 0; off2 = 2;
}else{
    off0 = 0; off1 = 1; off2 = 2;
}
```

Com que en el projecte sempre es fa servir RGB, s'assignen els *offsets* a 0,1,2 respectivament a les variables off0, off1, off2. Per tant aquest bloc s'elimina.

El *cropping* és l'altre funcionalitat que no és necessària, no és res més que preparar uns offsets sobre la x i la y de la imatge (amb una amplada i alçada extres) que permeten definir un rectangle sobre aquesta i retallar-la, és a dir, agafar només una part que interessi tractar.

Com que en aquest cas sempre interessa tota la imatge, podem esborrar tota la part de *cropping* de la funció, aquí es veuen alguns fragments que es poden eliminar:

Declaració d'una x que controla que no passem del recuadre definit.

```
    int x = cc3_g_pixbuf_frame.x0;
```

Hi ha unes variables que permeten controlar si ens saltem la lectura d'alguna fila o columna, anomenades “y_step” i “x_step” respectivament. En el projecte es llegeixen files i columnes seguides, per tant no cal fer cap comprovació sobre això, les variables valen sempre 1. Per exemple la següent linia, calcula si s'ha de saltar una quantitat de píxels concreta (en x i en y). Com que sempre dona 0 la resta, podem eliminar les crides a la funció *_cc3_pixbuf_skip_pixels*:

```
_cc3_pixbuf_skip_pixels ((cc3_g_pixbuf_frame.x_step - 1) / 2);
_cc3_pixbuf_skip_pixels ((cc3_g_pixbuf_frame.y_step - 1) *
                         cc3_g_pixbuf_frame.raw_width / 2);
```

A més a més, hi ha un switch que comprova si sempre es demanen tots els canals de color, o

només un de concret. Com que en el projecte sempre es demana RGB, es tracta només aquest cas i es pot eliminar el switch:

```
    switch (cc3_g_pixbuf_frame.coi) {
    case CC3_CHANNEL_ALL:
    case CC3_CHANNEL_RED:
    case CC3_CHANNEL_GREEN:
    case CC3_CHANNEL_BLUE:
```

I per acabar, com es veurà més endavant, no necessitem llegir més d'una fila quan es fa la crida a aquesta funció, per tant s'elimina el bucle de les files que es veu a continuació, i el càlcul de la direcció de memòria es fa només sobre les columnes:

```
  for (r = 0; r < rows; r++) {
      for (j = 0; j < width; j++) {
        uint8_t *p = ((uint8_t *) mem) + (r * width + j * 3);
```

Cal notar que aquesta part va donar certs problemes, ja que estava mal programada per part dels desenvolupadors de la plataforma. El càlcul de la direcció de memòria estava calculat així:

```
uint8_t *p = ((uint8_t *) mem) + (r * width + j * 3);
```

*I en realitat les files estaven mal comptades, doncs no és r*width, sinó r*width*3 ja que el píxel de cada fila té 3 bytes.*

cc3_pixbuf_read_pixel és la funció que llegeix un píxel i el carrega a la memòria:

```
_cc3_pixbuf_read_pixel (p, p - 3, off0, off1, off2);
```

Es poden treure els *offsets* com a paràmetre, ja que sempre són els mateixos, i a dintre la funció, hi ha una condició que comprova si *x_step* és diferent a 1 (per el *cropping*) per tant es pot treure també:

```
  if (cc3_g_pixbuf_frame.x_step == 1) {
```

Feta aquesta optimització, s'ha arribat a un temps d'execució que passa de 195ms amb l'optimització anterior a 188ms de mitja.

Optimització transformació RGB a HSV

La funció de transformació cc3_rgb2hsv s'executa també cada cop que es captura una imatge, per tant és interessant fer-hi un anàlisi detingut per veure si es pot escurçar el temps d'execució.

L'algoritme, sense mirar el codi, pretén fer els següents passos:

- Calcular el valor mínim i màxim entre les components RGB
- Computar V com a valor màxim

- Computar S com a quocient entre la diferència del màxim i el mínim i V.
- Computar H com a desplaçament en el cercle de to, utilitzant uns *offsets*.

El codi per trobar el mínim i el màxim és el següent:

```
uint8_t rgb_min, rgb_max;
rgb_max = 0;
rgb_min = 255;
if (pix->channel[CC3_CHANNEL_RED] > rgb_max)
  rgb_max = pix->channel[CC3_CHANNEL_RED];
if (pix->channel[CC3_CHANNEL_GREEN] > rgb_max)
  rgb_max = pix->channel[CC3_CHANNEL_GREEN];
if (pix->channel[CC3_CHANNEL_BLUE] > rgb_max)
  rgb_max = pix->channel[CC3_CHANNEL_BLUE];
if (pix->channel[CC3_CHANNEL_RED] < rgb_min)
  rgb_min = pix->channel[CC3_CHANNEL_RED];
if (pix->channel[CC3_CHANNEL_GREEN] < rgb_min)
  rgb_min = pix->channel[CC3_CHANNEL_GREEN];
if (pix->channel[CC3_CHANNEL_BLUE] < rgb_min)
  rgb_min = pix->channel[CC3_CHANNEL_BLUE];
```

A primera vista, dona la sensació que és molt ineficient. Ha de comprovar totes les condicions (no hi ha blocs *else*), i fa comprovació per cadascuna de les components amb el mateix valor mínim o màxim.

S'ha provat de fer un canvi en el codi comprovant entre components RGB, en comptes de comparar amb mínim o màxim. Però l'únic que s'aconsegueix és tenir un codi més complex i, tot i que fa 3 comprovacions menys, com que hi ha salts per culpa dels blocs *else* ha de fer l'instrucció de salt i va igual de lent que l'anterior, per tant s'ha deixat aquest codi.

El que realment ha aconseguit rebaixar el temps d'execució ha sigut la reducció de constants de multiplicació en els tres paràmetres H, S i V. Aquesta funció està preparada per donar els valors de H S i V de 0 a 255, però són més comuns de veure entre els intervals [0,360],[0,100] o [0,100].

Per fer aquesta conversió, en el bucle principal de la funció capturar imatge es feia la transformació mitjançant una regla de tres simple, però això significava fer més multiplicacions de les necessàries. Per això s'han agafat les constants de multiplicació i s'han modificat perquè deixin els valors en els intervals desitjats:

Valor és un valor entre 0 i 255, per tant es modifica així perquè quedi entre 0 i 100:

```
val = (rgb_max * 100) / 255;
```

El càlcul de la saturació dona un valor entre 0 i 1, només cal multiplicar per 100:

```
sat = 100 * (rgb_max - rgb_min) / val;
```

El to no es pot optimitzar com els altres, degut a que el valor de la component s'escriu a la direcció de memòria que ve per paràmetre, que és de 8 bits. Com que el to arriba fins a 360, no es podria

escriure el valor si supera els 255. Per tant s'ha de deixar igual, i fer la conversió a 360 fora la funció.

Amb aquesta optimització, s'ha reduït el temps d'execució de 188ms aconseguits anteriorment, a 176ms de mitja.

Altres optimitzacions no vàlides

S'han intentat dos optimitzacions més, però no han sorgit efecte, tal com s'exposa a continuació:

La primera ha sigut intentar provar en modificar el nombre de files a llegir. En comptes de llegir-ne una, llegir-les totes de cop, o llegir la meitat de files de la imatge. Aquesta proposta s'ha tirat enrere perquè, per començar, la imatge sencera no hi cap a memòria. Això es demostra seguint la imatge que es veu a continuació:

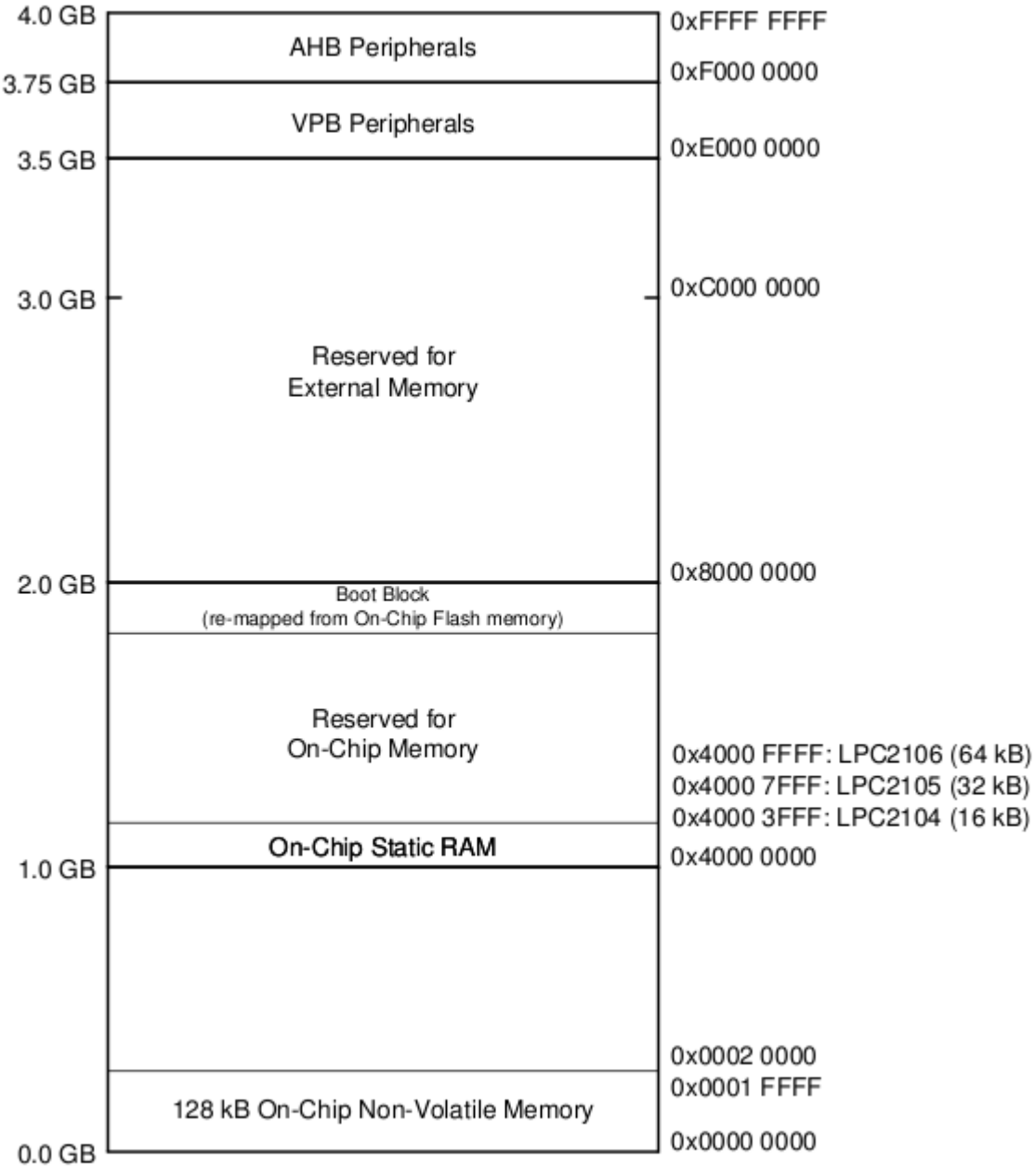


Figura 48: Memòria del LPC2106

L'adreça base per carregar variables del programa és la 0x40000000, però ja esta força ocupada per les variables de totes les funcions de les llibreries de la plataforma, per això l'adreça del nostre punter per guardar la imatge és a la 0x40003658. A partir d'aquí, sumem la mida de la imatge en bytes: 0x40003658 + 144*176*3 = 0x40015F60, i es veu que sobrepassa el límit de la memòria del LPC2106 (64 kB) que arriba només a 0x4000FFFF. Efectivament, provant el codi, dona un senyal de “data abort” quan sobrepassa l'adreça màxima.

La idea doncs podria ser guardar mitja imatge, però els resultats tampoc són molt millors, ja que partint les iteracions només s'aconsegueix reduir el nombre d'instruccions de salt, i com que tampoc hi ha tantes iteracions, el canvi és insignificant.

La segona optimització que s'ha intentat és la següent:

En el bucle de la funció capturar imatge, es recorre cada fila de la imatge dues vegades. La primera per llegir els píxels i assignar-los a memòria, la segona per transformar les dades de RGB a HSV. La idea de l'optimització seria recórrer la fila un sol cop, fent les dues operacions a la vegada, per estalviar el doble recorregut de la fila.

El problema principal és que en la lectura dels píxels, com s'ha vist en l'explicació de com llegir els píxels que envia la càmera, s'accedeix cada dues iteracions al píxel anterior, i això fa que si un cop llegit el píxel actual, se li fa la transformació de RGB a HSV, quan des del següent píxel s'accedeix a l'anterior, aquest no esta en els valors correctes (perquè ha estat transformat) i no es fa bé el procés de lectura del píxel.

Per solucionar això, s'ha de guardar el píxel transformat en una variable a part, per no sobreescriure la fila de memòria que s'utilitza per llegir els píxels, així es pot continuar accedint als píxels anteriors sense problema.

Però encara que així eliminem el doble recorregut de la fila, el fet de que a cada iteració s'hagi de copiar el píxel a una variable temporal, fa que torni a augmentar el temps d'execució, i no s'aprecia gens la diferència, per tant no és una optimització vàlida.

Un cop explicades les optimitzacions, es pot visualitzar a continuació un resum d'aquestes, amb les millores produïdes:

Optimització	Temps
Codi inicial	730ms
Reducció imatge	195ms
Funcions llegir fila i llegir píxel	188ms
Transformació RGB a HSV	176ms

Per veure l'increment de millora degut a la optimització, utilitzem el paràmetre Speed-up que és simplement la divisió del temps inicial entre el temps aconseguit:

$$S_p = \frac{T_{ini}}{T_{opt}} = \frac{730\,ms}{176\,ms} = \mathbf{4{,}15}$$

La reducció més radical com s'ha vist és la reducció de la imatge. Tot i que les altres dos optimitzacions també han reduït força, per tant s'ha aconseguit un temps d'execució de 4 vegades més ràpid que l'inicial, sense reduir cap funcionalitat del projecte.

5 Test de color

Com s'ha vist en l'apartat anterior, a l'hora de calcular el color s'han utilitzat uns paràmetres determinats. La pregunta és: Com s'han obtingut?

5.1 Escala sonocromàtica pura de Harbisson

L'associació de colors a freqüències de so, és més una qüestió artística que científica. Cadascú podria interpretar un color amb un cert to, segons el seu criteri. Neil Harbisson ha decidit utilitzar el que ell ha anomenat "Escala sonocromàtica pura de Harbisson", que es defineix de la següent manera:

Nom color	RGB (Hex)	Freqüència so (Hz)
Infraroig	---	Inferior a 363,79
Vermell	#FF0000	363,79
Taronja	#FF7F00	440,19
Groc	#FFFF00	462,02
Verd	#00FF00	478,39
Cian	#00FFFF	551,15
Blau	#0000FF	573,89
Violeta	#7F00FF	607,54
Ultra-violeta	---	Superior a 717,591

El rang d'infraroig i ultra-violeta es descarta per la impossibilitat de la càmera a captar aquest tipus de freqüències de llum. Així que s'utilitzarà només la gamma de color visible definida per 7 colors

5.2 L'escala en els tons reals

Un cop vist el que es vol aconseguir amb l'escala sonocromàtica, i vist que es farà servir el sistema HSV per tractar el color, s'ha de pensar com s'integrarà aquesta escala al funcionament real de la càmera.

Els tons es defineixen mitjançant uns llindars, amb valor d'angle en el cercle de to del sistema HSV. La primera aproximació teòrica seria crear aquests llindars, a partir de cada angle del to pur que esta definit a l'escala:

Nom color	Hue (graus)
Vermell	0º
Taronja	30º
Groc	60º
Verd	120º
Cian	180º
Blau	240º
Violeta	270º

Evidentment però, aquesta divisió no funciona. La càmera no segueix aquesta associació tant

directe, ja que la lent no capta els colors tant vius, té distorsió degut al tipus de lluminositat, i moltes altres variacions que s'han de tractar. Per tant, per començar s'hauria d'aconseguir veure quins són els tons “reals” de l'escala vistos per la càmera, i a partir dels valors que s'obtinguin, poder acotar els llindars.

El primer procediment que s'ha seguit és crear una paleta de colors amb l'escala definida anteriorment, per poder captar el valor que dona la càmera per cada color. S'han imprès exactament els colors definits a l'escala, per tenir com a referència a l'hora de fer les proves.

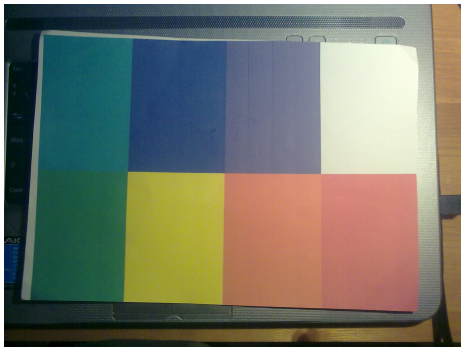


Figura 49: Paleta de colors

Un cop creada la paleta, la idea és és obtenir una quantitat important de dades sobre cadascun dels colors, per poder realitzar una espècie d'aprenentatge manual, és a dir, sabent a priori que cada imatge presa correspon a un color concret.

El principal problema però, com ja s'ha comentat, és que el valor de cada color té variabilitat depenent, per exemple de l'entorn, així com la lluminositat que l'influeix. Per tant, s'han pres un total de 30 fotografies, cadascuna d'aquestes s'ha pres a llocs diferents, amb il·luminacions diferents, concretament a 5 entorns:

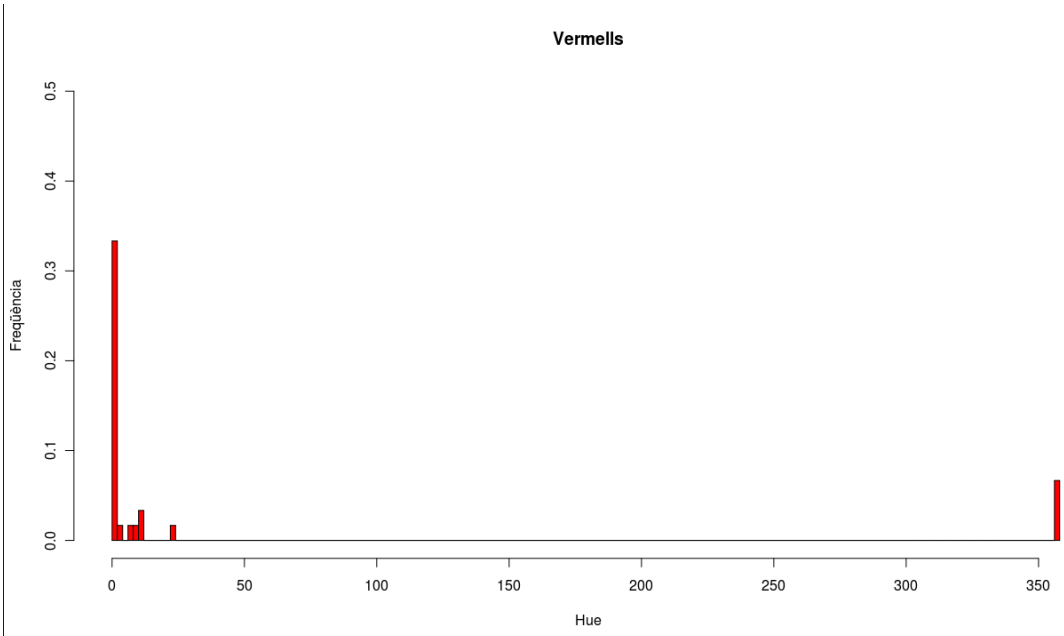
- Llum artificial blanca
- Llum artificial groga
- Llum natural en un dia núvol a exterior
- Llum natural en un dia núvol a interior
- Llum natural en un dia de sol a exterior

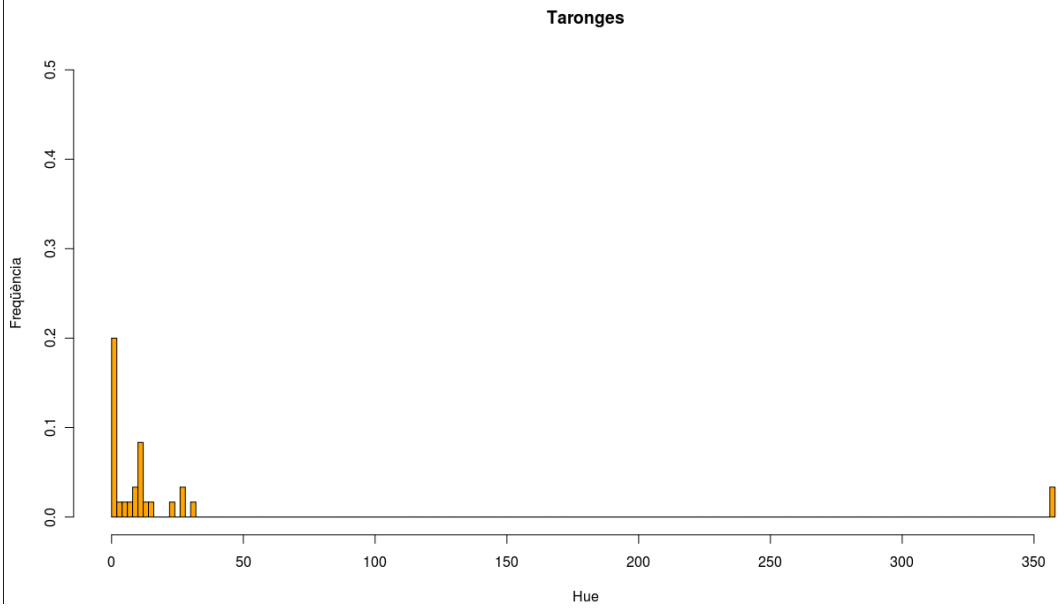
Amb això s'ha aconseguit una gran variabilitat, ja que són entorns molt diferents i s'ha posat el tipus de llum als extrems més oposats (dies amb molta llum o amb molta ombra, que proporcionen diferents lluminositats, i llocs amb llum ambient groga o blanca, per canviar la tonalitat de la llum a les imatges), tot això per comprovar exactament quin rang de valors comprèn cada color. Les dades preses es poden consultar a l'annex de dades del test de color.

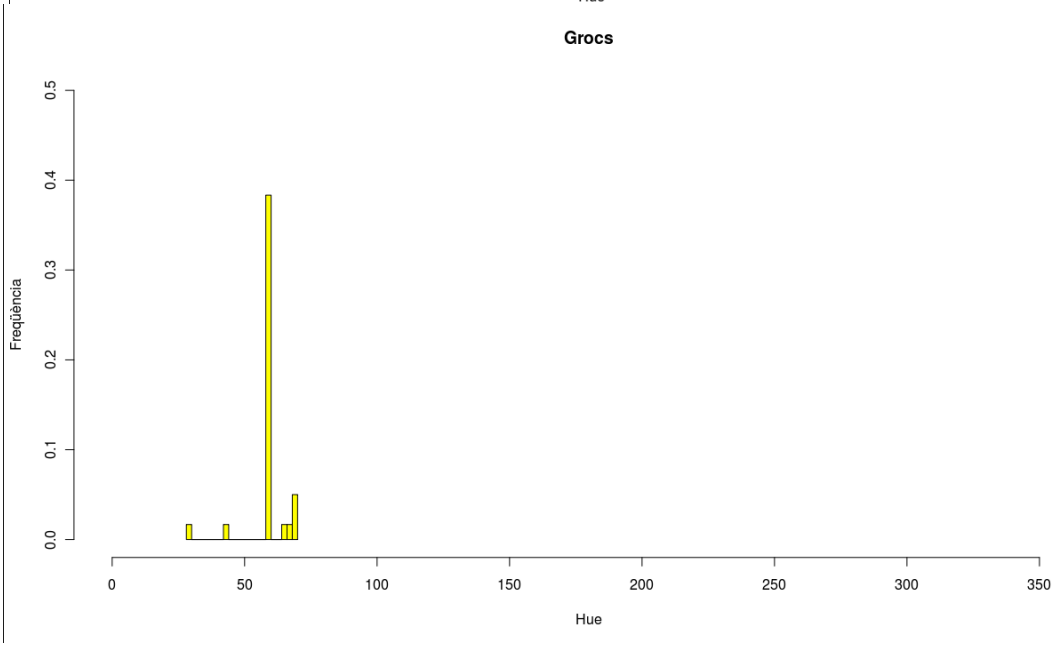
Un cop fetes les fotos, s'ha d'estudiat quina és la manera d'obtenir millors representacions i poder analitzar bé les dades. S'ha realitzat un estudi detallat per poder trobar exactament els llindars que es busquen, en diferents passos:

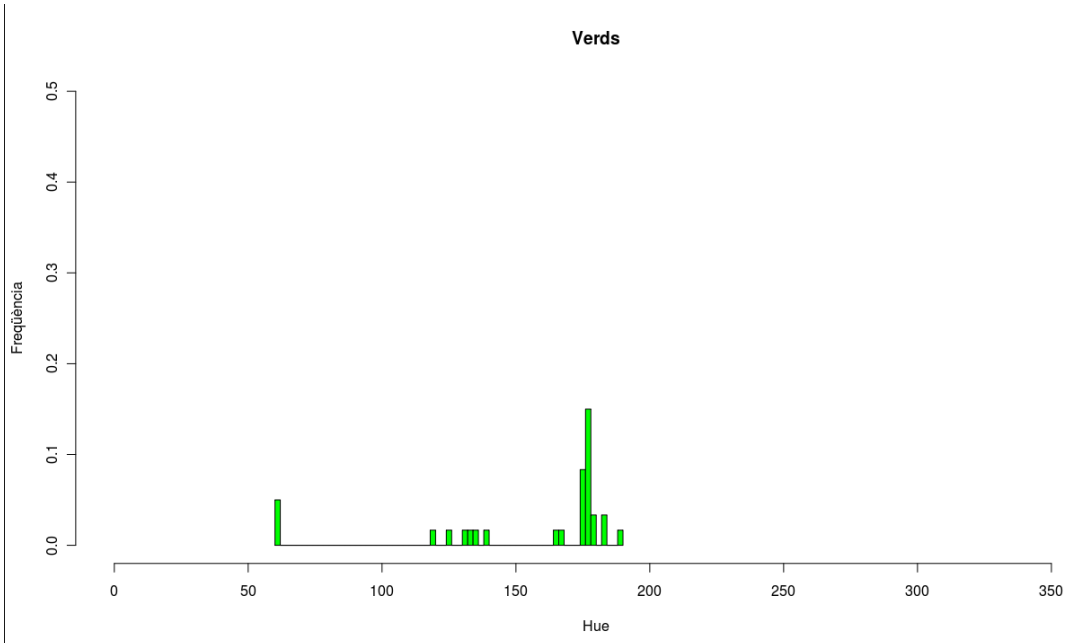
Histograma dels colors

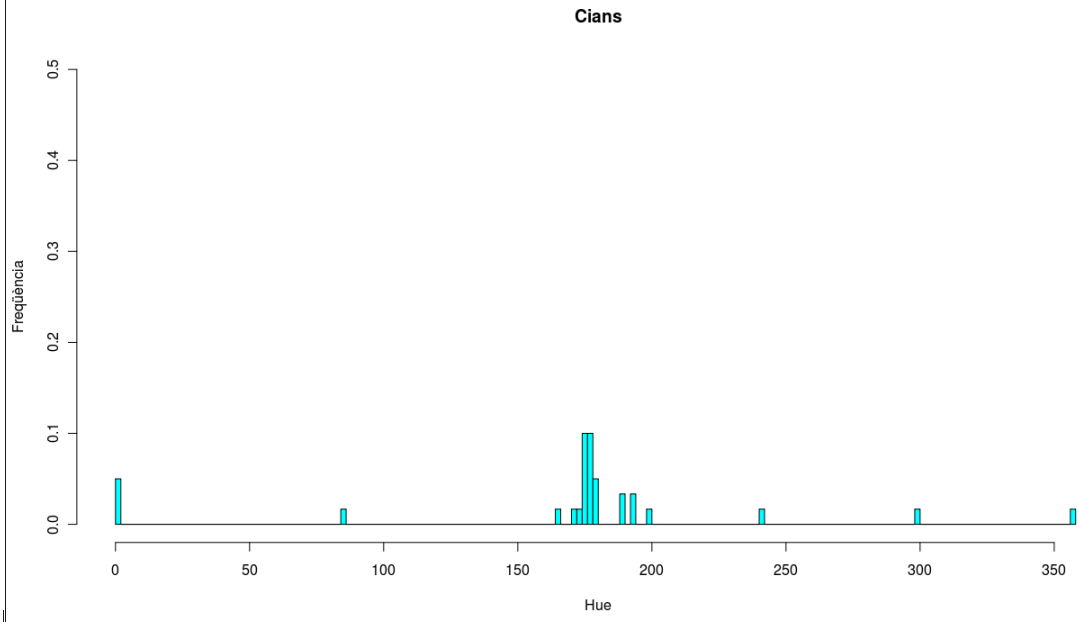
El que interessa primerament és descobrir quina distribució té cadascun dels colors, és a dir, veure en general quin comportament tenen en els diferents entorns als que han estat sotmesos. Per això, s'ha realitzat un histograma independent de cada color, sobre el paràmetre del to del color:

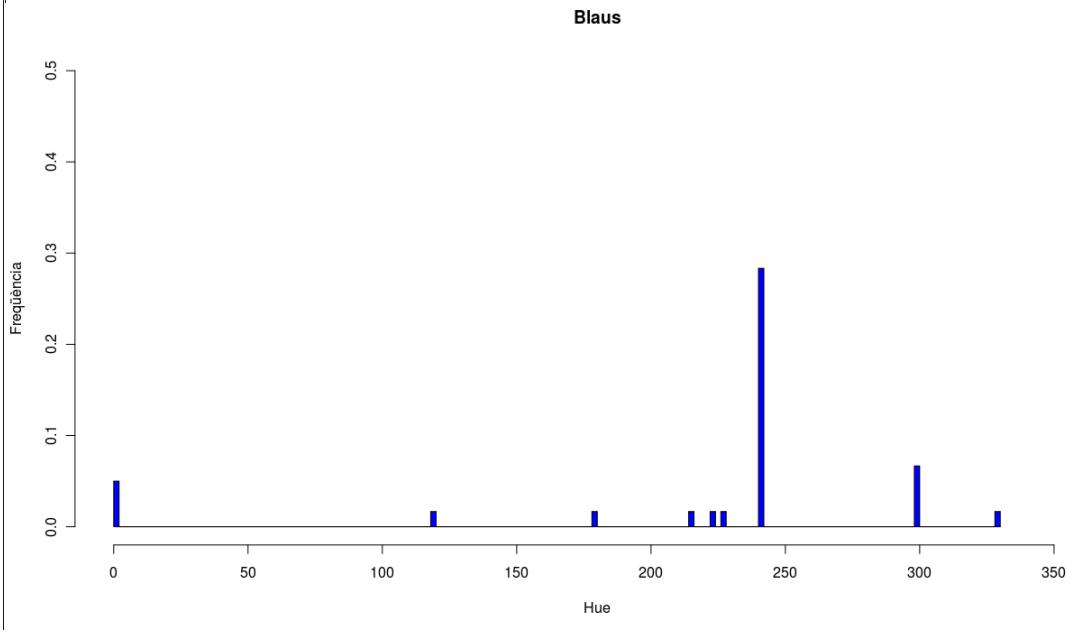

Vermells
Freqüència
0.0
0.1
0.2
0.3
0.4
0.5
0
50
100
150
200
250
300
350
Hue

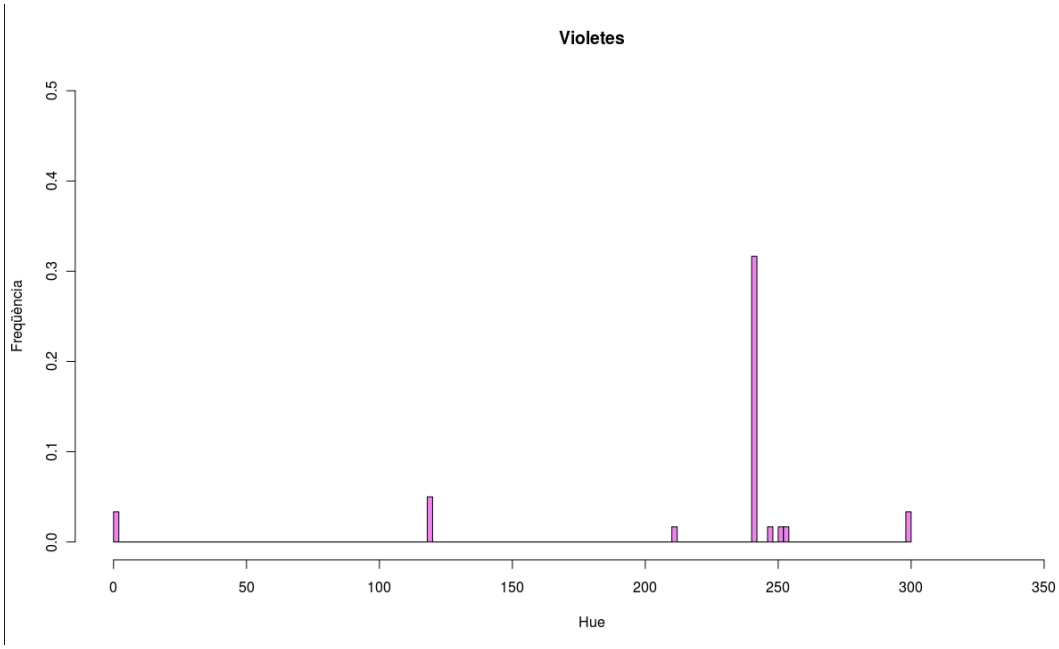

Taronges
Freqüència
0.0
0.1
0.2
0.3
0.4
0.5
0
50
100
150
200
250
300
350
Hue


Grocs
Freqüència
0.0
0.1
0.2
0.3
0.4
0.5
0
50
100
150
200
250
300
350
Hue

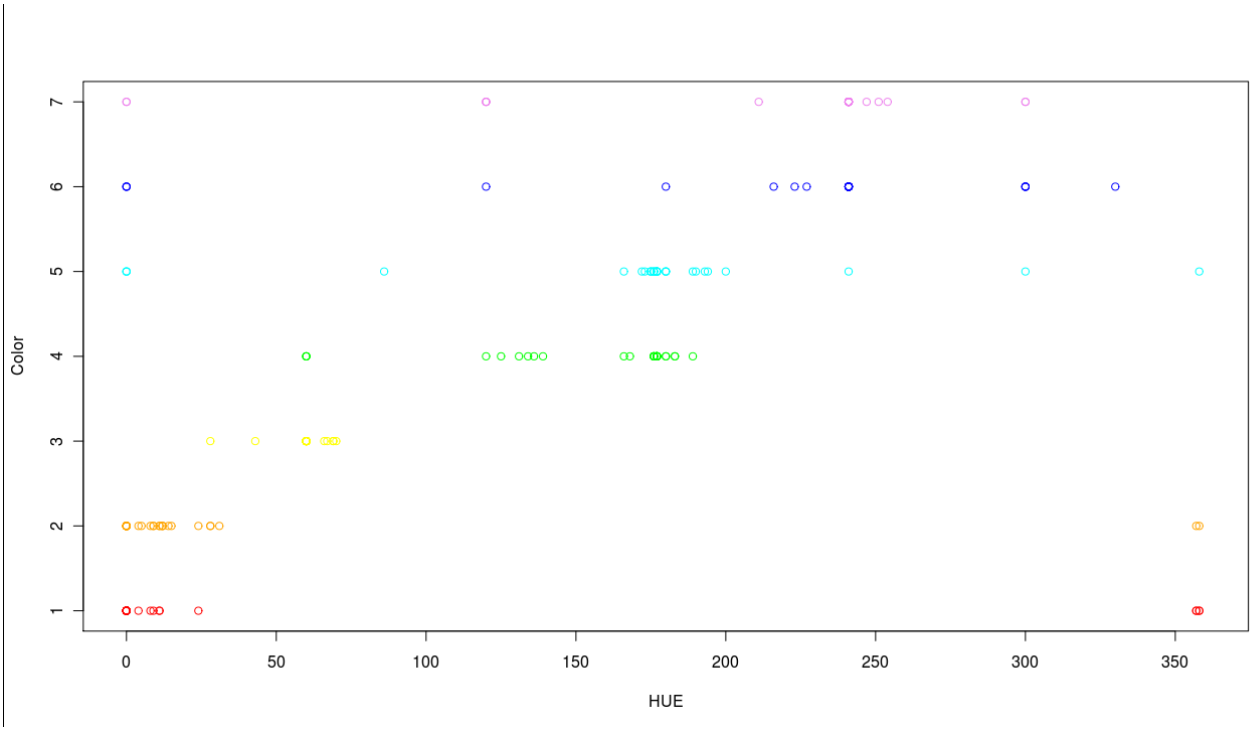

Verds
Freqüència
0.0
0.1
0.2
0.3
0.4
0.5
0
50
100
150
200
250
300
350
Hue


Cians
Freqüència
0.0
0.1
0.2
0.3
0.4
0.5
0
50
100
150
200
250
300
350
Hue


Blaus
Freqüència
0.0
0.1
0.2
0.3
0.4
0.5
0
50
100
150
200
250
300
350
Hue



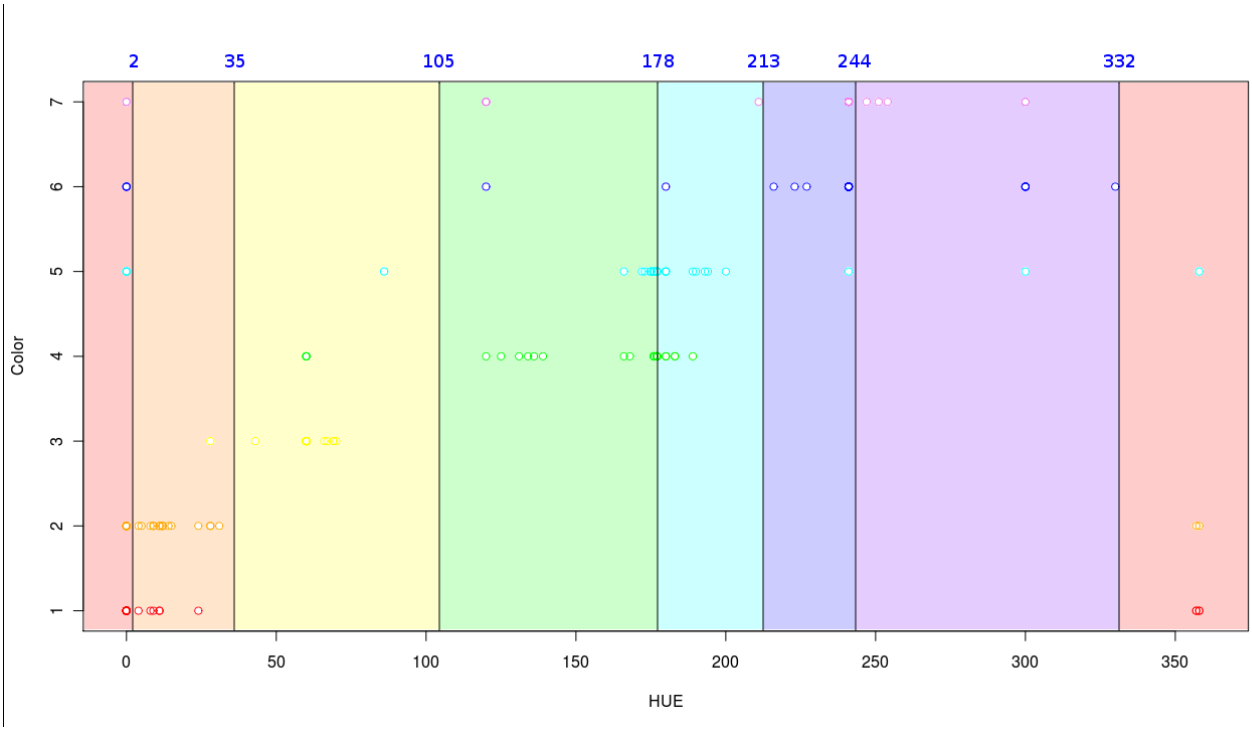
En l'eix d'abscisses es representa el to del color, de 0 a 360 graus. En l'eix d'ordenades es veu la freqüència en la que apareix aquell to concret. Amb aquests histogrames es pot identificar bé quin rang de to ocupa cada color. Per veure-ho millor, s'ha creat una gràfica que representa la distribució del to per tots els colors:



Aquí es veuen clarament totes les distribucions. Ara, estudiant aquest gràfic i els histogrames anteriors, es remarquen detalls com:

- El color vermell té molta més freqüència en el to 0 que en els restants i el taronja està una mica més distribuït.
- El groc té una regió clara molt definida.

Amb aquest petit anàlisi, s'intenta separar per cada color el que seria el seu rang de valor, i es pot provar de seccionar el gràfic sobre l'eix que representa el to, per fixar uns primers llindars:



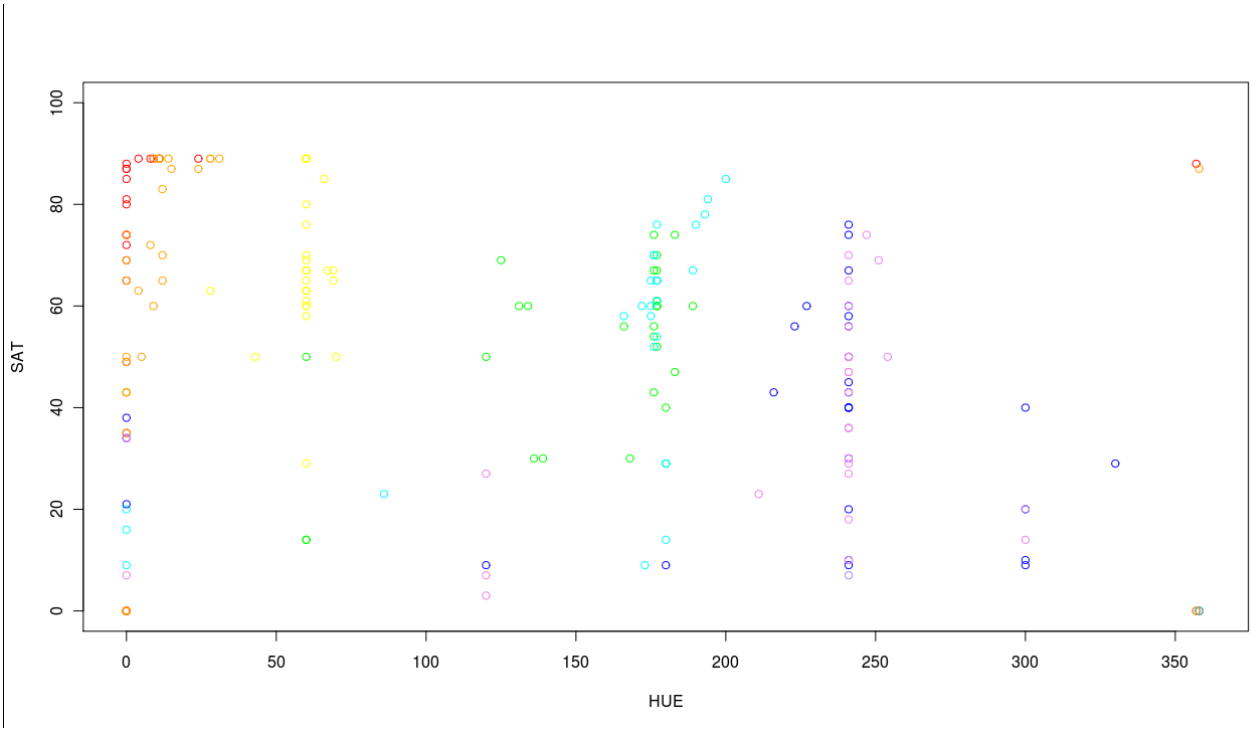
Aquesta és la primera aproximació dels llindars que s'ha escollit. De totes maneres, el principal problema encara no s'ha resolt:

El verd i el cian, i el blau i el violeta formen parelles molt estretes, que no permeten diferenciar gaire la seva distribució, ja que realment són colors molts semblants a nivell de to (la diferència de l'angle de la circumferència del to, és molt més petita que amb el color groc, per exemple).

El vermell i el taronja els hi passa una cosa semblant, tot i que sembli que el to més habitual estigui situat en llocs diferents (vermell més cap als 0º, taronja cap als 10º). S'ha d'aconseguir un mètode per separar aquestes tres parelles de colors que a la càmera li costa tant de distingir.

Histograma To – Saturació:

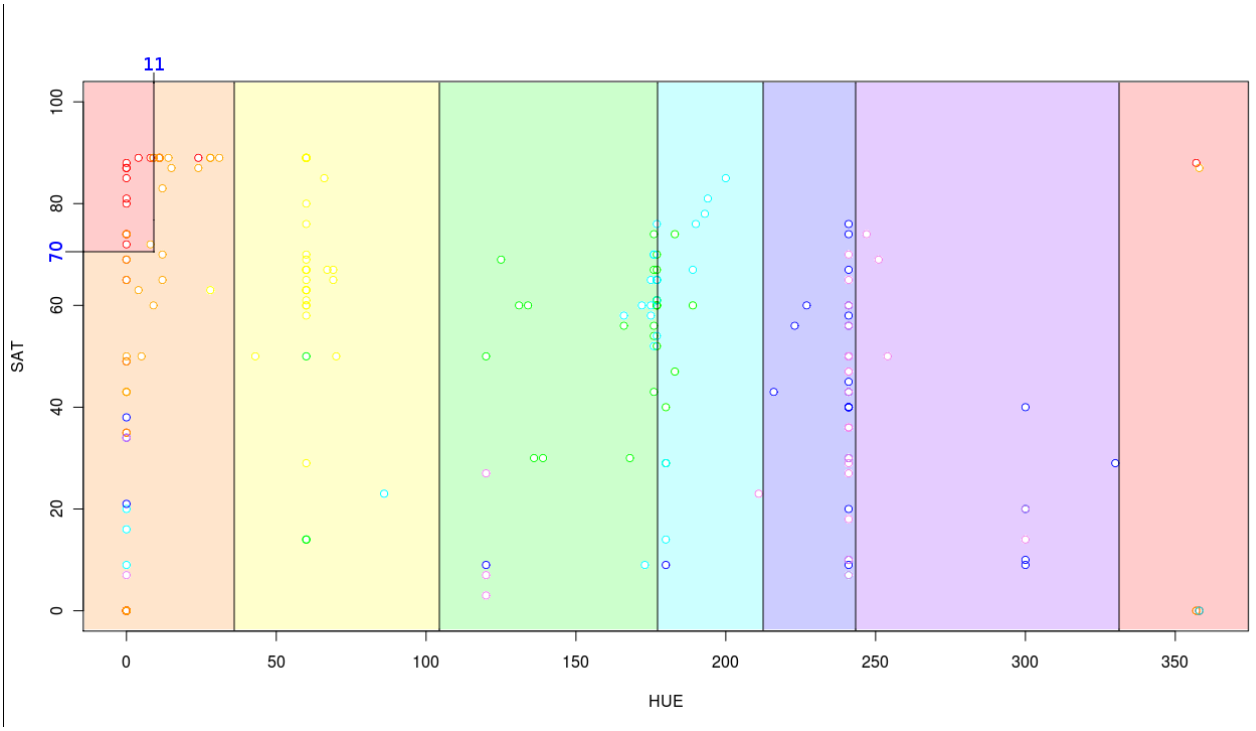
El primer intent és comparar el to amb la saturació, la segona component HSV que no s'ha tractat encara:



Com es pot veure a la gràfica, tenim el to a l'eix d'abscisses i la saturació a l'eix d'ordenades. Ara la distribució adquireix un altre sentit, i es poden determinar un parell de fets:

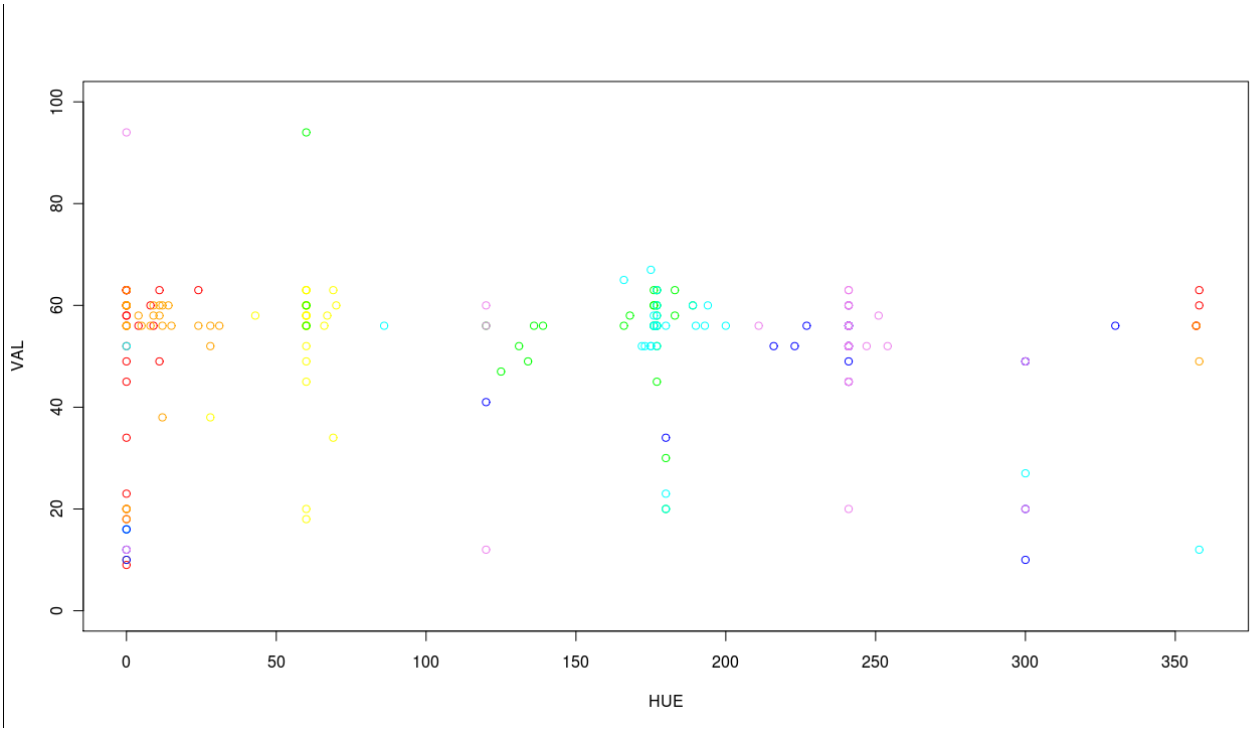
Verd i cian, blau i violeta, continuen estant massa junts. La dispersió en la saturació és irregular, ja que tots els valors de cadascun d'ells contenen gairebé tot el rang de la saturació. Per tant, es dedueix que de moment, amb aquesta visió no es poden separar aquestes dues parelles.

Vermell i taronja en canvi tenen una distribució peculiar. El vermell s'arracona cap als valors de saturació alta i baix to, mentre que el taronja es distribueix més espaiadament cap a valor de saturació baixa i tons més alts. Això permet que, amb el paràmetre de la saturació, es pugui separar el vermell i el taronja amb aquest llindar de dues components:



Histograma To – Valor:

El segon i últim intent és comparar el to amb el valor, la tercera component HSV que no s'ha tractat encara:

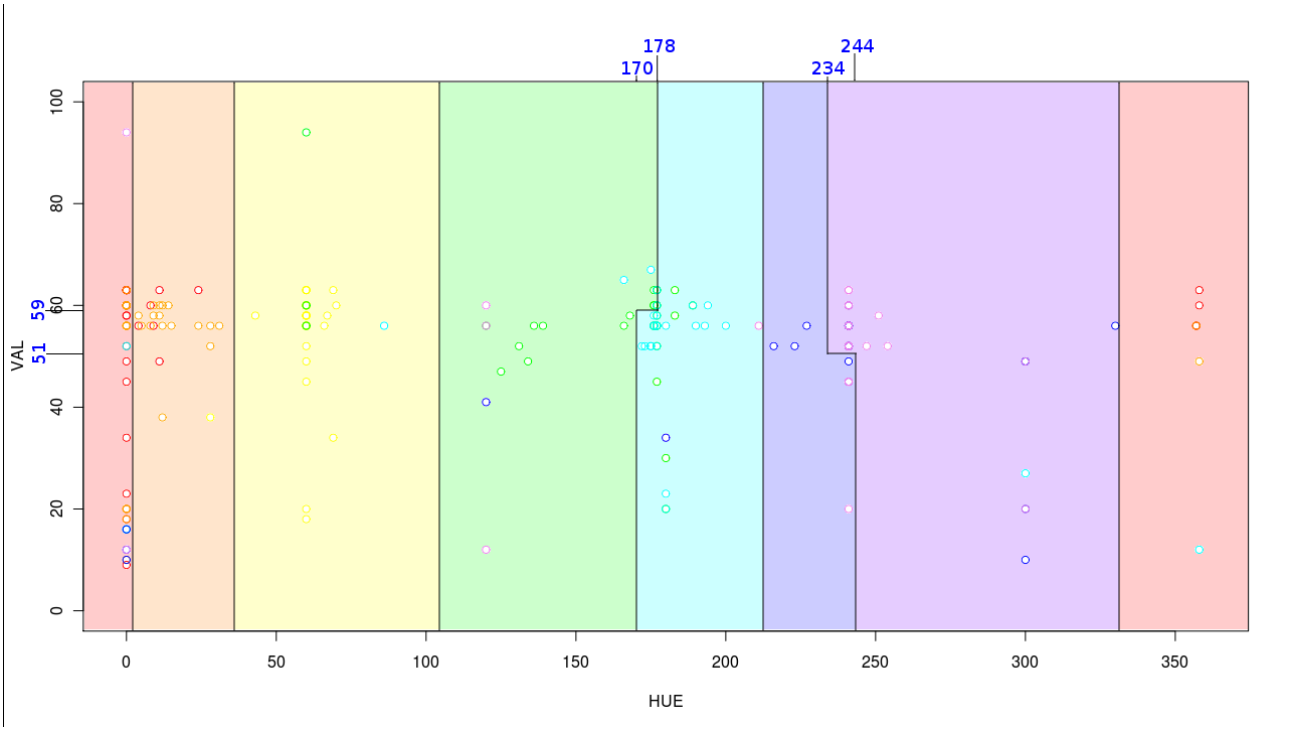


La gràfica ara conté en l'eix d'ordenades el valor, en comptes de la saturació. Amb aquesta s'intenta separar les dues parelles que havien quedat inseparables:

Verd i cian, on abans es trobaven en el mateix punt exacte, ara es veu que assoleixen valors diferents. Això és perquè, en el cas general, el verd adquireix tons més brillants.

Blau i violeta tenen un comportament similar. Es pot observar com, on s'encreuaven abans en el mateix punt de to, ara per la component de valor, el violeta assoleix punts de més brillantor en general que el blau.

Per tant es pot aprofundir més en el llindar d'aquestes dues parelles, i provar de crear un llindar de dues components com s'ha fet amb el vermell i el taronja, però utilitzant el valor en comptes de la saturació:



Finalment, ja s'ha aconseguit definir els llindars, que són el següents:

Color	Llindar
Vermell	HUE > 332 || (HUE <= 11 && SAT >=70)
Taronja	(HUE <=11 && SAT < 70) || (HUE > 11 && HUE <= 35)
Groc	HUE > 35 && HUE <= 105
Verd	(HUE > 105 && HUE <= 170) || (HUE > 170 && HUE <= 178 && VAL >= 59)
Cian	(HUE > 170 && HUE <= 178 && VAL < 59) || (HUE > 178 && HUE <= 213)
Blau	(HUE > 213 && HUE <= 234) || (HUE > 234 && HUE <= 244 && VAL < 51)
Violeta	(HUE > 234 && HUE <= 244 && VAL >= 51) || (HUE > 244 && HUE <= 332)

Com s'ha vist doncs, realitzant un estudi exhaustiu amb una quantitat prou gran d'imatges, per valorar el comportament que té cada color en diferents entorns, captat per l'òptica de la càmera, s'han pogut escollir uns llindars molt més ben acotats, i amb més bons resultats, que posant els llindars basats simplement en el to que dona l'escala del paràmetre *Hue*.

Tot i haver posat bé els llindars, la resposta a l'espectre de la llum és menys eficient en la component blava que en la verda i la vermella, per tant es podrà comprovar com la detecció de blaus i violetes dona uns resultats força pobres.

Cal remarcar que els llindars numèrics com els que es veuen a les gràfiques, no s'han calculat a ull, sinó mesuradament a partir de les taules de dades que es poden consultar a l'annex de les dades del test de color.

També cal dir que una gràfica que compari saturació i valor, perd el sentit per aquest projecte, ja que s'anul·la la component del to, que és realment la component interessant per definir els llindars, i per tant no aporta cap informació important alhora de definir-los.

5.3 Programa pel test de color

Per capturar les imatges s'ha creat un programa que segueix el mateix procediment que el programa del projecte, és a dir, capturar la imatge i donar els valors del color corresponent, però en comptes d'actuar per interrupcions, actua quan es prem el botó. A més, les imatges no es tenen simplement en memòria sinó que s'emmagatzemen a una memòria externa per guardar tant les imatges capturades com les estadístiques de color.

Ús del botó

La càmera conté un botó que pot ser llegit com a entrada/sortida de propòsit general (GPIO). El control del botó s'obté fàcilment amb la funció *cc3_button_get_state()*. Per tant el programa conté un bucle amb el següent codi:

```
while(true) {
    if(cc3_button_get_state()) {
          init_capture_ppm(numfile++);
          calculate_histogram();
          while(cc3_button_get_state());
    }
}
```

Bàsicament, el que es fa és executar el codi indefinidament i comprovar si el botó s'ha premut. En cas que sigui cert, captura una imatge, calcula l'histograma dels colors i després esta en un bucle fins que es deixa de prémer el botó.

Ús de la memòria externa

Per emmagatzemar les imatges i les dades d'aquestes, es fa servir la memòria externa. La plataforma CMUCam permet fer ús d'una targeta de memòria SD accedint-hi de la següent manera. Primer de tot, es crida una funció que inicialitza la configuració de la targeta SD:

```
cc3_filesystem_init();
```

Seguidament, es declara un fitxer del tipus FILE, s'obre amb la funció estàndard de C *fopen*, i s'escriu al fitxer amb la funció de C *fprintf*. Es pot veure per exemple com s'obre el fitxer per guardar les dades de les imatges:

```
FILE *log;
log = fopen("c:/colortest","a");
fprintf(log,"IMG:%d --> HUE:%d SAT:%d VAL:
%d\n",numfile,maxhue,maxsat,maxval);
```

Un cop finalitzada l'escriptura, es tanca el fitxer amb la funció estàndard de fitxers en C *fclose*:

```
fclose(log);
```

El codi complet d'aquest programa es pot consultar a l'annex del codi del test de color.

6 Test de consum

Un dels objectius del projecte és proporcionar autonomia suficient a la plataforma com per durar tot un dia. Evidentment, el càlcul s'ha fet abans de comprar la bateria, per aproximar teòricament el consum que generaria la plataforma, i quantes hores proporcionaria la bateria.

La plataforma té aquests valors de consum, indicat a la pàgina oficial d'aquesta:

Power State	Active Current	Idle Current	Voltage
All Active	130mA	25mA	5V
External Regulator Disabled	n/a	0.01uA	5V
CPU (@60Mhz)	30mA	10uA	1.8V
CPU Peripherals	30mA	10uA	3.3V
CMOS Camera	25mA	10uA	5V
MAX232	8mA	n/a	3.3V
Video FIFO	52mA	14mA	3.3V
MMC card	4mA	4mA	3.3V
Misc	10mA	10mA	3.3V

Figura 50: Taula de consum de CMUCam3

Per fer un càlcul d'autonomia màxima, s'ha agafat el valor màxim de consum (130mA). Cal tenir en compte que no serà molt diferent a aquest consum, ja que tots els components indicats a la taula estaran treballant al màxim rendiment, per tant s'espera una aproximació força bona.

En veure això, es va escollir una bateria de 2400mAh. Això proporciona:

$$Autonomia = \frac{Flux \cdot hora\, bateria}{Flux\, de\, consum} = \frac{2400\, mAh}{130\, mA} = \mathbf{18{,}46\ h}$$

7 Millora de la senyal d'àudio

Als apartats de PWM tant de hardware com de software, s'ha pogut entendre com es generava la senyal d'àudio seguint aquesta tècnica, però no s'ha justificat com s'ha pogut veure que el procés funcionava.

Per aconseguir analitzar exactament el comportament de la senyal, s'ha treballat amb un oscil·loscopi i així s'ha pogut observar en cada moment què s'estava generant.

Al principi de tot, s'ha generat una ona quadrada amb el duty cicle al 50%, per comprovar realment que les interrupcions funcionaven i que la senyal es generava segons el previst:

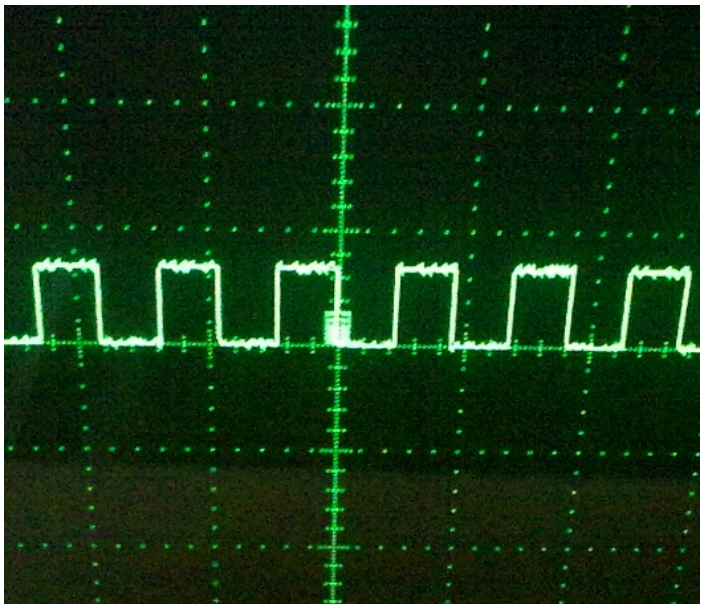


Figura 51: Senyal quadrada

Seguidament, es procedeix a generar la senyal variant el duty cicle, per obtenir la forma sinusoïdal i apareix això:

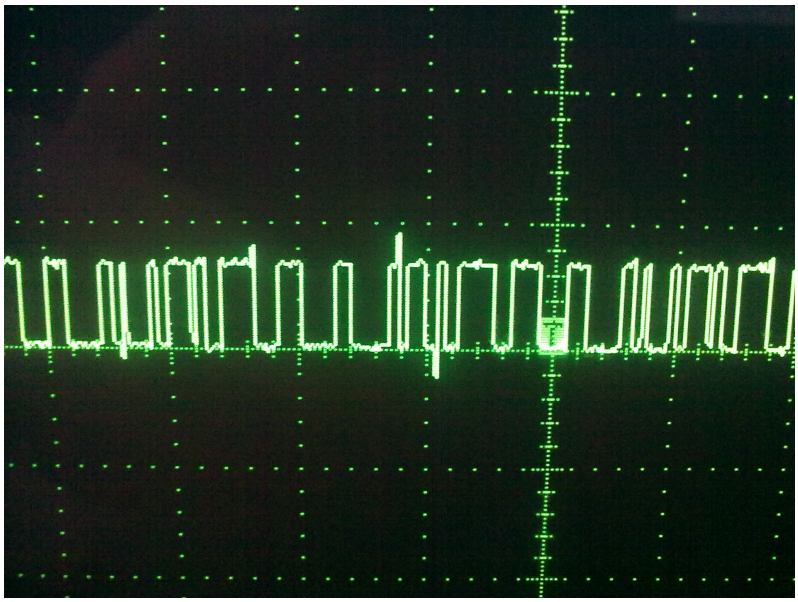


Figura 52: Senyal PWM

Evidentment, la senyal, tot i està donant els impulsos correctes per generar una ona sinusoïdal, continua sent quadrada. Per tant falta fer un pas més, un petit circuit integrador RC acoblat a la sortida.

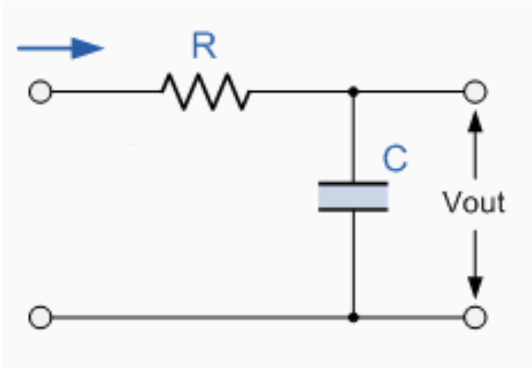


Figura 53: Circuit RC integrador

El circuit integrador RC no és res més que una resistència connectada en sèrie a l'entrada de la senyal, amb un condensador connectat en paral·lel a la sortida. Aquest circuit dona com a resultat una "integració" de la senyal d'entrada. Per entendre-ho, genera una senyal en funció de les càrregues que absorbeix el condensador, així, en duty cicles del PWM baixos absorbeix poc, i en duty cicles alts absorbeix molt més, per tant el resultat és l'acumulació progressiva de la corrent, que acaba generant una ona sinusoïdal.

El resultat després d'aplicar aquest circuit és el següent:

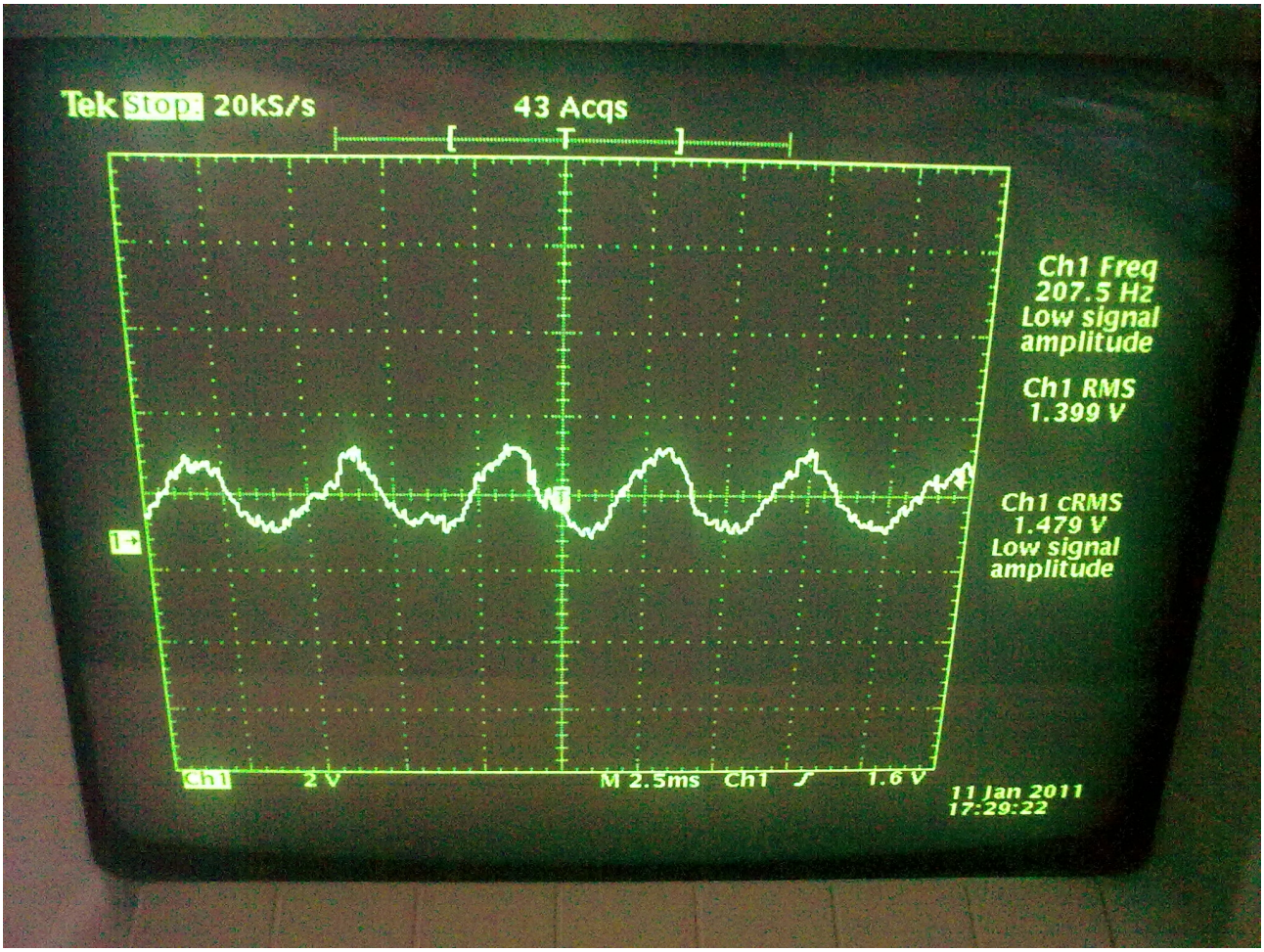


Figura 54: Senyal sinusoïdal a l'oscil·loscopi

Un cop vista la utilitat d'aquest circuit, s'ha de veure quins valors s'han calculat per la resistència i el condensador.

Cal veure que el circuit integrador RC, funciona alhora com un filtre passa-baixos. El filtre passa-baixos, com el seu nom indica, s'encarrega de deixar passar les freqüències baixes, i començar a eliminar, a partir d'una certa freqüència, les que es consideren altres. La formula que regeix el comportament d'un filtre passa-baixos és la següent:

$$F_c = \frac{1}{2 \cdot \pi \cdot R \cdot C}$$

Fc es refereix a la freqüència de tall a la qual es vol començar a inhibir les freqüències, i R i C són la resistència i el condensador respectivament.

El procediment comú es determinar la freqüència de tall a la que es volen inhibir les freqüències, i seguidament, donat un condensador escollit, es calcula la resistència necessària.

Per calcular la freqüència de tall, s'ha agafat la freqüència més alta necessària (en aquest cas 303 Hz de l'últim to). Per tant el càlcul és:

$$R = \frac{1}{2 \cdot \pi \cdot F_c \cdot C} = \frac{1}{2 \cdot \pi \cdot 303\,Hz \cdot 10\,\mu F} = 52{,}44\,\Omega$$

Aquest circuit s'incorporarà a l'estructura tal com es veurà al següent apartat.

8 Muntatge de l'estructura

Ara el projecte ja es troba en estat funcional, però falta una cosa, l'aspecte portable, ja que ara la plataforma està connectada en estat de proves. L'objectiu és aconseguir una estructura ergonòmica i portable, per això cal dissenyar una carcassa el més senzilla possible però que pugui portar tots els elements necessaris, que són la plataforma i tot el sistema d'energia.

Per fer això sorgeixen forces idees. Una d'elles ha estat col·locar la càmera en el front de l'usuari, i separar-la de la plataforma a través d'un cable. L'avantatge principal d'això és que es pot dur la càmera centrada al cap, però és més difícil d'encaixar en alguna estructura, ja que en el front no s'aguanta tant bé, per això s'ha triat de posar la càmera a un lateral del cap, de manera que es pugui sostenir amb una espècie de diadema.

L'altra idea sobre el sistema d'energia (bateria, dcdc i pcm) era posar-ho al costat oposat del cap on és la càmera. D'aquesta manera s'equilibraria el pes, i es portaria tot a la mateixa altura, però la idea de posar una bateria de liti s'ha considerat una mica arriscada, així que s'ha optat per separar-ho i posar-ho a la cintura, a través d'un cable.

No s'han d'oblidar els cascs per l'àudio, per tant s'havia de pensar una manera per col·locar els cascs d'àudio al cap i alhora aguantar la càmera. El que s'ha decidit és enganxar la càmera amb una estructura tancada, a uns cascs de tipus diadema (en comptes de cascs d'orella) de manera que els mateixos cascs ja fan d'estructura per aguantar la càmera.

Els materials que s'han necessitat per muntar l'estructura són:

- Planxa d'alumini: No pesa gaire i dóna robustesa a la carcassa. A més a més, es pot retallar fàcilment amb tisores i es pot doblegar.
- Claus de 3mm de diàmetre: Són uns claus per collar bé la carcassa. El diàmetre ha de ser el mateix que els forats que porta la plataforma per defecte (que ja els porta per poder-se collar a estructures externes).
- Auriculars de diadema: Són de plàstic, força lleugers i prims, permeten acoblar-se bé a l'estructura que es construirà.
- Capçal amb angle: Servirà per girar la càmera 90º.
- Cinta aïllant negre com a recobriment.

El primer pas és dissenyar la carcassa on anirà la càmera i s'acoblarà a la diadema dels auriculars. Per fer-ho, cal prendre mides exactes de la càmera, per tenir una idea de com s'embolcallarà, i com es podran treure els connectors necessaris. Abans de prendre mides però, s'ha de tenir en compte una cosa. La càmera, segons està connectada a la plataforma, apunta en direcció perpendicular a la base d'aquesta. Si es deixa així, no es podria posar la plataforma a un lateral del cap, perquè la plataforma hauria de tenir la base mirant endavant, i molestaria molt.

Per poder posar la base de costat al cap, s'ha de girar la càmera 90º, i això s'aconsegueix amb un capçal que gira el connector de la càmera aquests graus necessaris.

Ara la forma del muntatge no canviarà, per tant ja es poden prendre les mides que es consideren definitives i fer la carcassa.

Aquí es pot veure un esquema del disseny de l'estructura, mides en centímetres:

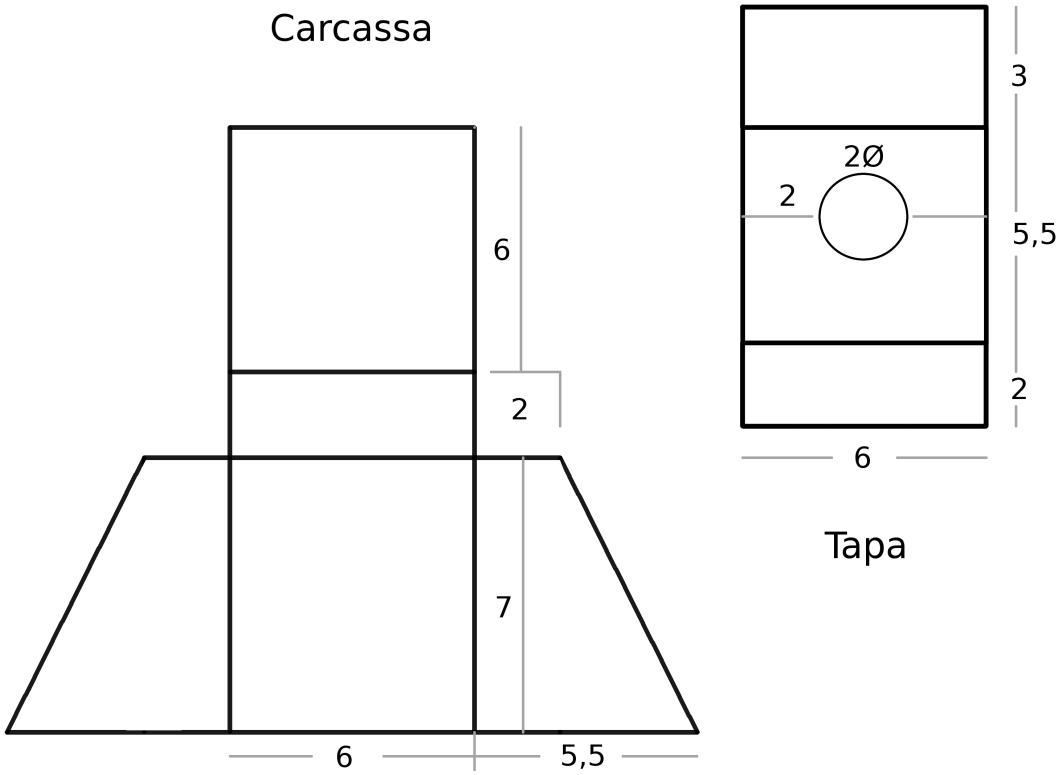


Figura 55: Disseny de la carcassa

Abans de embolcallar la plataforma, s'ha de connectar el cable d'àudio, que porta el circuit RC. Per fer-ho s'han soldat els components directament a un connector dels pins de sortida del PWM i s'han cobert una mica amb cinta aïllant per evitar el contacte.

Un cop fet això, s'ha plegat la carcassa i s'ha introduït a dins la plataforma. Un cop dins, es posen uns claus verticals que la travessen i amb uns tubs rígids que fa que s'aguanti sol. Després s'ha posat la tapa davantera i s'ha acabat de tancar els claus amb les femelles.

Finalment s'ha embolcallat el millor possible la carcassa amb cinta aïllant per donar-li un recobriment més discret de color negre, i evitant que les parets facin contacte amb algun element elèctric.

L'auricular es passa per dins la carcassa i queda fixat degut a la mida exacte del forat. Posteriorment es connecta el cable de la corrent a la bateria, i tot el sistema d'alimentació es col·loca dins una funda per poder portar a la cintura.

A continuació es mostren unes quantes imatges per veure el procés de muntatge:

Figura 56: Serrant l'estructura

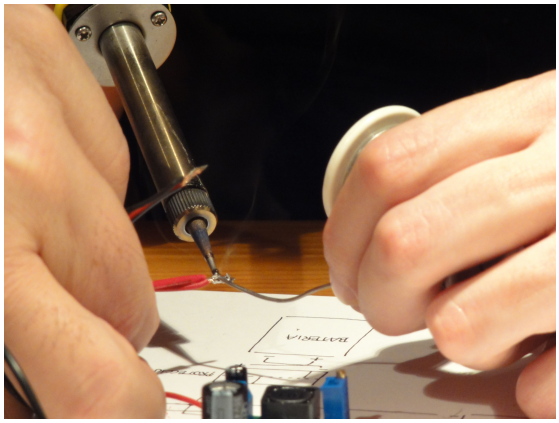


Figura 57: Soldant

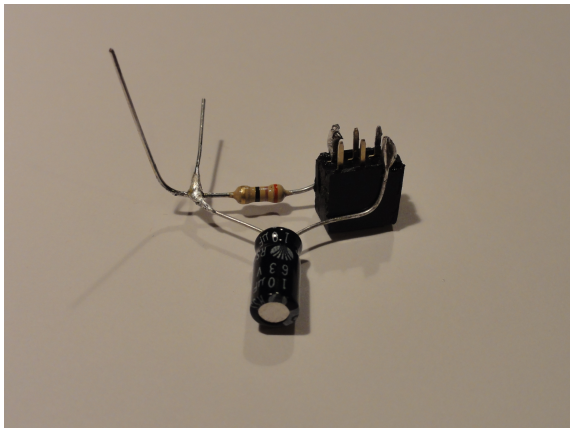


Figura 59: Circuit RC

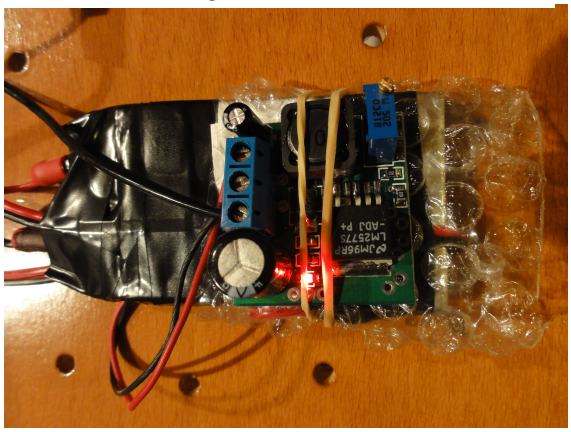


Figura 58: Sistema d'alimentació

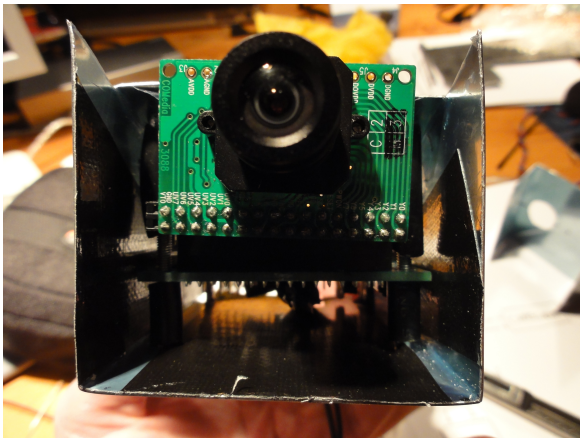


Figura 61: Vista frontal

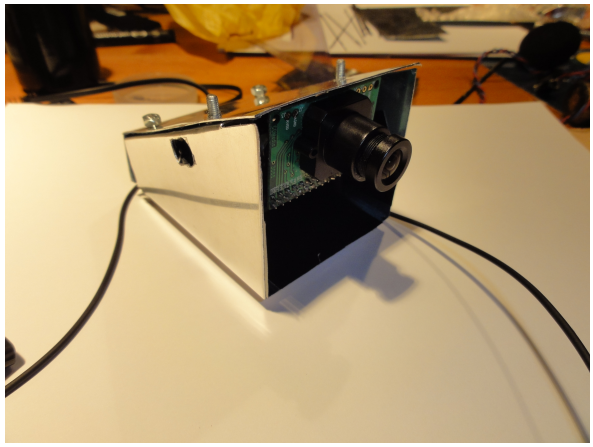


Figura 60: Vista lateral

Figura 62: Resultat final

Figura 63: Resultat final

9 Pressupost

Per calcular el cost d'aquest projecte, s'ha establert un pressupost en funció de les tasques i recursos necessaris. En aquesta taula, es poden veure les diferents etapes on participarien les persones qualificades per a cada tasca. Els càlculs del cost de l'empleat inclou el cost a pagar tots els afegits com ara l'assegurança o transports.

Tasca	Persona i hores	Cost/hora
Selecció components	Enginyer - 10h	x100 €/h = 1000 €
Disseny prototipus	Enginyer - 15h	x100 €/h = 1500 €
Muntatge prototipus	Tècnic - 8h	x20 €/h = 160 €
Disseny proves	Programador - 80h	x40 €/h = 3200 €
Programació codi	Programador - 80h	x40 €/h = 3200 €
Realització proves i re-programació	Programador - 40h	x40 €/h = 1600 €
Documentació	Informàtic - 120h	x40 €/h = 4800 €

A continuació, es suma també el cost del hardware:

Component	Preu
Plataforma	126,00 €
Components d'alimentació (Bateria, DCDC, PCM, Carregador)	74,00 €
Carcassa i peces muntatge	10,00 €

I finalment, suposant que el software es registra amb una llicència GPL i per tant d'ús gratuït, i que s'ha comprat el hardware necessari per desenvolupar una unitat del prototipus, es pot calcular el total:

Cost total: Total tasques + Total components = 15460 € + 210 € = **15670 €**

Si es volgués vendre el prototip a uns 300 € (tenint en compte els 210 € del cost per unitat del hardware), s'obtindria un benefici de 90 €, per tant seria necessari vendre com a mínim 175 unitats del producte perquè sortís rendible.

Agafant la dada del principi sobre el percentatge de població amb acromatòpsia (un de cada 30.000 habitants), suposant la població de mercat més pròxim (Estats Units i Europa) de un bilió d'habitants, considerant que un 30% d'aquesta població és benestant, es tindria:

- 300 milions de compradors potencials

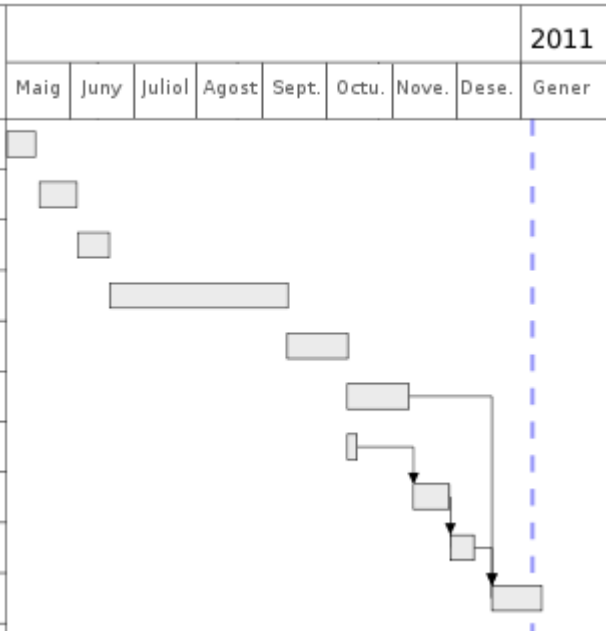
- 1 de cada 30.000 pot tenir acromatòpsia

Es podrien arribar a tenir uns 10.000 habitants en el mercat interessats a obtenir el producte. Només que una cinquantena part d'aquesta població útil el comprés, ja serien 200 unitats venudes i s'hauria recuperat la inversió.

10 Planificació del projecte

La planificació és important per complir les fites del projecte. Per dur a terme aquesta planificació, a l'inici de projecte es va crear un diagrama de Gantt per gestionar les tasques. A continuació s'observaran les tasques i es comentarà si s'han assolit les dates i objectius.

Nom	Feina
Plantejament del projecte	15d
Definició dels objectius	19d
Buscar plataforma visió	15d
Familiarització amb la plataforma	82d
Generació PWM	30d
Test d'imatge	28d
Càlcul de consum	5d
Disseny estructura	15d
Construcció estructura	12d
Redacció memòria	25d



Il·lustració : Diagrama de Gantt

El projecte es va començar a plantejar al febrer, i les tres primeres etapes que es mostren en el gantt realment comencen des d'aquest mes (al març ja es va crear el *blog* pel projecte). Però aquest es matriculava a setembre, per tant, durant el quadrimestre de febrer no es va desenvolupar quasi res. Per això, s'ha iniciat el gantt des de maig, que és on realment es va començar a treballar de debò.

La primera fase es basa en el plantejament el projecte. La idea estava força clara ja que s'havia estat discutint prèviament amb en Neil, que era la persona que el demanava. Per tant s'havia de planificar una reunió amb ell i el tutor per tractar el problema en qüestió i les eines que es necessitarien. Seguidament, a raó de la reunió amb el client, es podien decidir els objectius a assolir per el projecte. Tot aquest procés es va dur a terme correctament, fins i tot amb més rapidesa de l'esperada, i ja s'estava buscant la plataforma de visió.

Un cop clars els objectius, es podia saber quina plataforma seria ideal per complir-los, així que es va buscar per internet i es va trobar la plataforma CMUCam3. Al cap de dues setmanes, d'acord amb el temps planificat, es va obtenir la plataforma i seguidament es tractava de posar-se a estudiar el seu funcionament. Per aquesta etapa es va predir un temps de 2 mesos i mig. El temps ha estat encertat, tot i que no ha estat consecutiu (com es podia esperar), sinó que s'ha anat repartint a mesura que es desenvolupava cada tasca.

El PWM en canvi, ha portat més temps del previst. Es preveia dedicar un mes a l'estudi d'aquesta tècnica, comptant també amb la seva implementació. Tot i que més o menys s'ha trigat un mes i mig a tractar aquest apartat, després s'ha decidit aprofundir en un objectiu que no estava previst, com és el circuit RC per millorar la senyal. Per això s'ha trigat uns dos mesos a tenir-ho clar, estudiar-ho i implementar-ho.

El test d'imatge ha estat encertat amb el temps previst, un mes. S'ha dedicat una setmana a pensar com obtenir el color bé i parlar amb en Neil per saber quins eren els sons que volia obtenir. La setmana següent, s'ha discutit amb el tutor sobre el sistema de color a aplicar així com l'auto-lluminositat de la càmera.

El càlcul de consum és una cosa ràpida, simplement tenint en compte els paràmetres de la plataforma i de la bateria. El temps dedicat va ser ínfim, tot i que es va fer més endavant, quan es va tenir decidit quina bateria comprar, per calcular les hores d'autonomia.

Pel disseny i construcció de l'estructura s'ha avançat el temps establert, amb una setmana es va complir l'objectiu. El disseny va ser ràpid, i la construcció sobre una setmana.

La redacció de la memòria ha durat un mes i s'ha pogut iniciar amb suficient temps d'antelació ja que la part funcional del projecte estava acabada un mes i mig abans de l'entrega. Al final s'ha anat afegint la construcció de la carcassa així com el muntatge de la bateria.

En definitiva, s'ha fet un bon marcatge temporal dels objectius, que s'han pogut complir amb el temps necessari, tot i que n'hi ha que s'han desplaçat abans o després del temps establert.

11 Resultats i Conclusions

El més important d'aquest projecte és destacar que està dirigit a un client real, en Neil Harbisson. La proposta que va fer s'ha realitzat, i amb això s'ha aconseguit cobrir la necessitat d'aquesta persona. A nivell de producte, amb l'eyeborg 2.0 en Neil podrà reconèixer els colors mitjançant els tons, i utilitzar després aquest aparell amb altres finalitats del seu àmbit artístic. S'ha pogut participar en una entrevista a TV3 al programa "Entre línies" mostrant l'objectiu del projecte.

A nivell de projecte, cal comentar tots els punts que s'han tractat:

El PWM i la senyal de so han estat satisfactoris. El to que es genera és força bo, i pot ser escoltat sense molèsties (tot i ser un to generat amb poca electrònica), però queda per futures iniciatives poder-lo millorar, ja sigui augmentant la freqüència de treball del processador com incloent una electrònica més complexa.

El color, tot i treure uns bons resultats, és el punt crític del projecte. La càmera no té una òptica prou bona per captar bé el color ni la lluminositat. El test de color ha ajudat molt a discriminar entre colors, però també queda per futures millores poder obtenir una òptica més bona, que potser apliqui filtres més complexos per captar el color.

La part de programació ha estat molt satisfactòria. S'ha aconseguit entendre i controlar tota la plataforma, i ha estat una gran aportació a nivell personal, per aprendre a utilitzar un sistema *embedded* des de zero.

Un altre punt a tenir en compte ha estat l'ergonomia. S'ha vist que és molt complicat crear una plataforma que capti imatges i generi so, alhora que tingui una llarga autonomia i que sigui petita i portable. En el cas d'aquest projecte, s'ha aconseguit l'autonomia i la captació d'imatges amb el so, però la capacitat per realitzar una ergonomia total a l'usuari és molt costós. La carcassa que s'ha construït és bona i ben dissenyada, però es requeriria de millors materials i peces encara més petites per poder-se adaptar millor al cap de l'usuari. També hagués millorat si s'hagués disposat del material per fer una carcassa de plàstic amb un motlle a mida.

Així doncs, es deixen les següents tasques per la propera versió de l'aparell:

- Generar un hardware específic per aquest propòsit, on només calgui la càmera i un pin per generar so (evidentment amb processador i memòria).
- Utilitzar peces més petites:
  - Càmera de mòbil o càmera espia
  - Auriculars interiors o ressonadors ossis
  - Plataforma en general més petita
- Sistema més ergonòmic. Estructura millor adaptada al cap de l'usuari.

A nivell personal, he de dir que ha estat un projecte molt interessant, més a nivell de hardware que de software. La possibilitat d'aprofundir en el món dels sistemes *embedded* m'ha proporcionat una visió global molt bona de com funcionen totes les parts d'aquest sistema, com la càmera, el so i les interrupcions. Amb les bateries també he aprés tant en el procés de càrrega com en les connexions de protecció.

Amb les tasques definides anteriorment, deixo obert el projecte, juntament amb aquesta memòria, per qui vulgui avançar-lo amb les futures millores.

12 Annex I : Preparant l'entorn de treball

Per poder treballar amb el projecte s'ha de configurar primer l'entorn. En aquest apartat s'explica quins passos s'han seguit i quins programes han estat necessaris. El projecte s'ha realitzat des d'un sistema operatiu linux (concretament, Ubuntu 10.04).

Compilador ARM-GCC per linux

Aquest és el compilador específic per ARM que s'ha comentat a l'inici del document, el port de C per ARM. Només cal descomprimir-lo:

```
bunzip2 arm-200xqx-xx-arm-none-eabi-i686-pc-linux-gnu.tar.bz2
tar xvf arm-200xqx-xx-arm-none-eabi-i686-pc-linux-gnu.tar
```

LPC-ISP per linux

Aquest és el programa que permet carregar el codi compilat en binari a la plataforma. S'ha de descomprimir i compilar:

```
unzip lpc21isp_unix.zip
cd lpc21isp_unix
make
```

Llibreries de plataforma CC3

Aquestes llibreries implementen la majoria de funcions bàsiques de la plataforma, per fer servir com a capa d'alt nivell per programar-la fàcilment. S'ha de compilar.

Un cop s'han baixat i compilat les eines necessàries, s'ha d'afegir el carregador LPC-ISP i el compilador ARM-GCC al PATH d'usuari per poder fer-los servir des d'on sigui:

```
export PATH=$PATH:/home/matias/Projectes/CMUCam/CodeSourcery/bin/
export PATH=$PATH:/home/matias/Projectes/CMUCam/lpc21isp_179/
```

Un cop fet això, per fer un programa per la plataforma només cal crear un fitxer en C, fer un enllaç a les llibreries de la plataforma per fer-les servir, i compilar-ho amb *make*.

Perquè compili correctament, s'ha d'utilitzar un Makefile que ve per defecte, en el que s'especifiquen el codi font, les capçaleres i llibreries extres utilitzades. Després ho compila i genera un fitxer en hexadecimal.

Aquest fitxer hexadecimal és el que s'ha d'enviar a la plataforma, i es fa mitjançant la comanda:

```
lpc21isp eyeborg_lpc2106-cmucam3.hex /dev/ttyUSB0 115200 14746
```

És la comanda del carregador, on s'especifica l'arxiu hexadecimal, el port de la plataforma (port sèrie mitjançant convertidor USB), amb el *baudrate* del port sèrie i la freqüència de l'oscil·lador de la plataforma.

13 Annex II : Codi font

En aquest apartat es mostra el codi font del programa.

```
//////////////////////////////////////
// EYEBORG 2.0 - Matias Lizana García //
//////////////////////////////////////

#include "eyeborg.h"
#include <cc3.h>
#include "../../../cc3/hal/lpc2106-cmucam3/cc3_pin_defines.h"
#include "../../../cc3/hal/lpc2106-cmucam3/LPC2100.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cc3_hsv.h>

bool eyeborg_init = false;

int hue[361];
int maxhue = 0;
int sat[101];
int maxsat = 0;
int val[101];
int maxval = 0;

int angle = 0;
int duty[90] =
{5,5,6,6,6,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,7,7,7
,7,6,6,6,5,5,5,4,4,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,2,2,2,2,2,2,2,3,3,3,4,4,5};

int freq[7] = {360, //182 Hz - VERMELL
               298, //220 Hz - TARONJA
               284, //231 Hz - GROC
               274, //239 Hz - VERD
               238, //275 Hz - CIAN
               229, //286 Hz - BLAU
               216, //303 Hz - VIOLETA
              };

uint8_t *row;
```

```
int main(void) {

    cc3_uart_init(0,CC3_UART_RATE_115200,
                  CC3_UART_MODE_8N1,
                  CC3_UART_BINMODE_TEXT);

    printf("----- Eyeborg 2.0 -----\n");
    printf(">  Matias Lizana Garcia\n\n");
    printf("Inicialitzant Eyeborg...\n");

    printf("\t> Camera... ");
    if(cc3_camera_init()) printf("OK\n");
    else {
          printf("Error!\n");
          return 0;
    }
    cc3_camera_set_resolution (CC3_CAMERA_RESOLUTION_LOW);

    printf("\t> Estabilitzant sistema... ");
    cc3_timer_wait_ms(1000);
    printf("OK\n");

    printf("\t> Interrupcions... ");
    config_pwm();
    printf("OK\n");

    printf("\t> Carrega memoria... ");
    row = cc3_malloc_rows(1);
    printf("OK\n");

    printf("> Eyeborg actiu!\n\n");

    while(true) {
        capture_image();
          calculate_histogram();
          calculate_frequency();
    }

    return 0;
}

void calculate_frequency()
{
    if(maxhue > 332) {
          REG(PWM0_PR) = freq[0];
          printf(" Vermell!\n");
```

```
	}
	else if(maxhue <= 11) {
		if(maxsat >= 70) {
			REG(PWM0_PR) = freq[0];
			printf(" Vermell!\n");
		}
		else {
			REG(PWM0_PR) = freq[1];
			printf(" Taronja!\n");
		}
	}
	else if(maxhue > 11 && maxhue <= 35) {
		REG(PWM0_PR) = freq[1];
		printf(" Taronja!\n");
	}
	else if(maxhue > 35 && maxhue <= 105) {
		REG(PWM0_PR) = freq[2];
		printf(" Groc!\n");
	}
	else if(maxhue > 105 && maxhue <= 170) {
		REG(PWM0_PR) = freq[3];
		printf(" Verd!\n");
	}
	else if(maxhue > 170 && maxhue <= 178) {
		if(maxval >= 59) {
			REG(PWM0_PR) = freq[3];
			printf(" Verd!\n");
		}
		else {
			REG(PWM0_PR) = freq[4];
			printf(" Cian!\n");
		}
	}
	else if(maxhue > 178 && maxhue <= 213) {
		REG(PWM0_PR) = freq[4];
		printf(" Cian!\n");
	}
	else if(maxhue > 213 && maxhue <= 234) {
		REG(PWM0_PR) = freq[5];
		printf(" Blau!\n");
	}
	else if(maxhue > 234 && maxhue <= 244) {
		if(maxval >= 51) {
			REG(PWM0_PR) = freq[6];
			printf(" Violeta!\n");
		}
```

```
            else {
                  REG(PWM0_PR) = freq[5];
                  printf(" Blau!\n");
            }
      }
      else if(maxhue > 244 && maxhue <= 332) {
            REG(PWM0_PR) = freq[6];
            printf(" Violeta!\n");
      }
}

void pwm_MR0()
{
      if(eyeborg_init) {
            REG(GPIO_IOSET) = _CC3_SERVO_0;
            REG(PWM0_MR1) = duty[angle];
            REG(PWM0_LER) = 2;
            angle++;
            if(angle >= 90) {
                  angle = 0;
            }
      }
}

void pwm_MR1()
{
      if(eyeborg_init) REG( GPIO_IOCLR) = _CC3_SERVO_0;
}

void config_pwm()
{
      REG (VICIntEnable) = 0x00000100;

      REG(PWM0_TCR) = 0;
      REG(PWM0_TC) = 0;
      REG(PWM0_PC) = 0;
      REG(PWM0_PR) = freq[0];
      REG(PWM0_MR0) = 10;
      REG(PWM0_MR1) = 5;
      REG(PWM0_LER) = 3;
      REG(PWM0_PCR) = 0x0200;
      REG(PWM0_MCR) = 0x0B;
      REG(PWM0_TCR) = 9;

      eyeborg_init = true;
}
```

```
void capture_image()
{
    int time = cc3_timer_get_current_ms();
    cc3_pixbuf_load ();

    memset(hue,'\0',sizeof(hue));
    memset(sat,'\0',sizeof(sat));
    memset(val,'\0',sizeof(val));

    for (uint16_t y = 0; y < cc3_g_pixbuf_frame.height; y++) {
        cc3_pixbuf_read_row(row);
        cc3_rgb2hsv_row((cc3_pixel_t*)row,cc3_g_pixbuf_frame.width);
        for (uint16_t x = 0; x < cc3_g_pixbuf_frame.width * 3U; x+=3) {
            hue[(row[x]*360)/255]++;
            sat[row[x+1]]++;
            val[row[x+2]]++;
        }
    }
    printf("Total: %lu ms || ",cc3_timer_get_current_ms() - time);
}

void calculate_histogram()
{
    int maxh = 0, maxs = 0, maxv = 0;
    for(int k=0; k<361; k++) {
        if(hue[k] > maxh) {
            maxh = hue[k];
            maxhue = k;
        }
    }
    for(int k=0; k<101; k++) {
        if(sat[k] > maxs) {
            maxs = sat[k];
            maxsat = k;
        }
        if(val[k] > maxv) {
            maxv = val[k];
            maxval = k;
        }
    }
    printf("H: %d S: %d V: %d ",maxhue,maxsat,maxval);
}
```

I també les capçaleres:

```
#ifndef EYEBORG_H
#define EYEBORG_H

void config_pwm(void);
void calculate_histogram(void);
void calculate_frequency(void);
void capture_image(void);
void pwm_MR0(void);
void pwm_MR1(void);

#endif
```

14 Annex III: Dades i codi del test de color

En aquest apartat es mostren les dades obtingudes en les imatges del test de color i el codi font d'aquest.

Es recorda que per el test de color s'han pres 30 imatges per cada to, amb els valors H, S i V corresponents. Aquí es poden veure les mostres:

```
VERMELL:
IMG: 1 --> HUE: 9 SAT: 89 VAL: 56
IMG: 2 --> HUE: 11 SAT: 89 VAL: 63
IMG: 3 --> HUE: 11 SAT: 89 VAL: 49
IMG: 4 --> HUE: 24 SAT: 89 VAL: 63
IMG: 5 --> HUE: 8 SAT: 89 VAL: 60
IMG: 1 --> HUE: 0 SAT: 88 VAL: 58
IMG: 2 --> HUE: 4 SAT: 89 VAL: 56
IMG: 3 --> HUE: 0 SAT: 87 VAL: 49
IMG: 4 --> HUE: 0 SAT: 69 VAL: 20
IMG: 5 --> HUE: 0 SAT: 81 VAL: 34
IMG: 6 --> HUE: 0 SAT: 87 VAL: 52
IMG: 1 --> HUE: 358 SAT: 0 VAL: 63
IMG: 2 --> HUE: 0 SAT: 74 VAL: 63
IMG: 3 --> HUE: 0 SAT: 0 VAL: 60
IMG: 4 --> HUE: 0 SAT: 80 VAL: 58
IMG: 5 --> HUE: 0 SAT: 0 VAL: 63
IMG: 6 --> HUE: 0 SAT: 72 VAL: 56
IMG: 7 --> HUE: 357 SAT: 88 VAL: 56
IMG: 8 --> HUE: 358 SAT: 0 VAL: 60
IMG: 9 --> HUE: 0 SAT: 74 VAL: 60
IMG: 1 --> HUE: 0 SAT: 0 VAL: 60
IMG: 2 --> HUE: 0 SAT: 0 VAL: 63
IMG: 3 --> HUE: 0 SAT: 49 VAL: 23
IMG: 4 --> HUE: 0 SAT: 65 VAL: 18
IMG: 5 --> HUE: 0 SAT: 35 VAL: 9

TARONJA:
IMG: 6 --> HUE: 24 SAT: 87 VAL: 56
IMG: 7 --> HUE: 28 SAT: 89 VAL: 52
IMG: 9 --> HUE: 31 SAT: 89 VAL: 56
IMG: 10 --> HUE: 28 SAT: 89 VAL: 56
IMG: 7 --> HUE: 14 SAT: 89 VAL: 60
IMG: 8 --> HUE: 11 SAT: 89 VAL: 60
IMG: 9 --> HUE: 11 SAT: 89 VAL: 58
IMG: 10 --> HUE: 0 SAT: 65 VAL: 18
IMG: 11 --> HUE: 12 SAT: 83 VAL: 38
IMG: 12 --> HUE: 9 SAT: 89 VAL: 60
IMG: 10 --> HUE: 4 SAT: 63 VAL: 58
IMG: 12 --> HUE: 0 SAT: 43 VAL: 56
IMG: 13 --> HUE: 5 SAT: 50 VAL: 56
```

```
IMG: 14 --> HUE: 0 SAT: 43 VAL: 60
IMG: 15 --> HUE: 15 SAT: 87 VAL: 56
IMG: 16 --> HUE: 12 SAT: 70 VAL: 60
IMG: 17 --> HUE: 9 SAT: 60 VAL: 58
IMG: 18 --> HUE: 0 SAT: 43 VAL: 56
IMG: 19 --> HUE: 8 SAT: 72 VAL: 56
IMG: 6 --> HUE: 12 SAT: 65 VAL: 56
IMG: 7 --> HUE: 0 SAT: 50 VAL: 56
IMG: 8 --> HUE: 0 SAT: 49 VAL: 20
IMG: 9 --> HUE: 0 SAT: 69 VAL: 20
IMG: 10 --> HUE: 0 SAT: 35 VAL: 10

GROC:
IMG: 11 --> HUE: 60 SAT: 89 VAL: 56
IMG: 12 --> HUE: 60 SAT: 89 VAL: 58
IMG: 13 --> HUE: 60 SAT: 89 VAL: 56
IMG: 14 --> HUE: 60 SAT: 89 VAL: 63
IMG: 15 --> HUE: 60 SAT: 89 VAL: 52
IMG: 14 --> HUE: 60 SAT: 89 VAL: 63
IMG: 15 --> HUE: 60 SAT: 89 VAL: 60
IMG: 16 --> HUE: 60 SAT: 29 VAL: 20
IMG: 17 --> HUE: 60 SAT: 69 VAL: 45
IMG: 18 --> HUE: 66 SAT: 85 VAL: 56
IMG: 20 --> HUE: 60 SAT: 67 VAL: 58
IMG: 21 --> HUE: 60 SAT: 61 VAL: 60
IMG: 22 --> HUE: 60 SAT: 80 VAL: 58
IMG: 23 --> HUE: 69 SAT: 67 VAL: 34
IMG: 24 --> HUE: 60 SAT: 63 VAL: 60
IMG: 25 --> HUE: 60 SAT: 63 VAL: 56
IMG: 26 --> HUE: 60 SAT: 67 VAL: 60
IMG: 27 --> HUE: 60 SAT: 67 VAL: 60
IMG: 28 --> HUE: 60 SAT: 76 VAL: 56
IMG: 29 --> HUE: 69 SAT: 65 VAL: 63
IMG: 11 --> HUE: 67 SAT: 67 VAL: 58
IMG: 12 --> HUE: 60 SAT: 63 VAL: 63
IMG: 13 --> HUE: 60 SAT: 60 VAL: 56
IMG: 14 --> HUE: 60 SAT: 70 VAL: 49
IMG: 15 --> HUE: 60 SAT: 65 VAL: 18

VERD:
IMG: 16 --> HUE: 134 SAT: 60 VAL: 49
IMG: 17 --> HUE: 131 SAT: 60 VAL: 52
IMG: 18 --> HUE: 60 SAT: 50 VAL: 94
IMG: 19 --> HUE: 120 SAT: 50 VAL: 56
IMG: 20 --> HUE: 125 SAT: 69 VAL: 47
IMG: 19 --> HUE: 136 SAT: 30 VAL: 56
IMG: 20 --> HUE: 60 SAT: 14 VAL: 60
IMG: 21 --> HUE: 60 SAT: 14 VAL: 56
```

```
IMG: 22 --> HUE: 300 SAT: 30 VAL: 16
IMG: 23 --> HUE: 241 SAT: 18 VAL: 34
IMG: 24 --> HUE: 139 SAT: 30 VAL: 56
IMG: 30 --> HUE: 177 SAT: 70 VAL: 60
IMG: 31 --> HUE: 176 SAT: 67 VAL: 56
IMG: 32 --> HUE: 176 SAT: 43 VAL: 56
IMG: 33 --> HUE: 177 SAT: 61 VAL: 63
IMG: 34 --> HUE: 177 SAT: 60 VAL: 56
IMG: 35 --> HUE: 177 SAT: 52 VAL: 45
IMG: 36 --> HUE: 177 SAT: 67 VAL: 56
IMG: 37 --> HUE: 168 SAT: 30 VAL: 58
IMG: 38 --> HUE: 177 SAT: 60 VAL: 52
IMG: 39 --> HUE: 189 SAT: 60 VAL: 60
IMG: 40 --> HUE: 177 SAT: 65 VAL: 58
IMG: 41 --> HUE: 177 SAT: 61 VAL: 63
IMG: 16 --> HUE: 183 SAT: 74 VAL: 58
IMG: 17 --> HUE: 176 SAT: 74 VAL: 63
IMG: 18 --> HUE: 180 SAT: 40 VAL: 30
IMG: 19 --> HUE: 180 SAT: 29 VAL: 20
IMG: 20 --> HUE: 0 SAT: 48 VAL: 12

CIAN:
IMG: 21 --> HUE: 175 SAT: 65 VAL: 52
IMG: 22 --> HUE: 172 SAT: 60 VAL: 52
IMG: 23 --> HUE: 175 SAT: 58 VAL: 67
IMG: 24 --> HUE: 166 SAT: 58 VAL: 65
IMG: 25 --> HUE: 175 SAT: 60 VAL: 52
IMG: 26 --> HUE: 86 SAT: 23 VAL: 56
IMG: 27 --> HUE: 180 SAT: 14 VAL: 56
IMG: 28 --> HUE: 0 SAT: 9 VAL: 52
IMG: 29 --> HUE: 0 SAT: 20 VAL: 16
IMG: 30 --> HUE: 300 SAT: 20 VAL: 27
IMG: 31 --> HUE: 241 SAT: 7 VAL: 56
IMG: 42 --> HUE: 193 SAT: 78 VAL: 56
IMG: 43 --> HUE: 190 SAT: 76 VAL: 56
IMG: 44 --> HUE: 176 SAT: 52 VAL: 56
IMG: 45 --> HUE: 177 SAT: 65 VAL: 60
IMG: 46 --> HUE: 176 SAT: 70 VAL: 58
IMG: 47 --> HUE: 177 SAT: 61 VAL: 56
IMG: 48 --> HUE: 194 SAT: 81 VAL: 60
IMG: 49 --> HUE: 177 SAT: 54 VAL: 56
IMG: 50 --> HUE: 176 SAT: 70 VAL: 56
IMG: 21 --> HUE: 177 SAT: 76 VAL: 58
IMG: 22 --> HUE: 177 SAT: 65 VAL: 52
IMG: 23 --> HUE: 180 SAT: 29 VAL: 23
IMG: 24 --> HUE: 180 SAT: 29 VAL: 20
IMG: 25 --> HUE: 0 SAT: 16 VAL: 16
```

BLAU:

```
IMG: 26 --> HUE: 227 SAT: 60 VAL: 56
IMG: 27 --> HUE: 223 SAT: 56 VAL: 52
IMG: 28 --> HUE: 241 SAT: 10 VAL: 56
IMG: 29 --> HUE: 120 SAT: 9 VAL: 41
IMG: 30 --> HUE: 180 SAT: 9 VAL: 34
IMG: 32 --> HUE: 241 SAT: 9 VAL: 56
IMG: 33 --> HUE: 300 SAT: 10 VAL: 49
IMG: 34 --> HUE: 300 SAT: 9 VAL: 49
IMG: 35 --> HUE: 0 SAT: 34 VAL: 12
IMG: 36 --> HUE: 300 SAT: 20 VAL: 20
IMG: 52 --> HUE: 241 SAT: 74 VAL: 56
IMG: 53 --> HUE: 241 SAT: 60 VAL: 52
IMG: 54 --> HUE: 241 SAT: 30 VAL: 56
IMG: 55 --> HUE: 241 SAT: 20 VAL: 45
IMG: 56 --> HUE: 241 SAT: 40 VAL: 52
IMG: 57 --> HUE: 241 SAT: 45 VAL: 52
IMG: 58 --> HUE: 241 SAT: 56 VAL: 60
IMG: 59 --> HUE: 241 SAT: 58 VAL: 49
IMG: 60 --> HUE: 241 SAT: 43 VAL: 56
IMG: 26 --> HUE: 241 SAT: 50 VAL: 56
IMG: 27 --> HUE: 241 SAT: 40 VAL: 52
IMG: 28 --> HUE: 0 SAT: 21 VAL: 16
IMG: 29 --> HUE: 300 SAT: 40 VAL: 10
IMG: 30 --> HUE: 0 SAT: 38 VAL: 10
```

VIOLETA:

```
IMG: 31 --> HUE: 241 SAT: 10 VAL: 45
IMG: 32 --> HUE: 0 SAT: 7 VAL: 94
IMG: 34 --> HUE: 60 SAT: 7 VAL: 60
IMG: 35 --> HUE: 120 SAT: 3 VAL: 60
IMG: 36 --> HUE: 241 SAT: 7 VAL: 45
IMG: 38 --> HUE: 120 SAT: 7 VAL: 56
IMG: 39 --> HUE: 241 SAT: 27 VAL: 60
IMG: 40 --> HUE: 241 SAT: 18 VAL: 60
IMG: 41 --> HUE: 0 SAT: 34 VAL: 12
IMG: 42 --> HUE: 300 SAT: 20 VAL: 20
IMG: 43 --> HUE: 300 SAT: 14 VAL: 49
IMG: 62 --> HUE: 251 SAT: 69 VAL: 58
IMG: 63 --> HUE: 247 SAT: 74 VAL: 52
IMG: 64 --> HUE: 241 SAT: 43 VAL: 52
IMG: 65 --> HUE: 241 SAT: 36 VAL: 52
IMG: 66 --> HUE: 241 SAT: 47 VAL: 52
IMG: 67 --> HUE: 254 SAT: 50 VAL: 52
IMG: 68 --> HUE: 241 SAT: 50 VAL: 52
IMG: 69 --> HUE: 241 SAT: 50 VAL: 52
IMG: 70 --> HUE: 241 SAT: 30 VAL: 56
IMG: 71 --> HUE: 241 SAT: 56 VAL: 52
```

```
IMG: 31 --> HUE: 241 SAT: 50 VAL: 56
IMG: 32 --> HUE: 241 SAT: 36 VAL: 60
IMG: 33 --> HUE: 241 SAT: 29 VAL: 20
IMG: 34 --> HUE: 120 SAT: 27 VAL: 12
IMG: 35 --> HUE: 120 SAT: 27 VAL: 12
```

A continuació es mostra el codi font del programa. Com s'ha comentat, es basa en la mateixa idea que el programa principal però amb el tractament del botó i la memòria SD. No està tant depurat com el programa principal ja que era només un programa de test. Aquest és el codi:

```
////////////////////////////////////
// EYECOLOR - Matias Lizana García //
////////////////////////////////////

#include "eyecolor.h"
#include <cc3.h>
#include "../../../cc3/hal/lpc2106-cmucam3/cc3_pin_defines.h"
#include "../../../cc3/hal/lpc2106-cmucam3/LPC2100.h"
#include <stdio.h>
#include <stdlib.h>
#include <cc3_hsv.h>

void capture_ppm(FILE *f);
void init_capture_ppm();

FILE* log;

bool eyeborg_init = false;
uint8_t *row;
int hue[360];
int maxhue = 0;
int sat[100];
int maxsat = 0;
int val[100];
int maxval = 0;

int numfile=0;

int main(void) {
    cc3_uart_init(0,CC3_UART_RATE_115200,
                  CC3_UART_MODE_8N1,
                  CC3_UART_BINMODE_TEXT);
    cc3_camera_init ();
    cc3_filesystem_init();
    cc3_camera_set_resolution (CC3_CAMERA_RESOLUTION_LOW);
```

```
    row = cc3_malloc_rows(1);

    cc3_timer_wait_ms(1000);

    while(true) {
        if(cc3_button_get_state()) {
            init_capture_ppm(numfile++);
            calculate_histogram();
            while(cc3_button_get_state());
        }
    }

    return 0;
}

void capture_ppm(FILE *f)
{

    cc3_pixbuf_load ();

    memset(hue,'\0',sizeof(hue));
    memset(sat,'\0',sizeof(sat));
    memset(val,'\0',sizeof(val));

    for (uint16_t y = 0; y < cc3_g_pixbuf_frame.height; y++) {
        cc3_pixbuf_read_row(row);
        cc3_rgb2hsv_row((cc3_pixel_t*)row,cc3_g_pixbuf_frame.width);
        for (uint16_t x = 0; x < cc3_g_pixbuf_frame.width; * 3U; x+=3) {
            if (fputc(row[x], f) == EOF) perror("fputc failed");
            if (fputc(row[x+1], f) == EOF) perror("fputc failed");
            if (fputc(row[x+2], f) == EOF) perror("fputc failed");
            hue[(row[x] * 360) / 255]++;
            sat[row[x+1]]++;
            val[row[x+2]]++;
        }
    }
}

void calculate_histogram()
{

    int maxh = 0, maxs = 0, maxv = 0;

    for(int k=0; k<360; k++) {
        if(hue[k] > maxh) {
            maxh = hue[k];
```

```
                maxhue = k;
            }
        }
        for(int k=0; k<100; k++) {
            if(sat[k] > maxs) {
                maxs = sat[k];
                maxsat = k;
            }
            if(val[k] > maxv) {
                maxv = val[k];
                maxval = k;
            }
        }

        log = fopen("c:/colortest","a");
        fprintf(log,"IMG:    %d    -->    HUE:    %d    SAT:    %d    VAL:    %d
\n",numfile,maxhue,maxsat,maxval);
        fclose(log);
}

void init_capture_ppm() {

  int result;
  FILE *f;
  bool light_on = true;
  char filename[16];
  uint32_t size_x, size_y;

    do {
      snprintf(filename, 16, "c:/img%.5d.ppm", numfile);
      f = fopen(filename, "r");
      if (f != NULL) {
    printf( "%s already exists...\n",filename );
    numfile++;
    result = fclose(f);
    if (result) {
      perror("first fclose failed");
    }
      }
    } while(f != NULL);

    fprintf(stderr,"%s ", filename);
    fflush(stderr);
    f = fopen(filename, "w");

    if (f == NULL || numfile > 512) {
```

```
      if (f == NULL) {
    perror("crap");
      } else {
    fprintf(stderr, "full\n");
      }

      while (true) {
    cc3_led_set_state(0, true);
    cc3_led_set_state(2, true);
    cc3_timer_wait_ms(500);
    cc3_led_set_state(0, false);
    cc3_led_set_state(2, false);
    cc3_timer_wait_ms(500);
      }
    }


    if (light_on) {
      cc3_led_set_state (2, true);
    } else {
      cc3_led_set_state (2, false);
    }
    light_on = !light_on;

    size_x = cc3_g_pixbuf_frame.width;
    size_y = cc3_g_pixbuf_frame.height;

    fprintf(f,"P6\n%d %d\n255\n",size_x/2,size_y/2);

    capture_ppm(f);

    result = fclose(f);
    if (result) {
      perror("second fclose failed");
    }
    fprintf(stderr, "\r\n");

}
```

15 Annex IV: Funcions no optimitzades

En aquest apartat es mostren les funcions que s'han tractat a l'apartat d'optimització, però sense optimitzar, per poder veure el seu antic comportament:

cc3_pixbuf_read_rows

```
int cc3_pixbuf_read_rows (void * mem, uint32_t rows)
{

  int16_t j;
  uint16_t r;

  uint8_t off0, off1, off2;

  int width = cc3_g_pixbuf_frame.width;
 unsigned int row_limit = (cc3_g_pixbuf_frame.y1 -
cc3_g_pixbuf_frame.y_loc)/cc3_g_pixbuf_frame.y_step;

  if (row_limit < rows) {
    rows = row_limit;
  }

  if (_cc3_g_current_camera_state.colorspace == CC3_COLORSPACE_RGB) {
    off0 = 0;
    off1 = 1;
    off2 = 2;
  }
   else if (_cc3_g_current_camera_state.colorspace == CC3_COLORSPACE_YCRCB)
{
    off0 = 1;
    off1 = 0;
    off2 = 2;
  }
  else {
    off0 = 0;
    off1 = 1;
    off2 = 2;
  }

  // First read into frame
  _cc3_seek_top ();

  for (r = 0; r < rows; r++) {
    int x = cc3_g_pixbuf_frame.x0;

    // First read into line
```

```
_cc3_seek_left ();

switch (cc3_g_pixbuf_frame.coi) {
case CC3_CHANNEL_ALL:
  _cc3_second_green_valid = false;
  for (j = 0; j < width; j++) {
    uint8_t *p = ((uint8_t *) mem) +
      (r * width + j * 3);
    _cc3_pixbuf_read_pixel (p, p - 3, off0, off1, off2);

// advance by x_step
x += cc3_g_pixbuf_frame.x_step;
_cc3_pixbuf_skip_pixels ((cc3_g_pixbuf_frame.x_step - 1) / 2);
  }

  break;

case CC3_CHANNEL_RED:
  for (j = 0; j < width; j++) {
uint8_t *p = ((uint8_t *) mem) + (r * width + j);

if ((j & 0x1) == 0 || cc3_g_pixbuf_frame.x_step > 1) {
  // read
  _cc3_pixbuf_skip_subpixel ();
  *p = _cc3_pixbuf_read_subpixel ();
  _cc3_pixbuf_skip_subpixel ();
  _cc3_pixbuf_skip_subpixel ();
} else {
  *p = *(p - 1);
}

x += cc3_g_pixbuf_frame.x_step;
_cc3_pixbuf_skip_pixels ((cc3_g_pixbuf_frame.x_step - 1) / 2);
  }
  break;

case CC3_CHANNEL_GREEN:
  for (j = 0; j < width; j++) {
uint8_t *p = ((uint8_t *) mem) + (r * width + j);

if ((j & 0x1) == 0 || cc3_g_pixbuf_frame.x_step > 1) {
  // read
  *p = _cc3_pixbuf_read_subpixel ();
  _cc3_pixbuf_skip_subpixel ();
  _cc3_second_green = _cc3_pixbuf_read_subpixel ();
  _cc3_pixbuf_skip_subpixel ();
```

```
    } else {
      *p = _cc3_second_green;
    }

    x += cc3_g_pixbuf_frame.x_step;
    _cc3_pixbuf_skip_pixels ((cc3_g_pixbuf_frame.x_step - 1) / 2);
      }
      break;

    case CC3_CHANNEL_BLUE:
      for (j = 0; j < width; j++) {
    uint8_t *p = ((uint8_t *) mem) + (r * width + j);

    if ((j & 0x1) == 0 || cc3_g_pixbuf_frame.x_step > 1) {
      // read
      _cc3_pixbuf_skip_subpixel ();
      _cc3_pixbuf_skip_subpixel ();
      _cc3_pixbuf_skip_subpixel ();
      *p = _cc3_pixbuf_read_subpixel ();
    } else {
      *p = *(p - 1);
    }

    x += cc3_g_pixbuf_frame.x_step;
    _cc3_pixbuf_skip_pixels ((cc3_g_pixbuf_frame.x_step - 1) / 2);
      }
      break;
    }
    _cc3_pixbuf_skip_pixels ((cc3_g_pixbuf_frame.raw_width - x) / 2);


    // advance by y_step
          _cc3_pixbuf_skip_pixels   ((cc3_g_pixbuf_frame.y_step   -   1)   *
cc3_g_pixbuf_frame.raw_width / 2);
    cc3_g_pixbuf_frame.y_loc += cc3_g_pixbuf_frame.y_step;
  }
  return rows;
}
```

cc3_pixbuf_read_pixel

```
void _cc3_pixbuf_read_pixel (uint8_t * pixel,
                             uint8_t * saved,
                             uint8_t off0, uint8_t off1, uint8_t off2)
{
  if (cc3_g_pixbuf_frame.x_step == 1) {
    if (_cc3_second_green_valid) {
      // use the second green
      _cc3_second_green_valid = false;
      *(pixel + off0) = *(saved + off0);
      *(pixel + off1) = _cc3_second_green;
      *(pixel + off2) = *(saved + off2);

      return;
    }

    // otherwise, load a new thing
    *(pixel + off1) = _cc3_pixbuf_read_subpixel ();   // G
    *(pixel + off0) = _cc3_pixbuf_read_subpixel ();   // R
    _cc3_second_green = _cc3_pixbuf_read_subpixel (); // G
    *(pixel + off2) = _cc3_pixbuf_read_subpixel ();   // B

    _cc3_second_green_valid = true;
  } else {
    _cc3_pixbuf_skip_subpixel ();
    *(pixel + off0) = _cc3_pixbuf_read_subpixel ();
    *(pixel + off1) = _cc3_pixbuf_read_subpixel ();
    *(pixel + off2) = _cc3_pixbuf_read_subpixel ();
  }
}
```

cc3_rgb2hsv

```
inline void cc3_rgb2hsv (cc3_pixel_t * pix) {
  uint8_t hue, sat, val;
  uint8_t rgb_min, rgb_max;
  rgb_max = 0;
  rgb_min = 255;
  if (pix->channel[CC3_CHANNEL_RED] > rgb_max)
    rgb_max = pix->channel[CC3_CHANNEL_RED];
  if (pix->channel[CC3_CHANNEL_GREEN] > rgb_max)
    rgb_max = pix->channel[CC3_CHANNEL_GREEN];
  if (pix->channel[CC3_CHANNEL_BLUE] > rgb_max)
    rgb_max = pix->channel[CC3_CHANNEL_BLUE];
  if (pix->channel[CC3_CHANNEL_RED] < rgb_min)
```

```
    rgb_min = pix->channel[CC3_CHANNEL_RED];
  if (pix->channel[CC3_CHANNEL_GREEN] < rgb_min)
    rgb_min = pix->channel[CC3_CHANNEL_GREEN];
  if (pix->channel[CC3_CHANNEL_BLUE] < rgb_min)
    rgb_min = pix->channel[CC3_CHANNEL_BLUE];

// compute V
  val = rgb_max;
  if (val == 0) {
    hue = sat = 0;
    pix->channel[CC3_CHANNEL_HUE] = 0;
    pix->channel[CC3_CHANNEL_SAT] = 0;
    pix->channel[CC3_CHANNEL_VAL] = val;
    return;
  }

// compute S
  sat = 255 * (rgb_max - rgb_min) / val;
  if (sat == 0) {
    pix->channel[CC3_CHANNEL_HUE] = 0;
    pix->channel[CC3_CHANNEL_SAT] = 0;
    pix->channel[CC3_CHANNEL_VAL] = val;
    return;
  }

// compute H
  if (rgb_max == pix->channel[CC3_CHANNEL_RED]) {
    hue =
      0 + 43 * (pix->channel[CC3_CHANNEL_GREEN] -
                pix->channel[CC3_CHANNEL_BLUE]) / (rgb_max - rgb_min);
  }
  else if (rgb_max == pix->channel[CC3_CHANNEL_GREEN]) {
    hue =
      85 + 43 * (pix->channel[CC3_CHANNEL_BLUE] -
                 pix->channel[CC3_CHANNEL_RED]) / (rgb_max - rgb_min);
  }
  else {                        /* rgb_max == blue */

    hue =
      171 + 43 * (pix->channel[CC3_CHANNEL_RED] -
                  pix->channel[CC3_CHANNEL_GREEN]) / (rgb_max - rgb_min);
  }
  pix->channel[CC3_CHANNEL_HUE] = hue;
  pix->channel[CC3_CHANNEL_SAT] = sat;
  pix->channel[CC3_CHANNEL_VAL] = val;
}
```

16 Bibliografia

Personal

Blog oficial del projecte – http://eyeb.org

Plataforma

Pàgina oficial de CMUCam3 - http://cmucam.org/

Pàgina oficial ARM - http://www.arm.com/products/processors/classic/arm7/index.php

Pàgina GNU ARM toolchain - http://www.codesourcery.com/sgpp/lite/arm/

Manual LPC2106 - http://cmucam.org/attachment/wiki/Documentation/lpc2106.pdf?format=raw

Manual OV6620 - http://cmucam.org/attachment/wiki/Documentation/OV6620.PDF?format=raw

Manual AL440B - http://cmucam.org/attachment/wiki/Documentation/al440B.pdf?format=raw

Acromatòpsia

Document de referència de la xarxa d'acromatòpsia a internet - http://www.achromat.org/uc.pdf

Models de color

RGB Wikipedia - http://es.wikipedia.org/wiki/Modelo_de_color_RGB

HSV Wikipedia - http://es.wikipedia.org/wiki/Modelo_de_color_HSV

Filtre de bayer

Broadcast Engineer's – Reference Book

E.P.J. Tozer – Elsevier (2004)

Silicon imaging - http://www.siliconimaging.com/RGB%20Bayer.htm

PWM

Llibre Physical Computing – Sensing and Controlling the Phisycal World with Computers

Dan O'Sullivan i Tom Igoe – Thomson course technology (2004)

PWM Wikipedia - http://en.wikipedia.org/wiki/Pulse-width_modulation

Netrino: Web de sistemes *embedded* -

http://www.netrino.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation

PCBHeaven: Directori d'electrònica - http://pcbheaven.com/wikipages/PWM_Modulation/

17 Contingut del CD Adjunt

El CD adjunt conté els següents arxius:

- Memòria en PDF
- Codi font del projecte
- Dades i imatges preses en el test de color
- Els tres manuals de referència sobre la plataforma
  - Microcontrolador LPC2106
  - Sensor OV6620
  - Memòria AL440B
- Entrevista a TV3 programa “Entre Línies”