Laboratório sobre Implementação de Sistemas Digitais com VHDL
Acesso à Memória Externa na Plataforma de Prototipação XS40/XST-1

Prática: Implementação de uma aplicação que faz acesso à memória externa da plataforma XS40/XST-1

Recursos: Ambiente de Desenvolvimento Active-HDL da Aldec, Inc, CAD Foundation e Ferramentas XSTools e Plataforma XS40/XST-1 da Xess, Inc.

Parte I – Introdução e Objetivos

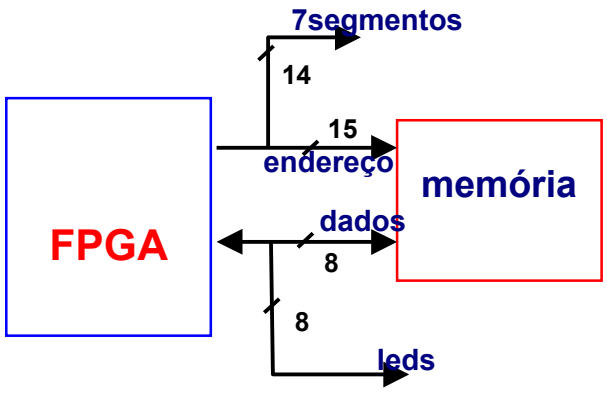
Os Laboratórios anteriores serviram para introduzir e compreender o uso da linguagem VHDL na implementação de sistemas digitais.

Este Laboratório tem por objetivo exercitar na prática os conceitos apreendidos pelo uso de recursos que empregam comunicação assíncrona como modo de trocar informações. Trata-se, neste caso, da comunicação entre o FPGA da plataforma de prototipação XS40/XST-1 e a memória estática (SRAM) disponível na mesma plataforma. Esta comunicação assíncrona é bastante simples, uma vez que funciona de forma unidirecional. Especificamente, a memória da XS40/XST-1 não tem como informar que conseguiu ler o dado colocado pelo FPGA no barramento no tempo que o FPGA o disponibilizou. Assim, a temporização de escrita, e sobretudo leitura da memória deve ser cuidadosamente planejada.

Ao final do laboratório, os alunos deverão ter compreendido uma forma particular de operação da comunicação assíncrona, bem como a descrição e implementação de sistemas assíncronos em VHDL. Os objetivos específicos envolvem o desenvolvimento de máquina de estados para controle de operações e a utilização de memórias externas.

Parte II - Uma palavra sobre a plataforma de prototipação XS40/XST-1 e o algoritmo de acesso a memória

A plataforma XS40/XST-1 possui um chip de memória RAM estática W24257AK-15, fabricado pela empresa Winbond Electronics Corp. Trata-se de uma memória estática de com organização 32.768 palavras (2^{15}) de 8 bits (32Kx8). O acesso se faz via um barramento bidirecional de 8 bits (1 palavra de cada vez), um barramento de endereços de 15 bits e três sinais de controle ativos em zero (CE-"chip enable", WE-"write enable" e OE-"output enable").



A plataforma de prototipação XS40/XST-1 usa multiplexação do barramento de endereços da memória com os fios para os mostradores de 7 segmentos, e do barramento de dados com os leds (ver Figura ao lado).

Desta forma, não é possível fazer acesso simultâneo aos recursos de saída (mostradores de 7 segmentos e leds) e a memória.

Mostra-se aqui um algoritmo muito simples de acesso à memória, iniciando por escritas seqüenciais (endereço 0 recebe valor, endereço 1 recebe valor+1, etc.), procedendo-se à leitura destes valores após 16 escritas. Este algoritmo pode ser assim descrito:

1. Aguardar o pressionamento da tecla ‘spare’. A seguir, iniciar processo de escrita:
   Registrador **mdr** recebe o valor das chaves dip-switch.
   Registrador **adram** recebe o valor 0.

2. Escrever, no endereço de memória indicado por **adram**, o conteúdo do registrador **mdr**.

3. Incrementar o valor dos registradores **mdr** e **adram**.
   Se o valor de **adram** for maior que 16, ir para o passo 4. Senão, voltar para o passo 2[1].

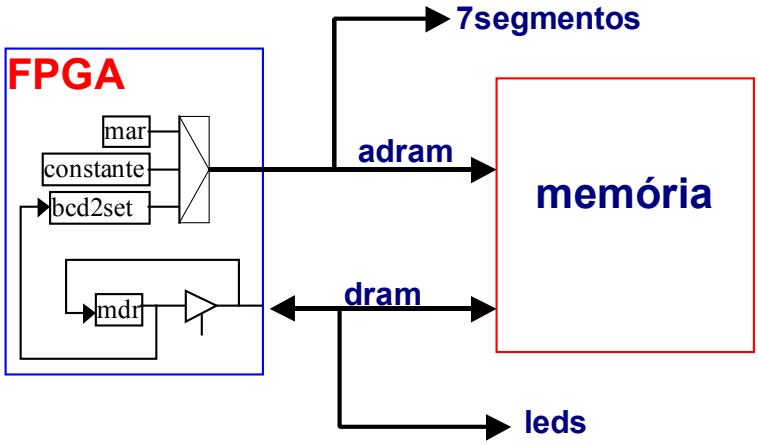
4. Esperar que seja pressionada a tecla ‘spare’. A seguir, zerar o registrador **adram**.

5. Ler o conteúdo de memória apontado por **adram** armazenando-o no registrador **mdr**.

6. Incrementar o registrador **adram**. Exibir o valor nos displays de sete-segmentos. Se o valor de **adram** for maior que 16 passar para o passo 7. Senão, voltar ao passo 5[2].

7. Terminar a execução, reiniciando o processo de escrita apenas quando for pressionada a tecla ‘reset’.

Parte III - Análise do circuito



Cada passo do algoritmo apresentado anteriormente pode ser implementado como um estado da máquina de estados de controle (S1 a S7), descrita na **Parte I** do código VHDL que aparece mais adiante nesta Seção.

O barramento *adram* (endereçamento da memória e mostradores de 7 segmentos) pode receber dados de 3 fontes distintas: registrador *mar*, quando desejo endereçar a memória; constante, quando desejo exibir uma constante nos displays; e o valor de *mdr* convertido para 7-segmentos, quando desejo exibir o valor lido da memória. A escolha de qual fonte será utilizada pelo *adram* é feita através de uma atribuição concorrente. O registrador *mar* tem por função armazenar e incrementar o endereço que vai ser utilizado para acesso à memória. O funcionamento dos barramentos *adram* e *mar* estão descritos na “**PARTE 2**” do código VHDL.

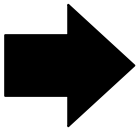
O barramento *dram* pode ser tanto entrada como saída, dependendo da ação efetuada sobre a memória. O registrador *mdr* tem por objetivo armazenar o valor contido nas chaves (*dipsw*), ou incrementar seu valor ou ser inicializado com o valor de *dram* (quando se lê da memória). Quando

[1] Os passos 2 e 3 escrevem seqüencialmente na memória, a partir do endereço de memória 0000H, o valor indicado pelas chaves, incrementado este endereço de 1 a cada novo endereço.
[2] Os passos 5 e 6 lêem seqüencialmente da memória, a partir do endereço de memória 0000H, exibindo cada valor lido nos mostradores de 7 segmentos.

se está escrevendo na memória (estados S2 e S3) *dram* recebe *mdr*, senão *dram* fica em alta impedância. Isto permite que dados provenientes da memória seja escritos em *mdr*. Os barramentos *dram* e *mdr* têm sua funcionalidade descrita na "**PARTE 3**" do código VHDL.

Como dito antes, A memória tem 3 sinais de controle ativos em zero: *OE*, para indicar leitura; *WE*, para indicar escrita; e *CE* para habilitar a pastilha de memória a funcionar (para leitura e/ou escrita). A geração destes sinais está descrita na "**PARTE 4**" do código VHDL.

Código VHDL



- **Buscar o arquivo com a descrição VHDL do projeto inicial em http://www.inf.pucrs.br/~calazans/undergrad/laborg/acesso.vhd. Iniciar um novo projeto no FOUNDATION. Neste laboratório não será utilizado o simulador Active-HDL. O código VHDL abaixo contém apenas o módulo principal.**
- **Complete a descrição, não esquecendo de colocar o arquivo UCF no projeto (ver em anexo, ao final deste texto).**
- **Sugestão: liste os sinais necessários antes de iniciar o projeto.**
- **Uma vez a descrição VHDL completa, faça a síntese do hardware e verifique se o circuito está funcionando corretamente.**
- **Depois desta etapa introdutória, há o trabalho propriamente dito, que é modificar este projeto.**

```
--------------------------------------------------------------------------------
-- Entidade acc_mem, grava e le da memoria
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.soma.all;

entity acc_mem is -- importante não alterar – comunicação com o arquivo UCF !!!
port
 ( dipsw: in std_logic_vector(8 downto 1); -- DIP switches
 spareb: in std_logic; -- SPARE pushbutton
 resetb: in std_logic; -- RESET pushbutton

 dram: inout reg8;          -- XS Board Ram data
 adram: out reg15;          -- XS Board Ram Adress

 oe: out std_logic; -- Output enable for all RAMs (negado)
 ce: out std_logic; -- Chip enable (negado)
 we: out std_logic; -- Read/Write (negado)

 clock: in std_logic -- Clock de 12Mhz (pino 13 do FPGA)

 );
end acc_mem;


architecture acc_mem of acc_mem is

    Aqui colocar os sinais necessários. Os componentes já estão no package.

begin

 U1 : OSC4 port map( mapear os pinos ao  sinais previamente declarados);
 U2 : debounce port map( mapear os pinos ao  sinais previamente declarados);
 U3 : display port map( mapear os pinos ao  sinais previamente declarados);
```

`U4 : display port map(` ***mapear os pinos ao sinais previamente declarados*** `);`

PARTE 1

```
---------- geração do proximo estado ----------
process(clock,resetb)
 begin
 if (resetb='0') then
 EA<=S1;
 elsif clock'event and clock='1' then
 EA<=PE;
 end if;
 end process;

---- maquina de estados para a escrita/leitura na memoria -----
---- controlada pela chave spare_b e o EA ----------
process(spareCK, EA)
begin
 case(EA) is
 when S1 => if spareCK='0' then PE<=S2; else PE<=EA; end if;
 when S2 => PE<=S3;
 when S3 => if mar(4)='1' then PE<=S4; else PE<=S2; end if;
 when S4 => if spareCK='0' then PE<=S5; else PE<=EA; end if;
 when S5 => if mar(4)='0' then PE<=S6; else PE<=S7; end if;
 when S6 => if spareCK='0' then PE<=S5; else PE<=EA; end if;
 when S7 => PE<=s7;
 end case;
end process;
```

Tarefa 1: Desenhe esta máquina de

PARTE 2

```
-- zera o mar nos estados S1 e S4, e o incrementa nos estados S3 e S5
process(clock,resetb,EA)
begin
 if (EA=S1 or EA=S4) then
 mar<= (others=>'0');
 elsif clock'event and clock='1' then
 if EA=S3 or EA=S5 then
 mar<=mar+1;
 end if;
 end if;
end process;

-- implementa o multiplexador para seleção de quem vai em adram
with EA select
 adram <= mar when S2 | S3 | S5,
 "111011111110111" when S4,
 not( setseg1(6 downto 0) & setseg2(7 downto 0) ) when S1 | S6 | S7;
```

PARTE 3

```
-- implementa o registrador mdr, inserindo nele o valor lido, no estado S5
process(clock,EA)
 begin
   if clock'event and clock='1' then
     if EA=S1 then mdr<=not dipsw;
     elsif EA=S3 then mdr<=mdr+1;
     elsif EA=S5 then mdr<=dram; -- captura dado vindo da memoria
     end if;
   end if;
 end process;

-- implementa o tri-state, ou seja, coloca dados quando for escrever
with EA select
 dram <= mdr when S2 | S3,
       "ZZZZZZZZ"  when others;
```

```
-- le da memoria apenas no estado S5
OE <= '0' when EA=S5 else '1';
-- escreve na memoria apenas no estado S2
WE <= '0' when EA=S2 else '1';
-- habilita a memoria apenas nos estados S2 e S5
CE <= '0' when EA=S5 or EA=S2 else '1';
 end acc_mem;
```

PARTE 4

Parte IV - Implementação de uma nova versão do algoritmo de leitura

Faça as modificações abaixo no projeto, sintetize e teste a nova versão:

- No início, ao invés de ler apenas uma informação das chaves, leia três:
  - Valor inicial a ser gravado na memória - *v0*;
  - Passo entre dois elementos consecutivos - *step*.
  - Número de elementos a gravar - *nb*;

	Conteúdo da Memória	
Endereço	**Versão original**	**Versão modificada**
0	**key**	***v0***
1	**key+1**	***v0+step***
2	**key+2**	***v0+2*step***
3	**key+3**	***v0+3*step***
4	**key+4**	***v0+4*step***
5	**key+5**	***v0+5*step***
6	**key+6**	***v0+6*step***
…	**…**	
15	**key+15**	

nb

Dicas para o correto funcionamento:

1. É necessário acrescentar apenas 2 (***dois***) estados adicionais na máquina de estado, para armazenamento das variáveis. No reset armazena-se o número de elementos, na primeira pressionada de *spare* o passo e na segunda pressionada de spare o valor de início de contagem.
2. Resumo das alterações necessárias no código:
   a) Declarar mais dois estados;
   b) Nos estados S3 e S5 fazer comparação com o número de elementos (processo da máquina de estados);
   c) No comando "with EA select …" indicar que é para utilizar os sete segmentos nos estados acrescentados;
   d) Modificar o processo que atribui valor a mdr, inserindo também os estados adicionais. Uma sugestão é:

```
.....
   if EA=S1 then      n <=not dipsw; mdr<=not dipsw;
    elsif EA=E1 then          step <=not dipsw; mdr<=not dipsw;
    elsif EA=E2 then          mdr <=not dipsw;
....
```

ARQUIVO COM OS PINOS DE CONEXÃO DO FPGA À PLACA (ARQUIVO UCF)

```
#Arquivo UCF
#
NET clock                LOC=P13;
#
# DIP SWITCH CONNECTIONS
NET dipsw<1>     LOC=P7;
NET dipsw<2>     LOC=P8;
NET dipsw<3>     LOC=P9;
NET dipsw<4>     LOC=P6;
NET dipsw<5> LOC=P77;
NET dipsw<6>     LOC=P70;
NET dipsw<7>     LOC=P66;
NET dipsw<8>     LOC=P69;
#
# PUSHBUTTON SWITCH CONNECTIONS
NET SPAREB               LOC=P67;
NET RESETB               LOC=P37;
#
#
# SRAM CONTROLLERS
NET ce                   LOC=P65;
NET oe                   LOC=P61;
NET we                   LOC=P62;
```

```
# SRAM DATA
NET dram<0>              LOC=P41;
NET dram<1>              LOC=P40;
NET dram<2>              LOC=P39;
NET dram<3>              LOC=P38;
NET dram<4>              LOC=P35;
NET dram<5>              LOC=P81;
NET dram<6>              LOC=P80;
NET dram<7>              LOC=P10;
#
# SRAM ADDRESS
NET adram<0>     LOC=P3;
NET adram<1>     LOC=P4;
NET adram<2>     LOC=P5;
NET adram<3>     LOC=P78;
NET adram<4>     LOC=P79;
NET adram<5>     LOC=P82;
NET adram<6>     LOC=P83;
NET adram<7>     LOC=P84;
NET adram<8>     LOC=P59;
NET adram<9>     LOC=P57;
NET adram<10>    LOC=P51;
NET adram<11>    LOC=P56;
NET adram<12>    LOC=P50;
NET adram<13>    LOC=P58;
NET adram<14>    LOC=P60;
```

Parte VI - A Fazer e a Entregar

- O projeto completo na sua versão modificada compactado pelo Foundation (opção de menu Design → Archive Design... do Ambiente Active-HDL), com formas de onda salvas em arquivo dentro do projeto;
- Relatório do projeto, contendo o diagrama de transição de estados da **Parte 1** do VHDL;
- Mostrar o projeto funcionando ao professor dentro de duas semanas.

Percentuais de nota atribuídos às Atividades:

Item Avaliado	Percentual
Projeto completo e simulações	**50%**
Relatório	**30%**
Demonstração do projeto, funcional ou não, ao professor, em aula	**20%**