PROJECTE D'ENGINYERIA DE COMPUTADORS

DOCUMENTACIÓ DE L'AMPLIACIÓ DEL PROCESSADOR: CONTROLADOR DE SO I SISTEMA OPERATIU

JOEL ACEDO DELGADO
CHRISTIAN MELÉNDEZ NÚÑEZ
ANTONI NAVARRO MUÑOZ
JOAN ORRIT PALAU
MARC PERELLÓ BACARDIT
FÉLIX FERNANDO RAMOS VELÁZQUEZ

2015-2016 QP
20/06/2016

ÍNDEX

INTRODUCCIÓ – PROCESSADOR ESCOLLIT

Abans de començar a implementar parts de l'ampliació, vam escollir el processador al que aplicar-li l'ampliació. La decisió es va prendre de la següent manera:

Només dos grups, dels quatre que fem l'ampliació, teníem el processador acabat fins l'etapa 7.3, pel que vam descartar d'altres processadors més endarrerits, bàsicament per falta de temps. Entre els dos més avançats, vam decidir (amb l'aprovació del professor) de que no faríem l'etapa del TLB, ja que no era essencial per la nostra ampliació i així ens podríem centrar en aquesta.

Entre aquests dos grups, vam triar el processador del grup 3 (Marc i Antoni) ja que no sabíem quin dels dos processadors funcionaria millor (no teníem la correcció) però vam veure que el codi era més entenedor i adaptable.

SISTEMA OPERATIU

Un cop feta l'entrega del processador per grups de dos, aquest processador era capaç d'executar tot tipus d'instruccions aritmètico-lògiques, de comparació, salts, d'entrada/sortida, privilegiades, etc. També era capaç de distingir entre mode usuari i mode sistema, la qual cosa permetia restringir certs accessos a memòria i execució d'instruccions privilegiades per part d'un procés d'usuari. Vam implementar interrupcions, excepcions i fins i tot la possibilitat d'afegir crides a sistema(instrucció CALLS).

Tot i haver implementat totes aquestes funcionalitats, però, el processador estava limitat a executar un sol procés carregat prèviament a memòria. Si tenim implementada una interrupció de timer, perquè no dissenyar un sistema operatiu que sigui capaç d'executar com a mínim dos programes concurrentment? D'aquesta manera podem posar en pràctica la implementació d'un "quantum" i del canvi de context entre processos.

Igual que en el cas d'un processador real, el hardware serà el mateix (amb alguna petita modificació puntual), la única cosa que s'haurà de canviar és afegir una nova capa de software entre el hardware i els usuaris, és a dir, un sistema operatiu. Aquest serà el que s'encarregarà de, fent ús de les interrupcions de timer, decidir quin procés s'executa, si s'ha de passar a executar el següent procés (canvi de context), què fer en cas d'interrupció/excepció i tractar les crides a sistema (en el cas que n'hi hagin).

DECISIONS PRESES

- Primer de tot, vam decidir que el sistema operatiu que implementarem es limitarà a executar concurrentment dos processos. La raó d'això és simplement per una qüestió de limitació temporal. Millor no perdre temps en augmentar la quantitat de processos, ja que en veritat amb dos en tenim prou perquè el sistema operatiu sigui multiprocés. Afegir-ne més només complicaria les coses i podria causar que fóssim incapaços d'acabar el sistema operatiu.

- Degut també al temps limitat del que disposem, hem relaxat la protecció de les regions de codi. Per una part, mantenim la protecció entre usuari i sistema, és a dir, evitem que l'usuari sigui capaç d'accedir a codi o dades de sistema. Per altra banda, no tenim cap tipus de protecció entre usuaris. Això vol dir que un usuari és capaç de sobre-escriure el codi i les dades de l'altre.

- Hem decidit que el tractament de les excepcions sigui parar el programa, és a dir, l'execució d'un halt, ja que suposem que els programes d'usuari i el sistema operatiu no haurien de contenir error, i per això no hauria de saltar cap excepció. Per tant, en el cas que hi hagi un error i es produeixi una excepció, simplement s'atura l'execució.

- Tot i que tècnicament la memòria de la VGA està en zona de memòria, es deixarà accedir-hi en mode usuari, per tal de facilitar l'ús de les pantalles per part dels usuaris.

- No hem necessitat implementar cap crida a sistema, ja que tant el nostre joc de proves com la demo comuna es poden implementar sense que l'usuari faci servir instruccions privilegiades (les úniques instruccions privilegiades que s'executen estan localitzades a dintre la RSG/RSIs, per tant ja s'executen en mode sistema).

CARACTERÍSTIQUES I FUNCIONALITATS

El sistema operatiu que hem dissenyat té com a principal objectiu permetre l'execució de dos processos concurrentment. Per això és necessari implementar les estructures i funcionalitats necessàries per garantir la consistència del sistema a l'hora de realitzar el canvi de context.

Així doncs, la implementació consisteix en implementar una RSG, que serà l'encarregada de decidir cada quant realitzar un canvi de context, i implementar les funcions que s'han de realitzar al canvi de context, per exemple: salvar els registres i el PC del procés en execució a memòria, i restaurar el context del procés a restaurar.

• *MEMÒRIA*

La memòria està dividida en diverses zones, on és mapejaran el codi i les dades del SO i dels diferents processos. Les zones mapejades en **posicions inferiors** a la direcció **0x8000** tenen els permisos limitats a **mode usuari**, i les **posicions iguals o superiors** disposen de permisos de **mode sistema** (a excepció de la memòria VGA). A continuació es mostren les diverses zones de memòria:

```
0x0000 - 0x3FFF   -- 1: Procés 1
0x4000 - 0x7FFF   -- 2: Procés 2
0x8000 - 0x8FFF   -- 3: Dades del sistema
0xA000 - 0xBFFF   -- 4: Memòria VGA
0xC000 - 0xCFFF   -- 5: Inicialització del sistema
0xD000 - 0xDFFF   -- 6: RSG i canvi de context
0xF000 - 0xFFFF   -- 7: Pila de sistema
```

1. **0x0000 - 0x3FFF, Procés 1**
   En aquestes 4 pàgines es troba mapejat el codi i les dades del **primer procés** que s'executarà en el sistema operatiu. Aquesta zona té permisos d'usuari.

2. **0x4000 - 0x7FFF, Procés 2**
   En aquestes 4 pàgines es troba mapejat el codi i les dades del **segon procés** que s'executarà en el sistema operatiu. Aquesta zona té permisos d'usuari.

3. **0x8000 - 0x8FFF, Dades del sistema**
   En aquesta pàgina es troben les dades necessàries per a la correcta execució del sistema. Té permisos de sistema.

4. **0xA000 - 0xBFFF, Memòria VGA**
   En aquestes 2 pàgines trobem la zona de memòria de la VGA, és a dir, és la zona de memòria on s'emmagatzemaran les dades que s'enviaran a la pantalla. Aquesta zona té permisos d'usuari, per facilitar a l'usuari l'ús de la pantalla.

5. **0xC000 - 0xCFFF, Inicialització del sistema**
   En aquesta pàgina trobem el codi de la inicialització del sistema . És la posició en la qual comença l'execució del sistema operatiu. Aquesta zona té permisos de sistema.

6. **0xD000 - 0xDFFF, RSG i canvi de context**
   En aquesta pàgina trobem mapejat tot el codi de la rutina d'interrupcions i el codi del canvi de context. Aquesta pàgina té permisos de sistema.
   D'altra banda, a la direcció 0xD500 trobem les dades que utilitza el canvi de context per salvar i restaurar l'estat de cada procés quan es produeix el canvi.

7. **0xF000 - 0xFFFF, Pila de sistema**
   Finalment, en aquesta pàgina, trobem mapejada la pila del sistema, lògicament aquesta pàgina té permisos de sistema.

- ***INICIALITZACIÓ DEL SISTEMA***

El primer pas per al correcte funcionament del sistema operatiu és efectuar la inicialització del sistema. En el nostre cas la rutina d'inicialització conté el següent codi:

```
.text
       $MOVEI R0, 0xD000            -- 1
       wrs S5, R0
       $MOVEI R0, 0x0002            -- 2
```

```
wrs S0, R0
$MOVEI R0, 0x0000          -- 3
wrs S1, R0
reti                       -- 4
```

1. Inicialitzem el registre **S5** amb la direcció de la RSG, **0xD000.**

2. Inicialitzem el registre **S0** amb la direcció la paraula d'estat del processador (PSW). En aquest cas la paraula d'estat té el bit 1 activat, per tal d'activar les interrupcions i el bit 0 desactivat indicant mode usuari.

3. Inicialitzem el registre **S1** amb la direcció del codi del primer procés a executar.

4. Un cop s'executa la instrucció **reti**, el processador salta a **0x0000** i restaura la paraula d'estat salvada a **S0** copiant-la a **S7**. D'aquesta manera el processador passa a executar el codi del primer procés en mode usuari i les interrupcions habilitades.

- ***RSG***

La rutina de servei general (RSG) s'encarrega del tractament de les interrupcions, excepcions i crides a sistema que és puguin produir durant el funcionament del sistema. Està situada a la direcció 0xD000 de memòria. S'explicaran per separat els passos que intervenen en la RSG.

La primera acció a realitzar un cop es passa a executar la RSG és salvar el context actual del procés. Normalment, aquesta funció es realitzaria salvant a la pila els registres que es poden modificar durant l'execució de la RSG, no obstant en el nostre sistema hem optat per salvar directament el context del procés a les dades del canvi de context, d'aquesta manera ens anticipem a la possibilitat de realitzar un canvi de context i evitem haver de salvar el context dos cops, un al començament de la RSG i un altre cop al començar el canvi de context.

No obstant, abans de salvar el context hem de conèixer quin procés s'està executant, per això carregem de memòria la variable user_turn. Si aquesta variable és 0, el procés que s'està executant és el primer procés i per tant es salva l'estat en la zona del primer procés, en cas contrari la variable val 1, indicant que s'està executant el segon procés i de nou es salva l'estat en la seva zona corresponent.

A més, per tal de no sobreescriure cap registre utilitzat pel procés, és necessari fer una còpia del R7 que utilitzarem com a registre auxiliar, per això utilitzem el registre de sistema S4 per realitzar aquesta còpia.

A continuació es pot veure el codi que s'executa per salvar l'estat del procés al començament de la RSG, en aquest cas es suposa que s'està executant el primer procés (el funcionament de les noves macros utilitzades es detalla a l'apartat Macros):

```
RSG_guardar:
      wrsS4, R7
      $LOADI r7, user_turn
      bnz R7, guardar_user2

guardar_user1:
      $PUSHI user1_r0, R0, R7
      ...
      $PUSHI user1_r6, R6, R7
      rds R7, S4
      $PUSHI user1_r7, R7, R6
      rds R7, S1
      $PUSHI user1_pc, R7, R6

$MOVEI R3, RSG_tractar
      jmp R3
```

Bàsicament es salven tots els registres del procés que s'estava executant, i el PC de la instrucció a la qual ha de saltar després de tractament de servei general.

Un cop s'ha salvat l'estat es passa a executar el tractament de la RSG, on es comprova si s'ha de tractar una interrupció, excepció o crida a sistema, i es decideix si és necessari un canvi de context. En cas d'una interrupció s'agafa l'id d'interrupció, i segons l'id es salta a una rutina d'interrupció o una altre, per tractar-la. Si pel contrari era una excepció, és para el processador tal i com deia l'especificació (HALT):

```
RSG_tractar:
      rds    R1, S2
      movi   R2, 15
      cmplt  R3, R1, R2
      bz     R3, __int
      $MOVEI R5, __exc
      jmp    R5

__int:
      getiid R1
      movi R2, 0x0
      cmpeq R2, R2, R1
      bnz R2, RSI__interrup_timer
      movi R2, 0x03
      cmpeq R2, R2, R1
      bnz R2, RSI__interrup_keyboard
```

```
        $MOVEI R6, RSG_restaurar
        jmp    r6

__exc:
        HALT
```

No inserim el codi de rutines d'interrupció ja que ja adjuntem tot el codi amb l'entrega de l'ampliació.

Si finalment no es decideix realitzar el canvi de context, es passa a executar la restauració de l'estat del procés que s'estava executant abans de saltar a la RSG. Així doncs, el procés és molt similar al realitzat per salvar l'estat, de nou es llegeix la variable user_turn i es restaura l'estat corresponent al procés que s'estava executant.

Finalment es realitza un reti per tal de sortir de la RSG i continuar amb l'execució del codi del procés restaurat. A continuació es pot veure el codi executat per restaurar l'estat del procés:

```
rest_user1:
        $LOADI R7, user1_pc
        wrs S1, R7
        $LOADI R7, user1_r7
        $LOADI R6, user1_r6
        $LOADI R5, user1_r5
        $LOADI R4, user1_r4
        $LOADI R3, user1_r3
        $LOADI R2, user1_r2
        $LOADI R1, user1_r1
        $LOADI R0, user1_r0

        reti
```

Bàsicament es restauren tots els registres amb els valors llegits de memòria, i s'escriu el PC que s'executarà al registre S1 perquè un cop es processi el reti es passi a executar la instrucció desitjada.

- ***CANVI DE CONTEXT***

El canvi de context és l'encarregat de gestionar el canvi entre els processos que s'estan executant al processador. En el nostre cas hem simplificat la seva implementació, ja que el nostre sistema només té **2 processos**, per tant la funció bàsicament ha de comprovar quin procés s'està executant, salvar el seu estat a memòria i restaurar l'estat de l'altre procés.

Com la part encarregada de salvar l'estat del procés es realitza al principi de la RSG, el codi que realment s'ha d'executar al canvi de context és actualitzar la variable **user_turn** per indicar que es passa a executar l'altre procés i finalment restaurar l'estat de l'altre procés. A continuació es pot veure el codi que s'executa en el canvi de context. En aquest exemple suposem que la variable **user_turn** val **0**, és a dir, s'està executant el **primer procés** i per tant es para a executar **curr_user1** per tal de restaurar el **segon procés**:

```
contextSwitch:              -- 1
      wrs S4, R7
      $LOADI r7, user_turn
      bnz R7, curr_user2

curr_user1:                 -- 2
      movi R6, 0x01         -- 3
      $MOVEI R7, user_turn
      st 0(R7), R6

      $LOADI R7, user2_pc   -- 4
      wrs S1, R7
      $LOADI R7, user2_r7
      $LOADI R6, user2_r6
      $LOADI R5, user2_r5
      $LOADI R4, user2_r4
      $LOADI R3, user2_r3
      $LOADI R2, user2_r2
      $LOADI R1, user2_r1
      $LOADI R0, user2_r0

      reti                  -- 5
```

1. Per tal de seleccionar quin dels processos s'ha de restaurar, el primer que cal fer és carregar de memòria la variable **user_turn** i seleccionar quin dels processos es restaurarà a continuació.

2. En aquest cas, el procés que s'està executant és el primer procés i per tant es passa a executar **curr_user1** per tal de realitzar el canvi de context cap al segon procés.

3. Un cop es coneix quin procés s'ha de restaurar, s'actualitza la variable **user_turn** per tal d'indicar quin procés s'executarà a continuació. En aquest cas es salva a **user_turn** un **1** per tal d'indicar que es passa a executar el **segon procés**.

4. Un cop s'ha actualitzat la variable **user_turn** es procedeix a restaurar l'estat del procés corresponent, en aquest cas es restaura l'estat del **segon procés.**

Concretament es restaura tots els **registres** i el **PC** que s'ha d'executar un cop es processi el **reti**.

5. Finalment, s'executa la instrucció **reti** i es passa a executar el codi del procés restaurat. D'aquesta manera queda completat el canvi de context.

- ***MACROS***

Amb l'objectiu de facilitar el disseny del sistema operatiu s'han implementat un parell de macros en el fitxer “macros.s” que es detallen a continuació:

```
PUSHI:
	.macro $pushi imm16 p1 p2
		$movei \p2 \imm16
		st 0(\p2), \p1
	.endm
```

Aquesta macro serveix per emmagatzemar en la posició de memòria indicada a **imm16**, el contingut del registre **p1**. Per a poder fer-ho, però, es necessita un registre auxiliar que s’indica amb **p2**, en el qual es carrega la direcció codificada en l'immediat i a continuació es realitza un store en la direcció indicada per **p2**, emmagatzemant el contingut de **p1**.

```
LOADI:
	.macro $loadi p1 imm16
	 $movei \p1 \imm16
	 ld \p1, 0(\p1)
	 .endm
```

Aquesta macro serveix per llegir el contingut de la posició de memòria indicada a **imm16** i carregar-ho al registre indicat a **p1**. Per a això, s'utilitza el registre **p1** per carregar la direcció codificada a l'immediat, i posteriorment s'efectua un load de la direcció contenia en **p1** i s'emmagatzema en el mateix registre.

CONTROLADOR D'AUDIO

Crear un controlador d'audio des de zero no hauria sigut una tasca viable amb les poques setmanes de temps que teníem, així que vam optar per buscar un controlador d'audio ja fet i adaptar-lo a les nostres necessitats.

CERCA DEL CONTROLADOR D'AUDIO

Per a dur a terme aquesta ampliació vam decidir buscar un controlador d'audio disponible a Internet. Les primeres webs on esperàvem trobar algun projecte de controlador d'audio per a Quartus II i per a la nostra placa DE1 són les següents:

- Opencores: http://opencores.org/
- Terasic : Altera DE1 Board: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=83&PartNo=4

A l'apartat de la web de Terasic corresponent a la nostra placa vam trobar dos projectes: un sintetitzador i un lector de targetes SD i reproductor de música. Cap d'aquests dos projectes era poc complexa com el que finalment vam trobar en aquest enllaç d'un grup de recerca en enginyeria informàtica de la Universitat de Toronto:

http://www.eecg.toronto.edu/~jayar/ece241_08F/AudioVideoCores/audio/audio.html

ADAPTACIÓ DEL CONTROLADOR D'AUDIO

Abans de tot, vam intentar traduir el codi del controlador a VHDL, ja que estava en Verilog. No va funcionar ja que en Verilog hi havia uns mòduls anomenats megafunctions que al traduir el codi no existien en VHDL.

En comptes de fer les megafunctions nosaltres des de zero, vam optar per inserir el controlador directament al nostre projecte, sense traduir, ja que Verilog i VHDL poden existir en un projecte amb Quartus.

Un cop teníem el controlador dins el projecte, vam pensar que no calia tractar el que venia per entrada d'audio, per tant podíem adaptar el controlador de tal manera que generés una melodia que nosaltres establiríem. El principal inconvenient va ser que tots els mòduls associats al controlador estaven escrits en Verilog, llenguatge amb el que no havíem tractat fins el moment.

Un cop vam saber que un projecte de Quartus pot barrejar mòduls escrits en VHDL i Verilog i compilar-los alhora sense cap mena de problema, vam decidir provar si era vàlid instanciar, dins d'un fitxer VHDL, un component l'entitat associada al qual estigués escrita en Verilog. Efectivament era possible, i la propera decisió que vam prendre va ser instanciar un fitxer de prova que venia adjunt al projecte descarregat i que s'encarregava

d'instanciar el controlador d'audio del projecte i transmetre-li dades per a generar un so senzill.

A continuació vam veure que el fitxer de proves que volíem instanciar contenia errors de sintaxi, un d'ells provocat per una variable de tipus reg a la qual s'intentava fer un 'assign' (pel que sembla, Verilog distingeix els tipus de dades 'wire' i 'reg' com en VHDL hi ha les 'signal' i les 'variable'). Com que per la lògica combinacional s'utilitzen 'wire' i per a la lògica seqüencial s'utilitzen 'reg', vam haver de canviar els tipus de variable per a que el codi fos correcte.

Per últim vam agafar el projecte de processador escollit dins del grup per a fer-hi l'ampliació, vam instanciar el fitxer de proves de controlador d'audio al respectiu mòdul 'sisa', vam compilar i tot semblava correcte. El següent pas va ser generar un simple so i, un cop funcionés, programar una melodia per defecte. Aquest pas s'explica en detall en la secció de generació de notes musicals.

Com a últim pas, havíem pensat que fos possible carregar dades en memòria (és a dir, melodies) per a fer automàtic el procés de generació i reproducció d'audio, no obstant si ho haguéssim intentat hauríem d'haver segmentat l'accés a memòria, ja que no podem controlar quan s'estan fent loads normals i quan s'estan fent loads de musica. Segmentar els accessos no hagués sigut trivial així que vam optar per tenir dos vectors, un amb les notes de la melodia i l'altre amb la seqüencia a seguir.

FUNCIONAMENT DEL CONTROLADOR D'AUDIO

En la següent figura es mostra un esquema del mòdul o interfície del controlador d'audio global:

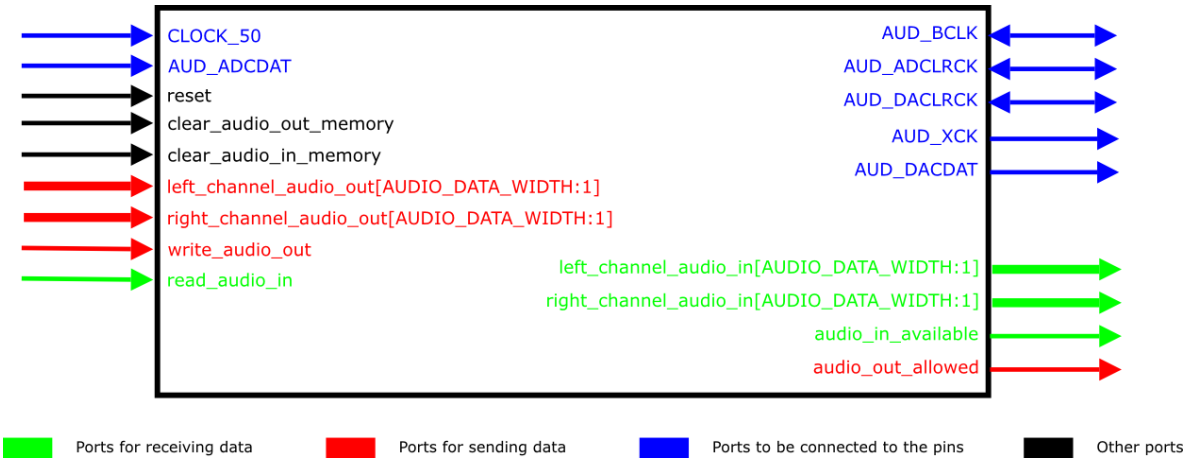


Figura 1: Interfície del controlador

A continuació expliquem totes les entrades/sortides del mòdul:

- `CLOCK_50` – Input clock del sistema, ha de ser de 50MHz per a que el control del temps funcioni adequadament.
- `reset` – Reset.
- `AUD_ADCDAT, AUD_DACDAT, AUD_BCLK, AUD_ADCLRCK, AUD_DACLRCK, I2C_SDAT, I2C_SCLK i AUD_XCK` – Ports a connectar amb els pins del chip d'audio de la placa.
- `clear_audio_in_memory` – Per a netejar el buffer d'entrada d'audio.
- `clear_audio_out_memory` – Per a netejar el buffer de sortida d'audio.

Ports de rebuda de dades:

- `left_channel_audio_in i right_channel_audio_in` – Dades d'audio rebudes per fonts externes.
- `read_audio_in` – Senyal d'enable per lectures. Les dades d'entrada, si estan disponibles, seran presents al següent cicle de clock.
- `audio_in_available` – Indica si les dades d'entrada estan disponibles. Si no esta a "1", les lectures no tindran efecte.

Ports per enviar dades:

- `left_channel_audio_out i right_channel_audio_out` – Dades d'audio a reproduir.
- `write_audio_out` – Senyal d'enable per escriure dades.
- `audio_out_allowed` – Indica quan es poden escriure dades. Escriptures amb audio_out_allowed = 0 no tindran efecte.

A través dels ports explicats, el controlador d'audio es capaç de transmetre dades a full-duplex, input i output. Els ports son de 32 bits i, per defecte, connectats als buffers de dades. Les senyals `clear_audio_in_memory` i `clear_audio_out_memory` poden ésser utilitzades per netejar els buffers i les `senyals audio_in_available` i `audio_out_allowed` indiquen la disponibilitat de les dades (en cas d'input) o l'espai lliure (en cas d'output). En els buffers. Les dades són enters amb signe que representen mostres d'audio. Totes les senyals estan sincronitzades amb el mateix clock.

Com que només fem servir la part de reproducció d'audio, i no input d'aquest a través de micròfons, ens centrarem en explicar els protocols per enviar i convertir dades. En la següent figura mostrem el protocol per enviar dades:

Figura 2: Protocol d'enviament de dades

El controlador fa servir els canals de dades PCM per entrada i sortida, que són essencialment una seqüencia de números que representen la intensitat del senyal en un moment donat. Cada un d'aquests números representa una mostra. El so pot ser

representat per una seqüencia de mostres. Aquesta seqüencia te una freqüència associada, que representa la freqüència a la que es mostrejava la senyal original. Aquesta freqüència es imprescindible per la correcta reconstrucció del senyal.

Pel controlador la freqüència per defecte és de 48kHz amb una mida per defecte de 32 bits. És més, hi ha 2 canals, un d'entrada i un de sortida. La freqüència de mostreig i la mida de les mostres es poden canviar en temps de configuració si fa falta. Les mostres es representen en complement a 2 com a enter amb signe.

Per a reproduir audio, les dades de PCM s'envien directament als Conversors Digital-Analògic, que converteixen el valor al voltatge. Aquest voltatge de sortida analògic es connecta al jack de línia de sortida a la placa DE1, que a la vegada es connecta a auriculars o altaveus. El procés s'il·lustra en la següent figura:

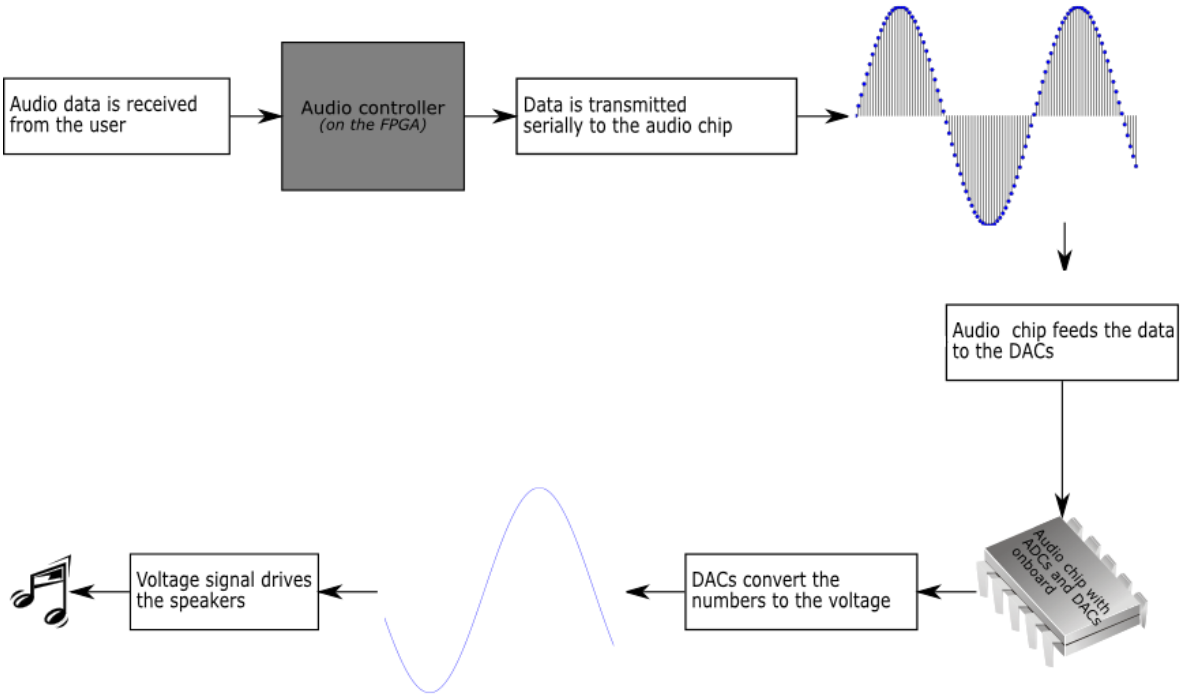


Figura 3: Conversió d'audio

GENERACIÓ DE NOTES MUSICALS

En el fitxer de prova es generava un so que en principi era casi imperceptible. Vam sospitar que aquest so no es generava correctament, per tant vam indagar dins del codi per descobrir com generar un so.

Aquest fragment de codi en concret s'encarregava de generar una ona amb una determinada amplitud i freqüència, magnituds que ens permetrien generar una nota concreta i a un determinat volum. L'error d'aquest codi es trobava en la línia en què es fa un assign a 'delay', ja que delay no tenia cap valor i per tant la igualtat del bloc 'always' es complia sempre, fet que provocava que es generés un so imperceptible. Això es pot veure en la següent imatge:

```
/*****************************************************************************
 *                             Sequential Logic                              *
 *****************************************************************************/

always @(posedge CLOCK_50)
    if(delay_cnt == delay) begin
        delay_cnt <= 0;
        snd <= !snd;
    end else delay_cnt <= delay_cnt + 1;

/*****************************************************************************
 *                             Combinational Logic                           *
 *****************************************************************************/

assign delay = {SW[3:0], 15'd3000};
```

Figura 4: Part del codi encarregada de generar ones de so

En el moment en què vam establir un valor numèric que substituïa a 'delay', es va generar la nota B4 (nota SI). Per descomptat, es va eliminar la línia de l'assign a 'delay'.

Arribats a l'última fase, havíem de decidir quina melodia generar i de quina manera representar-la. Hem triat com a motiu musical les primeres notes (primers 4 compassos o *introducció*) de l'Himne del Barça, mostrat en la següent imatge:

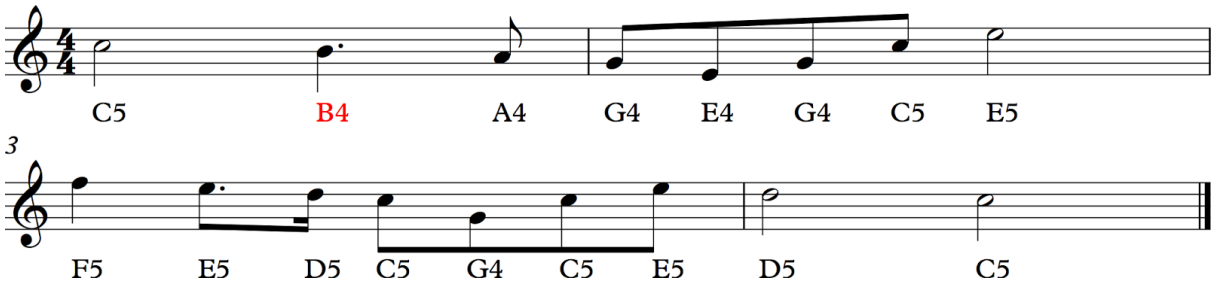


Figura 5: Part de la melodia de l'Himne del Barça

Per determinar la *afinació* (freqüència) de les diferents notes necessàries per reproduir la nostra melodia, partim de la nota B4, que té el valor 50000 en el codi d'exemple. Després de realitzar diverses proves, determinem que la distància d'un *to*, per ex. entre les notes A4 (LA) i B4 (SI), era aproximadament 6500. Encara que després també comprovem que aquesta distància variava gradualment segons ens allunyàvem cap amunt o cap avall. A més descobrim que aquests valors, al contrari que succeeix amb els valors reals de freqüència, s'han de modificar en sentit invers a l'*altura* o *afinació* de les notes de l'*escala musical* per aconseguir la nota desitjada.

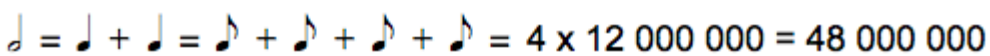
Per reproduir un *silenci* (o *pausa*), hem aplicat el petit truc de reproduir una nota de freqüència extremadament alta (valor 1), és a dir inaudible per a nosaltres.

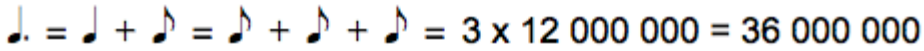
Respecte a la *durada* de les notes, la solució és bastant més senzilla doncs en aquest cas els diferents valors són proporcionals i matemàticament exactes. Llavors prenem com a referència el valor de la *corxera*, que és la *figura musical* (signe que representa gràficament la durada musical d'un determinat so en una peça musical) més usat en el

nostre fragment musical, i que a més determinarà el *tempo* (velocitat de reproducció) de la melodia.

Així doncs hem assignat el valor 12000000 a la *corxera*, amb la qual cosa la resta de valors de figures rítmiques queden establerts com segueix:

Algunes equivalències entre les figures:





Finalment, respecte a la intensitat o volum, s'ha establert un valor constant de 32'd200000000. Per tant, per reproduir la nostra melodia necessitem els vectors *notes[19:0]* i *temps[19:0]*, que contindran respectivament la afinació i la durada de cada nota.

ÚS DEL CONTROLADOR

Per tal de fer el controlador més "user-friendly", ja que la música pot arribar a cansar, hem mapejat el switch(0) de la placa DE1 com el botó "on-off" d'un reproductor de música. Qualsevol usuari doncs, pot encendre i/o apagar la música i resetejar-la quan vulgui.

Per a canviar la música de la que es fa play, s'hauria de generar una nova melodia simplement com s'indica en l'apartat anterior, creant dos vectors, un de notes musicals i un altre per la seqüencia de reproducció d'aquestes notes.

Havíem pensat estendre el repertori d'instruccions per a fer possible coses com regulació del volum de la música, crides a sistema per a pausar o començar la reproducció de música, per tal de poder-ho fer a nivell de codi d'usuari i per últim també havíem pensat el que s'ha comentat anteriorment, és a dir, poder reproduir música a partir d'una posició de memòria, per així poder carregar melodies des de codi d'usuari i no haver de modificar part del controlador.

No obstant, per limitacions temporals i d'altres, com per exemple no trobar manera d'accedir a memòria des del controlador si no segmentem l'accés, no vam obtenir aquestes funcionalitats, però si la més bàsica i l'objectiu al que volíem arribar: Reproduir sons a la placa.

ANNEX: JOC DE PROVES

Per a qualsevol joc de proves cal carregar a les posicions següents els arxius del sistema operatiu i a les direccions indicades en cada joc de prova els fitxers corresponents a aquest joc (entre parèntesis es troba la direcció del DE1-Control-Panel):

0xC000(6000) : init.code.DE1.hex
0xD000(6800) : context_switch.code.DE1.hex
0xD500(6A80) : context_switch.data.DE1.hex

JOC DE PROVES DISPLAY I LEDS

Aquest joc de proves consisteix en l'execució de dos processos, el primer d'aquests executa un sumador infinit i mostra els resultats en els 4 displays 7 segments, mentre que el segon procés també executa un sumatori infinit però mostra el resultat mitjançant els 8 led de color verd. D'aquesta manera, s'intenta veure si tant els display com els leds mostren els valors dels comptador simultàniament, en cas afirmatiu el canvi de context està funcionant correctament, ja que sinó, una, o les dues formes de visualització es pararien.

Cal carregar a les posicions descrites a continuació els arxius indicats (entre parèntesis es troba la direcció del DE1-Control-Panel):

0x0000(0000) : test1-user1-simple-led.code.DE1.hex
0x4000(2000) : test1-user2-simple-led.code.DE1.hex

JOC DE PROVES FIBONACCI I CORRE LLETRES

Cal carregar a les posicions descrites a continuació els arxius indicats (entre parèntesis es troba la direcció del DE1-Control-Panel):

0x0000(0000) : fibonacci.code.DE1.hex
0x2000(1000) : fibonacci.data.DE1.hex
0x4000(2000) : corre_letras.code.DE1.hex
0x6000(3000) : corre_letras.data.DE1.hex