# Sistemes Encastats

# Memòria de les Pràctiques

#### Realitzat per: Adrià Auguets i Pavel Macutela. 29 de Maig del 2017

# Introducció

En les pràctiques de Sistemes Encastats hem realitzat un detector de DTMF utilitzant l’algoritme de Goertzel. La gràcia d’aquestes pràctiques ha estat experimentar quins són els resultats que podem arribar a obtenir si implementem el DTMF en tres dispositius diferents.

Els dispositius que vam tenir oportunitat de programar, van ser:

- Un microcontrolador Arduino.

- Una FPGA (DE0-Nano).

- Un Sistema Operatiu, el nostre portàtil i una Raspberry Pi 3 Model B.

En tots aquests dispositius hem detectat avantatges i inconvenients en el seu ús, tots els resultats es podran comparar més endavant.

## DTMF - Realitzat amb el Microcontrolador Arduino.

La primera pràctica es va realitzar sobre l’entorn Arduino. Com sempre tota primera tasca té les seves complicacions i aquesta no en va ser l’excepció. Nosaltres abans d’assistir a la primera pràctica vam realitzar un estudi previ per familiaritzar-nos amb el que era el **Dual-Tone Multi-Frequency signaling i l’Algoritme de Goertzel**.

Un cop assolits aquests coneixements i de fer unes primeres proves en Octave, vam començar a realitzar el nostre codi.

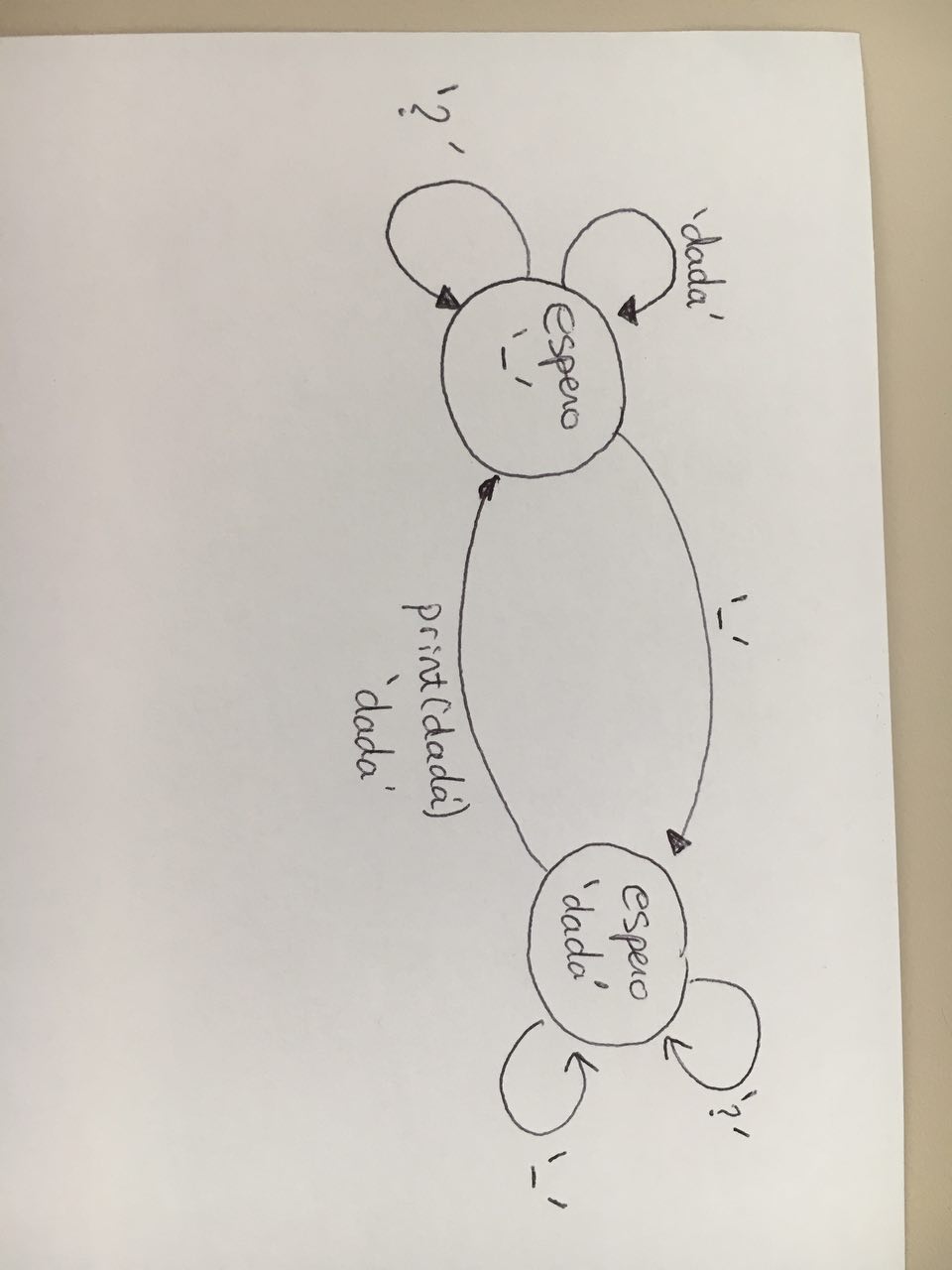
**DTMF.c :**

El nostre mòdul principal, nosaltres considerem que el tenim ben estructurat i fàcil d’entendre. Així com, estructurem el codi de manera que en les primeres línies, com es de costum, tenim els *includes* i les declaracions de les variables amb un nom fàcil d’entendre. Tot seguit es poden veure les funcions que usem i després de les funcions hi trobem el main. Al final del programa, sempre hi deixem lloc per la Interrupció.

Els coeficients que fem servir en l’script els hem calculat prèviament amb Octave. Ho podem veure en la Font 1.

Nosaltres, comencem a tractar les mostres dins la interrupció fins que en tenim **205**. Llavors, en **calculem la potencia utilitzant vuit filtres** i directament anem a una màquina d’estats. Aquesta màquina d’estats, primer s’encarrega de detectar quin número o lletra esta sonant o al contrari, no sona res. Després el mostra per pantalla segons toqui segons la màquina d’estats.

La màquina d’estats que hem implementat és la següent:



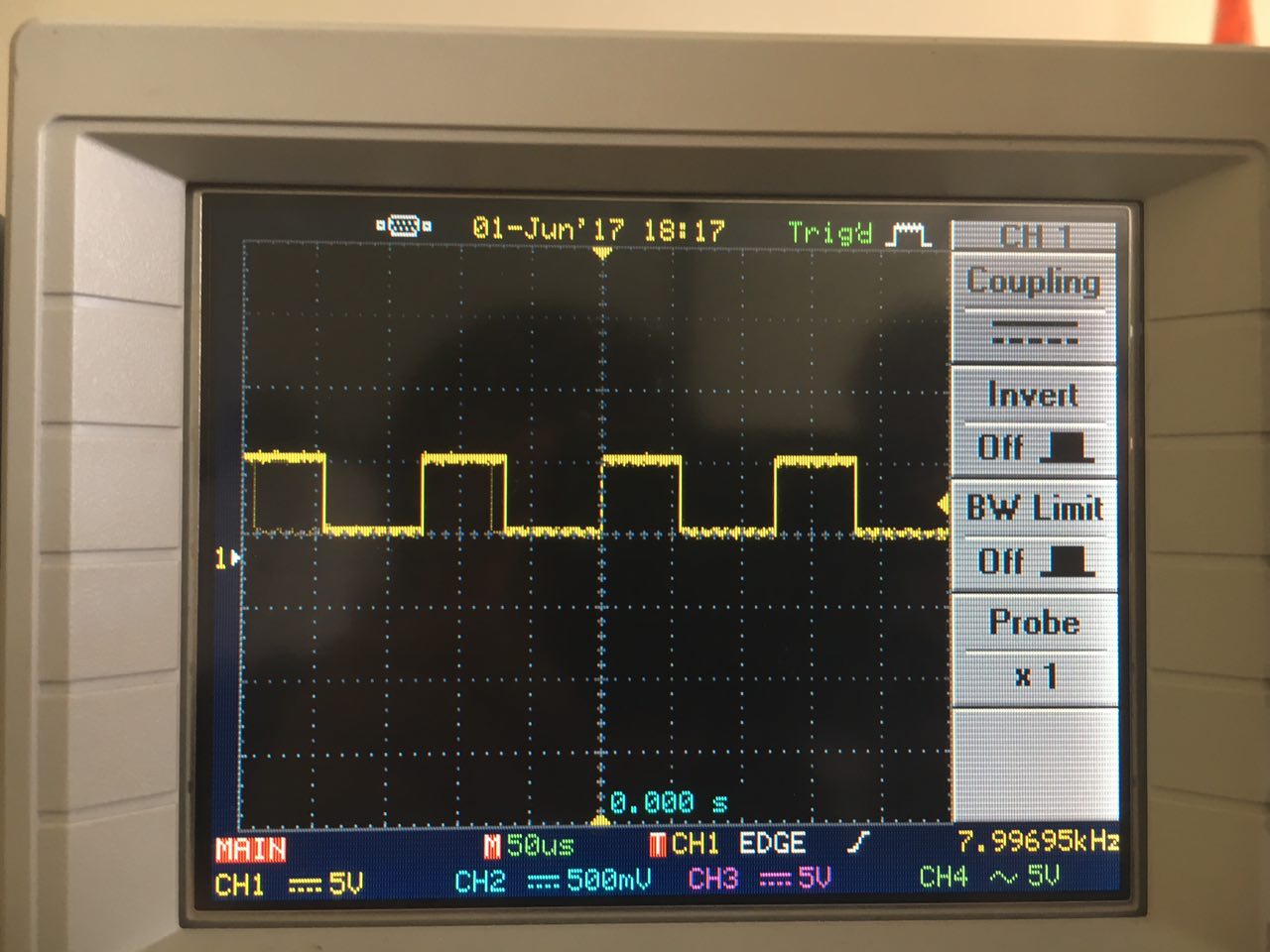
Màquina d’Estats.

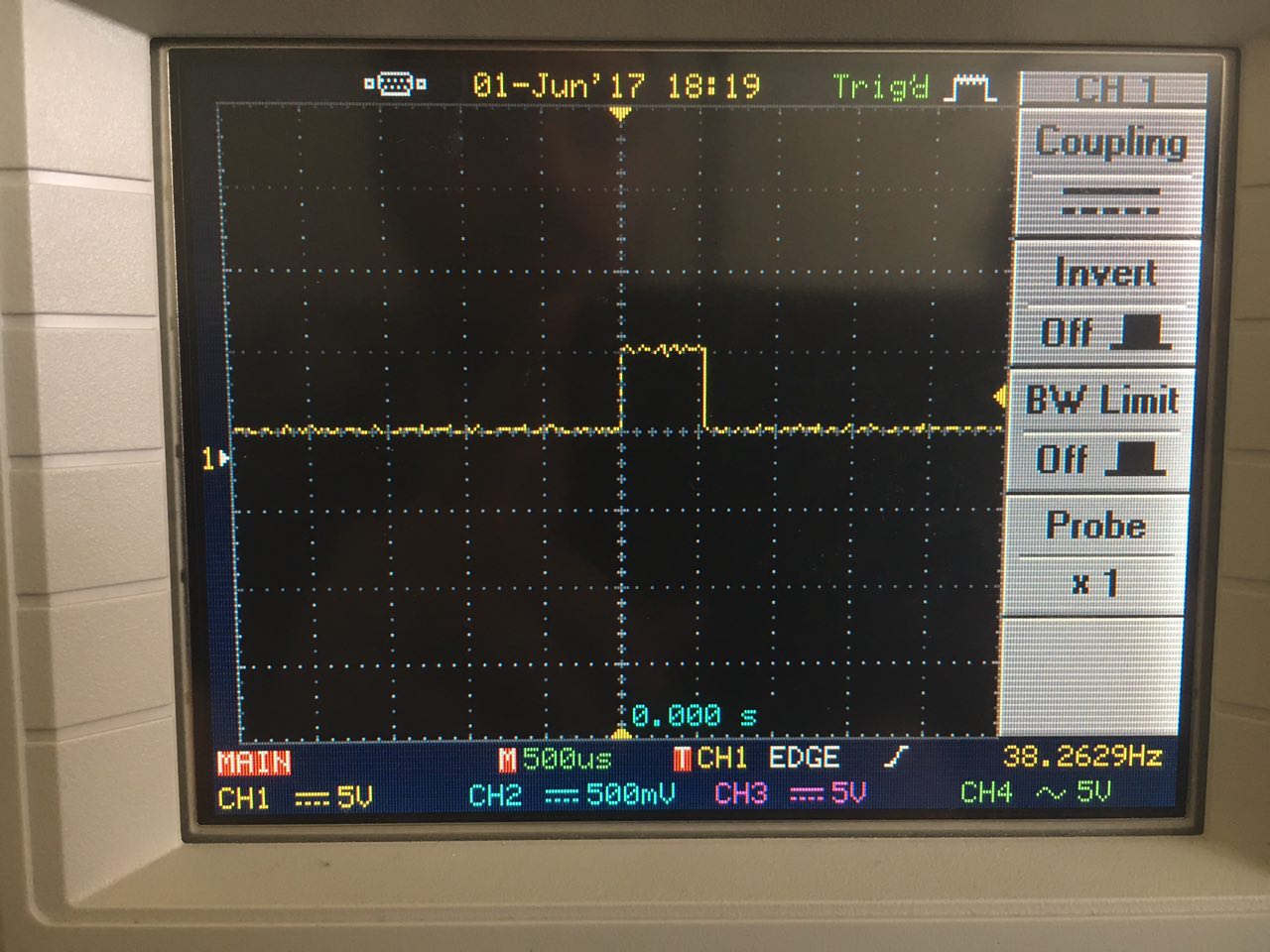
Creiem que un apartat que ens diferència de la resta, es la manera que nosaltres detectem el resultat, és a dir, a partir de quines freqüències estan sonant, saber a quin botó fan referencia. Nosaltres només definim *“resultats[16] = {'1','2','3','A','4','5','6','B','7','8','9','C','\*','0','#','D'};”* i per saber a quina posició de la llista resultats fan referencia les dues freqüències nosaltres hem creat la funció, **uint8\_t detectaResultat(void)**, Font 2.

Aquesta funció mira quines són les freqüències baixes que superen el llindar i actualitza la variable *posició* (+= i\*4) i es posa a true un boleà, depenent de quina superi el llindar. Després fem el mateix amb les altes( += i-4) i també es posa a true un boleà. Alhora per a cada freqüència que superi el llindar, es suma 1 a la variable trobat.

Finalment comparem si els dos booleans donen true i que el valor de trobat es igual a dos. Si això es compleix, anem a buscar el valor que ocupa la posició “*posició*” a la llista resultats.

Per acabar, vam comprovar amb l’oscil·loscopi els temps que tardava l’Arduino a fer el càlcul de la interrupció(Font 3) i els càlculs per determinar quin caràcter és(Font 4).





Font 3: Temps en la interrupció. Font 4: Càlculs per determinar el resultat.

Veient aquests resultats, podem veure que els temps que hi dedica a la interrupció són 50us, 75us en fer càlculs i en determinar el resultat 500us. Podem extreure la conclusió que el microcontrolador hi dedica 300us en fer els càlculs ja que 200us els dedicarà en fer la interrupció.

Que hem aprés en aquesta pràctica d’Arduino?

Com a coneixements que hem assolit realitzant-ho a un microcontrolador, podem destacar que s’ha de tenir sempre en compte els temps d’execució a la hora d’implementar el codi.

També hem aprés a fer servir “càsting” i com tractar amb valors que són decimals, transformant-los en enters, fer càlculs i després tornar a tenir el resultat com si haguéssim fet el càlcul amb floats però amb un temps molt més ràpid.

Per acabar, hem vist com l’Arduino dedica un temps per fer la interrupció i un altre per fer els càlculs externs de la interrupció. Amb això hem vist com pot arribar a tardar l’Arduino a fer els càlculs segons el temps que hi dediqui a l’interrupció.

## DTMF - Realitzat amb una FPGA.

La segona part de la pràctica com hem comentat anteriorment, utilitzem la FPGA (un dispositiu hard amb més recursos que l’Arduino), per tal de veure quin seria el resultat obtingut de la mateixa implementació del DTMF.

Per aquesta pràctica utilitzem la FPGA DEO-nano, llenguatge VHDL i Quartus per poder compilar i carregar el programa dintre de la FPGA. També farem servir l’Arduino ja que la FPGA no te ADC.

Per crear el programa de la FPGA utilitzarem tres capes, la primera per calcular l’Algoritme de Goertzel, la segona per descodificar el caràcter i la tercera que serà la màquina d’estats que vam implementar a la pràctica anterior.

Cada filtre tindrà un coeficient que hem calculat amb Octave, es veu en la Font 1, aquets filtre calcula la potència el valor de la «a\_int» per tal de calcular la potencia de la señal. La segona capa el «codificador.vhd» que depenent de la entrada posarem un valor a la sortida, juntament amb l’enable quan es un valor que ens interessa.

La tercera capa es la màquina d’estats, la nostra implementació te la peculiaritat que el enable de la capa3 està actiu durant molt poc temps(un clock), ja que cambiem d’estat rapidament depenent del que ens arribi.

El tercer mòdul és on implemeten la maquina d’estats, (esperem dada, o esperem silenci) si la entrada enable del mòdul anterior es 1 i estem esperant una dada, si la entrada es un silenci o un valor indeterminat ens quedem en el mateix estat (esperant dada) i desactivem la sortida enable; si es un valor que ens interesa agafem les dades de entrada (de menys pes) i la posem a cada una de les quatre sortides i activem l’enable y passem a l’estat de esperem un silenci. La sortida seran 5 cables , 4 cables que ens diran quin número és i l’enable.

Aquestes 5 sortides seran l’entrada de l’Arduino, on sempre està mirant l’estat del pin on esta connectat l’Enable, quan aquesta esta activada, llegim els pins on estan connectats els altres cables, descodifiquem i enviem pel port sèrie el caràcter, on mostrarem pel terminal quin caràcter és.

Al compilar el projecte es veu la quantitat de multiplicadors que utilitzarem per fer el DTMF un total de 48 multiplicadors.

Al treballar amb FPGA i tenir més recursos que l’Arduino veiem que treballar amb operacions de números grans es fa més ràpid ja que es treballen paral·lelament en cada filtre.

La dificultat que es pot presentar en el projecte es a la hora de recepció del caràcter en l’Arduino ja que el temps d’enable que surt de la FPGA dura poc temps (25 ms). Això pot provocar que es perdin dades.

## DTMF - Realitzat amb dos Sistemes Operatius.

La darrera pràctica consisteix en realitzar un DTMF en l’entorn de un Sistema Operatiu. El nostre primer objectiu va ser traduir el codi C que vam fer en la primera pràctica a Python. El nostre codi Python seguia la mateixa estructura i idea que el codi C. L’avantatge que vam tenir, és que, en Python no havíem de declarar variables segons el tipus, o utilitzar casts.

A diferencia d’altres grups nosaltres vam realitzar dos programes Python. Un fet amb bucles i l’altre sense bucles. Això ho vam fer per tal de demostrar que sense bucles el programa Python realitzava més ràpidament els càlculs.

Com a resultat de fer els primers experiments en el Sistema Operatiu Mac, podem veure que en el Python amb bucles tarda uns 0.0256403 segons de mitjana en llegir i tractar un caràcter. En canvi, utilitzant el programa Python sense bucles, veiem tarda uns 0.0256395 segons.

En Raspberry Pi 3 Model B, utilitzant el Python amb bucles podem veure que a tractar un caràcter tarda uns 0.02569947segons. En canvi sense bucles tarda 0.02563410s.

A continuació podem veure una taula amb una mitjana de temps que el programa Python tarda en tractar els dos codis.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Test 1 | | Test 2 | |
|  | bucles.py | NoBucles.py | Bucles.py | NoBucles.py |
| Ordinador | 0.00105690s | 0.000808000s | 0.00107693s | 0.00079393s |
| RPi3 Model B | 0.01017594s | 0.008363962s | 0.01005101s | 0.00839090s |

Taula de mitjanes de temps

Gràcies a fer dos codis hem pogut comprovar que el Python sense fer bucles es molt més ràpid. També vam fer proves amb una de les primeres Raspberry’s i només el codi sense bucles tenia oportunitat de funcionar correctament.

En aquesta pràctica de DTMF implementat en un Sistema Operatiu ens hem adonat que implementar-ho és molt més ràpid que en altres dispositius. També hem vist com funcionen els buffers tant a el Sistema Operatiu MAC com en la Raspberry.

Hem comprovat que un codi en Python que no utilitza bucles per fer la mateixa feina que un amb bucles, es molt més ràpid.

Hem conegut millor l’entorn de la Raspberry Pi i ara sabem que la RasPi 3 ModelB es molt millor que la de la primera generació per fer aquest tipus de feina.

# Conclusions Globals.

Les pràctiques de Sistemes Encastats ens han servit per aprendre moltes coses. Hem aprés com programar de manera òptima un microcontrolador i una Raspberry Pi per tal de reduir el temps de càlcul de les operacions. A part d’això hem utilitzat una FPGA i el nostre ordinador per fer la mateixa feina.

Hem arribat a la conclusió que realitzar la mateixa feina en una FPGA resulta més costosa que fer-la amb un Arduino, però els resultats són molt millors. En canvi, realitzar-ho al nostre portàtil i en Python ens ha estat molt mes fàcil que amb l’Arduino però els resultats no són tant bons com els que ens donava el microcontrolador.

Per tant, si tinguéssim poc temps ho faríem amb la Raspberry Pi o l’Arduino. En canvi si ens donessin molt més de temps, faríem el DTMF amb la FPGA i a més a més podríem fer dos canals de DTMF sense cap problema (2 DTMF = 96 multiplicadors de 132 que te la nostra FPGA).

La nostra opinió sobre el laboratori:

Creiem que el fet de treballar sense cap guió no ha estat tant mala idea, ja que ens has donat tot el suport necessari a les hores de teoria.

Un bon aspecte al laboratori es que ens has donat llibertat per fer-ho com hem volgut dins de uns límits.

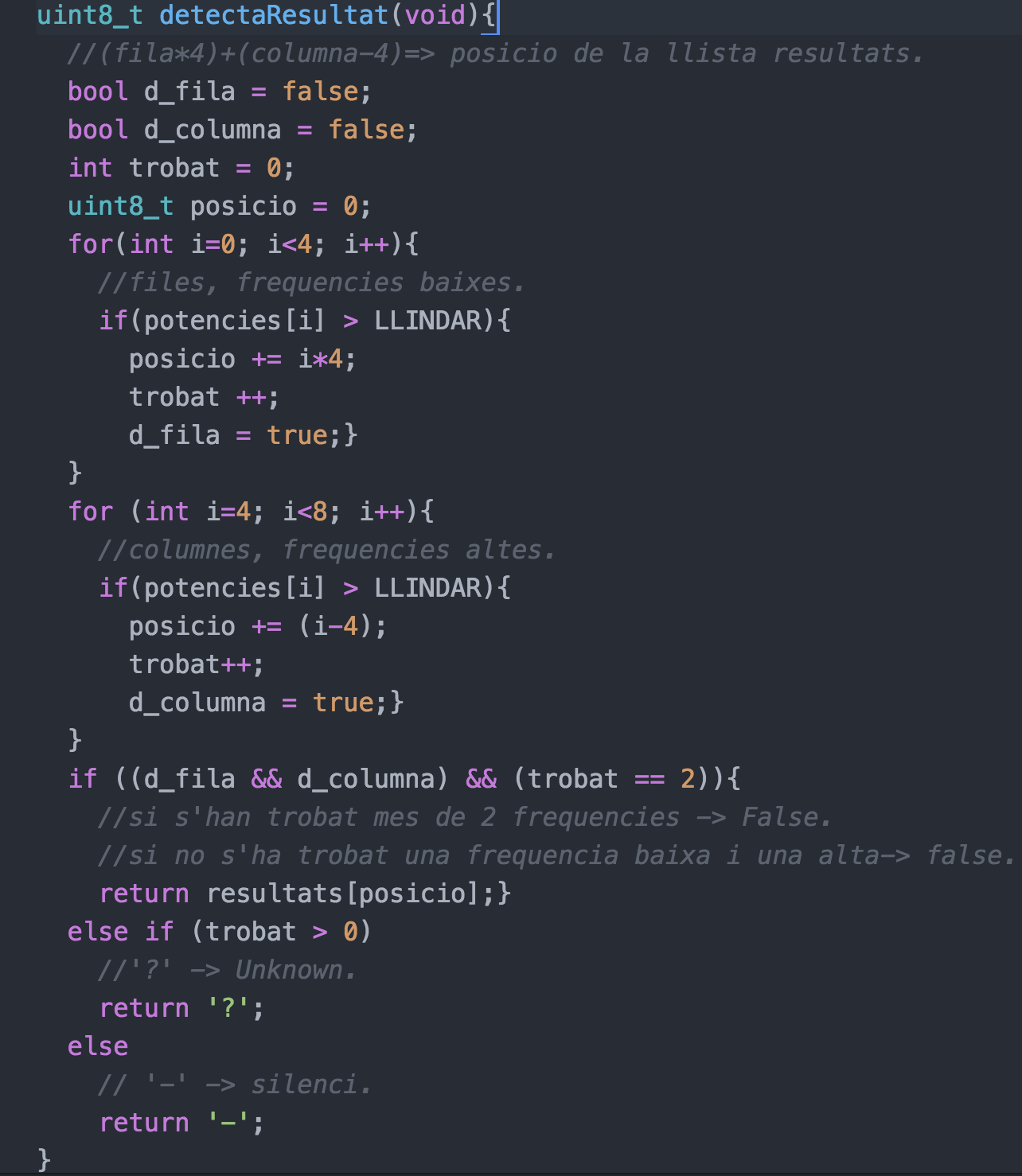
Ens ha semblat molt interessant el fet de repetir el mateix projecte en tres sistemes diferents i poder comparar-los, extraient així les parts bones i dolentes de cada dispositiu.

Nosaltres creiem que el fet de haver treballat en 3 plataformes diferents ens ha ajudat a assolir el concepte de la assignatura “Sistemes Encastats”.

Imatges:



Font 1: Mostra del càlcul dels Coeficients amb Octave.



Font2: Funció detectaResultat.