1 2 UNIVERSIDADE D 9 0
COIMBRA

deer.uc
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Departamento de Engenharia Eletrotecnica e de Computadores

Projeto de Sistemas Digitais

Professor: Jorge Manuel Miranda Dias

Editor de texto.

Autor:
Gonçalo Bastos
Leonardo Cordeiro

Numero de Estudante:
2020238997
2020228071

23 Maio, 2024

Abstract

Este relatório descreve o desenvolvimento de um terminal de texto com editor de linha num ecrã VGA utilizando a placa Altera DE2. O projeto para alem dos modulos criados integra o VGA_SYNC para sincronização do ecrã, o CHAR_ROM para renderização de caracteres e o KEYBOARD para a decodificação de entradas de teclado. O design facilita capacidades básicas de edição de texto, como inserção, eliminação.

1 Introdução

Este projeto visa desenvolver um terminal de texto com funcionalidades de edição em linha, utilizando a placa Altera DE2. O sistema permite uma interação dinâmica com o usuário, incluindo a capacidade de apagar caracteres utilizando a tecla de retrocesso (backspace), inserir parágrafos através da tecla Enter e mover o cursor com as setas direcionais. Além disso, o terminal suporta a introdução de números e sinais de pontuação básicos. Na Implementação foram usados os módulos fornecidos pela biblioteca UP-Core da Altera, como o CHAR_ROM, VGA_SYNC E KEYBOARD.

2 Metodologia

Para a implementação da Emulação de um terminal de texto com editor de linha no ecrã VGA, sabíamos que teríamos de usar os módulos CHAR_ROM E VGA_SYNC combinados para podermos escrever para o ecrã, no entanto, começamos por pensar como vamos adquirir o que vamos escrever para o ecrã, ou seja, a nossa entrada de dados, dai termos começado pelo KEYBOARD.

Tratamento da entrada de dados

Para interagir com o teclado utilizamos então o modulo fornecido pela bibliote UP-Core da Intel, este lê os codigos das teclas (scan codes) enviados pelos teclado PS/2, como se pode ver na Figura 1. Tem duas entradas que recebem os dados do teclado PS/2 keyboard_clk (PS2_CLK) e keyboard_data (PS2_DAT), e a entrada de relógio. No BUS de saída scan_code[7..0] fica o código binário da tecla carregada ou libertada no teclado. A entrada reset limpa os registos internos.

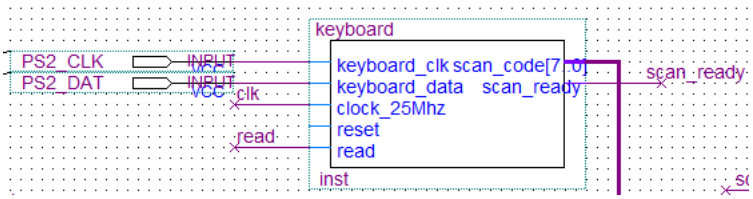

Figure 1: Modulo Keyboard

Apos uma pequena pesquisa sobre PS2 Keyboard Codes descubrimos que cada tecla possui um scan code único para quando é pressionada, chamado de "make code". Se mantiver a tecla pressionada, o make code é enviado repetidamente ao computador a uma taxa conhecida como taxa de repetição automática. Se várias teclas forem pressionadas e mantidas, apenas a última tecla pressionada se repetirá.

Quando solta uma tecla, um scan code diferente, conhecido como "brake code", é enviado para informar ao computador que a tecla foi libertada. Para a maioria das teclas, o brake code consiste no prefixo F0h seguido pelo make code, no entanto, as teclas estendidas (como SHIFT ou CTRL) e algumas exceções, como a tecla PAUSE, seguem regras diferentes, mas como as unicas teclas que vamos necessitar serão o SPACE, ENTER, BACKSPACE, e setas direcionais.

Fica-se a espera que uma tecla seja premida, quando uma tecla é premida, é feito o processamento do scan_code, ou seja, o sinal de entrada read é afirmado ('1'), indicando que o sistema deseja começar a ler um scan_code, a saída scan_ready é reiniciada para '0' para sinalizar que o módulo está ocupado e não está pronto para fornecer um novo scan_code.

Quando o scan_code esta pronto, scan_ready = 1, e o scan_code é recebido em binario (8bits) e só volta a zero quando read tiver uma vertente ascendente, implementou-se em VHDL (Figura 2) lógica adicional para detectar a chegada de novas teclas, implementando um handshake para ler apenas uma vez a cada novo código, para isso criou-se uma maquina de estados simples (Figura 3) que detecta quando uma tecla é premida, destingue e considera apenas make codes sobre os break codes, força um pulso na entrada 'read' do keyboard de forma a colocar a 0 a saida scan_ready do keyboard. O top-level view desta maquina de estados pode ser visto na figura 4, ao clock foi ligado o pixel o sinal de sincronismo com o VGA, proveniente do modulo VGA_SYNC.

Nota: Mais a frente na implementação tivemos de modificar o código da maquina de estados, adicionando a saída wr_en que controla as escritas para a memoria quando é detetado um carácter valido.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Keyboard_Controller is
    Port (
        scan_ready  : in STD_LOGIC;
        scan_code   : in STD_LOGIC_VECTOR(7 downto 0);
        wren        : out STD_LOGIC;
        read        : out STD_LOGIC;
        state           : in STD_LOGIC_VECTOR(2 downto 0); -- Estado atual
        next_state          : out STD_LOGIC_VECTOR(2 downto 0) -- Próximo estado
    );
end Keyboard_Controller;

architecture Behavioral of Keyboard_Controller is
    -- Sinais para os códigos de quebra
    signal break_code_F0: std_logic;
    signal break_code_E0: std_logic;

begin

    -- Mapear os break codes
    break_code_F0 <= '1' when (scan_code = "11110000") else  -- 0xF0
                     '0';
    break_code_E0 <= '1' when (scan_code = "11100000") else -- 0xE0
                     '0';

    -- Transição de estados baseada em sinais de entrada
    next_state <= "000" when (state = "000" and scan_ready = '0') else
          "000" when (state = "010") else
          "000" when (state = "100") else
          "001" when (state = "000" and scan_ready = '1') else
          "010" when (state = "001" and break_code_F0 = '0' and break_code_E0 = '0') else
          "011" when (state = "001" and (break_code_F0 = '1' or break_code_E0 = '1')) else
          "011" when (state = "011" and (break_code_F0 = '1' or break_code_E0 = '1')) else
          "100" when (state = "011" and break_code_F0 = '0' and break_code_E0 = '0') else
          "000";

    -- Lógica das saidas
    wren <= '1' when (state = "010") else
            '0';

    read <= '1' when (state = "010") else
            '1' when (state = "100") else
            '0';

end Behavioral;
```

Figure 2: Implementação VHDL do controlador do Keyboard.

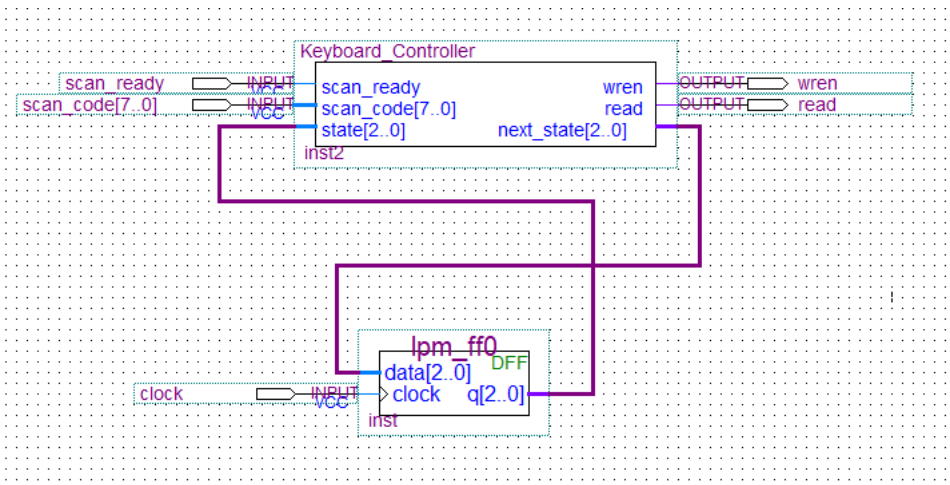

Figure 3: Keyboard Processor.

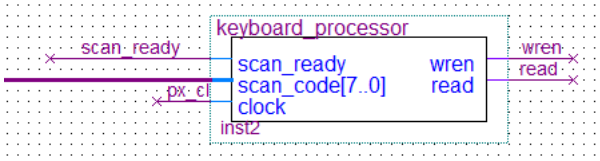

Figure 4: Top level view do processador.

Conversão dos scan codes

Para ter caracteres disponíveis para gerar texto no ecrã VGA, temos o módulo CHAR_ROM. Cada carácter é representado por uma matriz de pixeis 16x16. A entrada character_address do CHAR_ROM indica qual o carácter

```
char_address <= "000001" when (scan_code = x"1C" ) else -- A
        "000010" when (scan_code = x"32" ) else
        "000011" when (scan_code = x"21" ) else
        "000100" when (scan_code = x"23" ) else
        "000101" when (scan_code = x"24" ) else
        "000110" when (scan_code = x"2B" ) else
        "000111" when (scan_code = x"34" ) else
        "001000" when (scan_code = x"33" ) else
        "001001" when (scan_code = x"43" ) else
        "001010" when (scan_code = x"3B" ) else
        "001011" when (scan_code = x"42" ) else
        "001100" when (scan_code = x"4B" ) else
        "001101" when (scan_code = x"3A" ) else
        "001110" when (scan_code = x"31" ) else
        "001111" when (scan_code = x"44" ) else
        "010000" when (scan_code = x"4D" ) else
        "010001" when (scan_code = x"15" ) else
        "010010" when (scan_code = x"2D" ) else
        "010011" when (scan_code = x"1B" ) else
```

Figure 5: Parte do vhdl de implementação desta conversão

Figure 6: Simbolo do bloco criado para a conversão

pretendido (temos $2^6 = 64$ disponíveis) .
Como o carácter que tem de ser escrito para o VGA é proveniente do scan_code da tecla necessitamos de criar um ficheiro VHDL (Figura 5.) de um modulo que convertesse os scan codes PS2 que saem do modulo KEYBOARD para a entrada character_adress de forma a poderem ser usados pelo modulo CHAR_ROM. Este modulo recebe como entrada o scan code e tem como output o caractere correspondente que vai mais tarde endereçar o CHAR_ROM.

Implementação do buffer para os caracteres.

Neste ponto o nosso projeto, ao adicionar ao nosso esquematico o CHAR_ROM e o VGA_SYNC (Figura 6) com o objetivo de escrever no ecrã a letra premida e o mesmo foi conseguido, mas neste ponto apenas detetava a tecla premida, o carácter inserido anteriormente desaparece, e o mesma era mostrada por todo o ecrã.

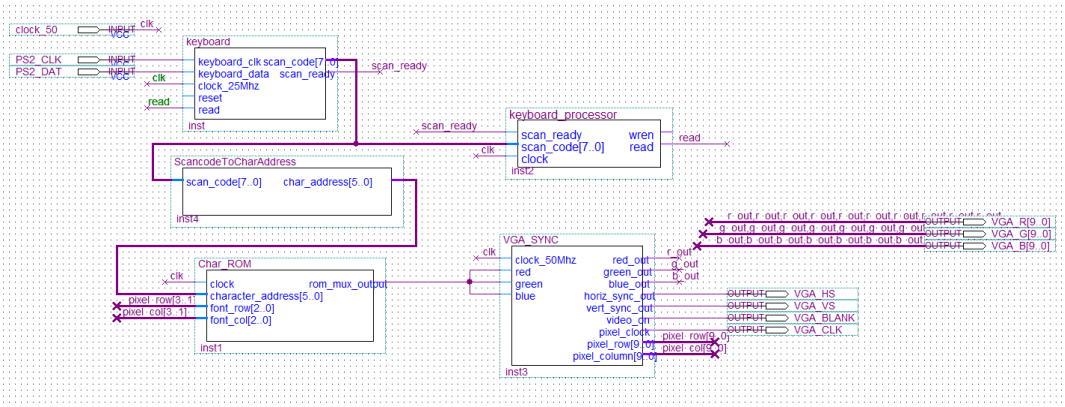

Figure 7: Top level view da versão 0 do editor de texto.

Tivemos então de arranjar um buffer que armazene os caracteres introduzidos, para isso utilizamos uma lpdm_ram_dp Single-Port RAM, como cada carácter ocupa uma região de 16x16 pixeis no ecrã e como temos uma resolução de 640x480 teremos de indexar 40x30 (1200) caracteres em memória, tendo sido escolhido o tamanho de 4096 words (de 12 bits), esta também foi inicializada com todas as suas células a decimal "20" que corresponde ao carácter SPACE, de forma a que sempre que seja iniciado o editor, este tenha o ecrã limpo.

Neste momento também verificamos a necessidade de gerenciar a escrita na memória e mais tarde também controlar onde os caracteres são exibidos no display VGA, o que nos leva a proxima secção.

Cursor

Desenvolvemos o módulo VHDL 'CursorControl' Figura 8, responsável por interpretar os scan_codes das teclas pressionadas e converter esses códigos em ações correspondentes, como movimentar o cursor para a próxima linha com a tecla Enter ou retroceder com a tecla Backspace. As teclas de seta são usadas para mover o cursor na tela sem alterar os caracteres já escritos.

Como podemos ver em parte na figura 9, a lógica interna do processo VHDL é acionada pela borda de ascendente do sinal wren, que é ativado a cada vez que se prime uma tecla. Diversas condições verificam qual tecla foi pressionada e, em seguida, atualizamos dois contadores (coluna e linha) que determinam a posição do cursor no display e na memória. As saídas col e row indicam a posição atual do cursor, enquanto a saída w_addr fornece o endereço calculado para escrita na RAM, baseado na posição do cursor; são ainda usadas constantes que verificam e mapeiam para as saidas sinais de deteção de 'backspace' e 'setas direcionais' para tratamento de casos especiais.

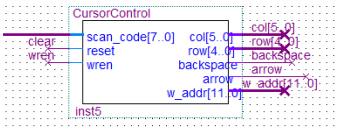

Figure 8: Símbolo do bloco CursorController.

```
            ----...---|

        elsif scan_code = SCAN_CODE_UP_ARROW then
            if row_counter = 0 then
                row_counter <= to_unsigned(29, row_counter'length);
            else
                row_counter <= row_counter - 1;
            end if;
            arrow <= '1';

        elsif scan_code = SCAN_CODE_DW_ARROW then

            if row_counter = 29 then
                row_counter <= (others => '0');
            else
                row_counter <= row_counter + 1;
            end if;
            arrow <= '1';

        else
            if col_counter /= 39 then
                col_counter <= col_counter + 1;
            else
                col_counter <= (others => '0');
                if row_counter /= 29 then
                    row_counter <= row_counter + 1;
                end if;
            end if;
            backspace <= '0';
            arrow <= '0';
        end if;
    end if;
end if;

col <= std_logic_vector(col_counter(5 downto 0));
row <= std_logic_vector(row_counter(4 downto 0));

w_addr <= std_logic_vector((row_counter * 40) + col_counter);

end process;
end Behavioral;
```

Figure 9: VHDL de controlo do cursor e calculo do endereço de escrita.

Sincronização e Controle de Escrita na RAM

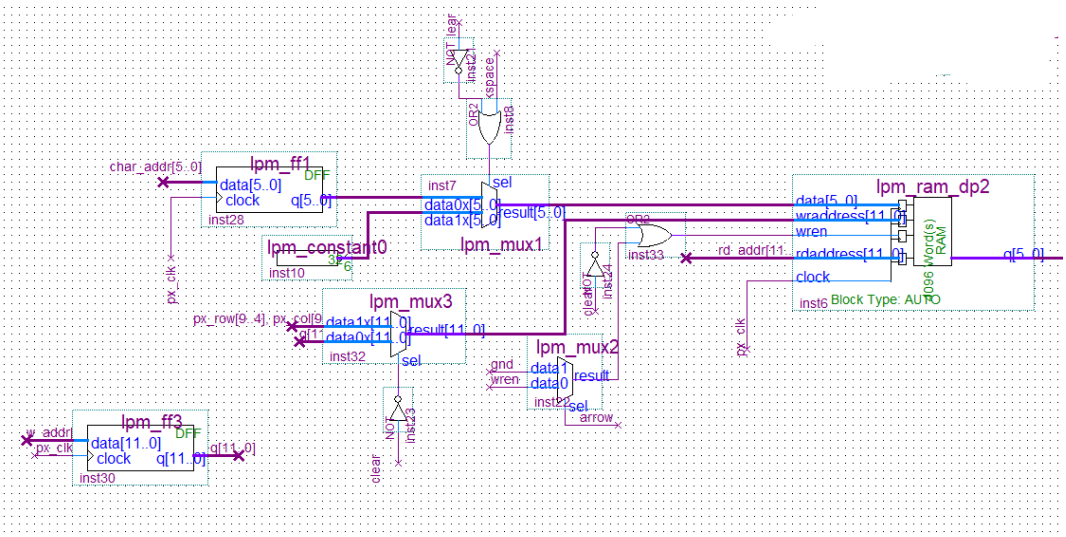

Figure 10: Sincronização e Controle de Escrita na RAM

A saída *w_addr* do módulo **CursorControl** representa o endereço de escrita calculado para a memória RAM. Esta é conectada à entrada *wradddress* da RAM, no entanto, devido à necessidade de sincronização, inserimos um flip-flop no caminho do sinal *w_addr* para introduzir um delay de um ciclo de pixel_clock, garantindo que o caractere seja escrito no endereço correto na memória e não no endereço subsequente, alinhando assim a ação de escrita com o estado atualizado do cursor. Temos ainda um mux que é usado para controlar a operação de limpeza do ecrã, quando acionado o *clear*, este multiplexador seleciona um valor que corresponde ao código de "espaço", efetivamente apagando o conteúdo do ecrã.

O flip-flop com a entrada *data[5..0]* cria um atraso que garante que a posição do cursor e o caractere escrito estejam sincronizados, evitando que o cursor seja sobreposto ao caractere recém-escrito. Utilizamos um multiplexador (*mux1*) que gerencia a lógica de escrita para situações especiais, como a inserção de um *Backspace* ou a ação de limpar o ecrã. O *mux1* decide entre o caractere a ser escrito e um sinal de limpeza com base na entrada *sel* controlada pelo estado da tecla pressionada.

A entrada *wr_en* da RAM é controlada pela máquina de estados *keyboard_controller*. Para gerenciar a interação com teclas direcionais, como as setas do teclado, implementamos lógica adicional no esquemático de nível superior. Quando uma tecla direcional é pressionada, queremos apenas mover o cursor sem executar uma operação de escrita. Isso é conseguido utilizando a flag *arrow* do **CursorControl**, que, quando ativa, assegura que *wr_en* seja mantido em '0', prevenindo assim uma escrita indesejada na RAM.

O endereço de leitura *rd_addr* é calculado usando no modulo **ReadAddress** (Figura 11) este calcula o endereço de escrita corretamente respeitando as dimensões do carácter e a resolução do ecrã VGA.

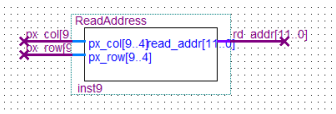

Figure 11: Símbolo do bloco ReadAddress.

```
entity ReadAddress is
    Port (
        px_col    : in  STD_LOGIC_VECTOR(9 downto 4);
        px_row    : in  STD_LOGIC_VECTOR(9 downto 4);
        read_addr : out STD_LOGIC_VECTOR(11 downto 0)
    );
end ReadAddress;

architecture Behavioral of ReadAddress is
begin
    Cursor: PROCESS(px_col, px_row)
        variable temp_address : INTEGER;
        variable row: integer;
        variable column: integer;
    begin
        -- converter a inteir, tomando apenas os bits [9 downto 4]
        row := to_integer(unsigned(px_row(9 downto 4)));
        column := to_integer(unsigned(px_col(9 downto 4)));
        --calculo do endereço
        temp_address := row * 40 + column;
        read_addr <= std_logic_vector(to_unsigned(temp_address, read_addr'length));
    .
```

Figure 12: VHDL de calculo do endereço de leitura.

Ligação com o VGA

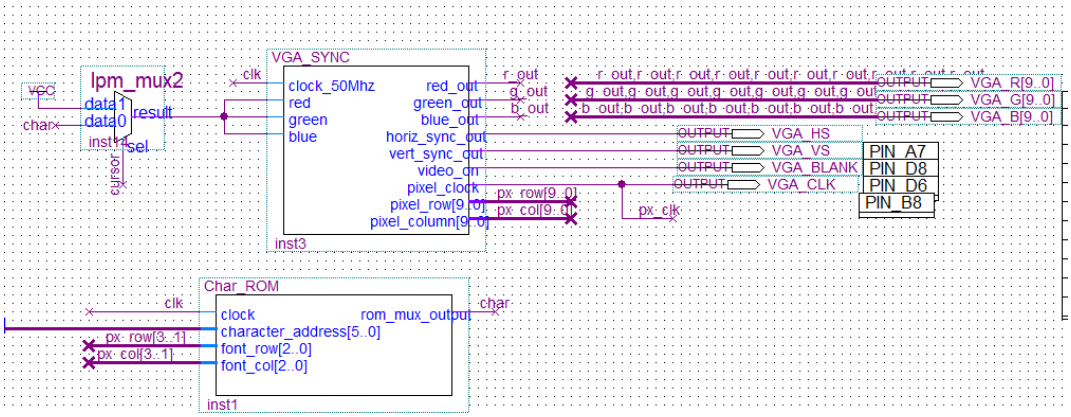

Figure 13: Ligação de CHAR_ROM e VGA_SYNC

Por fim o modulo **CHAR_ROM** trata de fazer a conversão dos endereços em memoria através da entrada character_address para o caracter pretendido, e as restantes entradas permitem fazer um varrimento da matriz 16x16 para ter na saída o pixel correspondente. A figura 13 mostra como combinamos os módulos VGA_SYNC e CHAR_ROM para escrever no ecrã VGA. O módulo VGA_SYNC, conforme representado no esquemático, é responsável por gerar os sinais de sincronização necessários para o monitor VGA, as entrada de cor são definidas de modo a que a cor de fundo seja preto e a cor dos caracteres (ou do cursor) a branco, é ainda usado um mux para o desenho do cursor, esta seleção e feita com base em duas comparações, basicamente se o cursor estiver na mesma posição que *pixel_col* e *pixel_row* que indicam a posiçao do pixel atual desenha-se o cursor, como se pode ver na figura 14.

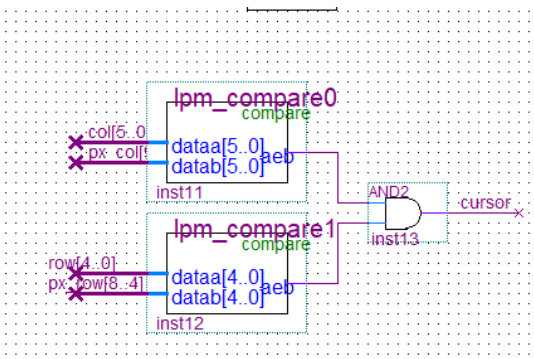

Figure 14: Desenho do cursor

3 Visão top-level do projeto

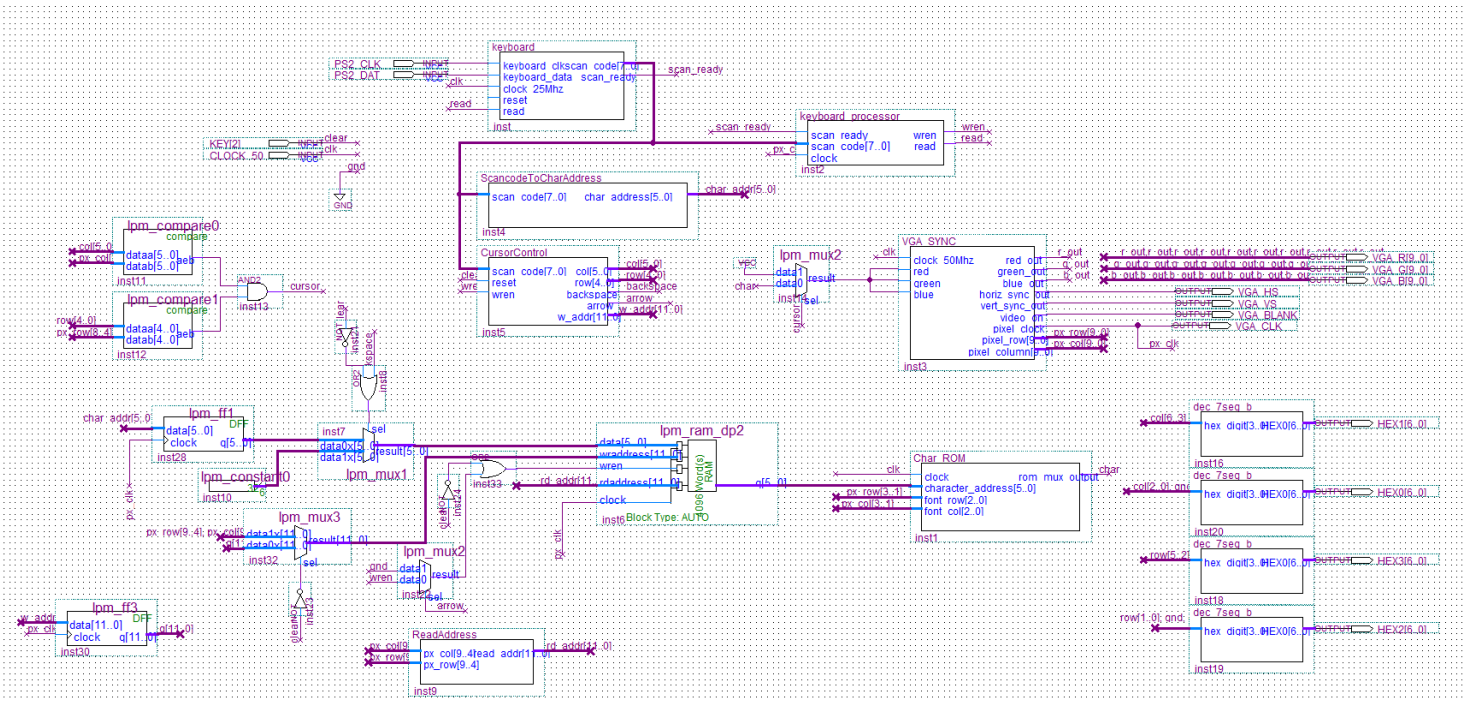

Figure 15: Top Level View

4 Conclusão

Concluímos com sucesso o desenvolvimento de um Editor de Texto para FPGA, um sistema que capta entradas de teclado e as exibe num monitor VGA. As etapas fundamentais incluíram a integração do CHAR_ROM com o VGA_SYNC para renderização de texto e a criação do CursorControl para uma navegação e edição de texto eficazes. Implementamos sincronização meticulosa para as operações de escrita em RAM, garantindo uma interação precisa com o usuário.