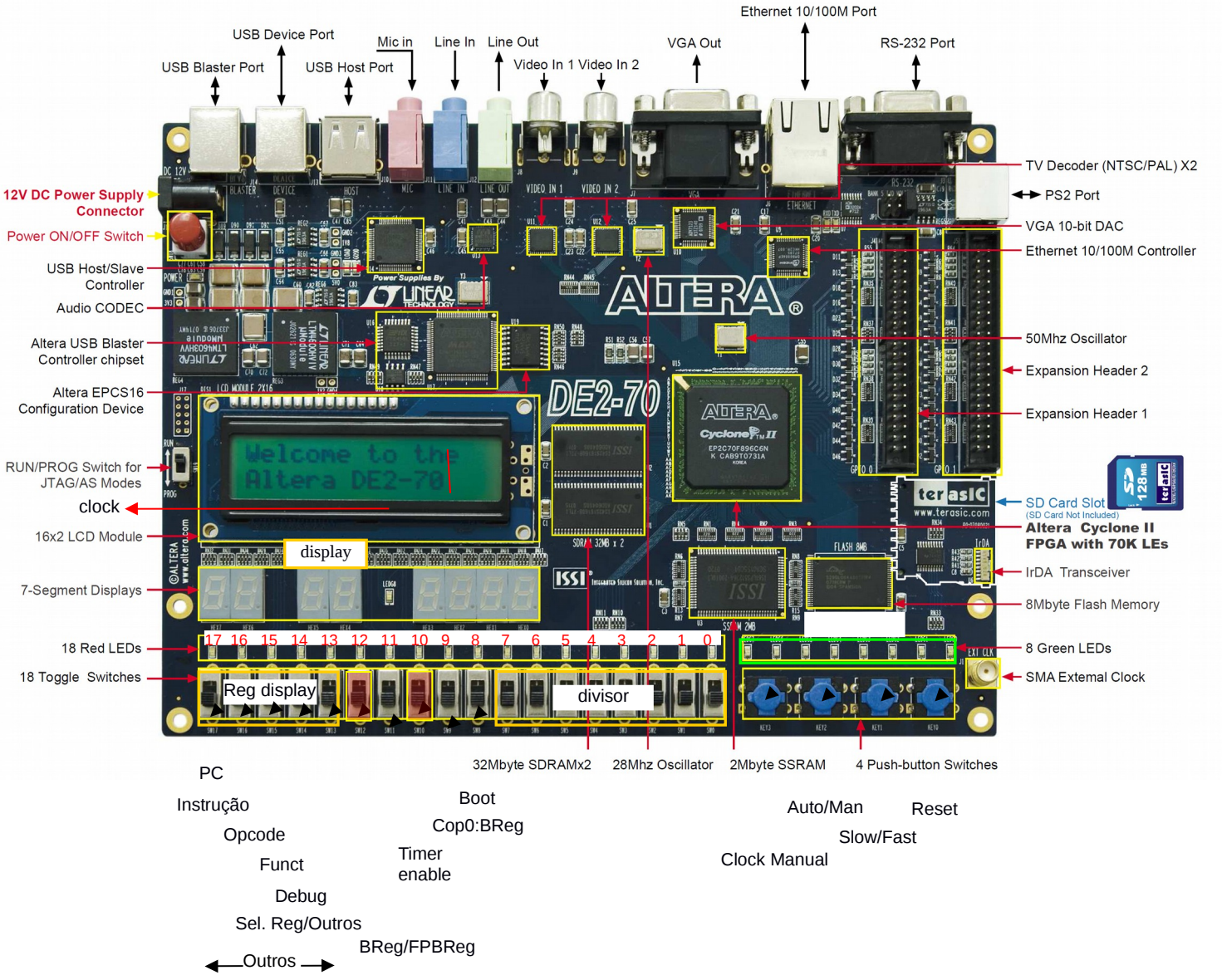
**Universidade de Brasília**
Departamento de Ciência da Computação

# MAPA DE UTILIZAÇÃO DA DE2-70
## Processador MIPS PUM Versão 4.5




Display : SW[12] ? Outros (PC, Instrução, Opcode, Funct, Debug, Flags) :
SW[11] ? FPU Reg[SW[17:13]] :
SW[9] ? RegCop0[SW[17:13]] : Reg[SW[17:13]]

Timer: SW[10] ? 10 segundos : Free running

Boot: SW[8] ? Boot pelo BootLoader : Boot pelo .text

Reset: pressionado durante borda de subida do clock

Visualização das Flags de condições da FPU nos displays: SW[12] = 1 e SW[17:13] = 0

Frequência: 50 MHz / SW[7:0]

Modo Debug: SW[12]=1, visualização do PC e do IR no display LCD e do Banco de Registradores no display VGA.
Banco de Registradores da CPU SW[11]=0 ou Banco de Registradores da FPU SW[11]=1

## Monitoramento dos Sinais de Controle:

| MULTICICLO | |
|---|---|
| **LED** | **Sinal** |
| LEDR[17] | Não usado |
| LEDR[16] | LeMem |
| LEDR[15] | EscreveMem |
| LEDR[14] | EscreveReg |
| LEDR[13] | RegDst |
| LEDR[12:11] | ALUOp |
| LEDR[10:9] | OrigAALU |
| LEDR[8:6] | OrigBALU |
| LEDR[5] | IouD |
| LEDR[4] | EscreveIR |
| LEDR[3] | EscrevePC |
| LEDR[2:0] | OrigPC |
| LEDG[8] | Clock |
| LEDG [7] | Manual/Automático |
| LEDG[6] | Slow/Fast |
| LEDG[5:0] | Estado |

| PIPELINE | |
|---|---|
| **LED** | **Sinal** |
| LEDR[17:15] | Não usado |
| LEDR[14] | ID/LeMem |
| LEDR[13] | ID/EscreveMem |
| LEDR[12] | ID/EscreveReg |
| LEDR[11:10] | ID/ALUOp |
| LEDR[9:7] | ID/OrigALU |
| LEDR[6:5] | ID/RegDst |
| LEDR[4:3] | ID/Mem2Reg |
| LEDR[2:0] | ID/OrigPC |
| LEDG[8] | Clock |
| LEDG [7] | Manual/Automático |
| LEDG[6] | Slow/Fast |
| LEDG[5:0] | Não usado |

### Simulação por forma de onda (VWF): TopDE2.vwf

IMPORTANTE: Para poder usar a pilha ($sp) na simulação por forma de onda é necessário alterar o endereço inicial do $sp nas linhas 177 e 178 do arquivo Parametros.v

### Analisador Lógico (SignalTap-II): stp1.stp

IMPORTANTE: Caso necessite de memória de código, pode-se retirar esta funcionalidade através do menu Assignments/Settings/SignalTap II Logic Analyzer/ Enable

- **Mapeamento da Memória:**

Memória de Instruções (CodeMemory)

| Endereço | Tamanho | Uso |
|---|---|---|
| 0x0040 0000 | 8 kibytes | .text UserCodeBlock |
| ... | | |
| 0x0040 1FFF | | |
| 0x8000 0000 | 8 kibytes | .ktext SysCodeBlock |
| ... | | |
| 0x8000 1FFF | | |
| ... | | Não Alocado |

Memória de Dados (DataMemory)

| Endereço | Tamanho | Uso |
|---|---|---|
| 0x0000 0000 | 128 bytes | Boot Loader |
| .... | | |
| 0x0000 007F | | |
| ... | | Não Alocado |
| 0x1001 0000 | 8 kibytes | .data UserDataBlock |
| ... | | |
| 0x1001 1FFF | | |
| 0x1001 2000 | 2 Mibytes | SRAM |
| ... | | |
| 0x1021 1FFF | | |
| ... | | Não Alocado |
| 0x9000 0000 | 2 kibytes | .kdata SysDataBlock |
| ... | | |
| 0x9000 07FF | | |

Valores definidos na inicialização do Processador:
Endereço da Base da Pilha: $sp = 0x1021 1FFC
Endereço do Contador de Programa: PC = 0x0040 0000
Endereço de Boot: PC=0x0000 0000

Memória de Programa do Usuário: user_code.mif
Memória de Dados do Usuário: user_data.mif
Memória de Programa do Sistema: system_code.mif
Memória de Dados do Sistema: system_data.mif

Os arquivos .mif devem ser gerados pela ferramenta MIF Exporter do Mars. Ajustar os tamanhos dos blocos para os valores usados.

O arquivo SYSTEMv53.s contém a rotina de tratamento de exceções/interrupções e syscalls.

O arquivo BootLoader.s contém a rotina do boot loader.

Deve-se configurar o Mars para:
-Settings/Self-Modifying Code
-Settings/Exception Handler/escolher o arquivo SYSTEMv53.s

E/S Mapeada em Memória (MMIO)

| Endereço | Tamanho | Uso |
|---|---|---|
| 0xFF00 0000 | 76800 bytes | Memória de vídeo da VGA |
| ... | | |
| 0xFF01 2C00 | | |
| ... | | Não Alocado |
| 0xFFFF 0000 | 4 bytes | Áudio inL |
| 0xFFFF 0004 | 4 bytes | Áudio inR |
| 0xFFFF 0008 | 4 bytes | Áudio outL |
| 0xFFFF 00C | 4 bytes | Áudio outR |
| 0xFFFF 0010 | 4 bytes | Áudio Ctrl1 |
| 0xFFFF 0014 | 4 bytes | Áudio Ctrl2 |
| ... | | Não Alocado |
| 0xFFFF 0100 | 4 bytes | Buffer0 Teclado |
| 0xFFFF 0104 | 4 bytes | Buffer1Teclado |
| ... | | Não Alocado |
| 0xFFFF 0110 | 4 bytes | Teclado x Mouse |
| 0xFFFF 0114 | 4 bytes | Buffer Mouse |
| ... | | Não Alocado |
| 0xFFFF 0120 | 1 byte | Rx RS232 |
| 0xFFFF 0121 | 1 byte | Tx RS232 |
| 0xFFFF 0122 | 1 byte | Rx/Tx Crtl |
| ... | | Não Alocado |
| 0xFFFF 0130 | 1 byte | LCD[1,1] |
| 0xFFFF 0131 | 1 byte | LCD[1,2] |
| ... | ... | ... |
| 0xFFFF 013F | 1 byte | LCD[1,16] |
| 0xFFFF 0140 | 1 byte | LCD[2,1] |
| 0xFFFF 0141 | 1 byte | LCD[2,2] |
| ... | ... | ... |
| 0xFFFF 014F | 1 byte | LCD[2,16] |
| 0xFFFF 0150 | 1 byte | LCD Clear |
| ... | | Não Alocado |
| 0xFFFF0200 | 4 bytes | NOTE_SYSCALL_ADDRESS |
| 0xFFFF0204 | 4 bytes | NOTE_CLOCK |
| 0xFFFF0208 | 4 bytes | NOTE_MELODY |
| 0xFFFF020C | 4 bytes | MUSIC_TEMPO_ADDRESS |
| 0xFFFF0210 | 4 bytes | MUSIC_ADDRESS |
| 0xFFFF0214 | 4 bytes | PAUSE_ADDRESS |
| ... | ... | Não Alocado |
| 0xFFFF0250 | 4 bytes | SD_INTERFACE_ADDR |
| 0xFFFF0254 | 1 byte | SD_INTERFACE_CTRL |
| 0xFFFF0255 | 1 byte | SD_INTERFACE_DATA |
| ... | ... | Não Alocado |
| 0xFFFF0260 | 4 bytes | IrDA_CTRL |
| 0xFFFF0264 | 4 bytes | IrDA_RX |
| 0xFFFF0268 | 4 bytes | IrDA_TX |

- **Chamadas do Sistema:**

No endereço .ktext encontra-se a rotina de tratamento de exceções, e chamadas do sistema da instrução syscall:

Syscalls

| Serviço | $v0 | Argumentos | Resultados |
|---|---|---|---|
| print integer | 1<br>101 | $a0=inteiro<br>$a1=coluna<br>$a2=linha<br>$a3=cores | Imprime o número inteiro complemento de 2 $a0 na posição ($a1,$a2) com as cores $a3={0...0BBGGGRRRbbgggrrr} BGR fundo; bgr frente |
| print float | 2<br>102 | $f12=float<br>$a1=coluna<br>$a2=linha<br>$a3=cores | Imprime o número float em $f12 na posição ($a1,$a2) com as cores $a3 |
| print string | 4<br>104 | $a0=endereço string<br>$a1=coluna<br>$a2=linha<br>$a3=cores | Imprime a string terminada em NULL (ASCIIZ) presente no endereço $a0 na posição ($a1,$a2) com as cores $a3 |
| read int | 5 | | Retorna em $v0 o valor inteiro com sinal lido do teclado. |
| Read float | 6 | | Retorna em $f0 o valor float lido do teclado. |
| Read string | 8 | $a0 endereço do buffer de entrada<br>$a1 número de caracteres máximo | Retorna no endereço $a0 o conjunto de caracteres lidos, terminando com /0. |
| Print char | 11<br>111 | $a0=char (ASCII)<br>$a1=coluna<br>$a2=linha<br>$a3=cores | Imprime o caractere $a0 (ASCII) na posição ($a1,$a2) com as cores $a3 |
| exit | 10 | | |
| read char | 12 | | Retorna em $v0 código ASCII do caractere lido |
| time | 30 | | Retorna o tempo do sistema (número de ciclos de clock)<br>$a0 = parte menos significativa<br>$a1 = parte mais significativa |
| MIDI Out | 31 | $a0 = pitch<br>$a1 = duração ms<br>$a2 = instrumento (1)<br>$a3 = volume | Gera o som definido e retorna imediatamente |
| sleep | 32 | $a0=tempo(ms) | Aguarda $a0- milissegundos |
| MIDI Out sincrono | 33 | $a0 = pitch<br>$a1 = duração ms<br>$a2 = instrumento (1)<br>$a3 = volume | Gera o som definido e retorna após o término |
| rand | 41 | | $a0 = número randômico de 32 bits |
| plot | 45 | $a0=X<br>$a1=Y<br>$a2=cor | Plota um pixel na posição (X,Y)=($a0,$a1) com a cor $a2<br>Obs.: Não é eficiente! |
| getplot | 46 | $a0=X<br>$a1=Y | Retorna em $a2 a cor do pixel na posição (X,Y)=($a0,$a1) |
| Inkey | 47 | | Retorna em $v0 e $v1 os códigos ASCII das duas teclas pressionadas simultaneamente, retorna $v0=0 e/ou $v1=0 se alguma tecla não estiver pressionada |
| clear screen | 48 | $a0=cor | Preenche a tela com a cor $a0 |
| SD Card Read | 49 | $a0=Endereço Origem<br>$a1=Endereço Destino<br>$a2=Quantidade Bytes | $v0=0 transferência bem sucedida<br>$v0=1 falha na transferência |

Os syscalls 101, 102, 104 e 111 são para visualização da tela VGA no Mars (Tool/Bitmap Display).

- **Interface Serial RS232:**

Comunicação com o PC é feita pela transmissão/recepção de um byte.
Taxa de comunicação: 115kbps
Ajustar a porta COM0 (ou outra) do PC para este valor de taxa (Gerenciador de Dispositivos / Portas (COM e LPT))

Procedimentos:

Byte de Controle

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| x | x | x | x | X | x | Ready | Busy | Start |

Transmissão: MIPS  PC

I- MIPS escreve o byte a ser enviado no endereço TX
II- MIPS ativa o bit Start do controle no endereço CTRL
III- MIPS desativa o bit Start do controle no endereço CTRL
IV- MIPS aguarda o bit Busy do controle ser desativado para fazer nova transmissão

Recepção: MIPS  PC

I- MIPS aguarda o bit Ready do controle ser ativado
II- MIPS lê o byte recebido do endereço RX
III- MIPS aguarda o bit Ready do controle ser desativado

Ver exemplo: testeRS232.s e testeRS232.c, demo_tx.c, demo_rx.c
Detalhes em http://www.fpga4fun.com/SerialInterface.html

- **Interface Teclado PS2**

Nos endereços Buffer0 e Buffer1 está o buffer do teclado.
O byte menos significativo de Buffer0 contém o código mais recente enviado pelo teclado.
Memoria[0xFFFF0110]=1 habilita detecção de interrupções do teclado
Memoria[0xFFFF0110]=0 habilita a detecção de interrupções do mouse

| Buffer1 | | | | | Buffer0 | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | | 4 | 3 | 2 | 1 |

Ver exemplo: testePS2.s

Obs.: Alterado em 2012/2 para uso de mapa do teclado...

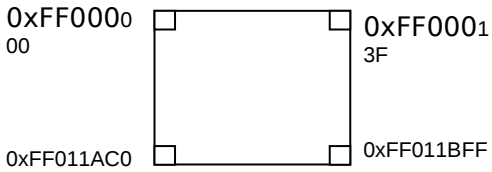
- **Interface VGA:**

O arquivo display.mif é carregado na memória VGA ao inicializar o processador.
Mapeamento da tela VGA para resolução 240 linhas(Y) x 320 colunas(X) pixels:

Mapeamento: (X,Y) => Endereço Base (0xFF00 0000) + Y * 320 + X
Com X de 0 a 319 e Y de 0 a 239.

Cor do Pixel: 8 bits

| 7-6 | 5-3 | 2-0 |
|---|---|---|
| BB | GGG | RRR |




Obs.: Transparência
Ver exemplo: testeVGA.s
Programas png2mif e bmp2mif disponíveis no moodle

- **Interface de áudio CODEC:**

Amostras de 16 bits de áudio estéreo lidas e escritas nos endereços indicados na tabela de mapa de memória.
Frequência de Amostragem 44.1kHz

Sincronismo do Processador com o CODEC:

I - MIPS coloca o Ctrl2 em 0, indicando ao CODEC para enviar uma amostra;
II – MIPS aguarda CODEC colocar Ctrl1 em 1, que indica que uma amostra está pronta;
III – MIPS lê o valor IN e escreve o valor OUT;
IV– MIPS coloca o Ctrl2 em 1, indicando ao CODEC que terminou de ler e escrever a amostra;
V - MIPS aguarda CODEC colocar Ctrl1 em 0, que indica que o CODEC está pronto novamente
VI – Reinicio do ciclo

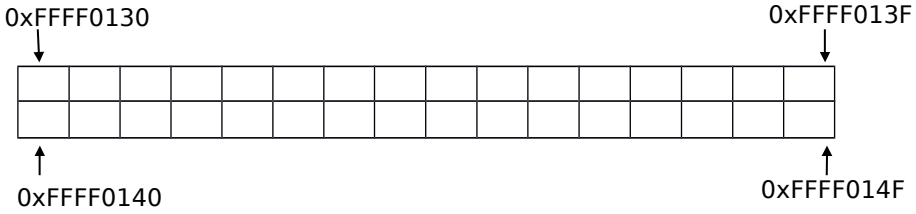
Onde o bit [0], nas palavras de controle Ctrl1 e Ctrl2, corresponde ao canal R e o bit [1] ao canal L . Com isto é, é possível o controle, aquisição e síntese de áudio para os 2 canais de forma independentes.

Ver exemplo: testeAUDIO.s

- **Interface Display LCD:**

Display LCD é composto de 2 linhas de 16 caracteres, programáveis através dos endereços indicados no mapa de memória.

Os caracteres são usados conforme a tabela ASCII (não estendida) ao lado.




Escrevendo-se qualquer valor no endereço LCD Clear (0xFFFF0150) apaga-se o display.

Ver exemplo: testeLCD.s

| | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|
| **0** | | 0 | @ | P | ` | p |
| **1** | ! | 1 | A | Q | a | p |
| **2** | “ | 2 | B | R | b | r |
| **3** | # | 3 | C | S | c | s |
| **4** | $ | 4 | D | T | d | t |
| **5** | % | 5 | E | U | e | u |
| **6** | & | 6 | F | V | f | v |
| **7** | ´ | 7 | G | W | g | w |
| **8** | ( | 8 | H | X | h | x |
| **9** | ) | 9 | I | Y | i | y |
| **A** | * | : | J | Z | j | z |
| **B** | + | ; | K | [ | k | { |
| **C** | , | < | L | ¥ | l | \| |
| **D** | - | = | M | ] | m | } |
| **E** | . | > | N | ^ | n | → |
| **F** | / | ? | O | _ | o | ← |

- **Interface Mouse PS2: (by Hugo Silva)**
  *Mouse não está Funcionando, foi desativado (por enquanto).*

  Na prática, para usar o mouse com lançamento de exceções basta seguir os seguintes passos:

  1) A word no endereço 0x66666666 (em bytes) deve valer zero
  2) A seção kdata do arquivo testeSYSCALL.s deve ser copiada para o programa em assembly a ser executado
  3) Sempre que houver atualização das coordenadas do mouse, será iniciada uma rotina que, em função dos valores contidos no buffer de mouse (que pode ser acessado diretamente, conforme será detalhado após a seção atual), calcula as coordenadas do pixel representativo do mouse (caso haja necessidade de se desenhar um ponteiro, o programador deve ser responsável por fazer isso a partir das coordenadas desse pixel). Essas coordenadas ficam gravadas nas variáveis DATA_X, DATA_Y e DATA_CLICKS, que podem ser carregadas como se fossem labels no programa. As variáveis devem ser interpretadas pelo programador da seguinte forma:
     - DATA_X: a posição do cursor no eixo X (da esquerda para a direita)
     - DATA_Y: a posição do cursor no eixo y (de cima para baixo)
     - DATA_CLICKS: quando há um click com o botão esquerdo, os 4 bits da direita são 1. Se o click for com o botão direito, os próximos 4 são 1. Se for com o do meio, os bits 11 a 8 ficam 1. Segue um desenho desse funcionamento:

| Não usados | Lower Byte 1: vira F se houver clique com botão do meio | Upper Byte 0: vira F se houver clique com botão direito | Lower Byte 0: vira F se houver clique com botão esquerdo |
|---|---|---|---|

*Funcionamento do DATA_CLICKS*

  4) A partir daí, o programador pode carregar a posição (X,Y) do cursor, carregar os clicks, inibir a alteração das coordenadas do mouse temporariamente (colocando na word da posição 0x66666666 o valor 1 e, quando desejar reativar as alterações, colocando a word em 0 novamente). Para desenhar um cursor, os passos sugeridos são:
     - Definir posições de todos os pixels que farão parte do ponteiro em relação ao pixel informado (normalmente na forma de offsets a serem somados às coordenadas desse pixel)
     - Definir as cores de cada pixel desses
     - Criar um array no .data para receber as cores da superfície que será sobreposta pelo mouse (para que seja possível restaurá-la)
     - Dar início ao seguinte algoritmo:

```
SALVA AS CORES DA SUPERFÍCIE NO ARRAY DECLARADO
DESENHA CURSOR POR CIMA DA SUPERFÍCIE
ENQUANTO TRUE
        SE HOUVER NECESSIDADE DE MOVER O CURSOR
                DESENHA SUPERFÍCIE SALVA NO ARRAY POR CIMA DO CURSOR
                SALVA A NOVA SUPERFÍCIE A SER SOBRESCRITA NO ARRAY
                DESENHA CURSOR POR CIMA DA NOVA SUPERFÍCIE
        FIMSE
FIMENQUANTO
```

Para usar o buffer do mouse diretamente, basta acessá-lo na posição (0x6666666A). Segue um desenho de como ler a word desse endereço, sendo que a informação nos bytes 2,1 e 0 encontra-se da maneira como foi enviada pelo mouse e ainda precisa ser devidamente interpretada:

| Byte 3: Lixo | Byte 2: Informações relativas aos cliques | Byte 1: Informações relativas ao ΔX desde o último envio | Byte 0: Informações relativas ao ΔY desde o último envio |
|---|---|---|---|

*Organização do buffer do mouse*

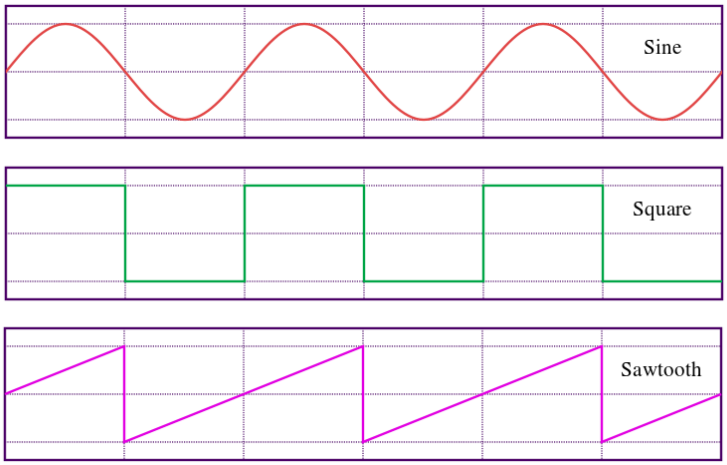
Ver exemplo: testemouse.s

- **Interface de áudio: Sintetizador Polifônico (by Lucas Carvalho)**

O sintetizador de áudio utilizado no curso de OAC é um sintetizador polifônico compacto capaz de tocar até 8 notas simultaneamente. Foi baseado no padrão MIDI e é formado por três sistemas:

**Oscilador:**

Gera formas de onda em várias frequências. É responsável por criar o som base de cada nota em 3 possíveis formatos:

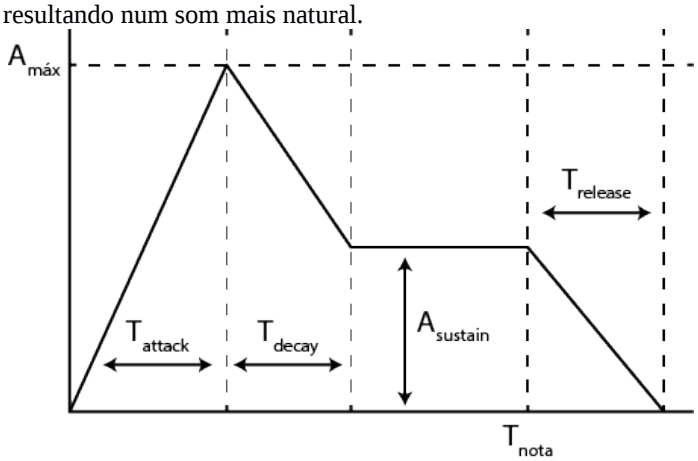



**Filtro:**

Um filtro passa-baixas pode ser ativado ou não na saída do oscilador. Sua frequência de corte é a própria frequência da nota e seu fator de qualidade é configurável.

**Envelope:**

O tempo de vida de uma nota é dividido em 4 seções: *Attack*, *Decay*, *Sustain* e *Release*. Elas determinam uma variação na amplitude da onda resultando num som mais natural.




**Interface**

Para a implementação do sintetizador no MIPS é necessário utilizar o módulo *Sintetizador*, que encapsula todos os sistemas.

```
/*
 * Entradas do CODEC de áudio da própria DE2.
 */
input AUD_DACLRCK;
input AUD_BCLK;
```

Nestas entradas devem ser passados os pinos da DE2 de mesmo nome. São dois pinos de entrada que controlam a taxa de amostragem do CODEC.

```
/*
 * Comando de início/fim de uma nota.
 */
input NOTE_PLAY;
input [6:0] NOTE_PITCH;
```

O sintetizador funciona recebendo comandos para que inicie ou termine uma nota. Por exemplo, caso queira tocar 3 notas simultâneas, você deve enviar 3 comandos de início seguidos.

Para enviar um comando atribua a *NOTE_PLAY* um valor de 1 para iniciar ou 0 para terminar uma nota. *NOTE_PITCH* determina a nota deste comando, seguindo o padrão MIDI (pesquise por *MIDI Note Table*).

Em toda subida de *AUD_DACLRCK*, um comando é lido. Comandos repetidos ou inválidos (terminar uma nota que nunca foi iniciada) são entendidos como se não houvesse comando.

```
/*
 * Configurações do oscilador.
 */
input [1:0] WAVE;
input NOISE_EN;
```

A entrada *WAVE* define qual forma de onda será gerada pelo oscilador, como mostra a seguinte tabela de referência.

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| Mudo | Senoide | Quadrada | Dente-de-serra |

O oscilador adicionará um certo ruído à onda gerada se *NOISE_EN* assumir valor 1. Este mecanismo é útil na criação de efeitos sonoros. Sintetizadores de vídeo games antigos utilizavam uma técnica similar para simular explosões, impacto, dano etc.

```
/*
 * Configurações do filtro.
 */
input FILTER_EN;
input [7:0] FILTER_QUALITY;
```

O filtro passa-baixas é ativado se a entrada *FILTER_EN* estiver alta. Sua frequência de corte é a própria frequência da nota sendo filtrada e seu fator qualidade pode ser configurado pela entrada *FILTER_QUALITY*, que é interpretada como ponto fixo Q8.

O fator de qualidade controla o quão limitada é a banda de frequências que passa pelo filtro. Caso assuma o valor zero, o filtro se comporta como um passa-baixas comum, sem ressonância. Uma onda quadrada filtrada com fator de qualidade alto se aproxima bastante de uma senoide.

```
/*
 * Configurações do envelope aplicado durante a vida de uma nota.
 */
input [6:0] ATTACK_DURATION;
input [6:0] DECAY_DURATION;
input [6:0] SUSTAIN_AMPLITUDE;
input [6:0] RELEASE_DURATION;
```

Toda nota tem uma envoltória aplicada a sua onda enquanto é tocada. Uma tecla de piano, ao ser pressionada, gera um som alto inicial que decai e mantém uma altura constante. Quando a tecla é solta a nota some aos poucos até ficar inaudível. Este processo é simulado em sintetizadores por meio do envelope de 4 seções mostrado anteriormente.

A duração de cada fase é configurável, exceto a de *Sustain* cuja amplitude é variável. A escala destes valores é, assim como definido no padrão MIDI, dependente da implementação, ou seja, é necessária uma certa experimentação.

```
/*
 * Amostra de saída do sintetizador.
 */
output [15:0] SAMPLE_OUT;
```

Os sistemas internos do sintetizador processam todas as notas e produzem uma nova amostra a cada subida do clock de amostragem *AUD_DACLRCK*. Esta amostra deve ser repassada ao CODEC da DE2. Este processo não é parte do sintetizador e já está implementado no processador MIPS do curso.

- **Interface de áudio: Sintetizador Polifônico (Grupo 1 - 2015/1)**

1 - Foi criada uma memória de dados de duas portas UserDataBlockDouble. Uma porta de escrita e leitura para o uso da CPU e uma porta somente de leitura para o sintetizador (na pasta item6/core/memoria). FOI RETIRADA ESSA CARACTERÍSTICA

2 - Outra PLL para gerar o AUD_CTRL_CLK para "afinar" o sintetizador.

3 - Outros Parâmetros adicionados (na pasta **item6/core/**):

```
NOTE_SYSCALL_ADDRESS        = 32'hFFFF0200,
NOTE_CLOCK                  = 32'hFFFF0204,
NOTE_MELODY                 = 32'hFFFF0208,
MUSIC_TEMPO_ADDRESS         = 32'hFFFF020C,
MUSIC_ADDRESS               = 32'hFFFF0210,
PAUSE_ADDRESS               = 32'hFFFF0214
```

4 - Adição do módulo SynthControl que toca a música sequencialmente na memória de dados. Está na pata **/item6/core/Sintetizador**.

Pequena explicações sobre a forma que a nota é codificada:

Existem 2 tipos de Configuração para a nota na memória, uma para syscall e outra para a música que será tocada sequencialmente. FOI RETIRADA ESTA CARACTERÌSTICA

**Configuração para notas que serão tocadas sequencialmente**:


1. **Melody** – 1 se for melodia; 0 se for acorde. Se esse bit for 1, a próxima nota só será tocada ao fim desta. Se for zero, a próxima nota será tocada na próxima borda de subida de AUD_DACLRCK.
1. **Instrument** – 16 possibilidades de instrumentos (0 - 15).
2. **Volume** - 128 possibilidades de amplitude da nota (0 – 127).
3. **Pitch** – 128 possibilidades de notas musicais segundo o padrão MIDI (0 – 127).
4. **End** – A última nota da música deverá setar esse bit para 1 para o controlador do sintetizador parar de percorrer a memória.
5. **Repeat** – Se esse bit estiver setado para 1, a próxima nota será a nota salva no endereço inicial.
6. **Duration (Q4)** – Duração em ponto fixo de cada nota. O valor é relativo à duração em milissegundos da Nota Base (Semínima). O valor da duração da Nota Base em milissegundos será passado para o endereço 0xFFFF020C. A partir daí o hardware atribuirá a duração em milissegundos equivalente para cada nota baseado em seus 8 primeiros bits (Duração – Ponto Fixo Q8).

„ = Semibreve 4 x Nota Base (0100.0000)

Ó = Mínima 2 x Nota Base (0010.0000)

Œ = Semínima Nota Base (0001.0000)

‰ = Colcheia ½ x Nota base (0000.1000)

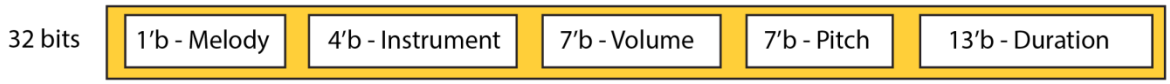
Ù = Semicolcheia ¼ x Nota Base (0000.0100)

Û = Fusa 1/8 x Nota Base (0000.0010)

Ì = Semifusa 1/16 x Nota Base (0000.0001)

Por exemplo, se a duração da Nota base for 400 milissegundos, a duração de (0000.0010) será 400 x 1/8 = 50 milissegundos.

## Configuração para notas do syscall (31 e 33):




A word será salva no endereço 0xFFFF0200.

1. **Melody** – 1 se for syscall 33; 0 se for syscall 31. Em resumo, se esse bit for 1, a próxima nota só será tocada ao fim desta. Se for zero, a próxima nota será tocada na próxima borda de subida de AUD_DACLRCK.
1. **Instrument** – 16 possibilidades de instrumentos (0 - 15). (SÓ 1 IMPLEMENTADO)
2. **Pitch** – 128 possibilidades de notas musicais segundo o padrão MIDI (0 – 127).
3. **Duration** – Duração da nota em milissegundos.

- **Interface Cartão SD: (Grupo-2 2016/1)**

A interface do cartão SD funciona por MMIO, onde seis bytes de memória são endereçados para o controlador do cartão SD, sendo eles:

- **SD_INTERFACE_ADDR – 0xFFFF250** (4 bytes): Recebe o endereço físico (não confundir com o endereço lógico) de memória do byte a ser lido do cartão SD. O endereço pode ser obtido lendo o cartão SD em um Hex Editor (editor de memória) em um computador.
- **SD_INTERFACE_CTRL – 0xFFFF0254** (1 byte): Fornece ao software informações do estado do controlador. Caso o valor lido seja 0x00, o controlador se encontra em estado READY, significando que o byte da última leitura está pronto para ser obtido e que uma nova leitura já pode ser realizada. Caso o valor lido seja diferente de 0x00, o controlador estará em estado BUSY, significando que o último byte pedido ainda não está pronto ou o cartão ainda não inicializou. O valor de SD_INTERFACE_CTRL em estado BUSY mostra qual o último comando enviado ao cartão SD e serve para depuração.
- **SD_INTERFACE_DATA – 0xFFFF0255** (1 byte): Fornece o byte lido do cartão.

A implementação utiliza o modo SPI de comunicação e a frequência SCLK de inicialização é de 400 KHz (máximo possível limitada pelo protocolo) e de 25 MHz para operações de leitura (máxima possível limitada pelo protocolo).

É necessário o uso de um cartão SD Standard Capacity (até 2 GB de armazenamento), uma vez que os estados de inicialização adicionais requeridos por cartões SDHC e SDXC não estão implementados. Além disso, o controlador envia um CMD16 ao cartão para a leitura de um único byte de dado por vez, mas cartões SDHC e SDXC não aceitam o comando CMD16 e requerem a leitura de um setor (512 bytes) por vez.

O syscall 49 recebe o primeiro endereço do byte a ser lido do cartão SD no registrador $a0, o endereço de destino no registrador $a1 e a quantidade de bytes a serem lidos no endereço $a2. O retorno do syscall é hardcoded em $v0 = 0 (sucesso), uma vez que o hardware de detecção de erro na leitura não foi implementado por falta de necessidade no presente momento.

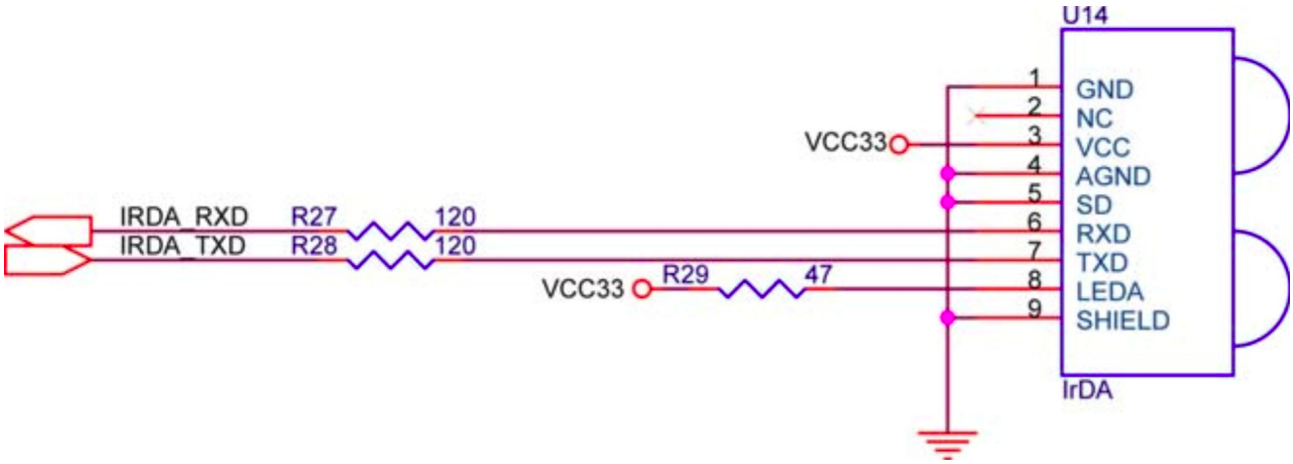
O arquivo de testes “testeReadSD.s” exemplifica como utilizar o cartão SD.

Detalhes quanto ao funcionamento de cartões SD podem ser encontrados em:

- http://elm-chan.org/docs/mmc/mmc_e.html
- http://www.chlazza.net/sdcardinfo.html
- https://www.sdcard.org/downloads/pls/pdf/index.php?p=part1_410-1.jpg&f=part1_410.pdf&e=EN_SS1

- **Interface Infravermelho: (Grupo 2 02/2016)**

A sigla IRDA (*Infrared Data Association*) faz referência a um conjunto de protocolos para comunicação sem fio entre dispositivos. A transferência de dados é feita na forma de pacotes que são enviados serialmente. A transmissão começa com o envio de 1 bit de início, seguido por 8 bits de dados e 1 bit de paridade e termina com a transferência de 1 bit de parada. Não é possível enviar e receber pacotes de dados simultaneamente. Para realizar a comunicação entre dispositivos, são necessárias uma porta de emissão (*receiver*) e uma porta de recebimento (*transmitter*). A figura abaixo esquematiza o transceptor IRDA contido em FPGAs Altera DE2.




**IRDA_CONTROL(0xFFFF0500)** (4 bytes)- wReadData[1] indica que tem dado pronto para leitura e wReadData[0] se existe dado sendo transmitido.

**IRDA_RXD (0xFFFF0504)** (4 bytes)- é de onde vai ser lido o dado do Irda

**IRDA_TXD (0xFFFF0508)** (4 bytes)– é o dado que vai ser transmitido pelo irda